International Conference

REAL-TIME AND NETWORK SYSTEMS

RTNS'06

Poitiers, France
May 30-31, 2006

Sponsored by:

COM'SCIENCE PROGRAM
CNRS/ GdR ASR
ENSMA

Edited by:

Guy JUANOLE and Pascal RICHARD

PREFACE

This is the fourtheenth in the series of conferences on Real Time Systems. The aim of these conferences is to provide a forum for the presentation, by academic researchers and practitioners, of original works which cover the technological and scientific topics in the area of the distributed real time systems: design process (the different phases between the requirement specification till the implementation) and operational life.

The first thirdteens (from 1993 to 2005) were held in the environment of the "Real Time Systems" Exhibition in Paris (at first, Palais des Congrés Porte Maillot, and then Paris Expo-Porte de Versailles). In 2005, in Paris, it was decided, at first, to make a conference independent of the Exhibition and more academic oriented,, second, to emphasize the role of Systems on Networks (hence the transformation of the name from RTS to RTNS) and, then, to organize the 2006 conference in Poitiers.

In response to the call for papers, 29 papers were submitted and 18 were selected by the Program Committee, which has permitted to organize seven sessions (2 on "Uniprocessor scheduling", 1 on "Networks", 2 on "Resources and Data management" and 1 on "Worst Case Execution Time"). In addition, we have been very fortunate to secure the service of the excellent international speaker Professor S.K.Baruah (North Carolina University at Chapel Hill, USA) who will be giving an invited paper titled "Multiprocessor Real-time Scheduling Theory: Questions (many) and answers (a few)". These 19 presentations will provide an interesting snapshot of research results and directions covering conference topics.

The quality of the program is due to the authors who submitted papers and to the members of the program Committee and extra referees who have given their time to provide excellent reviews (three for each paper). We are sincerely grateful to all of them.

We would like to thank the local organization committee, the conference secretary Claudine Rault, Frédéric Ridouard for the conference web site and Michaël Richard for the proceedings. Many thanks to six students from the Business and Administration Management Department of the University Institute of Technology (University of Poitiers) for their major activities in the organization committee. Special thanks to Nicolas Navet who organized the previous edition and provided lot of useful information, and also to Françoise Simonot-Lion for her helpful contacts for finding the scientific support of the CNRS. Finally, we are grateful to the institutions and people that helped to prepare and organize the event.

Guy Juanole and Pascal Richard
Program Co-chairs

PREFACE

C’est la quatorzième édition de la série de conférences sur les systèmes temps réel. L’objectif de ces conférences est d’établir un forum de présentations de chercheurs et d’industriels sur des travaux originaux qui couvrent les sujets scientifiques et techniques dans le domaine des systèmes temps réel distribués : méthodes de conception (les différentes phases entre les spécifications à la mise en œuvre) et leur vie opérationnelle.

Les éditions précédentes (entre 1993 et 2005) se sont déroulées dans le cadre du salon "systèmes temps réel" à Paris (Palais des Congrés Porte Maillot, puis ensuite Paris Expo-Porte de Versailles). En 2005, à Paris, il a été décidé, premièrement, de rendre la conférence indépendante du salon et de l’orienter vers un public plus académique, et deuxièmement, de prendre en compte le rôle des réseaux dans ces systèmes (d’où la transformation de l’acronyme de la conférence de RTS à RTNS) et, enfin, d’organiser l’édition 2006 à Poitiers.

En réponse de l’appel à communications, 29 articles ont été soumis et 18 ont été sélectionnés par le comité de programme, qui ont permis d’organiser 7 sessions (2 sur "l’ordonnancement monoprocesseur", 1 sur les "réseaux", 2 sur "la gestion des ressources et des données" et 1 sur "le calcul des pires temps d’exécution"). En supplément, nous avons l’honneur d’accueillir le professeur S.K.Baruah (North Carolina University à Chapel Hill, USA) qui présentera un exposé invité intitulé "Théorie de l’ordonnancement temps réel multiprocesseur : questions (beaucoup) et réponses (quelques-unes)". Ces 19 présentations fournissent une vue intéressante des résultats de recherche et des thèmes de recherche couvrant les thèmes de la conférence.

La qualité du programme est due aux soumissions des auteurs ainsi qu’aux membres du comité de programme et aux relecteurs extérieurs qui ont prodigués d’excellentes révisions des articles (trois par article). Nous sommes sincèrement reconnaissants envers chacun d’entre eux pour le travail accompli.

Nous tenons à remercier tous les membres du comité local d’organisation et Claudine Rault, la secrétaire de la conférence, Frédéric Ridouard pour le site web de la conférence, ainsi que Michaël Richard pour les actes du congrès. Nous remercions aussi les six étudiants du département "Gestion des Entreprises et Administrations" de l’Institut Universitaire de Technologie de Poitiers pour leurs importantes activités au sein du comité d’organisation. Nous tenons tout spécialement à remercier Nicolas Navet, qui a organisé la précédente édition de la conférence, pour l’ensemble des informations qu’il nous a transmis, ainsi que Françoise Simonot-Lion pour ses contacts qui ont permis d’avoir le soutien scientifique du CNRS. Enfin, nous sommes très reconnaissants aux institutions et personnes qui ont permis de préparer et organiser cet événement.

Guy Juanole et Pascal Richard
Présidents du comité de programme

PROGRAM CO-CHAIRS/PRESIDENTS DU COMITE DE PROGRAMME

Guy JUANOLE (LAAS, Toulouse, France) and Pascal RICHARD (LISI, Poitiers, France)

PROGRAM COMMITTEE/COMITE DE PROGRAMME

L. Almeida (University of Aveiro, Portugal)
P. Amer (University of Delaware, USA)
Ch. André (I3S, Sophia Antipolis, France)
S.K. Baruah (University of North Carolina, USA)
A. Cervin (Lund Institute of Technology, Sweden)
F. Cottet (LISI, ENSMA, Poitiers, France)
J.-D. Decotignie (CSEM, Neuchâtel, Suisse)
A.-M. Déplanche (IRCCyN, Nantes, France)
J.A. Fonseca (University of Aveiro, Portugal)
J.M. Fuertes (Technical Univ. of Catalonia, Spain)
J. Goossens (ULB, Bruxelles, Belgique)
Z. Hanzalek (Tech. Univ., Prague, Czech Republic)
T.W. Kuo (National Taiwan University, Taiwan)
F. Lepage (CRAN, UHP Nancy, France)
L. Lo Bello (University of Catania, Italy)
Z. Mammeri (IRIT, UPS Toulouse, France)
P. Minet (INRIA-Rocquencourt, France)
N. Navet (LORIA, Nancy, France)
N. Nissanke (London South Bank Univ., UK)
I. Puaut (IRISA, Rennes, France)
G. Rodríguez-Navas (Univ. of Balearic Islands, Palma de Mallorca)
T. Sauter (Austrian Acad. of Sciences, Austria)
M. Silly-Chetto (IRCCyN, Nantes, France)
D. Simon (INRIA-Rhônes Alpes, France)
F. Simonot-Lion (LORIA-INPL, Nancy, France)
D. Simplot-Ryl (LIFL, Lille, France)
L. Thiele (ETH, Zürich, Switzerland)
Y. Trinquet (IRCCyN, Nantes, France)
F. Vasques (University of Porto, Portugal)
F. Vernadat (LAAS, Toulouse, France)
L.T. Yang (St. Francis Xavier University, Canada)

ORGANIZATION COMMITTEE/COMITE D'ORGANISATION

F. Carreau (LISI/ENSMA, Poitiers)
B. Chauvière (LISI/ENSMA, Poitiers)
S. Enon (IUT GEA Poitiers)
J. Foulny (IUT GEA Poitiers)
A. Geniet (LISI/ENSMA, Poitiers)
D. Geniet (LISI/ENSMA, Poitiers)
P. Girard (LISI/ENSMA, Poitiers
E. Grolleau (LISI/ENSMA, Poitiers)
P. Occelli (IUT GEA Poitiers)
S. Pailler (LISI/ENSMA, Poitiers)
E. Parrault (IUT GEA Poitiers)
C. Rault (LISI/ENSMA, Poitiers)
M. Richard (LISI/ENSMA, Poitiers)
P. Richard (LISI/ENSMA, Poitiers)
F. Ridouard (LISI/ENSMA, Poitiers)
A. Sauquet (IUT GEA Poitiers)
E. Soulat (IUT GEA Poitiers)
K. Traore (LISI/ENSMA, Poitiers)

LIST OF REVIEWERS/LISTE DES RELECTEURS

L. Almeida (University of Aveiro, Portugal)

P. Amer (University of Delaware, USA)

Ch. André (I3S, Sophia Antipolis, France)

S.K. Baruah (University of North Carolina, USA)

I. Calvo (Universidad del Pais Vasco, Portugal)

A. Cervin (Lund Institute of Technology, Sweden)

F. Cottet (LISI, ENSMA, Poitiers, France)

J.-D. Decotignie (CSEM, Neuchâtel, Suisse)

A.-M. Déplanche (IRCCyN, Nantes, France)

S. Faucou (IRCCyN, Nantes, France)

J.A. Fonseca (University of Aveiro, Portugal)

J.M. Fuertes (Technical Univ. of Catalonia, Spain)

J. Goossens (ULB, Bruxelles, Belgique)

Z. Hanzalek (Tech. Univ., Prague, Czech Republic)

P.E. Hladik (IRCCyN, Nantes, France)

H.R Hsu (National Taiwan University, Taiwan)

C.M. Hung (National Taiwan University, Taiwan)

T.W. Kuo (National Taiwan University, Taiwan)

F. Lepage (CRAN, UHP Nancy, France)

L. Lo Bello (University of Catania, Italy)

Z. Mammeri (IRIT, UPS Toulouse, France)

P. Minet (INRIA-Rocquencourt, France)

N. Navet (LORIA, Nancy, France)

N. Nissanke (London South Bank Univ., UK)

M. Peca (CTU Prague, Czech Republic)

I. Puaut (IRISA, Rennes, France)

G. Rodríguez-Navas (Univ. of Balearic Islands, Palma de Mallorca)

T. Sauter (Austrian Acad. of Sciences, Austria)

M. Silly-Chetto (IRCCyN, Nantes, France)

D. Simon (INRIA-Rhônes Alpes, France)

F. Simonot-Lion (LORIA-INPL, Nancy, France)

D. Simplot-Ryl (LIFL, Lille, France)

Y.Q Song (Loria, Nancy, France)

M. Sousa (University of Porto, Portugal)

L. Thiele (ETH, Zürich, Switzerland)

Y. Trinquet (IRCCyN, Nantes, France)

J. Trdlicka (CTU, Prague, Czech Republic)

F. Vasques (University of Porto, Portugal)

F. Vernadat (LAAS, Toulouse, France)

L.T. Yang (St. Francis Xavier University, Canada)

C.Y. Yang (National Taiwan University, Taiwan)

Contents

Invited talk

Multiprocessor Real-time Scheduling Theory: questions (many) and answers (a few)

Sanjoy K. Baruah
University of North Carolina at Chapel Hill, USA.

Abstract:

Due to various inherent advantages of multiprocessor platforms, real-time application systems are increasingly coming to be implemented upon such platforms. However, theoretical developments have not kept pace: currently, our formal understanding of the behavior of such multiprocessor systems is approximately where our knowledge of uniprocessor systems was in the early 1970's. There is consequently a need for developing a theory of multiprocessor real-time scheduling that is as complete and sophisticated as uniprocessor real-time scheduling theory currently is, and that will prove as useful to the designers of real-time systems as uniprocessor real-time theory does today. In this presentation, I will propose a research agenda for multiprocessor real-time scheduling theory that aims to address this need. I will also briefly outline the progress that has been made thus far towards achieving the goals in this agenda.

Uniprocessor Scheduling I

Worst-case analysis of feasibility tests for self-suspending tasks

Frédéric Ridouard, Pascal Richard
LISI-ENSMA
Av C. Ader, Téléport 2 BP 40109
86961 Futuroscope Cedex, France
{frederic.ridouard,pascal.richard}@ensma.fr

Abstract

In most real-time systems, tasks invoke external operations processed upon dedicated processors. External operations introduce self-suspension delays in the task behaviors. In such task systems, checking that deadlines will be meet at run-time is $\mathcal{NP}$-Hard in the strong sense. For that reason, known response time analysis (RTA) only compute upper bounds of worst-case response times. These pessimistic estimations lead in practice the designers of a real-time system to oversize the computer features. The aim of this paper is to quantify the pessimism used in known RTA methods. We propose an exact exponential time feasibility test and define upper bounds of competitive ratio of three known RTA techniques.

Keywords: Real-time, On-line scheduling, self-suspension, Maximum response time.

1 Introduction

A real-time system is a system in which the correctness of the system depends not only on correctness of computations, but also on the time at which the results are produced (if a result is late, it is a fault). A real-time system can be seen as a task system where each task must respect its constraints. A task meets its deadline if it completes its execution before its deadline otherwise the task misses its deadline. There exists a feasible schedule for a task system if all deadlines are met.

Several models of recurring real-time tasks have been defined. The simplest but also the most fundamental model is provided by the *periodic task model* of Liu and Layland [8]. In this model, a periodic task τ has only two characteristics $\tau = (C, T)$: C is the worst-case execution requirement of task τ and T its period between two successive releases. Consequently, an instance of the periodic task τ (a job) is generated and released in the system after T units of times with an execution requirement equals to C. A job must complete its execution before the next release (T units of time later). Tasks are assumed to be independent.

Most of real-time systems contain tasks with self-suspension. A task with a self-suspension is a task that during its execution prepares specifics computations (e.g. In/Out operations or *FFT* on a digital signal processor). The task is self-suspended to execute the specifics computations upon external dedicated processors. External operations introduce self-suspension delays in the behavior of tasks. The task waits until the completion of the external operations to finish its execution. Generally, the execution requirement of external operations can be integrated in the execution requirement of the task. But, if self-suspension delays are large, then such an approach cannot be used to achieve a schedulable system. Thus self-suspension must be explicitely considered in the task model.

We have already proved [13] that the feasibility problem of scheduling task systems is $\mathcal{NP}$-Hard in the strong sense. We have also shown the presence of scheduling anomalies under *EDF* for scheduling independent tasks with self-suspension upon an uniprocessor platform when preemption is allowed. We have proved [14] that classical on-line scheduling algorithms are not better than 2 competitive to minimize the maximum response time. In this paper, we show that on-line and deterministic scheduling algorithms are not optimal to schedule tasks with self-suspension. The Response Time Analysis (*RTA*) can only compute upper bounds of worst-case response times in a reasonable amount of time. These pessimistic estimations lead in practice to oversize the computer features. The aim of this paper is to quantify the pessimism used in three known *RTA* methods based on fixed-priority task systems.

Several feasibility tests are presented and defined for analysing tasks allowed to self-suspend. For fixed-priority task systems, there exist tests based on the computation of

worst-case response time: Kim *et al.* [7], Jane W. S. Liu [9] and Palencia *et al.* [11, 12]. The latter approach can be used for *EDF* scheduling [12]. There exists also a test based on the utilization factor of the processor [4]. But, no study concerning the quality of these tests are known to exhibit relative merits of these methods. Consequently, our approach is to analyze the relevance and quality of these tests.

We next analyse the feasibility tests of Kim *et al.* [7] and Liu [9] to schedule tasks with self-suspension. Before, we define the task model (Section 2). In Section 3, the feasibility tests of Kim and Liu are presented. In Section 4, we present the main technique to evaluate the on-line algorithms. In Section 5, we show that it is impossible to define an optimal on-line algorithm to schedule tasks systems when tasks are allowed to self-suspend. Lastly, the feasibility tests are analyzed to determine their pessimism.

2 Task model

We consider that task systems are based on a collection of periodic and independent tasks. Let I be a task system of n tasks. Every occurrence of a task is called a job. Every task τ_i ($1 \leq i \leq n$) arrives in the system at time 0, its relative deadline is denoted D_i and its period T_i. If its relative deadline is equal to the period, the task has a implicit deadline else if just $D_i \leq T_i$ constrained deadline. The maximum execution requirement of a task τ_i is C_i.

In the system, preemption of tasks is allowed. Consequently, a job can be suspended at any time to allow the execution of others jobs and later on will be resume to continue its execution.

To simplify our results , we consider that tasks are allowed to self-suspend at most once. The Figure 1 presents this model. Every task τ_i ($1 \leq i \leq n$) has two subtasks (with a maximum execution requirement $C_{i,k}$, $1 \leq k \leq 2$) separated by a maximum self-suspension delay X_i between the completion of the first subtask and the start of the second subtask. Such delays change from one execution to another since they model execution requirements of external operations. Consequently every task τ_i is denoted: $\tau_i : (C_{i,1}, X_i, C_{i,2}, D_i)$.

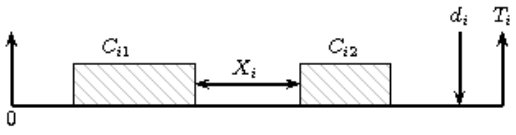


Figure 1. Task model

The utilization factor of a periodic task τ_i, is the ratio of its execution requirement to its period: $U(\tau_i) = C_i/T_i$. The utilization factor of a task system τ is the sum of the utilization factors of all tasks: $U(\tau) = \sum_{i=1}^{n} U(\tau_i)$.

The maximum response time R_i of a task τ_i is equal to the difference between the completion time and the release date. To minimize the maximum response time of a task set is to minimize $\max R_i$.

A task set is said *feasible* if there exists a schedule such that all tasks are completed by their deadlines at run-time.

3 Presentation of feasibility tests

In the following section,we present three feasibility tests:

- *Kim et al. [7]*: To define their feasibility tests, they use the works of Wellings [16] and Ming *et al.* [10]. They define two tests based on the same principle : to consider a task with a self-suspension in two independent tasks without any suspension delay.
- *Jane W. S. Liu [9]* : This feasibility test determines the blocking time due to self-suspension and higher-priority tasks.

3.1 Feasibility tests of Kim *et al.* [7]

Wellings *et al.* [16] studied the tasks with self-suspension but with $C_{i,1} = 0$. The self-suspension is called release jitter [3, 16]. A release jitter for a task is the difference of time between arrival and release time. Consequently, they use task set in which each task has a release jitter. To determine the response time of a task τ_i, they use the following recurrence relation:

$$R_i^0 = C_i$$
$$R_i^{n+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^n + X_j}{T_j} \right\rceil C_j \qquad (1)$$

The recurrence stops if $R_i^{n+1} = R_i^n$. And the worst-case response time of τ_i is $R_i^n + X_i$. To prove that the task τ_i is schedulable, $R_i^n + X_i$ must be less than or equal to D_i.

Ming *et al* (cf [10]) have modified the recurrence relation of Wellings (1) to take into account any task with a self-suspension:

$$R_i^0 = C_i + X_i$$
$$R_i^{n+1} = C_i + X_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^n + X_j}{T_j} \right\rceil C_j \qquad (2)$$

However, Ming *et al.* consider the suspension delay as a part of execution requirement. But external operations are scheduled upon dedicated processors. Consequently, such an approach can increase unnecessarily the worst-case response times of tasks. Kim *et al.* (cf [7]) define two new feasibility tests to compute worst-case response times of tasks with self-suspensions.

3.1.1 Method A of Kim

They consider that $D_i \leq T_i$ for all i and tasks can be preempted. This first method subdivide each task τ_i with self-suspension in two independent tasks without suspension :

- $\tau_{i,1}$, released at time r_i without release jitter and with a processing requirement of $C_{i,1}$.
- $\tau_{i,2}$, released at time r_i, its jitter $J_{i,2}$ equals X_i and a processing requirement equal to $C_{i,2}$.

The two generated tasks inherit the period and the deadline of τ_i.

To prove the schedulability of task τ_i, we must transform τ_i into $\tau_{i,1}$ and $\tau_{i,2}$, and we then calculate the worst-case response time of the generated tasks. $\tau_{i,1}$ has a release jitter equal to 0 and $\tau_{i,2}$ has one equal to X_i. The worst-case of $\tau_{i,1}$ and $\tau_{i,2}$ are calculated independently. To calculate the worst-case response time of $\tau_{i,1}$, the Wellings's formula (1) is used:

$$R_{i,1}^{n+1} = C_{i,1} \quad + \quad \sum_{j=1}^{i-1} \left\lceil \frac{R_{i,1}^n}{T_j} \right\rceil C_{j,1} \quad + \quad \sum_{j=1}^{i-1} \left\lceil \frac{R_{i,1}^n + X_j}{T_j} \right\rceil C_{j,2}$$

Computations stop for the smallest positive integer n satisfies $R_{i,1}^{n+1} = R_{i,1}^n$ and the worst-case response time $R_{i,1}^*$ of $\tau_{i,1}$ is equal to $R_{i,1}^n$. If $R_{i,1}^* \leq D_i$ then $\tau_{i,1}$ is schedulable. Otherwise, we cannot conclude that $\tau_{i,1}$ is schedulable.

The worst-case response time of $\tau_{i,2}$ is calculated with the following recurrent formula:

$$R_{i,2}^{n+1} = C_{i,2} \quad + \quad \sum_{j=1}^{i-1} \left\lceil \frac{R_{i,2}^n}{T_j} \right\rceil C_{j,1} \quad + \quad \sum_{j=1}^{i-1} \left\lceil \frac{R_{i,2}^n + X_j}{T_j} \right\rceil C_{j,2}$$

The worst-case response time $R_{i,2}^*$ of $\tau_{i,2}$ is calculated. To finish, if $(R_{i,1}^* + X_i + R_{i,2}^*) \leq D_i$, then τ_i is schedulable, otherwise we cannot conclude.

3.1.2 Method B of Kim

This approach is an improvement of Ming's method (*cf.* Formula 2). This method consider the suspension delays as part of processing requirement of tasks. But without this assumption, during the interval of time X_i, other tasks can be scheduled. To calculate the worst-case response time of a task, X_i can be reduced and furthermore the worst-case response time of τ_i can be shortened. Consequently, to calculate the worst-case response time of a task τ_i, the following recurrent formula is used:

$$R_i^{n+1} = C_i \quad + \quad M_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_{j,1} \quad + \quad \sum_{j=1}^{i-1} \left\lceil \frac{R_i^n + X_j}{T_j} \right\rceil C_{j,2}$$

Where $M_i = X_i - \sum_{j=1}^{i-1} \left\lfloor \frac{X_i}{T_j} \right\rfloor C_j$

If $R_i^{n+1} = R_i^n$ and $R_i^n \leq D_i$ then τ_i is schedulable. Otherwise, we cannot conclude if it is schedulable or not.

Remark 1 *Since $M_i \leq X_i$, if τ_i is schedulable with the Ming's method (cf. Formula 2), then the task is schedulable with the method B of kim.*

3.2 The Liu's method [9]

To take into account the extra delay suffered by a task τ_i due to its own self-suspension and the suspension of higher-priority tasks, Liu [9] considers this delay as a factor of blocking time of τ_i, denoted $b_i(ss)$.

The blocking time of a task due to its own suspension is not more than X_i. To determine the blocking time due to a higher-priority task τ_k, we must study two cases:

- τ_k cannot delay τ_i during more than C_k units of time since the task τ_k can be scheduled (or partially scheduled) during the suspension of τ_i because the processor is idle.
- Moreover, if $X_k < C_k$ then the blocking time cannot be more than X_k units of time.

Consequently, the blocking factor due to each higher-priority tasks, τ_k is never more than the suspension delay of τ_k and never more than C_k.

Finally, the blocking time $b_i(ss)$ is equal to:

$$b_i(ss) = X_i + \sum_{k=1}^{i-1} \min(C_k, X_k)$$

Note that Liu's method is not expected to perform as well as the Kim's methods, since it does not specify where the suspension occurs within the task.

4 Validation of on-line algorithms

4.1 Introduction

This paper is interested by the validation of on-line algorithms. For any objective function, we wish to know the quality of the solution obtained with an on-line scheduling algorithm (hereafter referred to as the performance guarantee of the algorithm). This quality will not be better than the quality obtained by an optimal off-line algorithm. Two commonly used methods to evaluate the performance of an on-line algorithm are known:

- The simulation : The on-line scheduling algorithms are compared and evaluated in the confine of a stochastic model.
- The competitive analysis : The on-line algorithm is compared with an optimal off-line algorithm for the same problem so that the on-line algorithm achieves its worst-case results.

4.2 The simulation

The simulation allows to compare the on-line algorithms. To evaluate the performance of an on-line algorithm, this method defines a stochastic model by assuming a certain probabilistic distribution to compute task features. With this model, a task system is generated and it is submitted to every on-line algorithm.
However, the on-line algorithm is then evaluated within the confine of the stochastic model. Moreover, this approach is inconsistent with the environments of on-line algorithms. Because the probabilistic distribution model based on past observations will always model the future arrivals of jobs. But, as pointed out by Karp [6], this assumption is inconsistent with the nature of on-line algorithms unless the future resemble to the past.

4.3 The competitive analysis

The first results of this approach are the results obtained by *Sleator and Tarjan* [15] in 1985. This approach compares the on-line algorithm to an optimal clairvoyant algorithm in the worst-case. The optimal off-line algorithm (said *the adversary*) defines the instances of problem to compare the two algorithms. But a good adversary defines instances of problem so that the on-line algorithm achieves its worst-case performance. To analyse deterministic algorithms, two equivalent adversaries can be used:

- The oblivious adversary defines the task system in advance based on the characteristics of the on-line algorithm, and serves it optimally.
- The adaptive on-line adversary defines the next request of tasks according to the decision taken by the on-line algorithm, but serves it immediately.

An algorithm that minimizes a measure of performance, is c-competitive if the performance obtained by the on-line algorithm is less than or equal to c times the value of the optimal algorithm. More formally, given an on-line algorithm A and a task system I, the performance obtained by the on-line algorithm A (*Resp.* the adversary) in scheduling I is denoted $\sigma_A(I)$ (*Resp.* $\sigma^*(I)$). Consequently, A is c-competitive if there exists a task system I and a constant c so that $\sigma_A(I) \leq c\sigma^*(I)$.

The competitive ratio c_A of an on-line algorithm A is the worst-case ratio while considering any instance I.

Definition 1 *The competitive ratio, c_A, of the on-line algorithm A to minimize a performance criterion while considering any instance I is:*

$$c_A = \sup_{tout I} \frac{\sigma_A(I)}{\sigma^*(I)}$$

5 On-line algorithms are not optimal

In this section, we demonstrate that there exists no on-line optimal algorithm to schedule task systems when tasks are allowed to self-suspend upon uniprocessor systems.

Theorem 1 *No on-line deterministic algorithms are optimal to schedule tasks systems when tasks are allowed to self-suspend upon a uniprocessor system.*

Proof :
To prove this theorem, we use the competitive analysis with an adaptative adversary (cf. Section 4.3). Hence, we define a task system and according to the scheduling decision of any on-line and deterministic algorithm, the adversary defines the next request of tasks so that the on-line algorithm misses a deadline and the adversary serves it optimally. We define a task system I and we show that no on-line deterministic algorithm can schedule optimally I. We consider that at time 0 two tasks are available:

$$\tau_1 : C_{1,1} = 1, X_1 = 7, C_{1,2} = 1, D_1 = 10, T_1 = 10$$
$$\tau_2 : C_{2,1} = 1, X_2 = 4, C_{2,2} = 1, D_2 = 9, T_2 = 10$$

Let A be an on-line algorithm. At time 0, to make its scheduling decision, A has two choices:

1. *The on-line algorithm* A does not schedule τ_2 at time 0 (this schedule is presented Figure 2.a). Either it schedules τ_1 or it lefts the machine idle. In the two cases, it schedules τ_2 at time t, with $0 < t \leq 3$ to respect the deadline of τ_2 (For the Figure 2.a, $t = 2$). But at time 3, an other task τ_3 with a period equals to 10 is released:

$$\tau_3 : C_{1,1} = 1, X_1 = 2, C_{1,2} = 3, D_1 = 9$$

At time 3, the on-line algorithm A schedules τ_3 since it has not laxity. But A has not enough time to complete τ_2 and τ_3 before their common deadline at time 9. Consequently A has done a bad choice and hence, the scheduling of I under A is not feasible.

The optimal off-line algorithm schedules at time 0, τ_2 and at time 1, τ_1. At time 3, τ_3 is released and immediately run. At time 5, τ_2 is resumed from its self-suspension and completed at time 6. Finally, τ_3 is completed at time 9 and τ_1 at time 10. Figure 2.b presents the scheduling of I under an optimal off-line algorithm.

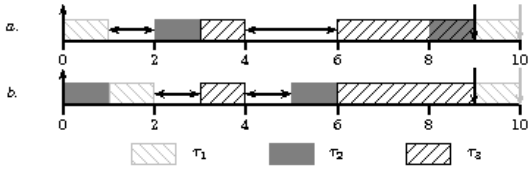


Figure 2. The on-line algorithm A does not schedule τ_2 at time 0 but at time $t = 2$

Consequently, we show that if an on-line and deterministic algorithm chooses to not run τ_2 at times 0, it is not optimal since there exits a feasible schedule of I.

2. When A schedules τ_2 at time 0, at time 1, it must schedule τ_1 to respect its deadline. At time 2 arrives τ_4 with implicit deadline ($T_4 = D_4$):

$$\tau_4 : C_{1,1} = 4, X_1 = 3, C_{1,2} = 1, T_1 = 10$$

A schedules τ_4 at time 2 (to respect its deadline) and at time 6, τ_2 is resumed from its self-suspension and scheduled. But between time 9 and 10, A must complete τ_1 and τ_4, hence it is impossible. A cannot schedule the task system I. Figure 3.a presents the scheduling of I under A.

The optimal off-line algorithm schedules at time 0, τ_1, at time 1, τ_2 and at time 2, τ_4. At time 6, τ_2 is resumed and completed at time 7. Finally, τ_1 is completed at time 9 and τ_4 at time 10. Figure 3 presents the scheduling of I obtained by the adversary.

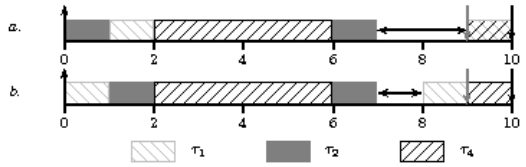


Figure 3. The on-line algorithm A schedules τ_2 at time 0

Consequently, there exits no optimal on-line and deterministic algorithm to schedule task systems upon uniprocessor system when tasks are allowed to self-suspend.

□

6 Analysis of feasibility tests

6.1 Introduction

In this section, the feasibility tests presented in the Section 3 are analyzed. We use the two validation techniques presented in Section 4. These feasibility tests compute the upper bound of the maximum response time of every task. At first time, we establish the pessimism of these estimations. To determine this pessimism, we use the approach presented by Epstein and Rob Van Stee in [5]. They provided lower bounds for on-line deterministic (or randomized) algorithms for several optimization criteria. They studied problems in term of competitive analysis. They automatically generated a huge number of synthetic task sets. For each task set, they computed the competitive ratio for the on-line algorithm studied. Finally, they kept the task set with the worst competitive ratio. In our work, the optimality criteria is the minimization of the maximum response time. We use the same method: we generate, with a brute force generation (as done in [1] for non-preemptive system), a huge number of task sets and for each feasibility test we keep the task set leading to the worst competitive ratio. To complete this analysis, we define a stochastic model to generate a lot of task systems. With these task systems, statistics are defined to compare these tests.
The feasibility tests presented in this paper are based on fixed-priority task systems. Consequently, to use the competitive analysis (*cf.* Section 4.3), we don't use an optimal off-line algorithm as adversary but we use the fixed priority algorithm (*RM*).

6.2 Simulation environment

The first constraint is to obtain schedulable task systems. The utilization factor of generated task systems is bounded by 0.7. Decreasing the utilization factor is an important parameter for generating feasible task systems.

The presence of anomalies for scheduling tasks with self-suspension in fixed-priority task system [13, 14] increases the costs of computations since reducing processing requirement can lead to worst-case response times of tasks.

The feasibility tests are based on the fixed-priority scheduling algorithm *RM*. But, we proved [13, 14] that scheduling anomalies can occur while scheduling tasks with self-suspension under *RM*. Consequently, if the execution requirement of a task is decreased of one unit of time, the response time can increase and a deadline can be missed. Hence, to determine the exact worst-case response time of task systems where tasks are allowed to self-suspend, we must test all possible processing requirements (and suspension delays) for each job of each task.

Remark 2 *C_i (resp. X_i) is the upper limit to its processing requirement (resp. worst-case suspension delay) of task τ_i. Consequently, we consider that the execution requirement (resp. suspension delay) of a task can vary between 1 and C_i (resp. X_i) since all parameters are integers. Moreover, $C_{i,1}$, X_i and $C_{i,2}$ belong to the interval $[1, 4]$.*

We define two rules to reduce the hyperperiod length:

- To minimize the length of the hyper period, the tasks are synchronous.
- To minimize the computations and the length of the hyper period, tasks have harmonic periods (Definition 2).

Definition 2 *Let $I : (\tau_1, \tau_2, \ldots, \tau_n)$ be a task system. I has harmonic periods if and only if the two following properties are respected:*

$$T_1 \leq T_2 \leq \cdots \leq T_n$$
$$\forall i, i \in \{2, \ldots, n\}, T_i \mod T_{i-1} = 0$$

Remark 3 *We assume that tasks are indexed in increasing order of periods. The second property limits the length of the feasibility interval and the number of jobs within it. Consequently the task with the smallest priority is τ_n. Since we use the fixed priority scheduling algorithm, τ_n has the longest period.*

To generate a task: first, the executive requirements and the suspension delays are computed. Finally, to determine the period, the period of the task previously generated, is multiplied until the utilization factor (of the task system) is less than 0.7.
Finally, we consider tasks with implicit deadlines. Moreover, the generated task systems contain only two or three tasks to firstly limit time while computing exact response times, and secondly to exhibit task systems leading to worst-case performance guarantees.

6.3 Lower bounds

6.3.1 Introduction

Next subsections detail task sets automatically generated by our simulator leading to the worst-case performance of the three considered feasibility tests.

6.3.2 Method A of Kim

Lower bound 1 *The lower bound of the competitive ratio for the feasibility test of the method A of Kim to minimize the maximum response time while scheduling tasks allowed to self-suspend at most once is $2,91667$.*

Proof:
Let I_A be the following task system containing three tasks:

$$\tau_1 : C_{1,1} = 3, X_1 = 2, C_{1,2} = 3, T_1 = 12$$
$$\tau_2 : C_{2,1} = 3, X_2 = 1, C_{2,2} = 1, T_2 = 96$$
$$\tau_3 : C_{3,1} = 1, X_3 = 1, C_{3,2} = 1, T_3 = 96$$

The upper bound of the maximum response time obtained with the method A of Kim denoted σ_i^A for each task τ_i of I_A is:

$$\tau_1 : \sigma_1^A = 8, \tau_2 : \sigma_2^A = 17, \tau_3 : \sigma_3^A = 35$$

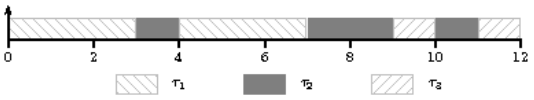


Figure 4. The exact maximum response time obtained by *RM* while scheduling I_A.

Figure 4 presents the exact maximum response time obtained with the fixed-priority scheduling algorithm *RM*. There are no scheduling anomalies and for that reason, tasks are scheduled with their worst-case execution requirements and suspension delays. At time 0, *RM* schedules τ_1 and during its suspension it schedules partially τ_2. At time 7, τ_2 is scheduled. At time 9, τ_2 is suspended and τ_3 scheduled. Finally at time 11, τ_2 finishes its execution and τ_3, at

time 12. Hence, the exact maximum responses time of tasks (denoted σ_i^{RM}) are:

$$\tau_1 : \sigma_1^{RM} = 8, \tau_2 : \sigma_2^{RM} = 11, \tau_3 : \sigma_3^{RM} = 12$$

Consequently, the worst-case competitive ratio for I is:

$$\begin{aligned} c_A^{RM} &= \sup_{anyI} \frac{\sigma_A(I)}{\sigma_{RM}(I)} \geq \frac{\sigma_A(I_A)}{\sigma_{RM}(I_A)} \\ &\geq \max\left(\frac{\sigma_1^A}{\sigma_1^{RM}}, \frac{\sigma_2^A}{\sigma_2^{RM}}, \frac{\sigma_3^A}{\sigma_3^{RM}}\right) \\ &\geq \frac{\sigma_3^A}{\sigma_3^{RM}} = \frac{35}{12} = 2.91667 \end{aligned}$$

□

6.3.3 Method B of Kim

Lower bound 2 *The lower bound to minimize the maximum response time for the method B of Kim is equal to* $2,75$.

Proof:
Let I_B be the following task system:

$$\begin{aligned} &\tau_1 : C_{1,1} = 1, X_1 = 1, C_{1,2} = 3, T_1 = 6 \\ &\tau_2 : C_{2,1} = 1, X_2 = 3, C_{2,2} = 2, T_2 = 270 \\ &\tau_3 : C_{3,1} = 3, X_3 = 2, C_{3,2} = 3, T_3 = 810 \end{aligned}$$

The upper bound obtained with the second method of Kim for each task τ_i of I_B and denoted σ_i^B is equal to:

$$\tau_1 : \sigma_1^B = 5, \tau_2 : \sigma_2^B = 22, \tau_3 : \sigma_3^B = 35$$

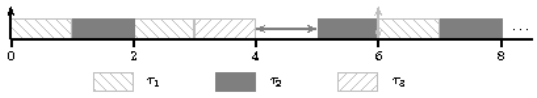


Figure 5. The exact maximum response time obtained by *RM* while scheduling I_B.

Figure 5 presents the exact maximum response time of task τ_2. At time 0, *RM* schedules τ_1 since it has the highest priority. At time 1, τ_1 is suspended and τ_2 scheduled. At time 2, τ_2 is suspended and τ_1 is completed. At time 3, τ_3 is scheduled during the suspension of τ_2. At time 6, τ_1 is released and at time 8, τ_2 is completed. Consequently, we obtain the following exact response times:

$$\tau_1 : \sigma_1^{RM} = 5, \tau_2 : \sigma_2^{RM} = 8, \tau_3 : \sigma_3^{RM} = 24$$

Consequently, the competitive competitive ratio for the second method of Kim is:

$$\begin{aligned} c_B^{RM} &= \sup_{anyI} \frac{\sigma_B(I)}{\sigma_{RM}(I)} \geq \frac{\sigma_B(I_B)}{\sigma_{RM}(I_B)} \\ &\geq \max\left(\frac{\sigma_1^B}{\sigma_1^{RM}}, \frac{\sigma_2^B}{\sigma_2^{RM}}, \frac{\sigma_3^B}{\sigma_3^{RM}}\right) \\ &\geq \frac{\sigma_2^B}{\sigma_2^{RM}} = \frac{22}{8} = 2.75 \end{aligned}$$

□

6.3.4 Jane W. S. Liu's method

Lower bound 3 *The competitive ratio on RM obtained with the method of Liu is* $2,875$ *to minimize the maximum response time for tasks are allowed to self-suspend at most once.*

Proof:
We use the instance I_B defined in the Theorem 2. The upper bound of maximum response time obtained with the method of Liu for each task τ_i of I_B and denoted σ_i^L are:

$$\tau_1 : \sigma_1^L = 5, \tau_2 : \sigma_2^L = 23, \tau_3 : \sigma_3^L = 47$$

Figure 5 presents the exact maximum response time obtained with *RM* while scheduling I_B. These results are:

$$\tau_1 : \sigma_1^{RM} = 5, \tau_2 : \sigma_2^{RM} = 8, \tau_3 : \sigma_3^{RM} = 24$$

Hence, the competitive ratio obtained for the method of Liu to minimize the maximum response time is equal to:

$$\begin{aligned} c_L^{RM} &= \sup_{anyI} \frac{\sigma_L(I)}{\sigma_{RM}(I)} \geq \frac{\sigma_L(I_B)}{\sigma_{RM}(I_B)} \\ &\geq \max\left(\frac{\sigma_1^L}{\sigma_1^{RM}}, \frac{\sigma_2^L}{\sigma_2^{RM}}, \frac{\sigma_3^L}{\sigma_3^{RM}}\right) \\ &\geq \frac{\sigma_2^L}{\sigma_2^{RM}} = \frac{23}{8} = 2.875 \end{aligned}$$

□

6.3.5 Comparison of feasibility tests

These results show that the method B of *Kim* obtained the best results. But, we cannot conclude that the best feasibility test is the method B. Because, the only possible conclusion is that these feasibility tests are not comparable. We cannot conclude since it is possible for each feasibility test to determine tasks sets where the feasibility test is the best test but it is the worst for another. Kim *et al.* have already proved that their two methods are not comparable [7]. Now to prove that all the tests are not comparable, we show that

the method A of *Kim* and the method of *Liu* are not comparable:
Let I be the following task set:

$$\tau_1 : C_{1,1} = 2, X_1 = 3, C_{1,2} = 1, T_1 = 7$$
$$\tau_2 : C_{2,1} = 1, X_2 = 3, C_{2,2} = 2, T_2 = 56$$
$$\tau_3 : C_{3,1} = 3, X_3 = 1, C_{3,2} = 2, T_3 = 392$$

The competitive ratio obtained for the three tests:

$$\sigma_A^{RM}(I) = 1.47$$
$$\sigma_B^{RM}(I) = 1.30$$
$$\sigma_L^{RM}(I) = 1.80$$

The ratio obtained with the method A of *Kim* is better than the ratio obtained with the method of *Liu*.

Let I' be the following task set:

$$\tau_1 : C_{1,1} = 2, X_1 = 3, C_{1,2} = 1, T_1 = 9$$
$$\tau_2 : C_{2,1} = 2, X_2 = 3, C_{2,2} = 3, T_2 = 45$$
$$\tau_3 : C_{3,1} = 2, X_3 = 2, C_{3,2} = 1, T_3 = 90$$

The ratios on *RM* are:

$$\sigma_A^{RM}(I') = 1.69$$
$$\sigma_B^{RM}(I') = 1.06$$
$$\sigma_L^{RM}(I') = 1.56$$

But with this task set, the method of *Liu* is better than the method A of Kim. To conclude all tests are not comparable. Consequently, we can define a last test: the best method.

6.3.6 The Best Method

This method consist in applying for **every task** all tests and to store the smallest computed response time. Such a method can help to decrease the competitive ratio (but we have no formal proof of that fact).

Lower bound 4 *The competitive competitive ratio obtained while considering for each task system the best feasibility test is* $2,16667$ *to minimize the maximum response time for tasks allowed to self-suspend at most once.*

Proof:
Let I_C be the following task system:

$$\tau_1 : C_{1,1} = 1, X_1 = 1, C_{1,2} = 3, T_1 = 9$$
$$\tau_2 : C_{2,1} = 1, X_2 = 3, C_{2,2} = 1, T_2 = 72$$
$$\tau_3 : C_{3,1} = 3, X_3 = 2, C_{3,2} = 1, T_3 = 648$$

The exact maximum response time obtained with the algorithm *RM* are:

$$\tau_1 : \sigma_1^{RM} = 5, \tau_2 : \sigma_2^{RM} = 6, \tau_3 : \sigma_3^{RM} = 14$$

The Table 1 presents for each task, the competitive ratio obtained with each feasibility test.

Tasks	Method A of Kim	Method B of Kim	Method of Liu
τ_1	1.00	1.00	1.00
τ_2	2.17	2.17	2.33
τ_3	1.57	1.43	1.64

Table 1. Competitive ratio for each task of I_C

Consequently, the competitive competitive ratio ,C_{Bst} for this method is:

$$\begin{aligned} c_{Bst}^{RM} &= \sup_{anyI} \frac{\sigma_{Bst}(I)}{\sigma_{RM}(I)} \geq \frac{\sigma_L(I_C)}{\sigma_{RM}(I_C)} \\ &\geq \sup_{1 \leq i \leq 3} \{\inf\{c_A^{RM}(\tau_i), c_B^{RM}(\tau_i), c_L^{RM}(\tau_i)\}\} \\ &\geq 2.16667 \end{aligned}$$

□

6.4 Simulation results

6.4.1 Introduction

In this section, we present numerical results obtained during the brute force generation described in Section 6.2. All tests (upper bounds and the exact test) have been applied to every generated task set. We are aware that such a simulation environment is not sufficient (*cf.* [2]) to exhibit relative merits of the considered feasibility tests that they are only valid in the confine of our stochastic model (see Section 6.2).

6.4.2 Results

To obtain relevant results from a statistical point of view, we generated one million of tasks sets. The tasks sets are generated with the procedure defined in Section 6.2. The Table 2 presents statistical results obtained by the simulator for the feasibility tests.
The first row of the Table 2 presents the percentage of times where every feasibility test has been the best one (while scheduling task sets). The method B of *Kim* leads to the best results.
The average competitive ratios in row 2 of the Table 2 allows us to remark that the method B of *Kim* is the feasibility test arriving in first position. But even if the percentage of the method of *Liu* is equal to zero, this feasibility test has a average less than the average of the method A of *Kim*.

With the standard deviations (row 3 of the Table 2), the feasibility test with the smallest standard deviation is the method A of *Kim*.

Feasibility tests	Method A of Kim	Method B of Kim	Method of Liu
Best method	3.64%	99.8%	$\approx$ 0.00%
Average ratios	1.65	1.21	1.50
Standard deviations of ratios	0.18	0.20	0.22

Table 2. Results of simulation for the feasibility tests for task systems with 2 or 3 tasks

7 Conclusion

In this paper, we have presented some results on tasks allowed to self-suspend at most once. For such task systems there exists no on-line optimal algorithm. We also presented the performances of three different feasibility tests. For these tests, our aim was to compute their pessimisms since they compute an upper bound of the exact maximum response time of tasks. To determine this pessimism, we use the approach of Epstein and Van Stee [5] and also the competitive analysis. But the feasibility tests are not compared to an optimal algorithm, but to the fixed-priority on-line algorithm *RM*. Hence, we shown that the competitive ratio of feasibility tests are between 2.75 and 2.91667 implying the designers of real-time system to oversize the computer features. We also shown that feasibility tests are not comparable. But, if for each task, we apply every feasibility test and we retain the best then the competitive ratio decreases to 2.16667. Finally, for task sets with a small number of tasks and exactly one self-suspension per task, the method B of Kim *et al.* is the best one.

In further works, an interesting issue is to analyze others feasibility tests and to consider a more general stochastic environment (with task sets having a larger number of jobs). Also to extend the approach to the *EDF* scheduling policy and the feasibility test of Palencia [12] (based on *EDF*). An other interesting issue can be to consider dependent tasks (tasks with shared resources or precedence constraints) [11].

References

[1] I. Alzeer, P. Molinaro, and Y. Trinquet. Calcul exhaustif du temps de rponse de tches et messages dans un systme temps rel rparti. *In Proceedings of the* 13^th *Real-Time Systems*, 2005.

[2] E. Bini and GC. Buttazzo. Biasing effects in schedulability measures. *IEEE Proceedings of the* 16^th *Euromicro Conference on Real-Time Systems (ECRTS04), Catania, Italy*, July 2004.

[3] A. Burns. Preemptive priority-based scheduling: An appropriate engineering approach. *in Advances in Real-Time Systems, S.H. Son, Ed.,Prentice Hall, New Jersey*, pages 225–248, 1995.

[4] U. C. Devi. An improved schedulability tast for uniprocessor periodic task systems. *proc. Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 23–30, 2003.

[5] L. Epstein and R. Van Stee. On non-preemptive scheduling of periodic and sporadic tasks. *Theoretical Computer Science*, 299:439–450, 2003.

[6] R. Karp. On-line algorithms versus off-line algorithms: How much is it worth to know the future? *Algorithms, Software, Architecture, IFIP Transactions A-12*, Information processing:1:416–429, 1992.

[7] I-G. Kim, K-H. Choi, S-K. Park, D-Y. Kim, and M-P. Hong. Real-time scheduling of tasks that contain the external blocking intervals. *Real-Time and Embedded Computing Systems and Applications(RTCSA'95)*, 1995.

[8] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM (Association for Computing Machinery)*, 20(1):46–61, 1973.

[9] Jane W. S. Liu. *Real-Time Systems*, chapter Priority-Driven Scheduling of Periodics Tasks, pages 164–165. Prentice Hall, 2000.

[10] L. Ming. Scheduling of the inter-dependent messages in real-time communication. *Proc. of the First International Workshop on Real-Time Computing Systems and Applications*, Dec. 1994.

[11] J.C. Palencia and M. Gonzales-Harbour. Schedulability analysis for tasks with static and dynamic offsets. *Proceedings of the 19th IEEE Real-Time Systems Symposium*, 1998.

[12] J.C. Palencia and M. Gonzales-Harbour. Offset-based response time analysis of distributed systems scheduled under edf. *Proceedings of the IEEE Real-Time Systems Symposium*, 2003.

[13] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, 1, December 2004.

[14] F. Ridouard, P. Richard, and F. Cottet. Ordonnancement de tches indpendantes avec suspension. *Proceedings of the 13rd RTS Embedded Systems (RTS'05)*, 1, April 2005.

[15] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communication of the ACM 28*, 2:202–208, 1985.

[16] A.J. Wellings, M. Richardson, A. Burns, N. Audsley, and K. Tindell. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 1993.

Bicriteria Fixed-Priority Scheduling in Hard Real-Time Systems: Deadline and Importance

A. Aguilar-Soto* and G. Bernat
Department of Computer Science,
The University of York
Heslington, York YO10 5DD
{aas, bernat}@cs.york.ac.uk

Abstract

In the context of fixed priority scheduling on hard real-time systems, we investigate the scheduling problem where timing and QoS requirements have to be optimized. The QoS is expressed in terms of relative importance relationships. Thus, the problem is formulated as finding an optimal priority ordering that maximises the importance criterion. Optimality in this context means that there is not other feasible schedule with higher importance. The main contribution is an algorithm that finds such optimal priority assignment in $O((n^2+n)/2)$ where n is the number of tasks. We indicate that if any QoS requirement can be correlated with the importance concept, our algorithm can find a good solution for problems with such QoS requirements. We exemplify this by applying the algorithm to the problem of minimizing the total number of preemptions.

1 Introduction

Hard real-time systems must simultaneously handle timing and Quality of Service (*QoS*) requirements. While classical scheduling policies cope satisfactorily only with the timing ones, dealing simultaneously with both requirements is still an issue.

During the specification of requirements, QoS requirements (e.g. safety, reliability, performance) are defined. These requirements form a multidimensional set of requisites interrelated that must be conveyed throughout all stages of the cycle development. Normally, conflicts among requirements exist and then, tradeoffs have to be defined to help the designers with their decisions. Assigning importances to the requirements is a mechanism to specify such tradeoffs. In effect, typically the requirements are not equally important; some may be essential while others may be desirable, and therefore each requirement should be rated by importance and/or stability to make these differences clear and explicit [9].

*Supported by CONACYT grant No. 61146

During the design phase, real-time system can be structured as a set of concurrent, preemptive tasks with hard and/or soft deadlines. A single task provides some *benefit* to the system when it achieves all or part of one or more requirements. The tasks share some scarce computer resources that must be allocated wisely in order to obtain the greater benefit possible. This is a scheduling problem that can be solved by scheduling approaches such as the Fixed Priority Scheduling (*FPS*) scheme.

FPS is considered an industry standard providing good performance, predictability and flexibility. On FPS, the assignment of priorities and the feasibility analysis are the two main areas of work. The former establishes the order at which the tasks will be executed and the last checks whether the timing constraints will be fulfilled. When the benefit is measured only with regard to the timing requirements, FPS allows solving problems with complex tasks models [6]. However, when the benefit includes other QoS measures, FPS exhibits some weaknesses.

For instance, FPS normally assigns high priorities to tasks with either shorter periods (Rate Monotonic [15]) or shorter relative deadlines (Deadline Monotonic [16]). However, important tasks may not have short periods or deadlines. It is true that, for such tasks, the priorities can be raised cutting the tasks down into smaller ones with shorter periods, but it increases the run-time overhead and introduces artificial constraints to the problem [25].

In hard real-time systems, when the benefit is measured with respect to the timing requirements, "meet all hard deadlines" is a criterion usually defined. How to measure the benefit with respect to QoS requirements is still motive of discussion and research.

In the context of FPS, we deal with the *bicriteria scheduling problem* where hard timing requirements have to be met and the QoS should be maximized.

1.1 Measuring the QoS

In order to include QoS requirements in a scheduling problem, a QoS metric has to be defined. A QoS requirement can be either quantitative or qualitative, and testable for presence in their implementation. If quanti-

tative, quantifiable in at least one scale of measure [8].

The literature shows that the QoS can be expressed with artifacts such as time-value functions and then, metrics such as "the value accrued" during a window time are utilised. Including timeliness and value into the scheduling decisions is sought by the value-based schemes.

In the the value-based (or reward-based) schemes, the value is represented as utility functions that describe the utility obtained when a task completes [10]; this utility can remain static or can vary with the time. Value-based schemes use the utility as either a priority or a admission policy. These schemes have several drawbacks namely:

1. They are heuristics.
2. They do not guarantee the timing constraints.
3. There is not a methodology for the design of utility functions [13].
4. It is assumed, a priori, that arithmetic operations can be performed with the the utility functions, which it is not necessarily true [18].

Drawbacks (1) and (2) have relegated value-based scheduling into the domain of soft dynamic-priority real time systems. Points (3) and (4) are related with the meaning of value. Prasad et al. [18] have shown that the assignment and the use of values are not separated issues but linked. Any assignment of values must conform to a *scale* of measurements and the scheduling scheme must be cognizant of the scale utilized to perform only meaningful operations.

The above drawbacks discourage us for using time-value functions to express the QoS in the bicriteria scheduling problem. Instead we observe the problem in a slightly different way by defining qualitative importance relationships to measure the QoS.

1.1.1 Importance Relationships

Time-value functions are a powerful tool for representing and reasoning about QoS preferences. However, they can be very difficult to both derive and measure.

An alternative form of expressing QoS requirements is using a class of preferential statements of the form "it is more important to me that the value of X be higher than the value of Y". These statements are called *relative importance statements* [4]. Such statements do not require complex quantitative assessment. A complete framework for expressing the relative importances is found in [4].

By rewriting the above statement as "X is more important than Y", we can define the *task importance* as a preference for executing a task with regard to the other tasks in the system. For example, assuming that in a real-time database system a task τ_a implements a critical transaction and a task τ_b implements something else, safety and reliability will be comprised if τ_a is interrupted frequently; the statement "τ_a is more important than τ_b" expresses this as a QoS requirement.

1.2 The Bicriteria problem

Assuming that the QoS is conveyed as importance relationships, the bicriteria scheduling problem can be formulated as *finding a feasible priority ordering that maximises the importance*. Two challenges can be devised:

- How to define the importance metric.
- How to solve the bicriteria scheduling problem.

Solving problems with multiple criteria is not simple; optimizing one criterion could decrease the value of the other criteria in the problem and hence, a number of solutions are possible. However, when the commitments among criteria are specified, optimal solutions can be defined. In hard real time systems, the commitment is with the timeliness; the deadlines must be satisfied and the importance is only a secondary objective. We observe that if the tasks set is feasible under FPS, then there exist a subset of priority orderings that are feasible. The aim is to find the feasible priority ordering that maximises its importance.

Note that by labeling a set of N tasks according to importance, the set of $N!$ orderings of tasks in lexicographical order is also ordered according to importance. For example, for $N = 3$ tasks $\{a, b, c\}$ where *a* is the most important and *c* the lower, the 3! orderings are *a b c*, *a c b*, *b a c*, *b c a*, *c a b*, *c b a*. If we assign priorities according to position *x y z* such that *x* has the highest and *y* the lowest priority, only a subset of these orderings may be feasible. The aim is to find the feasible ordering, which is higher (or lower) in lexicographical order of importance. If in our case *b a c*, *c a b* and *c b a* are feasible, then *b a c* is the optimal priority ordering; i.e it is the closest one to *a b c*.

In general, this problem can be solved by generating the $N!$ priority orderings in lexicographical order and testing them for feasibility. The first one that is schedulable is the optimal one. This is computationally intractable even for small N. We show how a pseudo-polynomial algorithm can find the optimal in the section 6.

The rest of the paper is organized as follow: Section 2 summarizes some related work. Section 3 establishes the process model. Section 4 illustrates an example. Section 5 defines the problem and Section 6 presents the solution. Finally, Sections 7 and 8 presents an evaluation and our conclusions respectively.

2 Related work

To the best of our knowledge no algorithm for solving this problem has been reported in the literature. However, algorithms for finding feasible priority assignments that optimize other additional QoS criteria have been described. For instance: in [14] a priority assignment algorithm improves system fault resilience in fault-tolerant hard real-time systems. Approaches to reduce the number of preemptions in FPS have been published but such solutions introduce additional problems such as requiring non-standard runtime support [24], or multiplying the

number of tasks to be scheduled by at least a factor of two [7]. In the energy consumption problem, the use of energy must be bounded to guarantee stability and/or extended the lifetime of a system. In this case, the system includes specialized hardware (e.g. Dynamic Voltage Scaling processor [19] or I/O Device Power States [11]) and the solutions are pairs *(priority,power-state)* such that at run-time both the priority and the power-state are applied. A number of papers related to this problem have been published and some solutions can be found in [22] and [3]. On Real-Time Databases Systems, transactions must satisfy timing constraints and consistency constraints of the database [20]. In [12] periodic transactions that access main memory resident data via read/write locks are scheduled using rate monotonic. On the other hand, the vast literature on value-based scheduling is primordially related to soft real-time systems. In this paper we present a more general approach.

3 Process model

We consider an extension of the traditional process model, where a set of N computer tasks must be scheduled on a single processor system. A *task set* is a collection of tasks and an *ordering* is a totally ordered task set.

3.1 Tasks

The tuple (C, T, D, P, B, J, I) characterises a task τ where C is the worst case computation time, T is the period or the minimal inter-arrival time between two consecutive releases, depending whether it is periodic or sporadic; D is the deadline of the task relative to the actual release (i.e. if τ is invoked at time t, it should have finished by $t + D$); P is the priority of the task where 1 is the highest and N is the lowest priority; without loss of generality we assume that two tasks do not share the same priority. The blocking factor B is the maximum interference that a task may suffer from lower priority tasks due to a share resource protocol [21]. The release jitter J is the maximum elapsed time between the programmed initial release time and the real ready-to-run time, which usually is zero for periodic tasks.

The importance I is an unique natural number representing the importance of the task with regard to the other tasks in the system, where 1 is the highest one. Note that this imply that if τ_i is most important that τ_j then $I_i < I_j$ for any two τ_i, τ_j. The comparison between importance will be called a *lexicographic* comparison as follow, if τ_i is most important than τ_j then τ_i *is lexicographically smaller than* τ_j. Finally, the tasks are preemptive, they are released at time zero and they do not suspend themselves.

3.2 Task Set

Let S be a set of N tasks and $S^{\prec} = \langle \tau_1 \, \tau_2 \, \ldots \, \tau_N \rangle$ an ordering on S with relation $\prec$ (*"precede to"*) such that $\tau_j \prec \tau_{j+1}, \ \forall \ j = 1, 2, \ldots, N-1$. To simplify the notation we will use $a, b, c, \ldots \, z$ to denote tasks and therefore $\langle a \, b \, c \, \ldots \, z \rangle$ is an ordering on set $S = \{a, b, c, \ldots, z\}$. Note that the ordering can be specified implicitly by the $\langle a \, b \, c \, \ldots \, z \rangle$ notation without requiring the $\prec$ operator. For instance, given the order relation *D="has a shorter deadline than"*, and a task set $\{a, b, c\}$ with $D_a > D_b > D_c$, its ordering under this relation is $S^D = \langle c \, b \, a \rangle$. Note that we use $\{\ \}$ to denote a set and $\langle\ \rangle$ to denote an ordering. Following this convention:

- The ordering defines a priority assignment over the tasks such as, if $a \prec b$ then $P_a > P_b$.
- For a given tasks set S, we denote $\hat{S}$ the set of all possible $N!$ orderings of S.
- Any two orderings in $\hat{S}$ can be compared according their importance as follow: let $\alpha, \beta \in \hat{S}$, we say that α *is most important than* β if comparing each element $\alpha[k]$ with $\beta[k]$, starting from the leftmost to the rightmost, the first difference is in the k^{th} task and the importance of $\alpha[k]$ is greater than the importance of $\beta[k]$. Note that it is similar to comparing two strings. Furthermore, note that because $\alpha[k]$ is more important than $\beta[k]$ then $\alpha[k]$ is lexicographically smaller than $\beta[k]$; therefore, we can say that if the ordering α is most important than the ordering β then α *is lexicographically smaller than* β; we denote it as $\alpha \prec_{lex} \beta$. The operator $\prec_{lex}$ also applies to tasks; e.g. $\tau_i \prec_{lex} \tau_j$.
- Any ordering $\langle a \, b \, c \, \ldots \, j \, k \, \ldots \, x \, y \, z \rangle$ can be represented as $\langle \phi \, \omega \rangle$ where the prefix ϕ is $\langle a \, b \, c \, \ldots \, j \rangle$ and the suffix ω is $\langle k \, \ldots \, x \, y \, z \rangle$. The meta-ordering $\langle \phi \, * \rangle$ denotes all the orderings in $\hat{S}$ starting with prefix ϕ.

Lets define some interesting orderings on S:

- For any $A \subseteq S$, $\delta(A)$ is an ordering with priorities assigned according the Deadline Monotonic Priority Ordering (DMPO) [5] where the function δ orders A according the relation *"has a shorter deadline than"*.
- S^D is the ordering by DMPO; i.e. $S^D = \delta(S)$.
- S^I is the ordering with priorities assigned according the relation *"is more important than"*. Note that S^I is the lexicographically smallest ordering in $\hat{S}$.

Finally, the following functions are defined:

- For $\alpha \in \hat{S}$, the function $F(\alpha)$ returns *true* when α is feasible, i.e. the ordering α passes the FPS test [1]; otherwise returns *false*.
- Due to $\hat{S}$ can be ordered lexicographically, for all orderings in $\hat{S}$, its lexicographic order defines

$$I : \hat{S} \rightarrow [0, 1, \ldots, (N! - 1)]$$

which is a function that indexes each element in $\hat{S}$, such that for any $\alpha, \beta \in \hat{S}$

$$\alpha \prec \beta \ \Leftrightarrow \ I(\alpha) < I(\beta)$$

τ	T	D	C	R	I	QoS_{code}
e	100	80	13	13	5	3,2,1
d	240	240	37	50	4	3,1,2
c	330	330	55	118	3	2,3,1
b	350	350	56	174	2	2,1,3
a	480	400	68	292	1	1,2,3

Table 1. Set S_5 with QoS codes where 1=Safety, 2=Reliability, and 3=Performance.

therefore, the lexicographic distance between any ordering can be determined. Note that $I(S^I) = 0$.

4 Motivational Example

In order to illustrate the problem, let us consider an hypothetical system with several QoS requirements, which must be implemented in a fixed-priority real-time operating system. The requirements are (in order of criticality) Safety (S=1), Reliability (R=2) and Performance (P=3).

The tasks have individual QoS requirements that can be coded according a string with elements S,R,P. For example, a task with high-safety, medium-reliability and low-performance requirements is coded as (1,2,3), i.e. (S,R,P).

The tasks are as follows: τ_a implements a control algorithm which is essential for the stability of the system and then the designer assigns the code (1,2,3). τ_b reads inputs from sensors and store them in a database. The code (2,1,3) is assigned because the database represents the external environment and then, the data freshness is essential for reliability. Similar reasoning can be given to the other tasks. Their codes are shown in the table 1.

Observing the codes: in terms of safety "τ_a is more important than τ_b"; in terms of reliability "τ_b is more important than τ_a"; in terms of performance "τ_a is equally important than τ_b". It is easy to conclude that in terms of the overall QoS, "τ_a is more important than τ_b". Similar reasons can be given to specify the rest of importance values but for the sake of simplicity, we omit them.

Note that this assignment of QoS codes provides a partial order (e.g. X is equally important than Y). However, the designers can always use a tie-break rule using their knowledge specific to the application. This methodology is only a simple example of how the importances can be assigned. A complete framework is found in [4].

Our objective is to guarantee that all activities meet their deadlines and fulfil their importance requirements. The tasks have deadlines less than or equal to their periods and hence, the ordering under DMPO is $S^D = \langle e\, d\, c\, b\, a \rangle$.

Table 1 shows the tasks set with their respective worst-case response times R_j, computed with the response time analysis equation [1]. The ordering S^D is feasible and hence, from the point of view of the timing constraints, it is satisfactory. However, it would be better if a and b have higher priorities to meet their importance requirements. These two task are the most important.

$S^I = \langle a\, b\, c\, d\, e \rangle$ is a priority assignment by importance; unfortunately S^I is unfeasible (under S^I, R_e is 229). We will show later that an assignment $S^* = \langle b\, e\, a\, d\, c \rangle$ is optimal in the sense that both, it is feasible and it is the closer one to S^I.

5 Deadline vs Importance: The Scheduling Problem

In real-time systems meeting the timing constraints is fundamental while any other QoS requirement can be considered as a soft requirement. Consequently, maximizing the importance is a requirement that can be relaxed as for example, defining different levels of satisfaction with respect to the level of importance; while closer to the most important ordering, greater the satisfaction. We can formulate the problem as *finding a feasible priority ordering that minimises the distance to the most important ordering*. More formally:

Problem 5.1 (BiCriteria Scheduling Problem)**.** Given a set of tasks and a ordering S^I, find S^* which is an assignment of fixed-priorities that meets their deadlines and is the closest to S^I.

In section 6.2 we present an algorithm which finds an ordering that solves the scheduling problem stated above. We affirm that, if all tasks are feasible under the deadline monotonic priority ordering, then there exist a subset of possible feasible orderings where the importance objective can be achieved. Our algorithm looks at this subset and finds the one which is the closest to S^I, the ideal solution.

Our algorithm is based on the traditional scheduling theory on real-time systems [5] and on the multi-criteria scheduling theory developed on the operational research area [23]. The next section introduces some concepts on multi-criteria scheduling and defines some metrics. These metrics provide the clues to show our solution.

5.1 Multicriteria Scheduling Problems

T'kindt and Billaut [23] define the *multicriteria scheduling problem* as "the problem which consists of computing a pareto-optimal schedule for several conflicting criteria". A simple definition of pareto-optimal schedule is given by Pinedo [17] in the context of minimization problems: "A schedule is called *pareto-optimal* if it is not possible to decrease the value of one objective without increasing the value of the other". When only two criteria are involved in the problem (as in our case), all pareto-optimal solutions can be represented in a cartesian plane such that, all tradeoffs between criteria can be shown.

Graphical representation of the problems assists both, to illustrate the problem and to identify clues to solve it. Figure 1 represents the space of solutions of a task set measured by one pair of metrics such that the optimums tends to zero. Note that the point that minimises simultaneously both criteria does not exist.

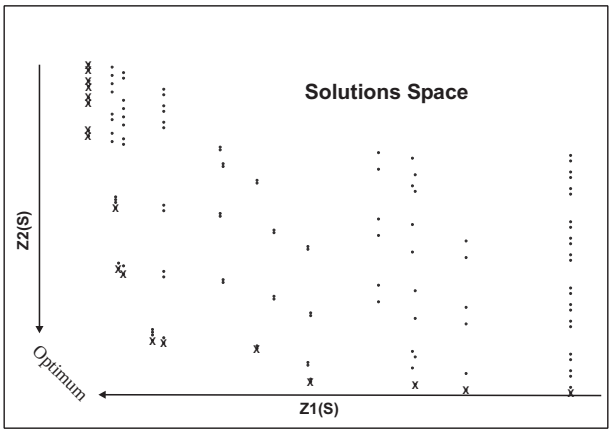


Figure 1. A Bicriteria Solution Space. A dot represents a possible solution and a $\times$ marks a pareto-optimal solution

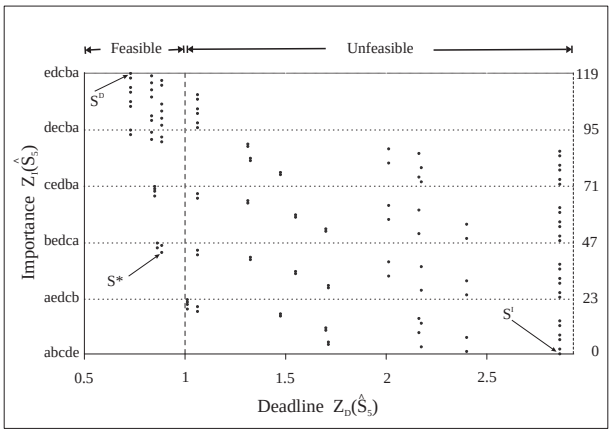


Figure 2. Plotting all priority orderings of S_5. S^I is not feasible. S^D is feasible but has low importance metric. The optimal bicriteria S^* is $\langle b\, e\, a\, d\, c \rangle$

5.2 Deadlines (Z_D)

The first objective is to meet the deadlines. On fixed priority scheduling the optimal solution is to assign priorities according to the DMPO. *Any fixed-priority ordering S' is feasible if and only if $R_j \leq D_j\ , \forall\, j \in S'$.* Therefore, for all j in S' we have

$$R_j \leq D_j \ \Rightarrow\ \frac{R_j}{D_j} \leq 1 \ \Rightarrow\ \frac{R_j}{D_j} \leq \max_{\forall j} \left\{ \frac{R_j}{D_j} \right\} \leq 1$$

Doing

$$Z_D(S') = \max_{\forall j \in S'} \left\{ \frac{R_j}{D_j} \right\}$$

we are able to define:

Definition 5.2. *An ordering S' is feasible iff $Z_D(S') \leq 1$ where*

$$Z_D(S') = \max_{\forall j \in S'} \left\{ \frac{R_j}{D_j} \right\} \tag{1}$$

Note that, if $Z_D \leq 1$ the ordering S' is feasible; otherwise it is unfeasible. Furthermore, when $Z_D = 1$ or very close to 1, it indicates at least one task finishes very close to its deadline. This metric give us more expressiveness that a simple yes/no answer.

5.3 Importance (Z_I)

The second objective is with regard to the tasks importance. In our model the importance only requires fulfilling the properties of a totally ordered set such that any $a, b \in S$ can be compared.

As defined in section 3, any $S^{'} \in \hat{S}$ can be indexed by a function. The most important ordering is S^I, which is lexicographically smaller and hence $I(S^I) = 0$. In the lexicographic order, any two consecutive orderings differ only by a pair, and the most important of them is always the lexicographically smaller. Consequently, the distance from any ordering S' to S^I is simply $I(S')$. Therefore,

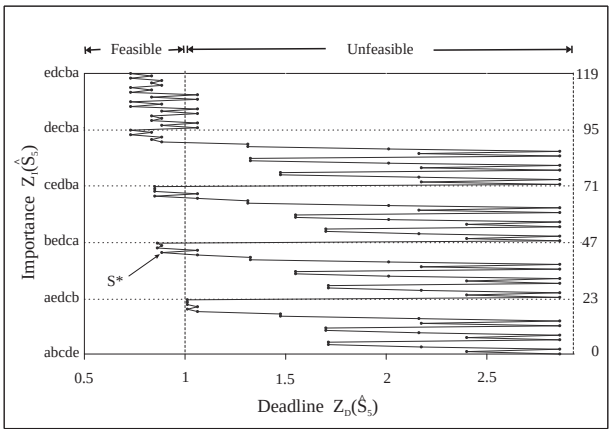


Figure 3. Similar to figure 2 but the priority orderings are connected according their lexicographic order

we will define Z_I, the metric which describes the distance from any ordering to S^I as follow.

$$Z_I(S') = I(S') \tag{2}$$

This metric defines the second objective. We will show how to apply Z_D and Z_I using our example.

5.4 Portraying the problem

Figure 2 shows all $5! = 120$ priority orderings of S_5 (from table 1) plotted with both metrics.

For example, the response times for S^D are $\{13, 50, 118, 174, 292\}$; it is feasible but it has low importance because its index is 119. On the other hand the response times for S^I are $\{68, 124, 179, 216, 229\}$ and therefore it is not schedulable. The optimal S^* is $\langle b\, e\, a\, d\, c \rangle$ with response times $\{56, 69, 150, 187, 292\}$ and index 44. Note that:

- With respect to Z_D, there are 32 feasible orderings ($Z_D(\hat{S}_5) \leq 1$);

- With respect to both Z_D and Z_I, the best solutions tend to zero. This example is an extreme case where S^I is completely opposite to S^D.

Clearly, the problem is how to determine S^* without having to enumerate all the orderings. We can figure out a solution looking at figure 3 carefully; it contains the same data but the points are connected successively such as they would be generated (in lexicographic order). Note that there are local minimums (left-side peaks) distributed around the indexes 23, 47, 71, 95 and 119. In addition note on the left side, which orderings correspond to such points. The Index 23 corresponds to $\langle a\, e\, d\, c\, b\rangle$ where $\langle a\rangle$ is the most important task plus a suffix $\langle e\, d\, c\, b\rangle$ ordered by DMPO. The Index 47 is $\langle b\, e\, d\, c\, a\rangle$ where $\langle b\rangle$ is the second most important task plus a suffix ordered by DMPO. The rest have the same pattern. Our algorithm uses this pattern to reduce the search space.

6 Solving The Deadline and Importance Problem

This section is divided in two parts:

1. We show how a set $\hat{S}$ can be organized in a tree, and how whole subtrees can be skipped by identifying some local minimums called entry-points.

2. We present the algorithm D&I, which performs a branch and bound search into the tree defined. In the worst-case it performs $\frac{N^2+N}{2}$ steps to find the solution. The algorithm is described and its optimality is proved.

6.1 Organizing the Search Space

The search space consists on the set $\hat{S}$ ordered lexicographically and organized as a tree. For the sake of simplicity and without loss of generality, we will define S^I as $\langle a\, b\, c\, \ldots\, z\rangle$ where z is the N^{th} task.

The root of the tree is $\langle a\, b\, c\, \ldots\, z\rangle$. The immediate sons of this three are the N subtrees $\langle a\, *\rangle$, $\langle b\, *\rangle$, $\langle c\, *\rangle$, $\ldots, \langle z\, *\rangle$ ordered lexicographically. Each one of these subtrees has $N-1$ sub-subtrees where each one has $N-2$ ones and so on. For example, $\langle a\, *\rangle$ has $\langle a\, b\, *\rangle$, $\langle a\, c\, *\rangle$, $\langle a\, d\, *\rangle$, $\ldots, \langle a\, z\, *\rangle$. Therefore, any subtree can be represented as $\langle \phi\, *\rangle$. A *vertex* is represented as $\langle \phi\, \omega\rangle$.

The reason for organizing the space into subtrees with this configuration comes from an interesting property related with the optimality of DMPO and the generality of the FPS test:

- First, DMPO is optimal in the sense that if a set is schedulable by any fixed-priority ordering then it also is schedulable under DMPO.

- Second, the FPS test is necessary and sufficient and it is independent of the assignment of priorities.

Consider an ordering $S^I = \langle a\, b\, c\, d\rangle$ and an ordering $S^D = \langle d\, c\, b\, a\rangle$. Suppose that we apply the FPS test to S^I and we find that S^I is feasible. The optimality of DMPO indicates that it is no necessary to apply the test to S^D because if S^I is feasible then S^D is also feasible; but the contrary is not true, if S^D is feasible we cannot assure anything about the feasibility of S^I or any other combination. However we can affirm that if S^D is unfeasible then any ordering of these four tasks will also be unfeasible.

Now, consider that we extend S^D with an arbitrary task such that we have an ordering $\langle x\, d\, c\, b\, a\rangle$ (remember that $\langle d\, c\, b\, a\rangle$ is ordered by DMPO) and we apply the FPS test. If $\langle x\, d\, c\, b\, a\rangle$ is feasible, then there are some orderings in $\langle x\, *\rangle$ that are feasible. However, if $\langle x\, d\, c\, b\, a\rangle$ is unfeasible we can affirm that no ordering in $\langle x\, *\rangle$ is feasible.

Lets define us any vertex $\langle \phi\, \omega\rangle$ with ω ordered by DMPO as follow.

Definition 6.1 (entry-point)**.** *An* entry-point *to a subtree $\langle \phi\, *\rangle$ is a vertex with ω ordered by DMPO; i.e.*

$$\textit{entry point} = \langle \phi\ \delta(\omega)\rangle$$

.

The next theorem resume the above observations.

Theorem 6.2. *If the entry-point $\langle \phi\ \delta(\omega)\rangle$ is unfeasible then any ordering in $\langle \phi\, *\rangle$ is also unfeasible.*

Proof. By contradiction, suppose that there exist a feasible ordering in $\langle \phi\, *\rangle$ say $S' = \langle \phi\, \omega\rangle$. On the other hand $S^\delta = \langle \phi\ \delta(\omega)\rangle$ is an unfeasible ordering. Both orderings share the same ϕ and differ only by the order on ω.

Note that if ω has less than 2 tasks, then $S' = S^\delta$ and this contradiction proves the theorem. Otherwise, the feasibility of S' and S^δ depends only on ω; all tasks in ϕ are not affected by any change in ω.

The proof of optimality of the DMPO assignment [5] shows us that, because S' is feasible, exchanging any pair of tasks (in ω) non-ordered by shorter deadline give us a feasible ordering S''. Following the same process we obtain orderings $S''', S'''', \ldots, S^n$ which are also feasible. The last one is S^n with its ω ordered by DMPO. S^n is identical to S^δ and therefore S^δ is feasible. This contradicts the hypothesis and concludes the proof.

□

Note that when ϕ is empty, the entry point is S^D and hence, if S^D is unfeasible then all orderings of S are unfeasible. In addition, the theorem gives us a useful corollary. Consider a subtree $\langle \phi\, *\rangle$ and a feasible vertex $\langle \phi\, \omega\rangle$. The proof of 6.2 shows that ordering ω by deadlines will give us another feasible ordering and hence:

Corollary 6.3. *If an ordering $\langle \phi\, \omega\rangle$ is feasible then $\langle \phi\ \delta(\omega)\rangle$ is also feasible.*

The Theorem 6.2 will be used to examine the search space. Testing a vertex $\langle \phi\ \delta(\omega)\rangle$ allows us to decide

Algorithm 1 D & I (*Deadline and Importance*)

Require: S^D feasible, S^I unfeasible
1: Set $\phi \Leftarrow \emptyset, \omega \Leftarrow S^I, \omega_\delta \Leftarrow S^D, k = 0$
2: **while** SizeOf(ω) > 1 **do**
3: let $\tau \Leftarrow \omega[k]$
4: let $\omega_{\delta-j} \Leftarrow delete(\omega_\delta, \tau)$
5: build $S^*_{test} \Leftarrow \langle \phi\ \tau\ \omega_{\delta-j} \rangle$
6: **if** $F(S^*_{test})$ **then**
7: $S^* \Leftarrow S^*_{test}$
8: $\phi \Leftarrow \langle \phi\ \tau \rangle$
9: $\omega \Leftarrow delete(\omega, \tau)$
10: $\omega_\delta \Leftarrow \omega_{\delta-j}$
11: $k \Leftarrow 0$
12: **else**
13: $k \Leftarrow k + 1$
14: **end if**
15: **end while**

whether the subtree $\langle \phi\ * \rangle$ can be skipped or whether we need to look inside it. This process is applied recursively to each subtree performing, in this way, a branch and bound search. The corollary will be used to prove the optimality of the algorithm proposed.

6.2 The Algorithm D&I

The algorithm D&I (Deadline & Importance) examines the entry-point of the subtrees in lexicographic order from the closest to S^I to the remotest one and from the top to the bottom. There are two preconditions which must be fulfilled:

- S^D must be feasible; otherwise no feasible solution exists.
- S^I must be unfeasible; otherwise we do not need to perform a search because S^I is the solution.

The next variables are used:

- ϕ is the prefix to the actual subtree indexed.
- ω is the complement of ϕ; i.e. $S = \phi \cup \omega$ and $\emptyset = \phi \cap \omega$.
- ω_δ is $\delta(\omega)$.
- τ is the k^{th} task of ω acquired orderly from left to right.
- $\omega_{\delta-\tau}$ is ω_δ without τ.
- S^*_{test} is the entry-point to be tested.

Let S be a tasks set with orderings S^D and S^I, feasible and unfeasible respectively. The algorithm builds keys to index each subtree in lexicographic order. A key is built by appending to ϕ a task τ (step 3) from ω. The lexicographical order is achieved following the sequence in ω which is S^I. The key indexes the subtree $\langle \phi\, \tau\, * \rangle$ and hence we build its entry-point S^*_{test} (steps 4-5). S^*_{test} is tested (*step 6*) and if it is feasible, it is saved as a partial solution S^*, τ is deleted from both ω and ω_δ, ϕ is updated and the index k is reset (*steps 7-11*); otherwise, k is advanced to the next τ. D&I stops when there are not more subtrees to visit (ω has length one). Note that the worst-case is when the only solution is S^D and its order is contrary to S^I. In this case, the entry-point of the first N subtrees will be tested and the N^{th} will be $S^*_{test} = S^D$; afterward, the next level of $N-1$ subtrees will be tested and the $(N-1)^{th}$ will be again $S^*_{test} = S^D$ and so on. Thus, in the worst-case, the solution will always be S^D. It is also easy to show that the algorithm will always stop.

6.2.1 Complexity

The worst-case occurs when the only solution is S^D and its order is contrary to S^I. In such case the loop is executed N times and ω is reduced to $N-1$ elements; afterward the loop is executed $N-1$ times and ω is reduced to $N-2$ elements and so on. Therefore the loop executes $N + (N-1) + (N-2) + \ldots + 1 = \frac{N^2+N}{2}$ times. At each iteration, a feasibility test of complexity E is performed and therefore the complexity of the algorithm D&I is $O(E \times \frac{N^2+N}{2})$. In other words, the algorithm finds the solution in a polynomial number of steps but the total complexity is pseudo-polynomial due to the FPS test.

6.2.2 D&I proof

Theorem 6.4. *The algorithm* D&I *yields an optimal solution for the problem 5.1 of guaranteeing all deadlines and minimizing the distance to* S^I

Proof. By contradiction. Suppose that at the n^{th} iteration, there exists an ordering S^o that is both, feasible and lexicographically closest to S^I. D&I finds S^* and hence we need to prove that either:

1. S^* is unfeasible
2. S^* is not the closest to S^I.

Note the next facts about S^o and S^*:

- S^o is lexicographically smaller than S^* (i.e $S^o \prec_{lex} S^*$).
- They are lexicographically different but before a task j they are identical; i.e. they share the same prefix $\phi = \langle a\, b\, c\, \ldots\, j \rangle$ and differ on their suffix.

$$S^o = \langle \phi\, \omega^o \rangle \text{ where } \omega^o = \langle k \ldots l \ldots \rangle$$

$$S^* = \langle \phi\, \omega^* \rangle \text{ where } \omega^* = \langle l \ldots k \ldots \rangle$$

Consequently $k \prec_{lex} l$.

- S^o and S^* share the same entry point $\langle \phi\, \omega_\delta \rangle$ where

$$\omega_\delta = \delta(\omega^o) = \delta(\omega^*)$$

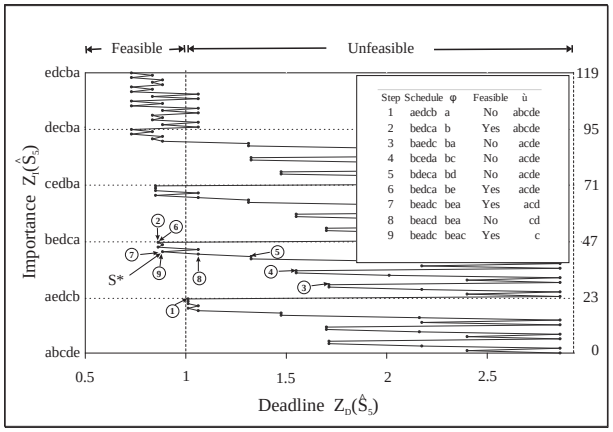


Figure 4. D&I applied to S_5 showing the sequence of operations.

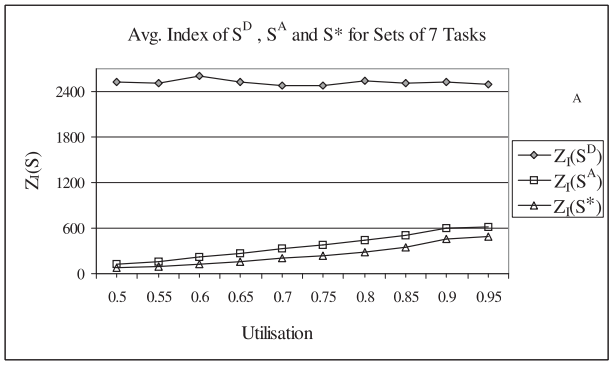


Figure 5. For high utilisations, the number of feasible tasks sets decreases and therefore, the distance between S^D and S^* gets smaller. The distance between S^A and S^* increases slightly.

- S^o is feasible and therefore $\langle \phi\ \delta(\omega^o)\rangle$ is also feasible (corollary 6.3).

S^o and S^* share the same feasible entry-point and hence, $\langle \phi\ \omega_\delta\rangle$ has been tested and saved (step 7) as the n^{th} partial solution S^*_n. The next iterations will give $S^*_{n+1} \rightarrow S^*_{n+2} \ldots \rightarrow S^*_{n+m} \rightarrow S^*$. All will be feasible because the step 7 accepts only feasible ones. Therefore S^* is feasible and *it contradicts the first hypothesis*.

Consequently, it is necessary to prove that although feasible, S^* is not better than S^o which is the closest feasible solution to S^I.

D&I builds and tests new orderings joining the actual ϕ with a task τ and ordering the rest of the tasks by DMPO (steps 3-6). Because $k \prec_{lex} l$, the ordering $\langle \phi\ k\ \delta(\ldots l \ldots)\rangle$ is tested before than $\langle \phi\ l\ \delta(\ldots k \ldots)\rangle$.

The ordering tested is

$$S^o_{test} = \langle \phi\ k\ \delta(\ldots l \ldots)\rangle$$

Two cases occur:

- If S^o_{test} is feasible, D&I will accept it and never will find S^* because they both are in different paths. This is not possible because S^* is found by D&I.
- If S^o_{test} is unfeasible then S^o is also unfeasible (Theorem 6.2) which *contradicts the hypothesis 2*.

This concludes the proof.

□

6.2.3 An example

Applying the algorithm D&I to the tasks set S_5, it finds the ordering S^* in nine steps (figure 4).

At the beginning, ϕ is empty and ω is $\langle abcde\rangle$. The first element is acquired to create a key $\langle a\rangle$ to test the entry-point $\langle a\, edcb\rangle$ (① in the graph); the test fails and then the subtree $\langle a\, *\rangle$ is skipped. The next element in ω is obtained and $\langle b\, edca\rangle$ is tested (②); it is feasible and then $\langle b\rangle$ is appended to ϕ and $\langle b\rangle$ is deleted from both ω and ω_δ. The next three orderings tested (③,④,⑤) are unfeasible, and therefore only the index k is updated. Afterwards, $\langle b\, edca\rangle$ and $\langle b\, eadc\rangle$ are tested successively and saved (⑥,⑦); the next ordering fails (⑧) and the next one passes the test (⑨). ω has length 1 and D&I stops.

7 Evaluation

This section is divided in two: The first one presents some results on the performance evaluation of the D&I algorithm. The second part shows how the algorithm can be used to reduce the total number of preemptions.

For the experiments we have generated random task sets of N tasks with utilisations from 0.5 to 0.95 and $N = 4, 5, 6, 7, 8$ tasks. Each task set is created by randomly chosen task's computation times between 2 and 50 time units, and then randomly chosen the periods to approximate the utilisation desired. Without loss of generality, we assign the deadlines equal to the periods. The importance is varied according each experiment. In addition, it is guaranteed that all task sets are feasible under deadline monotonic scheduling.

7.1 Performance evaluation

The first experiment quantifies how good (or bad) is using the DMPO assignment and the swapping priority assignment defined in [2] against D&I for the bicriteria problem. The swapping priority algorithm receives an unfeasible ordering and finds a feasible one (if it exists) by swapping pairs of task priorities. In this context, the swapping algorithm will receive an unfeasible ordering by importance and will produce a feasible one.

Lets S be a task set with importances randomly chosen and with priority orderings S^I (by importance), S^D (by DMPO), S^A (by swapping algorithm) and S^* (by D&I algorithm); the goodness of the orderings is given by the index metric Z_I. For each set, we compute both their orderings and their indexes. A point in figure 5 represents

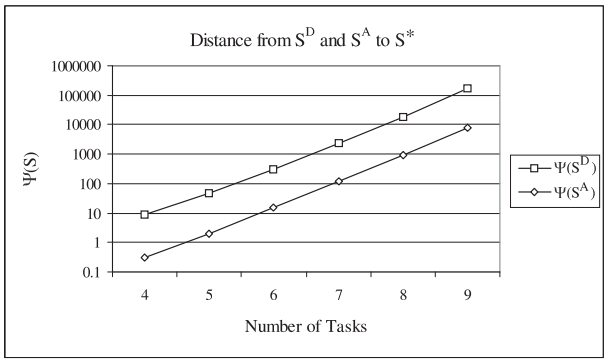


Figure 6. The solutions S^D and S^A move dramatically away from S^* conforming the number of tasks increases.

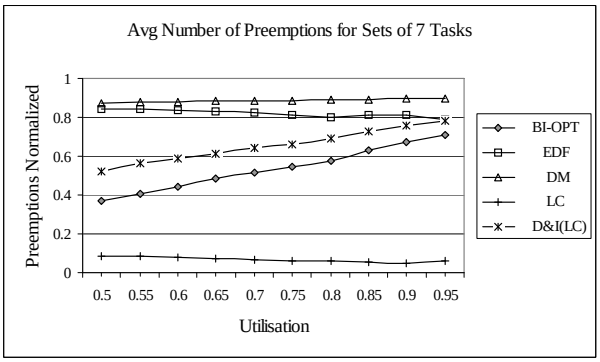


Figure 7. Minimizing the total number of preemptions.

the average index of 1000 sets of 7 tasks for a level of utilisation.

For high utilisations the number of feasible task sets decreases, and hence the difference between DMPO and D&I is reduced. On the other hand, the difference against S^A growths slightly. Note that by the optimality of D&I, it is never the case that the index of S^D or S^A can be lower than S^*.

The graphics for task sets with N equal to 4, 5, 6 and 8 are not depicted because they are similar to figure 5. More interesting is to show how fast the solutions S^D or S^A move away from S^*. Consider the data in figure 5; computing the distances $Z_I(S^D) - Z_I(S^*)$ and $Z_I(S^A) - Z_I(S^*)$ for their respective utilisations and calculating its average give us the average distance $\Psi(S^D)$ and $\Psi(S^A)$ respectively. Computing these distances for different N give us figure 6; this is the variation of the distance to S^* with respect the number of tasks per set. Note how fast they move away from S^*.

7.2 Minimizing the number of preemptions

This experiment quantifies the impact that using the D&I algorithm has on an overall system metric. For illustrative purposes we consider the problem of meeting the deadlines and reducing the total number of preemptions.

We note that in a fixed priority system, a task is preempted only by high priority tasks. In addition, if the task remains active during less time (e.g. shorter C), the high priority tasks will have less chance of interrupting it. Therefore, we can conjecture that tasks with shorter C should have lower priorities (i.e. they are less important) to reduce the possibility of being preempted, or reciprocally: "τ_i is more important than τ_j if $C_i > C_j$". We verify this conjecture by performing extensive simulations as follow:

A particular set of N tasks is simulated $N!$ times, once per different priority ordering, during a time window of 10000 ticks. Its number of preemptions is computed and stored in a table *[ordering , preemptions]* and then the preemptions are normalized according their minimum and maximum. Thus, the normalized number of preemptions for any possible priority assignment is recorded. This table allows us to find the optimal ordering *S-OPT* for the single criteria problem of minimizing the number of preemptions (no considering deadlines), which is the minimum in the table. We tested different priority assignments such as the shortest computation time first ($1/C$), the largest computation time first (*LC*), the largest period first and other simple rules and we found that none of them achieved *S-OPT*; among these rules, *LC* was the best.

Based on this, for each one of the task sets analysed we assign importances based on the *LC* criteria: *the largest the computation time, the higher the importance*. We then use the D&I algorithm to find a feasible schedule which is closer to the *LC* ordering (D&I(LC)). To compare the performance, we also display other indicative priority assignments, namely deadline monotonic (*DM*), largest computation time first (*LC*), and the Earliest Deadline First scheduling policy (*EDF*). In addition we find, by force brute, the optimal bicriteria solution (*BI-OPT*) examining the $N!$ elements in the table.

Figure 7 shows the results of this experiment for sets of 7 tasks. The graphics for sets of 4, 5 and 6 tasks are similar (no showed). We do not compute the task sets of 8 tasks because it is computationally expensive ($8! \times 1000$ simulations). The *S-OPT* solution is zero and *LC* is significantly close to it (6.7% distant on average); however both them miss deadlines. The optimal solution for this bicriteria problem is *BI-OPT*. Note that for high utilisations, the number of feasible tasks sets decreases and therefore, the *BI-OPT* moves away from *LC*. The performance of *EDF* is better than *DM* but it is still poor compared the *BI-OPT*. Finally, our solution D&I(LC) is substantially closer to the optimum (11.9% distant on average) and much better than *EDF* and *DM* for all utilisations excepting 0.95. Naturally, being the rule *LC* a heuristic, our solution will also be necessarily a heuristic. However, the results indicate that the D&I algorithm provides a remarkable improvement over other approaches. Comparison with other on-line heuristics are not really possible as they do not guarantee deadlines, and only provide best effort figures.

8 Conclusions

On the context of fixed priority scheduling, we propose an approach to deal with the bicriteria problem where meeting the deadlines and maximising the QoS expressed as relative importances are the objectives.

The solution that maximises the importance is not necessarily feasible and therefore, we reformulate the bicriteria problem as finding a feasible priority ordering that minimises the distance to the most important ordering.

In order to solve the bicriteria problem, we present the optimal algorithm D&I, which performs a branch and bound search into the space formed by the $N!$ possible orderings, ordered lexicographically by importance. Taking advantage of some interesting properties of the optimality of the deadline monotonic priority ordering, and some particular characteristics of the lexicographic order, we identify some patterns into the search space. This permits to solve the problem in $O((N^2 + N)/2)$ steps; however its complexity is pseudo-polynomial due to the fixed priority feasibility test.

Our approach is optimal in the sense that the priority ordering found is feasible and no other feasible priority assignment exist with higher importance. It is valid for any fixed priority system.

Complex frameworks exist to express the relative importances but in our model, we only require that the importance constitutes a totally ordering set. Sometimes a total ordering could not be established and hence, we suggest using the deadlines to make the decision; i.e. if two tasks are incomparable by importance, the shortest deadline precedes; in this way, the total ordering is obtained.

References

[1] A. N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[2] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, Dept. Computer Science, University of York, 1991.

[3] H. Aydin, R. Melhem, D. Mosse, and P.M. Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proc. of the 13th EuroMicro Conference on Real-Time Systems*, Delft, Netherlands, Jun 2001.

[4] R. Brafman and C. Domshlak. Introducing variable importance tradeoffs into cp-nets. In *Workshop on Planning and Scheduling with Multiple Criteria*, April 2002.

[5] A. Burns and A.J. Wellings. *Real-Time Systems and Programming Languages*. Addison Wesley, 3rd edition, 2001.

[6] Alan Burns. Preemptive priority-based scheduling: an appropriate engineering approach. In *Advances in real-time systems*, pages 225–248. Prentice-Hall, Inc., 1995.

[7] R. Dobrin and G. Fohler. Reducing the number of preemptions in standard fixed priority scheduling. In *Proc. 14th Euromicro International Conference on Real-Time Systems, Work in Progress Session*, 2002.

[8] Tom Gilb. Towards the engineering of requirements. Requirements Engineering 2, 165-169., 1997.

[9] IEEE. *IEEE Recommended Practice for Software Requirements Specifications*. IEEE Standard 830-1998, 1998.

[10] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time operating systems. In *IEEE Real-Time Systems Symposium*, pages 112–122, 1985.

[11] R. Krishnapura and S. Goddard. Dynamic real-time scheduling for energy conservation in i/o devices. In *Workshop on Constraint-Aware Embedded Software*, volume 2. 24th IEEE Real-Time Systems Symposium, Dec 2003.

[12] R. Rajkumar L. Sha and J.P. Lehoczky. Concurrency control for distributed real-time databases. *SIGMOD Rec.*, 17(1):82–98, 1988.

[13] P. Li, B. Ravindran, and E. D. Jensen. Adaptive time-critical resource management using time/utility functions: Past, present, and future. In *Proc. of the 28th Annual Int. COMPSAC'04 - Workshops and Fast Abstracts*, Washington, DC, USA, 2004. IEEE Computer Society.

[14] G. Lima and A. Burns. An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems. *IEEE Transactions on Computers*, 52(10):1332–1346, Oct 2003.

[15] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[16] M. F. Richardson N. C. Audsley, A. Burns and A. J. Wellings. Hard real-time scheduling: The deadline monotonic approach. In W. A. Halang and K. Ramamritham, editors, *Real-Time Programming*, pages 127–132. 1992.

[17] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 2002. ISBN: 0-13-028138-7.

[18] D. Prasad, A. Burns, and M. Atkins. The valid use of utility in adaptive real-time systems. *Real-Time Systems*, 25(2-3):277–296, 2003.

[19] H. Aydin R. Melhem, N. AbouGhazaleh and D.l Mosse. Power management points in power-aware real-time systems. In *Power aware computing*, pages 127–152. Kluwer Academic Publishers, 2002.

[20] Krithi Ramamritham. Real-time databases. *Distributed and Parallel Databases*, 1(2):199–226, 1993.

[21] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Comput.*, 39(9):1175–1185, 1990.

[22] Y. Shin and K.Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proc. of the 36th ACM/IEEE conference on Design Automation*, pages 134–139. ACM Press, 1999.

[23] Vincent T'kindt and Jean-Charles Billaut. *Multicriteria Scheduling Theory, Models and Algorithms*. Springer-Verlag, 2002.

[24] Y. Wang and M. Saksena. Scheduling fixed priority tasks with preemption threshold, 1999.

[25] Jia Xu and David Lorge Parnas. Priority scheduling versus pre-run-time scheduling. *Real-Time Syst.*, 18(1):7–23, 2000.

Near-Optimal Fixed Priority Preemptive Scheduling of Offset Free Systems

Mathieu Grenier* Joël Goossens+ Nicolas Navet*

LORIA-INRIA*
Campus Scientifique, BP 239
54506 Vandoeuvre-lès-Nancy- France
{grenier, nnavet}@loria.fr

Université Libre de Bruxelles+
Département d'Informatique, CP 212
1050 Bruxelles, Belgium
joel.goossens@ulb.ac.be

Abstract

In this paper, we study the problem of the fixed priority preemptive scheduling of hard real-time tasks. We consider independent tasks, which are characterized by a period, a hard deadline, a computation time, and an offset (the time at which the first request is issued) where the latter can be chosen by the scheduling algorithm.

Considering only the synchronous case is very pessimistic for offset free systems, since the synchronous case is the worst case in terms of schedulability. In this paper, we propose a new technique, based on the Audsley's priority assignment, that reduces significantly the search space of the combinatorial problem consisting in choosing the offsets. In addition, we propose new offset assignment heuristics and show the improvement of combining the new technique and the new heuristics.

1. Introduction

Problem definition. This study deals with the fixed priority preemptive scheduling of tasks in a real-time systems with *hard constraints, i.e.,* systems in which the respect of time constraints is mandatory. More specifically, we consider *offset free systems* where the offsets can be chosen by the scheduling algorithm. The activities of the system are modeled by independent *periodic tasks* τ_i as introduced in [8]. The model of the system is defined by a task set Δ of cardinality n, $\Delta = \{\tau_1, \tau_2, ..., \tau_n\}$. A periodic task τ_i is characterized by a quadruple (C_i, T_i, D_i, O_i) where each request of τ_i, called instance, has an execution time of C_i, a relative deadline D_i. T_i time units separate two consecutive instances of τ_i (hence T_i is the period of the task). The first instance of τ_i occurs at time O_i (the task offset in the following). The system is said schedulable if each instance finishes before its deadline.

Three different kinds of periodic task sets can be distinguished: *synchronous* sets, where all offsets are equal to 0, *asynchronous* sets, in which the constraints of the system determine the offsets, and finally *offset free* sets. In *offset free* systems, there is no constraint on offsets, hence they may be chosen beforehand by the scheduling algorithm. It may be noticed, that considering only the synchronous case is very pessimistic, since the synchronous case is the worst case, in the sense that, if the system is schedulable in the synchronous case it follows that this is also the case in all asynchronous situations (see [3], for instance). Our scheduling problem is the following, given the task characteristics T_i's, C_i's, and D_i's, determine a feasible offset (if any) and fixed priority assignment.

Related work. In [5], the concepts of *concrete* and *non-concrete* task sets are introduced. A *non-concrete* task set is a set for which the offsets are not determined, a *concrete* version of such a task set can be obtained by considering a particular offset configuration. Hence, a non-concrete task set *generates* a collection of concrete task sets. A non-concrete task set Δ is schedulable [5], if all the corresponding concrete sets are schedulable. While an offset free system is schedulable if at least one concrete task is schedulable.

Well-known results concern the optimality for asynchronous (and synchronous) task sets. But first a definition, a priority assignment rule is optimal for asynchronous (resp. synchronous) systems if, when a schedulable priority assignment exists for some asynchronous task set (resp. for the synchronous case), the priority assignment given by the rule is also schedulable. In [7], the non-optimality of the Deadline Monotonic (i.e., lower the deadline, higher the priority) is proven, and an optimal priority assignment rule is suggested by considering the $n!$ different priority assignments. In [1, 2] Audsley proposed an optimal priority assignment algorithm, which examines at most n^2 priority assignments, this algorithm is often referred as the Audsley's algorithm in the literature.

More recent results concern the optimality for offset free systems. In [4, 3], the authors show the interest of offset free systems and in [4] the non-optimality of rate-monotonic assignments when offset free systems are considered. Although there is an infinite number of asynchronous cases for a task set, the problem is restricted [3] by considering only non-equivalent offset assignments with an optimal offset assignment rule. Since, the number of combinations remains exponential, an efficient heuristic with a lower complexity is proposed, named *dissimilar*

offset assignment.

Well-known results concern the schedulability analysis for synchronous systems [6, 10]. For asynchronous systems as well, schedulability analysis has been studied. Due to space limitation, we shall not give details here. We know for instance (see [7]) that $[0, O_{\max} + 2P)$ where P is the LCM of the periods and $O_{\max} = \max_j(O_j)$, is a feasibility interval.

Contributions. In this paper, we show how to use the Audsley's algorithm to reduce the complexity of *offset assignment* by decreasing the number of tasks examined in the assignment. The optimal offset assignment cannot always be used due to its exponential complexity. Then, we propose new assignment heuristics that improve significantly upon the one presented in [3] as it will be shown in the experiments.

Organization. Section 2 recalls the results from [3] that are useful for the understanding of our contribution. Section 3 shows how the Audsley's algorithm can be used to decrease the complexity of the offset assignment algorithm. New heuristics are then proposed in Section 4, whose efficiency are assessed in Section 5.

2. Known offset assignments

In this section, we summarize known results on the scheduling of offset free systems. In particular, we summarize the approach developed in [3].

2.1. Scheduling of offset free systems

The topic of this study is the fixed (and preemptive) scheduling of offset free systems. In these systems, the offset of the tasks can be chosen by the scheduling algorithm. Consequently, we have to choose (off-line):

- the task priorities, and
- the task offsets.

2.2. Optimal offset assignment

Let us assume that the priorities of the tasks are already fixed, and we consider the specific priority assignment $\mathcal{P}$, which could be for instance the Deadline Monotonic. We consider *fixed* priority scheduler, hence at each time instant, the scheduling policy assigns the CPU to the instance of task with the highest priority (if any). Suppose that the system is not schedulable in the synchronous case with $\mathcal{P}$, we would like to find an asynchronous situation for which the system is schedulable. In the following, we shall distinguish between two kinds of optimality:

Definition 1 *A priority assignment rule $\mathcal{P}$ is optimal in the asynchronous case, if when a schedulable priority assignment exists, $\mathcal{P}$ provides a schedulable system in the very same asynchronous situation.*

Definition 2 *An assignment offset rule $\mathcal{O}$ is optimal under a priority allocation rule $\mathcal{P}$, if when a schedulable offset assignment exists with $\mathcal{P}$, $\mathcal{O}$ provides a schedulable asynchronous situation with the very same priority assignment $\mathcal{P}$.*

The optimal offset assignment considered in [3] is summarized in this section. The main idea is to test the schedulability of all the non-equivalent asynchronous situations of a task set.

All offset combinations may be found by restricting the offsets such as $O_1 = 0$ and $\forall i \in [2, n] \mid O_i \in [0, T_i)$. Consequently number of combinations is upper bounded by $\prod_{i=2}^{n} T_i = \mathcal{O}((\max_{2 \le j \le n} T_j)^{n-1})$.

To further reduce the number of offset assignments, it is possible to consider only offset assignments leading to non-equivalent asynchronous situations. Two asynchronous situations are defined to be *equivalent*, if they have the same periodic behavior. Indeed, the schedule becomes periodic with a period of $P = \text{lcm}\{T_1, .., T_n\})$. This periodic behavior only depends on the relative phasing of the task instances, i.e., on the tuple $(O_1 \pmod{T_1}, O_2 \pmod{T_2}, \ldots, O_n \pmod{T_n})$. This tuple characterizes the relative time shift between the instances of various tasks [4].

For two tasks τ_1 and τ_2, two choices ($O_2 = O_1 + v_1$ and $O_2 = O_1 + v_2$) are said equivalent if they define the same relative phasing:

$$\begin{array}{c} \exists k_1, k_2 \in \mathbb{N} : (O_1 + v_1 + k_1 \cdot T_2) \bmod T_1 \\ = \\ (O_1 + v_2 + k_2 \cdot T_2) \bmod T_1 \, , \end{array} \tag{1}$$

which is equivalent to:

$$v_1 \equiv v_2 (\text{mod } \gcd\{T_1, T_2\}). \tag{2}$$

From Equations 1 and 2 it follows that only the values $0, 1, ..., \gcd\{T_1, T_2\} - 1$ must be considered and are non-equivalent choices for O_1 and O_2.

The optimal offset assignment algorithm, in order to explore all possible non-equivalent asynchronous situations for the task set, constructs iteratively the situations. First, it sets the non-equivalent choices for O_2 (the offset O_1 is arbitrarily fixed to 0) by considering for O_2 all integer values in the $[0, \gcd\{T_1, T_2\})$ interval. Next, by assuming at each step that the offsets $O_1, O_2, ...O_{i-1}$ are set, consider for the offset O_i the interval $[0, \gcd\{T_i, \text{lcm}(T_1, ..., T_{i-1})\})$ (instances of task sub-set $\{\tau_1, ..., \tau_{i-1}\}$ having a period of $\text{lcm}(T_1, ..., T_{i-1})$).

2.3. Dissimilar offset assignment

The method of [3], presented in Section 2.2, reduces the non-equivalent offset assignment from $\prod_{i=2}^{n} T_i$ to $\frac{\prod_{i=2}^{n} T_i}{P}$. Despite this significant reduction, the number of offsets considered by the optimal algorithm remains exponential. In [3], the author defines then a heuristic that provides a single offset assignment for a task set.

The basic idea of the heuristic is to shift away, as far as possible, the offsets of the tasks for which some instances would be most probably in conflict for the use of the CPU. Precisely, the offset of tasks having instances released in small periods of time, and thus being close to the "synchronous" case, will be shift away as far as possible. Hence, a measure is introduced to estimate the proximity of an offset assignment with the synchronous case. The dissimilar offset assignment algorithm allocates the offsets of the periodic tasks to maximize this measure, which is defined as the length of the shortest interval that contains at least one instance of each task.

The technique considers first the (minimal) distance between two instances of tasks τ_i and τ_j in the periodic part of the schedule. The computation of this distance is performed according to Theorem 1.

Theorem 1 ([3]) *Let $r \in [0, \gcd\{T_i, T_j\})$. If $O_i = O_j + r$ (or $O_j = O_i + r$), the minimum distance between an instance of τ_i and an instance of τ_j is $\min\{r, \gcd\{T_i, T_j\} - r\}$.*

It follows from Theorem 1 that the minimum distance between an instance of τ_i and τ_j is upper bounded by $\left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$ and corresponds to the offset assignment $O_i = O_j + \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$ (or $O_j = O_i + \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$). In this case, r is equal to $\left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$ or $\left\lceil \frac{\gcd\{T_i,T_j\}}{2} \right\rceil$.

The dissimilar offset assignment algorithm fixes the offsets of the periodic tasks. The algorithm sorts the couples of tasks (τ_i, τ_j) in decreasing value of $\gcd\{T_i, T_j\}$, in order to maximize the measure defined above. Next, it sets iteratively the offset O_i and O_j of the sorted couples of tasks (T_i, T_j) to obtain the highest minimum distance (i.e., $r = \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$). During this assignment, three cases may occur:

1. when O_i and O_j are not yet set, a random offset is chosen for O_i and $O_j = O_i + \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$,
2. when O_i (resp. O_j) is fixed and O_j (resp. O_i) is not, $O_j = O_i + \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$ (resp. $O_i = O_j + \left\lfloor \frac{\gcd\{T_i,T_j\}}{2} \right\rfloor$),
3. when O_j and O_i are already chosen, there is nothing to do.

The maximal time complexity of this algorithm for assigning the offsets is $\mathcal{O}(n^2 \cdot (\log T^{\max} + \log n^2)$ where $T^{\max} \stackrel{\text{def}}{=} \max_{1 \le k \le n}(T_k)$.

3. Complexity reduction

In this section we propose a technique, based on the Audsley's priority assignment, to reduce significantly the search space. But first, we shall present the Audsley's algorithm [1] itself.

3.1. Audsley's algorithm

The Audsley's algorithm [1] performs an optimal static priority assignment for asynchronous systems (according to Definition 1).

A priority assignment is defined by:

$$\gamma : \{1, 2, ...n\} \rightarrow \{\tau_1, \tau_2 ... \tau_n\},$$

where the assignment function $\gamma(i)$ gives the task τ_k assigned to the priority level i using the convention: lower the priority level, higher the priority.

The Audsley's algorithm considers at most $\mathcal{O}(n^2)$ distinct priority assignments. First, it attempts to find a *lowest priority viable* task τ_i in Δ, i.e., tries to assign the priority level n.

Definition 3 *Task τ_i is lowest priority viable when τ_i is assigned the lowest priority of any task in Δ and:*

- *The remaining tasks in Δ are assigned priorities in any arbitrary order, the sole restriction being that all these priorities be higher than the priority assigned to τ_i.*
- *During run-time scheduling, the semantics is weakened as follows: instances generated by tasks other than τ_i may miss their deadlines (if they do so, they continue execution until completion); however, instances generated by τ_i may not miss any deadlines.*

Next, the algorithm recursively determines a lowest priority viable task in the sub-set $\Delta \backslash \{\tau_i\}$ of $n-1$ tasks (i.e., assigning priority level $n-1$). The Audsley's pseudo-algorithm is given in Algorithm 1.

Input: task set $\Delta = \{\tau_1, \tau_2 ..., \tau_n\}$
Result: task set with no assigned priority

procedure `audsley(`Δ`);`
if $\Delta = \emptyset$ **then**
  `priority assignment succeed`:
  return Δ;
end
if *no task is lowest priority viable* **then**
  `priority assignment failed`:
  return Δ;
else
  let τ_i a lowest priority viable task;
  `assign lowest priority to` τ_i:
  $\gamma(|\Delta|) = \tau_i$;
  return `audsley(`$\Delta \backslash \{\tau_i\}$`);`
end

Algorithm 1: Audsley's algorithm.

After executing the Audsley's algorithm, two cases may occur:

1. The priority assignment of the Audsley's algorithm leads to a schedulable system (i.e., priority assignment succeed): the set of task Δ is schedulable with the priority assignment given by function γ.

2. Otherwise, the Audsley's algorithm fails to assign the priority of level i where $i \in [1, n]$ (i.e., priority assignment failed). However, instances of the set of tasks $\{\gamma(i+1), \gamma(i+2), ..., \gamma(n)\}$ meet their deadline. Indeed, the schedulability of a task at a priority level, with a fixed scheduling preemptive policy, depends only on the set of higher priority tasks, whatever the assignment of priority among this set [1, 2].

The non-optimality of the Audsley's priority assignment for offset free systems

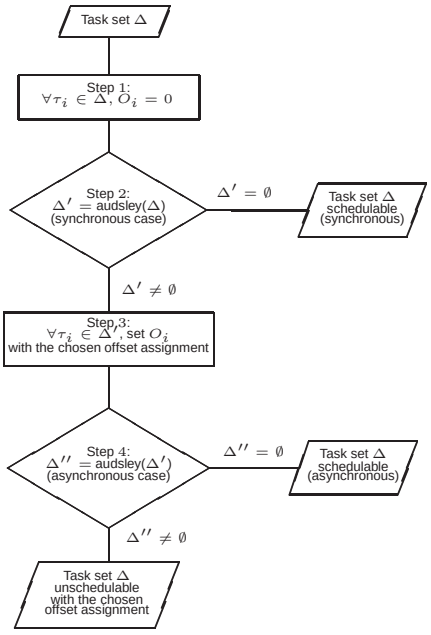
We shall see that while the Audsley's priority assignment is optimal for asynchronous systems it is not the case for offset free systems. But first a definition.

Definition 4 ([4]) *A priority assignment rule is optimal for offset free systems if when a schedulable priority assignment ($\mathcal{P}$) and offset assignment ($\mathcal{O}$) exist for some offset free task set, there is a schedulable offset assignment ($\mathcal{O}'$) for the priority assignment given by the rule.*

The priority assignment of the Audsley's algorithm depends on the offset assignment $\mathcal{O}$, actually Audsley consider *asynchronous systems* (the offsets must be already fixed). Thus, Definition 4 is not applicable in the case of the Audsley's priority assignment and consequently the Audsley's priority assignment is not optimal for offset free systems.

3.2. Reducing the search space using the Audsley's algorithm

In this section, we shall explain how to assign the priorities *and* the offsets together. Figure 2 presents the flow of our approach in a pseudo-algorithmic form. First, we initialize the offsets to consider the *synchronous* situation. Then, the Audsley's algorithm is used to assign priorities (in the synchronous case), more precisely the (recursive) function `audsley` (Algorithm 1) is used. If it successfully assigns priorities (case 1, Section 3.1), the system is schedulable in the synchronous case. Otherwise, the Audsley's algorithm fails in the synchronous case (case 2, Section 3.1), a schedulable asynchronous situation should be looked for. Consequently we first use at this step a rule to choose the offsets—for the subset of tasks returned by `audsley`: $\Delta' \stackrel{\text{def}}{=} \Delta \backslash \{\gamma(i+1), \gamma(i+2), ..., \gamma(n)\}$—and *then* the priorities using the Audsley's algorithm for the second times *but* on the subset Δ' (not on the original task set). Indeed, the sub-set of tasks $\{\gamma(i+1), \gamma(i+2), ..., \gamma(n)\}$ respects their timing constraints in the synchronous situation without considering the offsets and the priorities among the set of higher priority tasks. Thus, the tasks in $\{\gamma(i+1), \gamma(i+2), ..., \gamma(n)\}$ are lowest priority viable in the synchronous case. Since the synchronous case is the worst case, these tasks remain lowest priority viable in an asynchronous situation. That is why, in the following, the offset assignment scheme can safely take into account only the tasks in the set Δ'.



Algorithm 2: Offset and priority allocation algorithm.

With this method the number of tasks to consider for the offset assignment is much lower as it will shown in the experiments of Section 5.2. Since the time complexity of the offset assignment depends on the number of tasks and their periods, the time complexity is, thus, reduced.

4. Near-optimal offset assignment heuristics

In this section, we propose several assignment heuristics, which provide alternative offset allocations when the dissimilar offset assignment fails to produce a schedulable asynchronous situation.

The functioning scheme of these new heuristics is very similar to the one of the dissimilar offset assignment: couples of tasks are ordered according to a criteria, then the task offsets are chosen from the top of the resulting ordered list to its bottom. The new heuristics provide different offset allocations than the dissimilar offset strategy since they do not only consider the minimal distance between tasks. For instance, some try to "separate" tasks with the highest utilization rate (i.e. $\frac{C_k}{T_k}$). We propose 4 new offset assignment heuristics that take into account other characteristics of the task set than the minimal distance between tasks. Our 4 heuristics consider the couples (τ_k, τ_i) by decreasing values of:

1. $\left(\frac{C_k}{T_k} + \frac{C_i}{T_i}\right) \cdot \text{gcd}(T_k, T_i)$
2. $\max\left(\frac{C_k}{T_k}, \frac{C_i}{T_i}\right) \cdot \text{gcd}(T_k, T_i)$
3. $\frac{C_k}{T_k} + \frac{C_i}{T_i}$
4. $-\text{gcd}(T_k, T_i)$

The heuristics 1,2 and 3 sort the couples of tasks by considering their utilization rate. Different ways of introducing the utilization rate in the ordering provide several asynchronous situations, which may lead to a schedulable asynchronous situation. In heuristic 1 (resp. 2), the utilization rate of the couples of tasks (resp. the maximal utilization rate) is taken into account balanced by their gcd. Rule 3 arranges the (τ_k, τ_i) according to decreasing utilization rate.

Heuristic 4 first focuses on the couples of tasks (τ_k, τ_i) for which the minimal length between instances is small. The (τ_k, τ_i) are thus ordered according to decreasing value of $-\text{gcd}(T_k, T_i)$ to set the offset of the couples with the less choices in the offset assignment.

These new assignment heuristics are considered together. The combined used of these heuristics, in our experiments (Section 5.5), provides a "near-optimal" offset assignment. The complexity of these new heuristics is identical as the one of the dissimilar offset assignment (i.e., $\mathcal{O}(n^2 \cdot (\log T^{\max} + \log n^2))$, because the algorithm that performs the assignment is the same, except for the ordering of the couples of tasks.

5. Experimental results

In this section, we present our experimental results. We make use of the Algorithm 2 defined in Section 3.2.

5.1. Experimental setup

In the experiments, the global load U is chosen for each set Δ of n tasks. Since the sets Δ have to be unschedulable in the synchronous case, the load U has to be sufficiently high. The utilization rate ($\frac{C_k}{T_k}$) of each task τ_k is uniformally distributed in the $\left[\frac{U}{n} \cdot 0.9\ ,\ \frac{U}{n} \cdot 1.1\right]$ interval. The computation time C_k of each task τ_k is randomly chosen with an uniform law in the $[c_{\min}, c_{\max}]$ interval, the relative deadline D_k is uniformally chosen in the $[d_{\min}, d_{\max}]$ interval , and the period T_k is upper bounded by $t_{\max}$.

In the following, we make use of the tuple $(n, U, c_{\min}, c_{\max}, d_{\min}, d_{\max}, t_{\max})$ to denote the actual parameters used in our task sets random generation.

5.2. Complexity reduction using the Audsley's algorithm

In this section, the actual reduction of the search space using the Audsley's algorithm is studied. The improvement is evaluated with task sets randomly generated according to the tuple $(n, 0.8, 2, 30, T_k - 0.9 \times (T_k - C_k), T_k + 0.9 \times (T_k - C_k), 200)$ with n being the number of tasks in the $[5, 17]$ interval. We made approximately 13000 simulations for each graph (13 points per graph).

In Figure 1, the curve in plain style presents the percentage of task sets unschedulable in the synchronous case which have at least one lowest priority viable task in the synchronous situation. One can observe that at least 38 % of the task sets include a lowest priority viable task. For

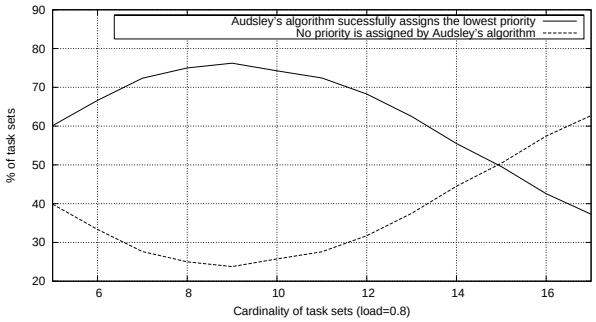


Figure 1. Percentage of unschedulable task sets in the synchronous case, which includes at least one lowest priority viable task in the synchronous case.

these task sets, the Audsley's algorithm (step 2, Algorithm 2) allows to reduce the number of tasks in the offset assignment. One can also note from Figure 1 that the percentage decreases with the number of tasks. This phenomenon is probably related to our task generation algorithm. Indeed, in order to keep the lcm of the tasks within bounds that still allow to assess the feasibility by simulation, restrictions are imposed on the task set characteristics. When the number of tasks becomes large, the tasks tend to have the same characteristics and they tend thus to behave in a rather similar manner. Hence, when a task is not lowest priority viable, the probability to find another lowest priority viable task is rather low.

In Figure 2, we consider only task sets which have at least one lowest priority viable task. The curve in plain style shows the percentage of tasks being lowest priority viable after step 2, Algorithm 2 (i.e., tasks in the set $\Delta \setminus \Delta'$). The dotted curve represents the percentage of tasks τ_j, which are not (i.e., tasks in the set Δ').

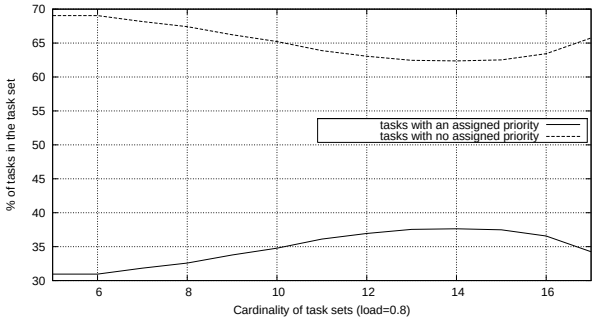


Figure 2. Proportion of lowest priority viable tasks.

As can be seen from the plot of Figure 2, at least 30 % of tasks are lowest priority viable (in the synchronous case). Thus, less than 70 % of the tasks have actually to

be considered for the offset assignment.

In order to accurately evaluate the complexity reduction obtained with the Audsley's algorithm, we study the actual reduction of the search space brought by the use of the Audsley's algorithm. In Figure 3, we consider again only task sets with at least one lowest priority viable task. The curve shows the percentage of search space reduction.

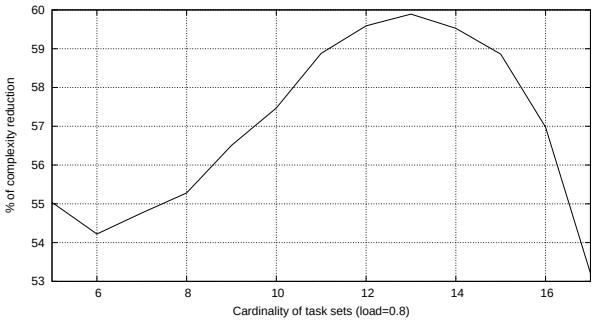


Figure 3. Search space reduction using the Audsley's algorithm.

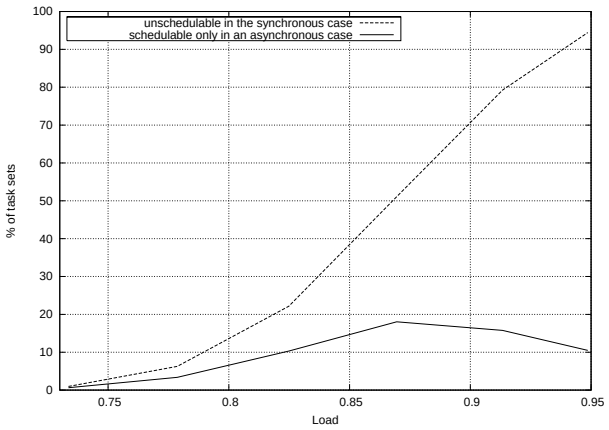


Figure 4. Percentage of systems unschedulable in the synchronous case (dotted curve) and systems only schedulable with an asynchronous configuration (plain curve). The cpu load ranges from 0.7 to 0.95 .

From the simulation results, presented in Figure 3, the search space reduction is always greater than 53 %.

The conclusion that can be drawn from these experiments is that for a very significant number of systems (more than 38 % in our experiments), at least 30 % of the tasks can be allocated a priority by the Audsley's algorithm (i.e., are lowest priority viable). This allows to reduce the search space of the offset assignment scheme by at least 53 %.

5.3. Offset free for increasing feasibility

This subsection aims to show the interest of offset free systems for schedulability, by using the optimal offset assignment.

Task sets are randomly generated according to the tuple $(5, U, 2, 30, T_k - \frac{T_k - C_k}{2}, T_k, 30)$ with U chosen in the $[0.73, 0.95]$ interval. We made approximately 6000 simulations for each graph (6 points per graph). It should be noticed that the time complexity of the optimal assignment rule, that is used in these experiments, is high, and checking if a system is schedulable or not may require a very long computation time (since we have to consider—in the worst case—all non-equivalent offset assignments). For this reason, we have strongly limited the number of tasks n and the maximum value of the periods in our simulations to reduce the number of non-equivalent offset assignment and thus diminish the complexity of the schedulability.

We now evaluate the percentage of systems unschedulable in the synchronous case which becomes schedulable in an asynchronous case (i.e., we use the optimal offset assignment). Once again, we use Algorithm 2 to determine these percentages. Figure 4 represents the percentage of systems unschedulable in the synchronous case (dotted curve), and the percentage of systems schedulable only in an asynchronous case (plotted style curve). From Figure 4, one can observe that the percentage of task sets unschedulable in the synchronous case increases with the load, which confirm the intuition that it is harder to find a schedulable system when the load is high. Moreover, the percentage of task sets schedulable in an asynchronous situation increases with the load (up to 18 %) until the load reaches 0.87, then it starts to decrease. Intuitively, it is clear that task sets tend to be unschedulable, whatever the offset allocations, when the load becomes too high.

5.4 Combined use of the heuristics: efficiency compared to the optimal allocation

Figure 5 shows the percentage of task sets schedulable in a particular asynchronous situation (non-equivalent to the synchronous situation) which remains schedulable with the dissimilar offset assignment rule (dashed curve) and with at least one of our new heuristics (curve in plain style).

As can be seen on Figure 5, the assignment heuristics find a schedulable asynchronous situation for at least 51 % and up to 95 % of the task sets in which such a situation exists. The chance of finding a schedulable assignment logically decreases with the load.

The combined used of the heuristics enables us to find an important percentage of the schedulable asynchronous situations. From Figure 5, it is obvious that the combination of our new heuristics outperforms the dissimilar offset assignment.

5.5. Relative performances of the heuristics

In this section, the improvement brought by the new heuristics is discussed more precisely. Task sets are ran-

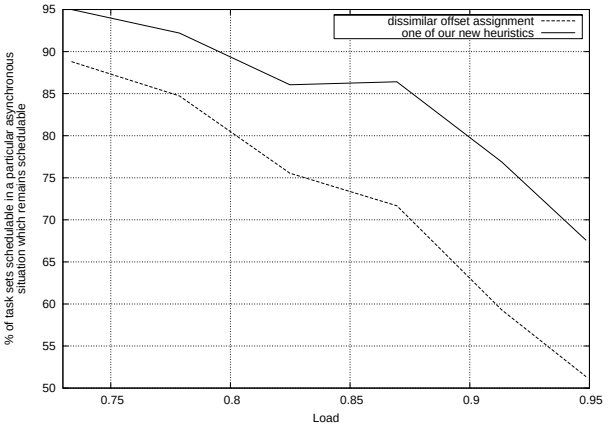


Figure 5. Dissimilar offset assignment vs. our new heuristics.

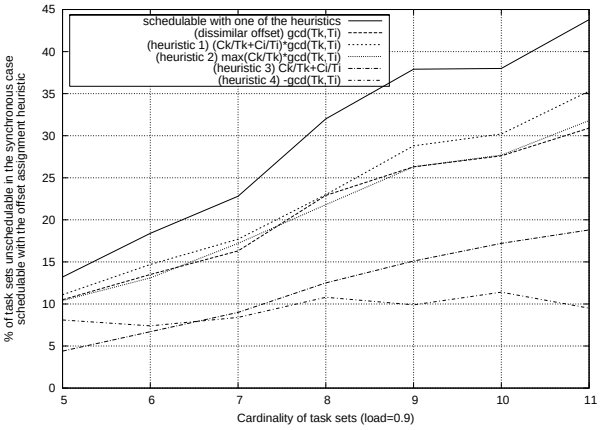


Figure 7. Percentage of the task sets unschedulable in the synchronous case that becomes schedulable with the different offset assignment heuristics (90 % CPU load).

domly generated according to the tuple $(n, U, 2, 30, T_k - \frac{T_k - C_k}{2}, T_k, 30)$ with U chosen in the $\{0.8, 0.9\}$ set and n in the $[5, 11]$ interval. We made approximately 7000 simulations for each graph (7 points per graph).

The offsets and priorities assignment are performed according to Algorithm 2 of Section 3.2. At step 4, the asynchronous situations correspond to the offset assignments produced by the dissimilar offset assignment and by the new heuristics.

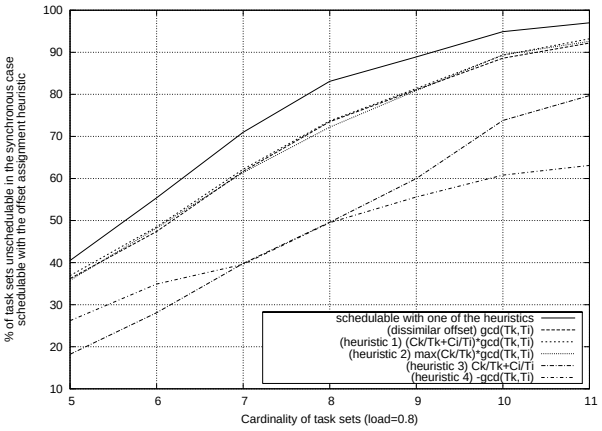


Figure 6. Percentage of the task sets unschedulable in the synchronous case that becomes schedulable with the different offset assignment heuristics (80 % CPU load).

Figure 6 and 7 display the percentage of tasks sets unschedulable in the synchronous situation which become schedulable in the asynchronous situation produced by each of the heuristics. The experiments are done with a global load of 0.8 in Figure 6 and of 0.9 in Figure 7. From these Figures, one sees that the offset assignment heuristics significantly increase the schedulability compared the synchronous case. For instance, in Figure 6, the percentage of task sets schedulable with an asynchronous situation produced by the heuristics is at least 40.5 % and up to 97 %. The improvement steadily increases with the number of tasks: for instance, in Figure 6, the percentage of schedulable task sets in an asynchronous situation is equal to 71 % for 7 tasks, while it is 88.9 % for 9 tasks. This can be intuitively explained by the fact that the higher the number of tasks, the higher the freedom degree to set the offsets, and thus, the farther from the synchronous case the system can be.

One also observes that, very logically, the percentage of systems schedulable in an asynchronous situation strongly decreases when the load is high. For instance, the percentage of schedulable systems for sets of 8 tasks is 83.1 % for a load of 0.8 of 32 % for a load of 0.9 (Figure 7).

The different heuristics can be compared using Figure 6 and 7. We observe that the dissimilar offset assignment performs very well, usually better than the new heuristics. However, using all heuristics together (i.e., try the offset assignment returned by each of the heuristics) allows to clearly outperform the dissimilar offset assignment alone. The heuristics (including the dissimilar offset assignment) are in some way very complementary. For instance, 37.9 % of the task sets are schedulable with at least one of our heuristics for 9 tasks (Figure 7) while only 26.3 % are schedulable with the dissimilar offset assignment. It is worth noting that the complexity of each of the new heuristics is the same as the dissimilar offset assignment and, in practice, the computing time does not raise problem whatever the cardinality of the task set.

In conclusion, our experiments show that the combined used of all the heuristics lead to a near near-optimal offset assignment, which allows to increase considerably the

percentage of systems schedulable compared to the sole asynchronous situation.

6. Conclusion

In this paper, we have studied the problem of the static preemptive scheduling of offset free systems. First, we have shown that the search space for assigning the offset may be reduce of up to 50 % with an appropriate use of the Audsley's algorithm. Then, new heuristics are proposed to improve upon the result of the dissimilar offset assignment scheme introduced in [3]. These heuristics provide alternative asynchronous cases, which allow to increase very significantly the number of schedulable systems with regards to pessimistic synchronous case. The combined use of all these heuristics provides a near-optimal offset assignment. Indeed, according to our experiments conducted with for a global load of 0.8 for task sets having a cardinality in $[5, 11]$, the set of heuristics enables to schedule at least 40.5% and up to 97% of task sets, which are unschedulable in the synchronous case.

A similar study remains to be conducted for the non-preemptive case, which is of interest for scheduling frames on networks but also for many small embedded systems without preemptive capabilities. In a first step, it has to investigated whether a similar complexity reduction procedure based on the Audsley's algorithm can be devised for the non-preemptive case (see [9] for some results on the use of the Audsley's algorithm in the non-preemptive case). Then, offset assignment heuristics dedicated to the non-preemptive case have to be proposed and their efficiency evaluated.

In the future, we also intend to evaluate if integer linear programming can be used to determine offsets in an efficient manner; the main problem will be here to define the cost functions that lead to schedulable systems.

References

[1] N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Report YO1 5DD, Dept. of Computer Science, University of York, England, 1991.

[2] N. Audsley. On priority assignment in fixed priority scheduling. *Inf. Process. Lett.*, 79(1):39–44, 2001.

[3] J. Goossens. Scheduling of offset free systems. *Real-Time Systems*, 24(2):239–258, March 2003.

[4] J. Goossens and R. Devillers. The non-optimality of the monotonic priority assignements for hard real-time systems. *Real-Time Systems*, 13(2):107–126, sep 1997.

[5] K. Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proc. of the 12th IEEE Real-time Systems Symposium (RTSS 1991)*, 1991.

[6] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proc. of the 11th IEEE Real-Time Systems Symposium*, pages 201–213, Florida, USA, 1990.

[7] J. Leung and J. Whitehead. On the complexity of fixed priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[8] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard-real time environnement. *Journal of the ACM*, 20(1):40–61, 1973.

[9] R. Saket and N. Navet. Frame packing algorithms for automotive applications. Available as research report INRIA RR-4998, to appear in Journal of Embedded Computing, 2006.

[10] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

Network

Worst-case analysis of a mixed CAN/Switched Ethernet architecture

Jérôme Ermont, Jean-Luc Scharbarg, Christian Fraboul
IRIT-ENSEEIHT
2, rue Camichel
31000 Toulouse - France
Jerome.Ermont@enseeiht.fr

Abstract

Embedded systems have specific real-time requirements that led to the development of dedicated communication protocols. Such systems often face increasing communication needs and the integration of switched Ethernet architecture. But moving from existing dedicated fieldbusses architectures to new Ethernet based architectures is not always easily feasible, due to industrial constraints.

In this paper, we evaluate a solution for integrating existing data busses (such as CAN, which is an important standard in automotive context) on a global architecture that respects increasing bandwidth requirements. We consider both event-triggered and time-triggered solutions, incorporating the scheduling on CAN and the CAN/Ethernet bridging strategy. The evaluation is performed using timed automata and UPPAAL, and aims at bounding end-to-end delays and jitter.

1. Introduction

Fieldbusses , e.g., CAN [16], WorldFIP [29], Profibus [29] have been developed in the context of real-time applications (distributed computer control systems) that have specific communication requirements such as:

- bounded end-to-end transmission delays in order to guarantee respects of deadlines,
- the bounded and small jitter for periodic traffic.

However, the amount of information that are nowadays exchanged in such systems have been increasing steadily and is now reaching the limits of traditional fieldbusses, especially in terms of bandwidth [12].

Switching from dedicated fieldbusses to Ethernet is a classical trend in embedded systems due to the wide acceptance of the Ethernet standard and its evolution toward a more predictable switched architecture.

However, successful experience with introduction of a switched Ethernet in avionic systems (AFDX, [4, 15]) is mainly due to the preservation of the applications communication model (periodic schemes) and the respect of the expected real time properties (bounded delay).

The goal of the study presented in this paper is to build an heterogeneous architecture obtained by interconnecting existing CAN data busses on a switched Ethernet. In this context, we aim at comparing, on the one hand, event-triggered systems using native CAN MAC and an event-triggered strategy for transmitting CAN frames on Switched Ethernet, on the other hand a system with a time-triggered behavior (TTCAN like [17]) on the whole network.

The proposed evaluation is based on timed automata. It aims at determining worst-case end-to-end delays and jitters on CAN frames.

Section 2 presents briefly TTCAN and switched Ethernet technologies and defines the heterogeneous CAN/Switched Ethernet architecture studied in this paper. Section 3 presents the application traffic over the network and proposes event-triggered and time-triggered strategies to schedule the traffic. Section 4 describes the modelling of the application and the network with timed automata. Section 5 shows how worst-case delays on CAN frames can be calculated. Section 6 concludes the paper and presents points we are presently studying and some ideas for future work.

2. Network architecture

The network architecture will comprise the two communication technologies CAN and switched Ethernet. In this section, we present briefly those two technologies. Then, we describe the network architecture that we will consider in the remaining of the paper.

2.1. CAN

CAN (Controller Area Network) [16] is a serial communication protocol suited for networking sensors, actuators and other nodes in real-time systems. The CAN specification defines several versions of the protocol for the physical and the data link layer. In this paper, we shortly present CAN 2.0 A. Several application layer protocols have been proposed (CANOpen, CAN Kingdom, . . .).

The CAN addressing system is based on message identifiers: a frame does not have a destination nor a source address. Frames are broadcasted on the bus. Stations get

the frames they are interested in by a filtering process of the identifiers.

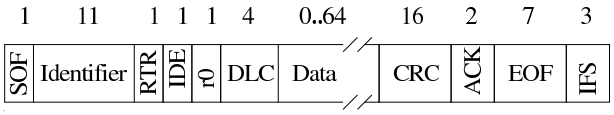


Figure 1. CAN frame (sizes in bits)

The frame format is depicted in Figure 1. The details of each field will not be presented. The relevant fields for the remainder of the paper are the following:

- the identifier field, which as mentioned earlier identifies the data contained in the frame,
- the DLC field which gives the length (in bytes) of the data field,
- the data field which is the payload of the frame.

Bit-stuffing is used to avoid the transmission of long sequences of bits with identical value [23]. As soon as 5 bits of identical value are transmitted, a bit of opposite value is automatically inserted. This mechanism is valid for the whole frame, except IFS, EOF, ACK and the last bit of CRC.

The medium access method (MAC) is CSMA/CR (Carrier Sense Multiple Access / Collision Resolution): the starting of frame transmissions on the bus are synchronous. When two or more stations start a transmission simultaneously, the one with the highest priority identifier (lowest value) wins and the others stop their transmission. This is implemented by a collision detection on a bit by bit basis. When a station transmits 1 (recessive bit) and detects 0 (dominant bit), it knows that a frame with a higher priority is being transmitted and, consequently, it immediately stops transmission. This mechanism guaranties strict priority order on identifiers, provided identifiers are unique. It implies limitations of the bandwidth and the maximal length of the bus (e.g., 1 Mbs for 40 meters).

Some drawbacks of the CAN native MAC have been identified. First, it is event-triggered: when a station has a frame to transmit, it tries to. It will succeed as soon as no frame with a higher priority is being transmitted. This mechanism can induce large jitter on periodic frames. Second, identifiers are associated with frames statically. This imposes a scheduling algorithm using static priorities, e.g., rate monotonic [22] when periodic traffic is considered. It is well known that higher utilization of the medium is obtained with a scheduling algorithm using dynamic priorities, e.g., Earliest Deadline First [22].

Solutions have been proposed to solve those drawbacks. Most of them add a protocol over CAN native MAC (e.g., [11, 24]).

Time triggered CAN (TTCAN) [14, 17] is a well-known solution. It imposes a static scheduling on CAN. This scheduling is memorized in a table, the matrix cycle, which is known by all the stations. This scheduling comprises in particular exclusive, arbitration and free windows. Each exclusive window is dedicated to exactly one frame identifier while an arbitration window is shared. Free windows allow some evolution of the application. The stations are resynchronized with a trigger message broadcasted periodically by the master station. Figure 2 gives an example of a matrix cycle. Each line of the table is called a micro-cycle and has a duration D_{uc}. The number of micro-cycles in the table is a power of 2. All the windows in a given column of the table have the same duration. The reference message can be easily recognized by its identifier. With TTCAN's level 1, the reference message only holds some control information in one byte. In extension level 2, the reference message holds additional control information (e.g., the global time information of the master) and covers four bytes.

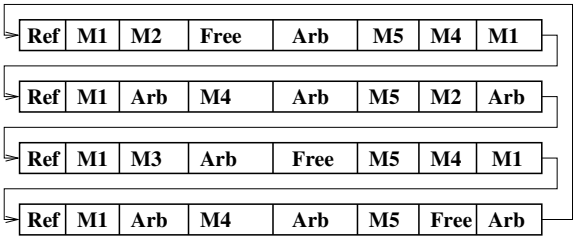


Figure 2. Example of a matrix cycle

In the remaining of the paper, we will consider both native CAN MAC and TTCAN.

2.2. Full Duplex Switched Ethernet

Full Duplex Switched Ethernet is an enhancement of Ethernet. The Ethernet link layer [13] is designed for computer local networks where high bandwidth and low cost hardware is more important than guaranteed deadlines and/or jitter.

The Ethernet addressing system is based on MAC addresses: each Ethernet entity has a unique MAC address. In each frame, the destination (unicast, broadcast or multicast) and source addresses are inserted. Frames are broadcasted on the physical layer. Entities get the frames there are interested in by a filtering process.

The Ethernet native medium access method is CSMA/CD (Carrier Sense Multiple Access / Collision Detection): the collision resolution mechanism is non deterministic and leads to unbounded transmission delays.

Full Duplex Switched Ethernet is a way to bypass the medium access strategy of Ethernet: each station is directly connected to an Ethernet switch with a full duplex link. Then, the medium is always free. Consequently guaranteed performances are strongly connected to policies of the switch. Many literature has been devoted to the subject (see for instance [30] concerning service disciplines in packet-switching networks). In this paper, we consider a very basic switch with a First-In First-Out policy on each output port.

2.3. Heterogeneous CAN / Switched Ethernet architectures

Our goal is to interconnect several CAN busses with a full duplex switched Ethernet network, in order to bypass CAN limitations while keeping the widely existing CAN technology. Such an architecture is depicted in figure 3. It includes four CAN busses and an Ethernet switch. There is a bridge station between each CAN bus and the switch. The switch has four receive ports and four queued transmit ports. When a frame arrives at the switch, the control logic determines the transmit port and tries to transmit the frame immediately. If the port is busy because another frame is already being sent, the frame is stored in the first-in first-out transmit port queue. The memory to store pending frames is obtained from a shared memory pool. If no more memory is available, the received frame is dropped.

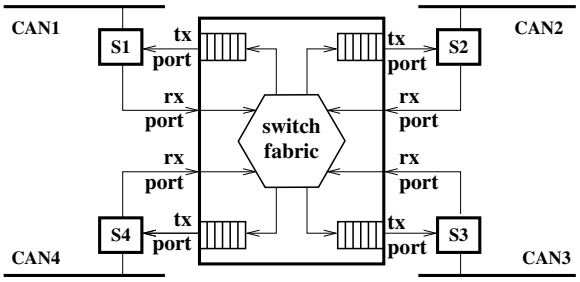


Figure 3. Network architecture

More generally, the architecture includes N_c CAN busses and the switch has N_c receive ports and N_c queued transmit ports. Network architectures with more than one switch are not considered in this paper.

3. Application traffic over the network

All the traffic is transmitted on CAN data busses or/and Switched Ethernet network. First, we detail the main characteristics of each kind of traffic. Second, we summarize interesting bridging strategies for CAN traffic that has to be transmitted over Switched Ethernet. Then, we describe more precisely a strategy that is intended to allow an end-to-end time-triggered behavior. Finally, we summarize the purpose of the evaluation of the network architecture with the given application traffic.

3.1. Kinds of traffic

The traffic on the whole network can be divided in three kinds:

local CAN traffic: all the frames of this traffic are produced by a station on a CAN data bus s and consumed by stations all on the same CAN data bus s; consequently, those frames don't have to be transmitted on switched Ethernet,

global CAN traffic: all the frames of this traffic are produced by a station on a CAN data bus s (their home bus) and consumed by stations among which at least one is on a CAN data bus d with $d \neq s$ (all those d busses are called the distant busses of the global CAN frame); consequently, those frames have to be transmitted on switched Ethernet,

non-CAN Ethernet traffic: all the frames of this traffic are produced by a station on the switched Ethernet network and consumed by stations all on the switched Ethernet network; consequently, those frames don't have to be transmitted on any CAN data bus.

CAN traffic (local and global) is composed of messages. Each message M_i consists in the periodic production of a frame with a given DLC. Message M_i period is denoted P_i. Each frame of M_i has a relative deadline equal to the period P_i. We impose that all periods are harmonic in order to simplify the TTCAN implementation of the application. We don't consider aperiodic nor sporadic CAN traffic. A global message is not transferred on a CAN bus which is neither its home bus nor one of its distant busses.

Non-CAN Ethernet traffic is composed of a set of flows. Each flow is a sequence of frames with a fixed length and an exponential inter-arrival law.

Concerning the scheduling of frames on CAN data busses, we consider the two following solutions:

CAN native MAC is considered and identifiers are allocated to CAN messages following a rate monotonic policy (messages with the smallest period get the higher priority [22]),

TTCAN is considered with D_{uc} (the duration of the basic cycle) equal to half the smallest period P_i among CAN messages and the duration of the matrix cycle equal to the biggest period P_i among CAN messages (this implies that each CAN message is scheduled at most every two basic cycle). As an example, if there are three CAN messages M_1, M_2 and M_3 with periods 2, 4 and 4 ms, we have $D_{uc} = 1\ ms$ and a matrix cycle of $4\ ms$.

3.2. Bridging strategies for global CAN traffic

As global CAN traffic has to be transmitted on the switched Ethernet network, it is necessary to define a bridging strategy between CAN and switched Ethernet. As explained in [26], the very different CAN and Ethernet characteristics make an encapsulating policy the best choice. The encapsulation consists in putting the Identifier, DLC and Data fields of CAN frames in the Data field of the Ethernet frame (the other fields of CAN frames can be easily reconstructed). This means that a CAN frame occupies at most 10 bytes of the Data field of an Ethernet frame. The following strategies will be considered:

the one for one strategy: it is the most straightforward strategy, since each CAN frame is put in a separate Ethernet frame and transmitted as soon as possible,

the n for one strategy: it consists in frame bunching with exactly n CAN frames in an Ethernet frame, implying that each global CAN frame has to wait until there is n pending CAN frames in the bridge station,

the timed n for one strategy: it consists in frame bunching with a bounding of the delay a global CAN frame has to wait before being transmitted over Ethernet,

the time-triggered strategy: it consists in applying a TTCAN like strategy over the whole network (CAN and Ethernet) for global CAN frames; more details will be given later on.

The three first strategies have been compared by simulation in [27], considering CAN native MAC. The results show that the **timed n for one** strategy gives the best ratio of CAN frames meeting their deadlines, whatever non-CAN Ethernet load is. Concerning the **n for one** strategy, greater values of n are better when non-CAN Ethernet load increases, while the percentage of CAN frames missing their deadline increases with n for low non-CAN Ethernet loads (see [27] for details).

Those three strategies can be applied using TTCAN for the scheduling of frames on CAN data busses. However, they won't keep the time-triggered behavior of TTCAN. Consequently, they won't be studied.

The time-triggered strategy is proposed in the context of TTCAN for the scheduling of frames on CAN data busses. It has been presented in [25]. It is described more precisely in the next paragraph.

3.3. The time-triggered strategy

It is based on the following principle. Each bridge station is the TTCAN master of its CAN bus. In order to obtain a time-triggered behavior on the whole network, all global frames ready on a bridge are encapsulated in a single Ethernet frame and broadcasted via the switch at predefined instants. The following problems have to be solved:

- the choice of transmission instants,
- the synchronization between the different CAN busses, when initializing the system and to compensate clock drifts,
- the coordination of the different TTCAN tables.

A possible solution to those problems is depicted in figure 4 with the four CAN data busses architecture of figure 3.

On this example, Global messages imply three CAN busses (all the messages of CAN bus 4 are local and bus 4 will be ignored) : X is produced on CAN bus 2 and consumed on CAN bus 1, Y is produced on CAN bus 3 and consumed on CAN busses 1 and 2, Z is produced on CAN bus 1 and consumed on CAN bus 2, T is produced on CAN bus 2 and consumed on CAN bus 3. The period of messages X, Y, Z and T is 20 ms. TTCAN tables are given on figure 5. Windows where messages are produced

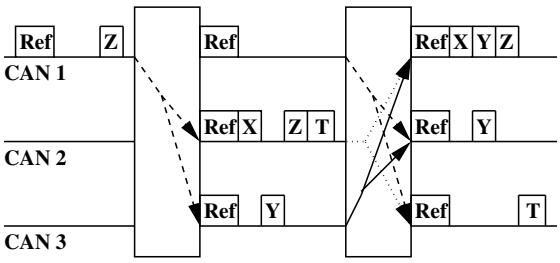


Figure 4. Time-triggered over switched Ethernet

are underlined. The duration of the micro-cycle D_{uc} is 10 ms for the three tables (half the smallest period). Local messages are not considered in this example.

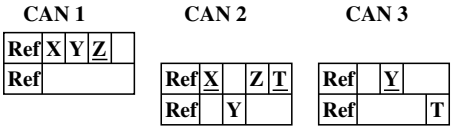


Figure 5. TTCAN tables

A master is defined among bridges stations - here, the bridge of CAN bus 1 is chosen. Its role is to synchronize the CAN busses at initialization time. Following synchronizations do not require a master. At the beginning of the application, the first TTCAN micro-cycle executes on CAN bus 1. At the end of this cycle, the bridge of CAN bus 1 broadcasts to bridges of CAN busses 2 and 3 an Ethernet frame encapsulating global CAN frame Z. This Ethernet frame synchronizes the three CAN busses. Then, the second TTCAN micro-cycle executes on CAN bus 1, while the first one executes on CAN busses 2 and 3. At the end of the TTCAN micro-cycle, each bridge broadcasts an Ethernet frame encapsulating ready global CAN frames (X and T for CAN bus 2, Y for CAN bus 3). Those Ethernet frames synchronize the three CAN busses, which then execute the next TTCAN micro-cycle.

The main characteristics of this solution are:

1. the transmission instants correspond to the ends of micro-cycles,
2. the synchronization of different CAN busses is done at the end of each micro-cycle, via Ethernet frames exchanges,
3. the TTCAN tables are built so that, when a global frame is sent on its source bus during a micro-cycle, it is sent on its destination bus during the next micro-cycle (see for example frame Y in figure 4),
4. a global CAN frame stands in the same column in all TTCAN tables (e.g., frame Y is always in column 2 on figure 4).

The time triggered behavior is obtained on each individual CAN bus by the construction of the TTCAN table.

	Native CAN	TTCAN
One for one strategy	*	
n for one strategy	*	
Timed n for one strategy	*	
Time-triggered strategy		*

Table 1. Allowed configurations

As mentioned earlier, a global CAN frame has consumers in different CAN busses including the one where it is produced. Characteristics 3 and 4 above imply there is a delay of one micro-cycle between the transmission of the message on its home bus and the transmission on its distant busses. This delay has to be taken into account when designing the application. It includes the synchronization delay which depends on the transmission delays on Ethernet links, the delay induced by the switch and the delays induced by the drift between bridges clocks. Those delays have to be bounded precisely.

3.4. Purpose of the evaluation

As stated earlier, two important communication requirements of real time applications are bounded end-to-end transmission delays and bounded and small jitter for periodic traffic. As a consequence, the evaluation of the different configurations of our network architecture especially addresses the two following points:

- calculate a bound on the end-to-end delay of each CAN frame, so that we can guaranty that no missed deadline arise on CAN traffic,
- bound the jitter for each periodic CAN message.

The allowed configurations are summarized in table 1. In the next section, we will propose both a modelling of the **one for one** strategy that can be easily extended to the two other event-triggered strategies and a modelling of the **time-triggered** strategy.

4. Modelling the network architecture with timed automata

Several approaches can be used to evaluate the behavior of a given application on a network architecture. In [26, 25], we developed a simulation model to compare different CAN / Ethernet bridging strategies. Such a model is inefficient to determine a worst-case end-to-end delay on CAN messages. The Network Calculus ([9, 10]) has been applied in the AFDX network system for Airbus embedded networks [4, 15]. However, it is often difficult to evaluate the quality of the obtained worst-case end-to-end delay (is it possible to approach or reach this delay?). Here, we use timed automata. We have already applied them in the contexts of a production cell [21] and avionics systems [6].

In this section, we first give a short overview of timed automata. Then, we successively present the modelling of the CAN/switched Ethernet and TTCAN/switched Ethernet architecture with timed automata. In these modellings, no non-CAN Ethernet traffic will be considered.

4.1 Modelling with timed automata

Timed automata have been first proposed by Alur and Dill [3] to describe systems behavior with time. A timed automaton is a finite automaton with a set of clocks, *i.e.* real and positive variables increasing uniformly with time. Transitions labels are:

- a guard, *i.e.* a condition on clock values,
- actions,
- updates, which assign new value to clocks.

Composition of timed automata is obtained by synchronous product. Each action *a* executed by a first timed automaton corresponds to an action with the same name *a* executed in parallel by a second timed automaton. In other words, a transition which executes action *a* can only be done if another transition labeled *a* is possible. The two transitions are performed simultaneously. So communication use rendez-vous mechanism.

Performing transitions requires no times. Conversely, time can run in nodes. Each node is labeled by an invariant, that is a boolean condition on clocks. Node occupation is dependent of the invariant. The node is occupied if the invariant is true.

Timed automata have been extended. One extension is committed nodes. The goal of these nodes is to ensure atomicity between consecutive execution of discrete actions [20]. As an example, consider the three automata of the figure 6.

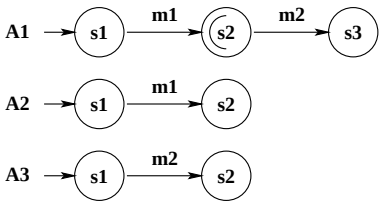


Figure 6. Example of committed nodes

A1 performs m1 and simultaneously A2 performs m1. Then A1 performs m2 and simultaneously A3 performs m2. As s2 of A1 is committed, the two transitions m1 and m2 are performed simultaneously without time evolution. So, this extension allows to model broadcast communication mechanism through timed automata.

Another extension is timed automata with shared integer variables. In timed automata with shared integer variables, a set of variables is shared by timed automata. In such a way, these values can be consulted and updated by any timed automata [20, 7].

A system modelled with timed automata can be verified using model-checking. The reachability analysis is performed by model-checking. It consists in encoding the

property in terms of reachability of a given node of one of the automata. So, the property is verified by the reachability of node if and only if the node is reachable from an initial configuration. Reachability is decidable and algorithms exist [20]. Unfortunately, reachability analysis is undecidable on timed automata with shared integer variables, but some semi-algorithms exist.

In the following subsections, we model the CAN / Switched Ethernet architecture and the TTCAN / Switched Ethernet architecture presented in section 3 using timed automata with shared integer variables. Properties will be verified using UPPAAL model-checker [1].

4.2 Modelling the CAN / Switched Ethernet architecture

In this subsection, we model the CAN / Switched Ethernet architecture of section 3. The structure of the model is depicted in figure 7. It is composed of four kinds of timed automata:

- a function automaton models periodic real-time functions,
- a transceiver automaton represents a part of the medium access layer,
- an arbiter automaton implements the CAN arbiter,
- a switch automaton models an output port of the Ethernet switch.

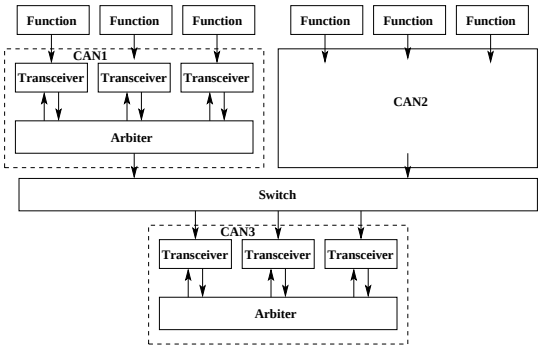


Figure 7. The CAN / Switch Ethernet model

The presented model only considers the **one for one** encapsulation strategy. It can be easily extended to the **n for one** and **timed n for one** strategies by the adding of bridge automata between CAN and the switch. Such automata will be described in the TTCAN context.

4.2.1 The function automaton

As all CAN messages are periodic in our context, each function of the system sends a frame periodically. The automaton is depicted in Figure 8.

$msgId$ corresponds to the identifier field in CAN frame. The function automaton waits the duration of $period[msgId]$. It leaves the node when the clock

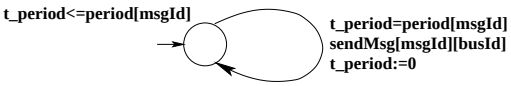


Figure 8. Function automaton

t_period is equal to $period[msgId]$. The message is then sent to the corresponding CAN bus *via* the action $sendMsg[msgId][busId]$.

4.2.2 The transceiver automaton

The transceiver automaton is the first part of the medium access layer. Figure 9 depicts its behavior.

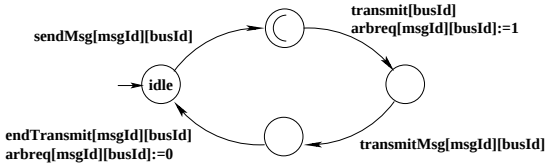


Figure 9. Transceiver automaton

This automaton is first idle, waiting for the signal $sendMsg[msgId][busId]$, which is the sending request from an upper layer. When it receives this signal, it immediately send the signal $transmit[busId]$. This signal requests access to the bus identified by $busId$. The shared integer variable $arbreq[msgId][busId]$ allows to identify the message which requests the bus. When the transmitter wins the bus, it receives the signal $msgTransmit[msgId][busId]$. When the message is completely transmitted, the signal $endTransmit[msgId][busId]$ is received and the variable $arbreq[msgId][busId]$ is reset.

4.2.3 The arbiter automaton

As explained in section 2.1, the bitwise arbiter of a CAN bus consists in choosing the lowest identifier, which corresponds to the highest priority, of the set of pending frames. Modelling of such an arbiter is proposed in [18]. It implements the following loop:

```
for i in 0 to max_identifier_value do
  if arbreq[i]=1 then
    begin
     transmit message i;
     wait end of transmission;
    end
end for;
```

We have adapted this solution to our context. Figure 10 shows the resulting timed automaton.

The node $h \leq 0$ models the loop. When the identifier is selected, the automaton simulates the transmission by waiting for a delay $transmission\_delay$. Then it sends a signal $endTransmit[msgId][busId]$ to the transmitter and simultaneously a signal $forwardTransmit[msgId]$ to the switch.

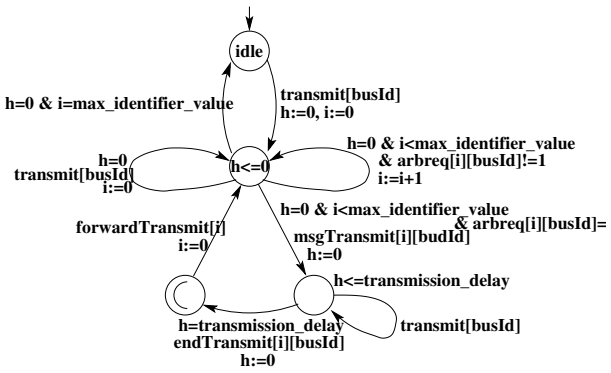


Figure 10. Arbiter automaton

4.2.4 The switch automaton

As explained in section 2.2, the switch uses FIFO policy on each queued transmit port. Each node of the automaton models a location in the queue. Consequently, the number of nodes of the automaton equals the size of the queue. Figure 11 shows a switch automaton for a transmit port with queue size n. Each transition from a node P_i to a node P_{i+1} of the automaton models the arrival of one frame at the transmit port. The identifier of the frame is the parameter of the $forwardTransmit$ event. $delay$ is the transmission time of the frame. In a first approach, we consider it is the same for every frame, meaning that frame length is constant.

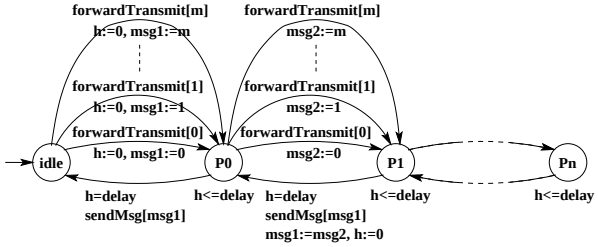


Figure 11. Switch automaton

4.2.5 The global system

First, we model the transmission on one bus i by composing functions, transceivers and arbiter:

$$Bus_i = ((Function_{i,0}||Transceiver_{i,0})||\ldots||(Function_{i,n}||Transceiver_{i,n}))||Arbiter_i$$

where $Function_{i,j}$ is a function sending a message with an identifier of value j, $Transceiver_{i,j}$ is the corresponding transceiver.

Finally, the global system is the composition of n bus and the switch:

$$System = (Bus_0||Bus_1||\ldots||Bus_n)||Switch$$

4.3 Modelling the TTCAN / Switched Ethernet architecture

We consider the model depicted Figure 12 which consists in the following components:

- a function automaton transmits a time-triggered message,
- a bridge TTCAN/Ethernet automaton encapsulates CAN frames in Ethernet frames,
- a bridge Ethernet/TTCAN automaton transmits CAN frames encapsulated in Ethernet frames to the destination CAN bus following the time-triggered schema,
- a switch automaton,
- and two Ethernet links.

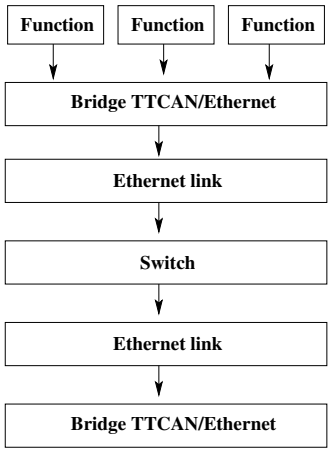


Figure 12. The TTCAN/Switch model

4.3.1 The function automaton

This function sends a signal $sendMsg[msgId][idCan]$ corresponding to the message $msgId$ at the instant $sendTime[msgId]$ (in order to respect the time-triggered schema) and then waits the beginning of the sending cycle (Figure 13).

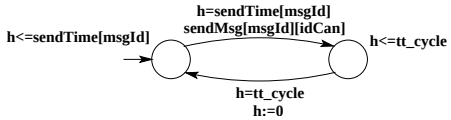


Figure 13. The function in time-triggered strategy

4.3.2 The bridge TTCAN/Ethernet automaton

The bridge TTCAN/Ethernet automaton constructs the Ethernet frame by encapsulating CAN frames. The model is depicted in Figure 14.

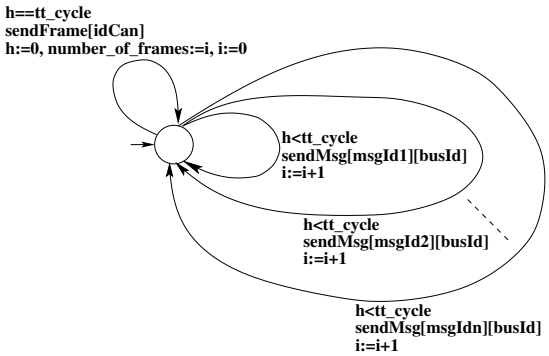


Figure 14. Model of the bridge CAN/Ethernet

During the cycle duration, each time the bridge receives the signal $sendMsg[msgId_n][idCan]$ which corresponds to a CAN frame, it encapsulates the frame into the pending Ethernet frame. When the cycle duration elapses, the bridge sends this Ethernet frame to the switch using the signal $sendFrame[idCan]$. The shared integer variable $number\_of\_frames[idCan]$ indicates how many frames have been encapsulated into the Ethernet frame.

4.3.3 The bridge Ethernet/TTCAN automaton

The bridge Ethernet/TTCAN automaton transmits CAN frames of the received Ethernet frame to the destination CAN respecting the time-triggered schema. Figure 15 represents its behaviour.

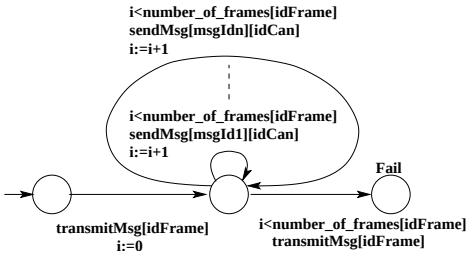


Figure 15. Model of the bridge Ethernet/CAN

The automaton receives the $transmitMsg[idFrame]$ from the switch automaton. $idFrame$ identifies the input CAN. The transmission of each frame is modelled by the reception of $number\_of\_frames[idFrame]$ signals named $sendMsg[idMes_n][idCan]$ from functions of same type as previously described. An error can be detected, which leads to the $Fail$ node in the Figure, when $transmitMsg[idFrame]$ is received before all the CAN frames have been transmitted, i.e. when the number of transmitted frames is lower than the number of CAN frames in the Ethernet frame.

station	message	bus	priority	period (ms)	trans. time (ms)
s1	m1	bus1	0	4	0.135
s2	m2	bus2	1	4	0.115
s3	m3	bus3	2	10	0.095
s4	m4	bus4	3	10	0.075
s5	m5	bus5	4	12	0.135
s6	m6	bus5	5	12	0.115

Table 2. Configuration of CAN / Switched Ethernet case study

4.3.4 The Ethernet links automata

The Ethernet link between the TTCAN/Ethernet bridge and the switch is modelled in Figure 16. It consists in adding a delay between the signal $sendFrame[idCan]$ sent by the bridge and $forwardTransmit[idCan]$ received by the switch.

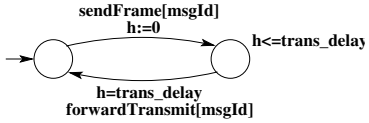


Figure 16. Model of an Ethernet link

The automaton modelling the Ethernet link between the switch and the Ethernet/TTCAN bridge is similar to the switch automaton depicted in Figure 11.

5. Worst-case delay calculation

In this section, we show how worst-case end-to-end delay can be obtained from the models of section 4 Consider the case study depicted in Figure 17 composed of eleven stations communicating on five CAN buses interconnected by an Ethernet switch. Six stations sends a message periodically. The configuration is given in table 2. The switch imply a transmission delay of 0.080ms. We won't consider any delay within CAN/Ethernet bridges. This case study is modelled as described in section 4.

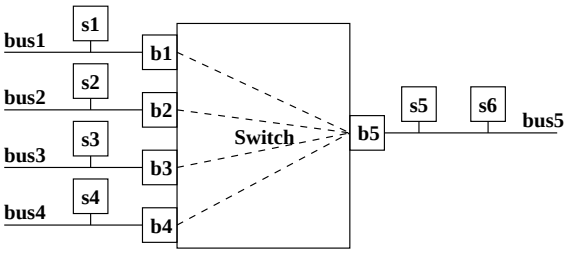


Figure 17. The CAN/Switch model

Model-checking is used to determine the global transmission delay of each message in the system. The method consists in verifying that a message is received before a global transmission delay. In other words, the property to

Message	Global trans. delay (ms)
m1	0.595
m2	0.460
m3	0.345
m4	0.670

Table 3. Worst-case delays

verify is "given a message m_i, the global transmission delay of the message m_i, noted $d(m_i)$ must be lower than a bounded delay d_i : $d(m_i) \leq d_i$". The test automata method can be used to help the verification process. This method is described in [7, 5] and consists in constructing a test automaton which encodes the considered property. Then, the model-checking consists in calculating if a reject node is reachable or not. The test automaton of our property is depicted in Figure 18.

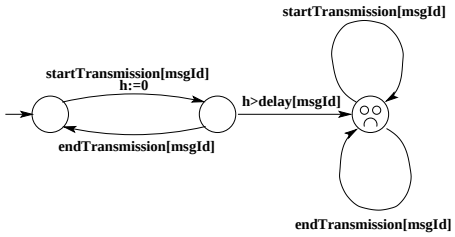


Figure 18. The test automaton

First, it waits for $startTransmission[msgId]$ signal which is immediately transmitted (using a committed node) after $sendMsg[msgId]$ in each automaton function. If $h > delay[msgId]$, i.e. no signal $endTransmission[msgId]$ is detected before $delay[msgId]$, the reject node, represented by the node with an unhappy face, is reached and the property is false. So, we compute, for each message, the lower value of the global transmission delay using the model-checker UPPAAL. Results are given in table 3 and show that the worst-case occurs when all the messages are sent at the same time. Due to transmission delays, messages positions in the switch are: 1: $m4$, 2: $m3$, 3: $m2$, 4: $m1$. $m4$ takes 0.075+0.080=0.155ms to access to bus5. During this time, $m5$ is completely transmitted on bus5 and $m6$ is in transmission since 0.020ms. $m4$ is delayed. At 0.235ms, $m3$ try to access to bus5. When transmission of $m6$ is finished at 0.250ms, the priority of $m3$ is higher than the one of $m4$ and then is transmitted to bus5. In accordance with messages priorities, transmission of $m2$ can starts at 0.345ms and transmission of $m1$ at 0.460ms. $m4$ is then sent at 0.595ms.

Consider now the system of Figure 19 composed of three CAN busses. It illustrates the worst-case delay calculation on the TTCAN/Switched Ethernet model. It includes two function on each input bus and four functions on the output bus, two CAN / Ethernet bridges and two Ethernet / CAN bridges. Configuration is given in table 4. The micro-cycle is 1ms.

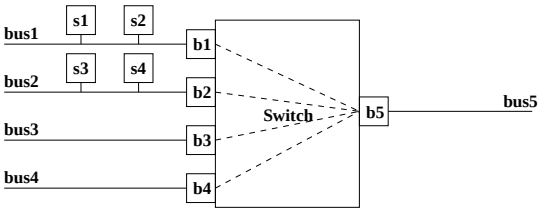


Figure 19. The TTCAN/Switch model

Message	Bus	Trans. Instant (ms)
m1	bus1	0.200
m2	bus1	0.600
m3	bus2	0.500
m4	bus2	0.700

Table 4. TTCAN Configuration

To bound the jitter, the Ethernet link transmission delay is increased until the Fail node is reached in the bridge Ethernet / CAN. The property is then "fail node should not be reached". Given a switching delay of 0.020ms, the maximum valid value for the Ethernet link transmission delay is 0.060ms.

Computing the worst case delay on the two case studies using a Pentium IV with 2Go of memory takes less than 5s on UPPAAL 3.4.11.

6. Conclusion and future works

In this paper, we mainly focused on two types of communication technologies:

- the Controller Area Network (CAN), with both the native CAN MAC and the time-triggered version (TTCAN), which is a good example of deterministic real-time communication system,
- Switched Ethernet, which is a popular non real-time communication system.

The aim of the paper was to study the use of switched Ethernet in conjunction with CAN for communications in a real-time system. More precisely, the challenge was to define and evaluate event-triggered and time-triggered mechanisms on a mixed CAN / switched Ethernet architecture.

The event-triggered behaviour is obtained by the native CAN MAC and an event-triggered encapsulation strategy. The time-triggered strategy extends TTCAN over the whole network (exchanges between CAN data busses take place at the end of each TTCAN micro-cycles).

We have proposed an evaluation method for the different proposed solutions using timed automata modelling and UPPAAL. With our models, we are able to determine worst-case end-to-end delays for CAN frames. We still have to validate this calculation on more significant case

studies. In this context, it will probably be necessary to simplify CAN modelling in order to overcome combinatorial explosion.

Moreover, our models have to be expanded, especially in the following ways:

- the introduction of jitters between the different CAN busses,
- the introduction of non-CAN Ethernet traffic,

This will probably imply the use of probabilistic timed automata [28, 19]. Moreover, the introduction of non-CAN traffic will imply the use of a more sophisticated Ethernet switch, in order to be able to differentiate traffics.

Among other points that should be studied, there is the use of other time-triggered strategies on CAN, such as FTTCAN [2] and an architecture with a more global Ethernet including several switches.

Finally, it would be valuable to compare the approach proposed in this paper, based on timed automata, with other approaches such as the one based on network calculus. A preliminary similar comparison has been conducted in the AFDX context [8].

References

[1] http://www.uppaal.com.

[2] L. Almeida, P. Pedreiras, and J. A. G. Fonseca. The ftt-can protocol : why and how. *IEEE transactions on industrial electronics*, 49(6), dec 2002.

[3] R. Alur and D. L. Dill. Theory of Timed Automata. *Theoritical Computer Science*, 126(2):183–235, 1994.

[4] ARINC Specification 664: Aircraft Data Network, Parts 1,2,3,4,5,8. Technical report, Aeronotical Radio Inc., 2002-2005.

[5] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[6] F. Boniol, G. Bel, and J. Ermont. Trois approches pour la modelisation et la verification des systemes embarques. *Technique et Science Informatique*, 22(5):539–569, 2003.

[7] A. Burgueño Arjona. *Vérification et synthèse de systèmes temporisés par des méthodes d'obervation et d'analyse paramétrique (in english)*. PhD thesis, Ecole Nationale Supérieur de l'Aéronautique et de l'Espace, Toulouse, France, 1998.

[8] H. Charara, J.-L. Scharbarg, J. Ermont, and C. Fraboul. Methods for bounding end-to-end delays on an AFDX network. In *ECRTS'06*, Dresden, July 2006.

[9] R. Cruz. A calculus for network delay, part I. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.

[10] R. Cruz. A calculus for network delay, part II. *IEEE Transactions on Information Theory*, 37(1):132–141, January 1991.

[11] M. Di Natale. Scheduling the can bus with earliest deadline techniques. In *Proceedings of the IEEE Real-Time Systems Symposium*, 2000.

[12] D. Dietrich and T. Sauter. Evolution potentials for fieldbus systems. In *IEEE Workshop on Factory Communication Systems*, Porto, September 2000.

[13] CSMA/CD access method. IEEE Standard 802.3, IEEE, 2002.

[14] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther. Time triggered communication on can. In *International CAN Conference*, 2000.

[15] J. Grieu, F. Frances, and C. Fraboul. Preuve de déterminisme d'un réseau embarqué avionique. In *Actes du 10ème Colloque Francophone sur l'Ingenierie des Protocoles*, Paris, 7-10 Octobre 2003.

[16] ISO. *ISO International Standard 11898 - Road vehicles - Interchange of digital information - Controller Area Network for high-speed communication*, nov 1993.

[17] ISO. *ISO International Standard 11898-4 - Road vehicles - Controller Area Network - Part 4 : Time-Triggered Communication*, dec 2000.

[18] J. Krákora, L. Wasznioski, P. Píša, and Z. Hanzálek. Timed Automata Approach to Real Time Distributed System Verification. In *5th IEEE International Workshop on Factory Communication Systems*, Vienna, September 2004.

[19] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying Quantitative Properties of Continuous Probabilistic Timed Automata. *Lecture Notes in Computer Science*, 1877:123–??, 2000.

[20] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.

[21] D. L'Her, J.-L. Scharbarg, P. Le Parc, J. Vareille, and L. Marce. Specification and verification of the korso production cell. In *INCOM*, Nancy, june 1998.

[22] C. Liu and L. J.W. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of ACM*, 20(1):46–61, 1973.

[23] T. Nolte, H. Hansson, C. Norstrom, and S. Punnekat. Using bit-stuffing distributions in CAN analysis. In *IEEE/IEE Real-Time Embedded Systems Workshop (RTES)*, 2001.

[24] P. Pedreiras and L. Almeida. Edf message scheduling on controller area network. *Computing and control engineering journal*, pages 163–170, aug 2002.

[25] J.-L. Scharbarg, M. Boyer, J. Ermont, and C. Fraboul. Ttcan over mixed can/switched ethernet architecture. In *Proc. of the 10th International Conference on Emerging Technologies and Factory Automation*, Catania, Italy, September 2005. IEEE.

[26] J.-L. Scharbarg, M. Boyer, and C. Fraboul. Can-ethernet architectures for real-time applications. In *Proc. of the 10th International Conference on Emerging Technologies and Factory Automation*, Catania, Italy, September 2005. IEEE.

[27] J.-L. Scharbarg, M. Boyer, and C. Fraboul. Interconnecting can busses via an ethernet backbone. In *Proc. of the 6th International Conference on Fieldbus Systems and their Applications*, Puebla, Mexico, November 2005. IFAC.

[28] J. J. Sproston. *Model Checking Probabilistic Timed and Hybrid Systems*. PhD thesis, University of Birmingham, Birmingham, UK, 2001.

[29] J.-P. Thomesse. Fieldbusses and interoperability. *Control Engineering Practice*, 7:81–94, 1999.

[30] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–1396, October 1995.

R-(m,k)-firm : A novel QoS Scheme for Real-time Flow Guarantee in Networks

Jian LI

INPL - LORIA
Campus Scientifique - BP 239
54506 Vandoeuvre-lès-Nancy Cedex, France
e-mail: Jian.Li@loria.fr;

YeQiong SONG

INPL - LORIA
Campus Scientifique - BP 239
54506 Vandoeuvre-lès-Nancy Cedex, France
e-mail: Song@loria.fr

Abstract

This paper presents a new real-time Quality of Service (QoS) guarantee scheme called Relaxed (m,k)-firm denoted by R-(m,k)-firm. The problem we deal with is that for a set of streams (e.g. sources which generate packets) sharing a common resource, deterministic (m,k)-firm guarantee of each stream can lead to an arbitrarily low resource utilization, making (m,k)-firm useless for QoS guarantee in a network. The goal of the proposed constraint relaxation is to achieve a higher resource utilization while still deterministically guarantee the (m,k)-firm constraint. As with (m,k)-firm guarantee, R-(m,k)-firm provides the guarantee on the ***transmission delay*** *of at least m out of any k consecutive packets (m≤k). Instead of imposing a transmission* ***delay constraint per packet*** *(i.e. deadline), R-(m,k)-firm only considers a global transmission* ***delay constraint on a group of any k consecutive packets****. This constraint relaxation may be acceptable for a large class of soft real-time applications such as multimedia flow transmissions in the networks for which occasional packet drops can be tolerated.*

One of the possible implementations of the R-(m,k)-firm scheme is also provided with the development of a new traffic control mechanism, called Double-Leaks Bucket (DLB). DLB selectively drops a proportion of packets of a flow or an aggregated-flows in case of the network congestion while still guaranteeing the R-(m,k)-firm constraint. The sufficient condition for this guarantee is given for configuring the DLB parameters.

Finally a comprehensive discussion on the existing constraint relaxation strategies is developed showing the generality of the R-(m,k)-firm scheme.

1. Introduction and Motivations

1.1 Why using (m,k)-firm for QoS control

Internet nowadays supports more and more real-time and business-critical applications. However, for providing them with transmission delay guarantee, neither Intserv nor Diffserv consists in an efficient solution [1]. In fact, for providing a bounded transmission delay to a flow (Intserv) or a class of flows (Diffserv), as a flow often generates bursty traffic (case of most VBR applications and aggregated Diffserv class of flows), a bandwidth reservation policy according to the peak rate of the flow is an over-provisioning one. This leads to low resource utilization in average case. Another problem we should deal with in providing real-time QoS is the network congestion because of the router overload. This can occur when a path includes one or several routers which do not support Diffserv. As it may lead to packet drop, although occasional packet drops could be tolerated, long consecutive packet drops must be avoided since it can drastically decease the QoS for applications such as audio/video diffusion. Unfortunately, existing queue management schemes such as TD (Tail-Drop) do not address the consecutive packet drop problem. RED (Random Early Detection) [2] has been proposed to deal with the problem with random dropping. However, it does not give any guarantee on non-consecutive packet drops.

The key idea we exploit here is to take the advantage of the "natural" packet loss tolerance of a large class of real-time applications to reduce the sufficient bandwidth reservation. In fact, if we consider applications such as video-on-demand, IP telephony, Internet radio, etc., many of them can tolerate packet losses to some extent. Let us take the example of the packetized voice transmission of the IP telephony service. Instead to guarantee the reliable transmission of all the packets with a large delay, it is preferable to drop a voice packet if it cannot be transmitted in time (typically 400ms for IP telephony). The (m,k)-firm model introduced by Hamdaoui and Ramanathan [3] can be used to specify such kind of tolerance by introducing the graceful QoS degradation between satisfying all packets' deadlines (i.e. (k,k)-firm guarantee) and (m,k)-firm guarantee. QoS management according to the (m,k)-firm model has several advantages: (1) during network congestion, packets are dropped according to the (m,k) model rather than non-deterministically as the case of TD and RED. Thus undesirable long consecutive packet drops can be avoided; (2) the fact to aim at only (m,k)-firm guarantee instead of (k,k)-firm may require less resource reservation since the average workload is reduced by a factor of m/k for providing the minimum QoS level. This last point is interesting when congestion occurs or when the peak rate reservation (i.e., over-provisioning) cannot be provided at admission control.

1.2 What are the problems when using (m,k)-firm model for QoS control

At first glance, the (m,k)-firm model [3] seems to be an interesting one for resolving both of the problems of the resource over-provisioning and long consecutive packet drops. The (m,k)-firm model says that the deadlines of at least *m* out of any consecutive *k* packets must be met. Moreover packets whose deadline cannot be met are not transmitted (i.e. dropped), this is why we use the term "firm real-time" instead of "soft real-time". Notice that the term "any consecutive *k* packets" implies a sliding window guarantee for a flow. Some work exists in applying (m,k)-firm to QoS management. In [4], it proposed to use (m,k)-firm model instead of RED for congestion control and experimentally showed its interest. However nothing is provided concerning the transmission delay guarantee. [5] proposed an integration of the existing (m,k)-firm scheduling algorithms into the Diffserv architecture for providing average performance improvement.

Intuitively, reserving resources according to (m,k) requirement rather than (k,k) should reduce the necessary resource reservation. This is true for example for the case when flows are served by a WFQ server [6] or when only statistic (m,k)-firm guarantee is required. Unfortunately, it has been proven that in general, for achieving **deterministic** (m,k)-firm guarantee, one has to reserve resources according to (k,k)-firm since the worst case must be considered [7, 8, 9]. Moreover, the problem of non pre-emptive scheduling of *N* (m_i,k_i)-firm constrained flows ($i = 1, \ldots, N$) on a single resource (processor or network link) has been proved NP-hard in strong sense, such that no optimal scheduling can be expected under such model. This problem seriously compromises the practical interest of using the (m,k)-firm model for network resource management.

Faced to this low utilization problem (due to NP-hard), three research directions are possible. The first one is to look for the sub-optimal scheduling using heuristic methods. This is generally not suitable for on-line QoS control because of the long computing time. The second one is to specialize the stream set. For instance, in [10], by specializing the stream model such that the packets of all the streams must have the same transmission time and the same period, the utilization factor is improved. However this so particular stream model cannot be directly applied to the multimedia transmission in which each stream could have its different packet length and period. The third way is to extend the (m,k)-firm one. In [11], the deadlines of packets are relaxed to reduce the resource requirement.

1.3 What is our proposal

In this paper we follow the third research direction and propose to modify the concept of (m,k)-firm. Instead of considering the traditional guarantee of the individual packet deadline, we define a global deadline for any group of *k* consecutive packets. Formally, in an interval [*s, t*], the source has sent *k* packets to the network, then the destination should be assured to receive at least *m* among them (delivered *in order or not*) before time $t+\Delta$, where Δ is the maximum tolerable transmission delay caused by the network for any group of *k* consecutive packets.

With this novel definition, it is obvious that the QoS requirement is given from per flow or per aggregated-flow point of view instead of the per packet deadline (m,k)-firm constraint. So we call this Relaxed (m,k)-firm guarantee and shortened by R-(m,k)-firm.

1.4 DLB: one of possible implementations of R-(m,k)-firm scheme

For implementing R-(m,k)-firm scheme, we also designed a new mechanism, called Double Leaks Bucket (DLB) for dropping a proportion of packets of a flow or a class of flows in case of network congestion while still guaranteeing the R-(m,k)-firm constraint of (r, b)-bounded [12] flows. Where r stands for the average arrival rate while b the burst. In [15] it has been shown that (r, b)-bounded can include periodic and sporadic flows (with or without jitters).

1.5 How the R-(m,k)-firm scheme is positioned

The R-(m,k)-firm scheme can be considered as one of the real-time constraint relaxation strategies similar to the Pinwheel model [13], frame-based model [14] and Virtual window constrained model [11]. We will show that R-(m,k)-firm is a general model which can include the previous ones.

1.6 Organization of the paper

The rest of this paper is organized as following. Section 2 describes in more detail the R-(m,k)-firm scheme. The DLB mechanism is presented in section 3. In section 4 we discuss on the existing constraint relaxation strategies and show the generality of the R-(m,k)-firm scheme. Section 5 summarizes our contributions.

2. R-(m,k)-firm scheme

2.1 Motivation of R-(m,k)-firm

Let us consider a traditional periodic or sporadic task[1] set described as $\Gamma=(\tau_1, \ldots \tau_n)$. A task is described by $\tau_i = (c_i, p_i, d_i, m_i, k_i)$, where c_i stands for the execution time of an instance, p_i stands for the period or minimum inter-arrival time of instances, d_i is the relative deadline before which the instance must be completed, otherwise the instance will miss its deadline and in firm real-time, it is dropped directly without execution; and m_i, k_i describe that the deadline of at least m_i out of any consecutive k_i instances must be met.

The practical advantage of (m,k)-firm constraint is to increase at much as possible the utilization factor of a task set which is given in terms of $\sum_{i=1}^{n} \frac{m_i}{k_i} \frac{c_i}{p_i}$. Obviously, this utilization cannot be higher than 100%, such that the **gain** of a system is to improve the utilisation factor to 100% at most.

[1] Here we use the term task for keeping close to the real-time scheduling terminology. However, it should understand that a task is a general term which can stand for a source generating packets in the context of networks

As mentioned in the previous section, until now, there is not a non pre-emptive scheduling system which can get an interesting gain for a general task set under (m,k)-firm constraint. So that we will propose a relaxed (m,k)-firm real-time constraint to resolve this low utilization problem.

2.2 Definition of R-(m,k)-firm QoS constraint

An R-(3,5)-firm constraint is shown in Fig. 1.

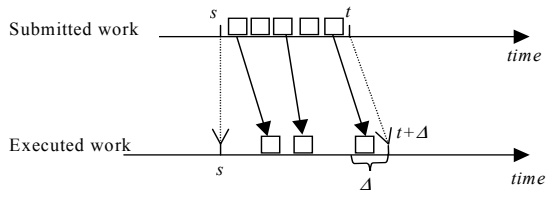


Fig. 1: R-(3,5)-firm constraint

Definition of R-(m,k)-firm constraint:
In any time interval [*s, t*] (with $t\text{-}s \geq l$), a task submits *k* units of workload to a server. The server should finish the execution of at least *m* among them (*in order or not)* before time $t+\Delta$, where Δ is the maximum tolerable delay caused by the server for the group of *k* units.

The following points help to understand this definition.

1) Any time interval means a sliding window, which can start from any time point, denoted by *s*.
2) A task τ_i can either be modelled by a periodic or sporadic source (c_i, p_i, m_i, k_i) or by a (r_i,b_i)-bounded source under (m_i,k_i)*-firm* constraint but with Δ as the deadline of any group of k consecutive instances starting at time *s*.
3) The constraint is given for each task: every task can require its own constraint without considering other tasks. The system should guarantee R-(m,k)-firm constraint individually for each task.
4) The real-time constraint includes two factors: (m,k) factor and delay factor.
5) (m,k) factor of constraint: at least *m* among *k* instances should be executed within the delay constraint Δ. In general case, (m,k) factor applies to the sliding window. However it can also applies to the no-overlapping fixed windows.
6) Delay factor Δ: this factor assures that (m,k) factor must be realised before a maximum delay after the end of the release of the k^{th} instance starting from time *s*. For the sliding window requirement, it requires that from no matter when one task generates k instances in time length not smaller than *l*, such that the system assures the execution of at least *m* instances before $l+\Delta$. No-overlapping window just requires that after the release of a task, at least *m* instances are executed among the first *k* instances, as well as the *m* among $(k+1)^{th}$ to $2k^{th}$ instances, and so on.

2.3 Application in network

After the comprehension of R-(m,k)-firm constraint, we give an example in networks, as shown in Fig. 2. The source has a virtual stack, and it sends the packets from its stack head through the network. The destination has also a corresponding stack, and adds the received packets to the stack tail. Supposing in the time interval [*s, t*], *k* packets have been sent to the destination, the QoS provided by the network must assure the destination to receive at least *m* packets among the *k* for adding into its stack before $t+\Delta$. There are *k-m* discardable packets in *k* consecutive ones.

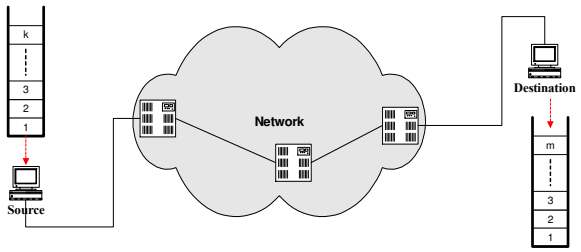


Fig. 2: Per Flow QoS

Actually, R-(m,k)-firm constraint replaces the conventional real-time constraint on the individual packet deadline.

2.4 Demonstration of R-(m,k)-firm advantages by Virtual scheduling pattern

Fig.3 shows the advantage of our R-(m,k)-firm constraint in contrast with conventional per packet deadline constraint. Each block stands for a packet.

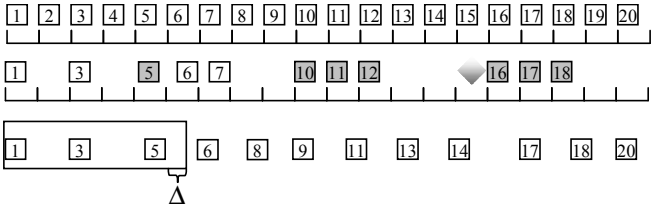


Fig. 3: Example of a virtual scheduling pattern under (3,5)-firm and R-(3,5)-firm constraint

The fist scenario (first line) shows a periodic packet stream that sends a packet at each beginning of period of P seconds. The second scenario (2nd line) shows a scheduling under (3,5)-firm constraint. In the second scenario, the server is obliged to serve all grey blocks. Otherwise, the system falls into failure state, such as the 15^{th} block. Once the system falls into failure state (i.e. the (m,k)-firm constraint is violated), the server is still obliged to serve the next 3 packets (16^{th}, 17^{th}, 18^{th}) to restore system. Note that this obliged service of packets, scheduled with other streams, will cause high resource requirement (or reduced utilization) forming the so-called interference point [7], [16]. The well distributed interference points could reduce

the resource requirement [16]. However, the optimal distribution of those interference points is an NP-hard problem in strong sense, making it impossible to always reduce the over-provisioning problem. The third scenario shows a sequence under R-(3,5)-firm constraint, whose window size is configured according to R-(3,5) constraint with 5P+Δ. Readers can verify by sliding the window or by positioning the no-overlapping windows, and will find there are always at least 3 packets in the window no matter which beginning of period it is slide to. Although there are a lot of deadline misses, the sequence can still be accepted by R-(3,5)-firm.

2.5 R-(m,k)-firm is flexible and adaptive

It is obvious that our R-(m,k)-firm pattern is more flexible than the (m,k)-firm one. Although there are some deadline misses, it can be acceptable for a real-time multimedia communication such as VoIP, VoD, as well as some networked control systems where over-sampled data are transmitted by a network.

In previous section, we only showed a simple example scenario under R-(3,5)-firm and (3,5)-firm constraints, but it is not to say that all transmissions could tolerate the loss-rate of 40%. After all, *m* and *k* of R-(m,k)-firm constraint can be configured as any natural number. The configurations should be done according to the specified communication requirement.

The R-(m,k)-firm scheme being specified, we propose in the next section a traffic control mechanism for deterministically guaranteeing R-(m,k)-firm constraint with high utilization factor. In fact satisfying R-(m,k)-firm constraint is not a trivial problem since the (m,k) factor and the delay factor are two antagonist factors for a given server. On the one hand, serving more instances (or packets) favours the (m,k) factor but leads to more delay. On the other hand, dropping more instances (or packets) may reduce the delay factor but risk to jeopardize the (m,k) factor.

3. DLB (Double-Leaks Bucket)

According to R-(m,k)-firm constraint, we develop one novel mechanism termed as DLB from the traditional leaky bucket [12]. DLB has two leaks named Serving Leak (SL) and Discarding Leak (DL) as depicted in Fig. 4.

3.1 Liquid model of DLB mechanism

Firstly, to simplify the problem, we start the analysis with a liquid model, whose workload is in terms of 'water' that can be split infinitesimally. The network should guarantee the 'water' that travels through SL, whilst the DL controlled by one switch gives the capacity to throw out the water from the bucket. With the service guarantee for SL, R-(m,k)-firm constraint could be satisfied. The water going through the DL is discarded, and can be treated with whatever method never jeopardizing the network QoS.

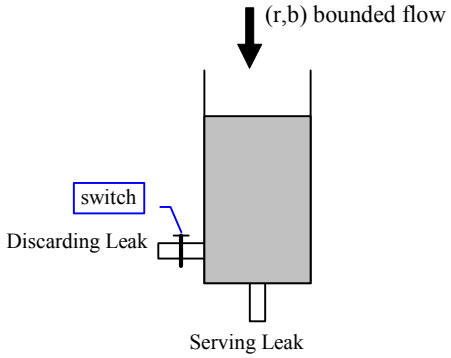


Fig. 4: Double-Leaks Bucket (Liquid Model)

Let C_1 and C_2 denote respectively the capacities of SL and DL. The control switch of DL works according to the quantity of the workload (represented by the height of the water in the bucket and denoted by q). This function is called as *double threshold control function* (DTC function), as shown in Fig. 5, where 1 stands for the opened state of the DL switch (or water gate) and 0 stands for the closed state.

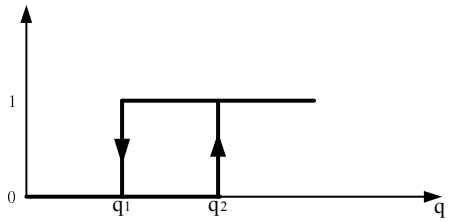


Fig. 5: Double Thresholds Control Function of the Switch

3.2 Service curve under (r,b)-bounded arrival stream.

We take the (r,b)-bounded arrival stream as a general source model. This is to say that the cumulative arrival curve is upper bounded by (*r,b*) [12], where *r* is the average arrival rate, and *b* is the burst. Denoting by $F_i(t)$ the function of the arrival curve, the workload arrived at any interval *[s, t]* is upper bounded by:

$$F_i(t) - F_i(s) \le b_i + r_i(t-s) \quad \forall 0 \le s \le t$$

This (*r,b*)-bounded source generates k units of workload in a time length no smaller than $l = \left(\frac{k-b}{r}\right)^+$ [12].

Assuming that the workload arrival bound is under the R-(m,k)-firm constraint. During the increment of the water height, the switch of Fig.4 remains closed until the height increases to q_2. Once it is opened at q_2, it remains opened state until the height reduces to q_1. When the switch is opened (control function's value is 1), the Discarding Leak throws out the water from the bucket, so that the water height will be effectively reduced to assure delay factor. During this procedure, it is obvious that no more than

C2/(C1+C2) quantity of water can pass through DL, such that this attribute will be used to guarantee the (m,k) factor.

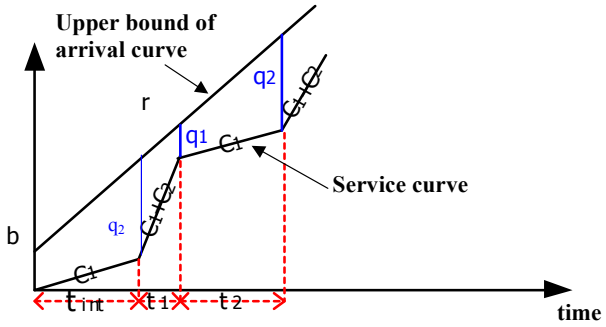


Fig. 6: Service curve evolution of DLB

Fig. 6 shows the evolution of the service curve under the critical cumulative arrival curve. Notice that SL is always on service unless the bucket is empty.

3.3 Sufficient condition for liquid model of DLB

After understanding the mechanism and its attribute to gurantee (m,k) factor and delay factor, we will propose the sufficient condition to configure DLB in order that deterministic guarantee can be achieved.

As shown in Fig.6, R-(m,k)-firm constraint is proposed in condition of congestion or overload, such that it is obvious that C1 < r. So the height of water grows higher and higher in the interval t_{int}, as the DL switch is closed initially. Once the height reaches q2, DL switch is opened due to the double threshold control function (Fig.5). In addition, we give C1+C2>r, so that the height of water goes lower and lower until the height decreases to q1 (in the interval t1). This procedure is iterated on until all arrival water has passed through either SL or DL.

Theorem 1:
If the liquid model of DLB is configured under follow condition, then the R-(m,k)-firm constraint will be deterministically guaranteed.

Condition (1) : $C_1+C_2>r;\ \frac{C_1}{C_2} \geq \frac{m}{k-m} \Rightarrow C_1 \geq r\frac{m}{k}$;

Condition (2): if $b>q_2$, then $\max\left(\frac{b-q_1}{C_1+C_2}+\frac{q_1}{C_1}, \frac{q_2}{C_1}\right) < \Delta$

else $\max\left(\frac{q_2-q_1}{C_1+C_2}+\frac{q_1}{C_1}, \frac{q_2}{C_1}\right) < \Delta$

Proof:

Condition (1) ensures the (m,k) factor.

As the liquid model, water is an infinitesimal material (the unit could be atom, molecule, gram or ton, etc.). We configure C_1, and C_2 as $\frac{C_1}{C_2} \geq \frac{m}{k-m}$. Let $Q_{SL}(t)$ represent the throughput quantity through SL during]0,t], and $Q_{DL}(t)$ represent the throughput quantity through DL. It can be ensured that $\frac{Q_{SL}(t)}{Q_{DL}(t)} \geq \frac{C_1}{C_2} \geq \frac{m}{k-m}$, since SL is always on service unless the bucket is empty, while DL is only on service during the switch is open. Thus in any k units of water passed through SL and DL, there are at least *m* units passing through SL.

Condition (2) ensures the transmission delay factor of R-(m,k)-firm : when *k* units of water is affused into the DLB at time $t_a=\left(\frac{k-b}{r}\right)^+$. We calculate the maximum delay of a packet serviced by either SL or DL.

One case is when the switch of DL is open, and this case is further divided into two sub-cases: $b>q_2$ and $b<q_2$.

If $b>q_2$, the burst causes the maximum bucket load, then SL and DL service the workload together until the height decreases to q_1. Thereafter, SL service alone until the burst is finished. We know that results in the delay of the service for the burst is given by:

$$\frac{b-q_1}{C_1+C_2}+\frac{q_1}{C_1} \qquad (1)$$

In this case, if $b<q_2$, the height must decrease once the switch is open, so the maximum delay can be given by:

$$\frac{q_2-q_1}{C_1+C_2}+\frac{q_1}{C_1} \qquad (2)$$

Another case is that the height is so near to q_2, but the switch is still closed. SL will service alone the workload until empty the bucket. This case results in the delay of service, given by:

$$\frac{q_2}{C_1} \qquad (3)$$

It must assure that in any case the maximum delay is no more than Δ, so that we give the condition (2).

End of proof.

In a concrete system, the source has its attributes of arrival curve upper bounded by (r,b) and R-(m,k)-firm requirement, so in the analysis, these parameters are regarded as the known parameters. Moreover, we can get the available bandwidth, so C_1 should be configured under the available bandwidth. Based on these known parameters we should constitute one DLB (configure C_2, q_1, q_2) to cope with this flow for providing the deterministic R-(m,k)-firm guarantee.

3.4 Numeric application of liquid DLB

To show the numerical application, audio-CBR streams' parameters are considered. For example, given one flow, it generates 2Mbps of average arrival rate with 6kbit of burst. Such that the flow is bounded by (r,b)=(2Mbps, 6kbit). As an example, we assume that such a stream is under R-(3,5)-firm constraint and with △=20ms as the per-flow deadline.

Assuming that at admission control, only 1.6Mbps bandwidth is available. Obviously, it is not possible to guarantee the deadline of all the packets, as congestions are unavoidable. We set $C_1+C_2 = 1.25r = 2.5$Mbps, so that the queue length can be effectively reduced in case of congestion. Then, DLB is implemented and configured as C_1=1.5Mbit/s; and C_2=1Mbit/s according to $C_1 = \frac{m}{k-m} C_2$; moreover, q_1 and q_2 are configured as that q_1=6kbit, q_2=12kbit. With this configured DLB, the maximum delay can be calculated by formula (2) as 8ms. So we can guarantee $t_{delay} \le 8\text{ms} \le \Delta = 20\text{ms}$. While, for guaranteeing all packets under delay of 20ms, it requires $r+b/t_{delay}$= 2.3Mbit/s of bandwidth [12]. With DLB, it guarantees R-(m,k)-firm constraint in providing 8ms delay, but it requires only 1.5Mbit/s of bandwidth. Intuitively, in this example, DLB can still work under smaller bandwidth, so DLB is robust under smaller bandwidth.

3.5 Packet model of Double-Leaks Bucket

In the packet switching network the information are encapsulated as packets, and the non-preemption is widely employed. So we now develop the DLB model according to the granularity of the packets and under the discrete time. Afterwards, the given R-(m,k)-firm is oriented the number of packets. Fig.7 depicts the mechanism. Two new parts are added, named *temporary vessel buckets* (TVB) for DL and SL, respectively. They take an entire packet from the DLB after the service of the current packet in themselves. TVB of the DL can only get the next during the switch is still in opened state.

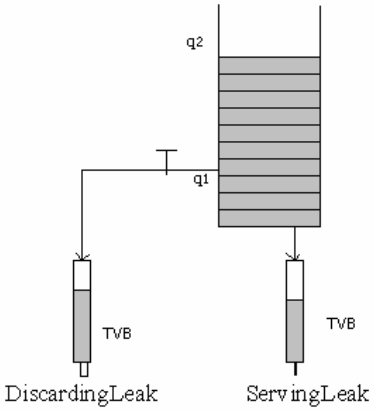


Fig. 7: Packet Model of DLB

The switch here is also controlled by the DTC function. Afterwards, for the packet model, the values of q_1 and q_2 are no longer the quantity of the water, but the number of packets; to represent the quantity of workload we use q_1S or q_2S, where S denotes the size of the packet (in unit of bit or byte).

3.6 Sufficient condition of DLB in packet model

Due to the packet granularity, the sufficient condition will be more complex than that of the DLB liquid model, which is given in the following theorem.

Theorem 2:
The sufficient condition for guaranteeing R-(m,k)-firm constraint of the packet model can be given as following:

(1) $q_1 \ge \frac{C_1}{C_2} \ge \frac{m}{k-m}$

(2) If $b<q_2$

$$\max\left(\frac{q_2-1}{C_1}S, \left(\frac{q_2-q_1}{C_1+C_2}+\frac{q_1}{C_1}\right)S\right) \le \Delta$$

else

$$\max\left(\frac{q_2-1}{C_1}S, \left(\frac{b-q_1}{C_1+C_2}+\frac{q_1}{C_1}\right)S\right) \le \Delta$$

Proof:
Condition (1) ensures the (m,k) factor.
As mentioned in the liquid model, it should be provided that $\frac{C_1}{C_2} \ge \frac{m}{k-m}$ as in the liquid model. Furthermore, we must take into account the granularity of the packet. In case that TVB of DL has just taken one packet when the bucket height is q_1+1, then the switch will be closed. Therefore, the service time of TVB of DL will continue $\frac{S}{C_2}$, and SL will be on service at least $\frac{q_1}{C_1}S$. The service process should be that SL is always on service during DL does, then $\frac{q_1}{C_1} > \frac{1}{C_2}$. This deduces that $q_1 \ge \frac{C_1}{C_2} \ge \frac{m}{k-m}$. We set q_1 as the minimum value such as: $q_1 = \left\lceil \frac{m}{k-m} \right\rceil$, and it is clear that $q_2 > q_1$. So we can choose one suitable value for q_2, which is neither too much big nor causes the switch rotate too frequently.

Condition (2) is derived with the same stragety as that for liquid model to guarantee delay factor.
End of proof.

3.7 Numeric application of packet DLB model

Let's consider the same concrete example as in the liquid model, the packet size is S=6kbit, then we can configure this DLB as C_1=1.44Mbit/s, C_2=0.96Mbit/s, q_1=2, q_2=5. With

this configured DLB, R-(m,k)-firm is guaranteed $t_{delay} \le \frac{q_2 - 1}{C_1} S = 10\text{ms} \le \triangle = 20\text{ms}$.

4. Comparison with other real-time constraint relaxation strategies

The R-(m,k)-firm scheme is a strategy to relax the too tight (and unnecessary) hard real-time constraint. Similarly, Pinwheel model [13], frame-based model [14] and Virtual window constrained model [11] can also be considered as other constraint relaxation strategies. In what follows we will discuss on those models and show that R-(m,k)-firm is a general model which can include the other ones.

4.1 Limitation of utilization gain when deterministic (m,k)-firm guarantee is required

The initial motivation of (m,k)-firm constraint is that the deadlines of at least *m* out of any *k* consecutive packets must be guaranteed. Similar to (m,k)-firm constraint, Dynamic Window-Constraint Scheduling (DWCS) [10] can accept a certain deadline misses such that *x* is the maximum acceptable packet loss in a fixed given window *y* packets.

(*m,k*)-firm and DWCS constraints just take into account the quantity of the deadline met or miss, whereas a stream is defined with several other parameters, such as *transmission time, period and dropping rate*. This unilateralism of analysis results in bad utilisation of resource, and this poor utilization factor causes the considerable over-provisioning of the system. Again, this over-provisioning causes the (*m,k*)-firm or its similar systems (e.g. DWCS) to loss their practical interest. Those facts have been shown in [7, 8, 9].

In [8], one theorem is proposed to show the utilization factor of a steam set serviced by multiprocessor under window-constraint. The result is very pessimistic, such that the system falls into failure state (no schedulable) even in arbitrary low utilization. The theorem is given as follows:

> For an arbitrary non-negative real number *u* and a natural number *g*, there exists a unit-size Window-Constrained stream set Γ, such that the aggregate utilization rate of Γ is less than or equal to *u*, and Γ is not schedulable by DWCS [10] on *g* processors.

In fact, such a theorem has also been given for (m,k)-firm constraint in [9]. Such as that:

> Under an arbitrary low utilisation, there is always a stream set Γ, which causes the system violate the (m,k)-firm constraint.

As mentioned above, providing deterministic (*m,k*)-firm guarantee can oblige resource reservation according to (*k,k*)-firm. This is to say that (*m,k*)-firm losses its practical interest for reducing the necessary resource reservation.

4.2 Complexity fatality

More general, in [7, 8], it has been proven that:

> The schedulablity of a stream set under (m,k)-firm or DWCS is a problem of NP-hard in strong sense.

4.3 Other relaxed constraint model

In fact, the deadline of instance is not the holy grail of real-time communication, and there have been a lot of work that try to relax the deadlines to gain higher utilization factor, such as frame-based model [14], Pin-wheel scheduling [13] and virtual deadline window scheduling [11], etc.

The frame-based model defines a set of tasks, which is to execute within each frame (time window) and is to complete before the end of frame. The problem is to schedule the task set in a single frame with deadline D.

Frame-based model assumes that all task instances are stored in a single queue at the beginning of the frame (i.e. released at the same time at the beginning of the time window). In this case, the utilization of processor can reach 100%. Clearly, if the total tasks' instances size is inferior or equal to the frame size, any no-idle scheduling can treat with them without any constraint violation. The scheduling in a frame can be repeated to do with infinite task instances.

Note that frame-based model removes the deadlines of instances to provide a scheduling in the same frame size for all tasks. However, for current diverse applications, tasks demand different frame size, which makes the frame-based model little interest for practical use in the QoS control in networks.

In our opinion, Pinwheel model is more interesting to reserve different quantity of time in different frame size for each task. The generalized pinwheel scheduling problem is an offline scheduling for satellite-based communication as follows: Given a multiset $\{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$ of ordered pairs of positive integers, determine an infinite sequence over the symbols {1, 2, 3, …, n} such that, for each *i*, $1 \le i \le n$, any subsequence of b_i consecutive symbols contains at least a_i times of i_s. Differing from frame-based model, Pinwheel guarantees the execution time in a sliding frame.

Together, Pinwheel model and frame-based model can find optimal scheduling scheme, and their utilization factor can arrive to 100%. However, as mentioned, frame-based model constraint all tasks in a unique frame, on the other hand, Pinwheel is designed for satellite-based communication model which only concerns unit size execution time. They cannot service current diverse multimedia applications.

Similar to R-(m,k)-firm constraint, Zhang and West [11] proposed a relaxed window constraint to gain utilisation. It allows task instances to be serviced after their deadlines, as long as it can guarantee a minimum fraction of service to a task in a fixed window. Its scheduling mechanism lengthens the instances' deadlines, and the deadlines are modified according to the execution time. Moreover, as like as DWCS, virtual deadline scheduling requires specially that the tasks should have the unit size execution time and their period must be the multiple of the execution time. Notice

that this model can be regarded as a special version of R-(m,k)-firm constraint in case that R-(m,k)-firm schem requires (m,k) factor in no-overlapping window (fixed window) with delay factor Δ=0.

In one word, R-(m,k)-firm synthesized all the above mentioned relaxation strategies to define a more general scheme for general task set. This scheme can be deterministically guarantee using DLB under the sufficient condition (theorem 1 and 2) proposed in section 3.

In the next subsection, we will shown the advantage gained by R-(m,k)-firm scheme. Simultaneously, it must be noticed that R-(m,k)-firm constraint can be well guaranteed even with the traditional simple scheduling policy, such as Rate Monotonic policy.

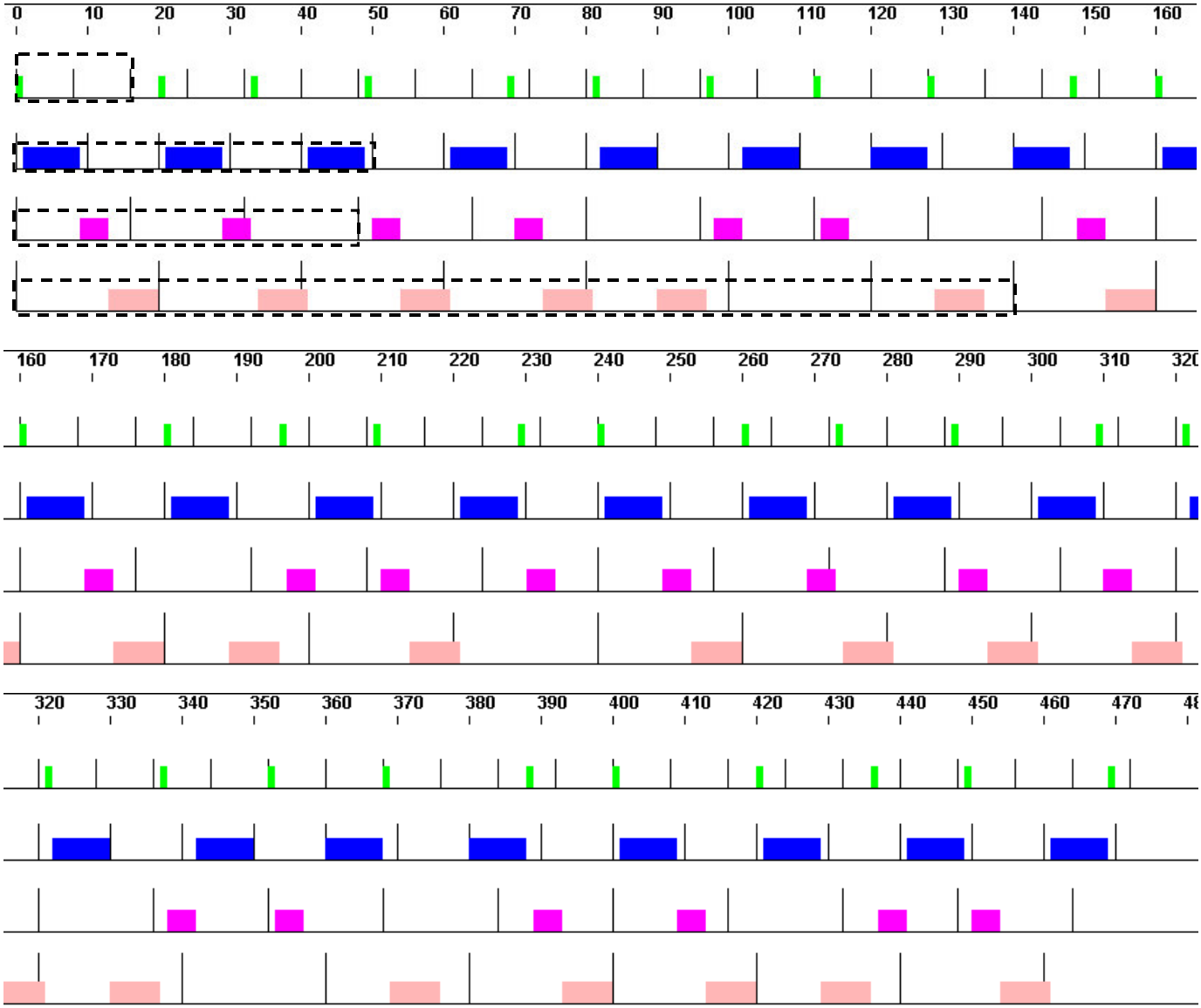


Fig.8: Scheduling trace of Table 1.

4.4 Comparison by simulation

In this subsection, we give a scenario to show that R-(m,k)-firm constraint can significantly gain the utilisation factor compared with (m,k)-firm constraint and DWCS constraint.

The scenario consists in four tasks, each of which has R-(m_i,k_i)-firm constraint. We will simulate for a strict delay factor (i.e., Δ=0) to show the high gain of utilisation.

The task set is scheduled under non pre-emptive fixed priority scheduling, and the priority is indicated by the index. Then, the schedule trace is shown in Fig.8.

Notice that R-(m,k)-firm constraints are well guaranteed for either sliding window or no-overlapping window with

zero delay (Δ=0). Such that, for any task, in any window of size $k_i p_i$, there are at least m_i instances executed. Dashed window is marked in Fig. 8, which can help the readers to verify this.

	Period (ms)	Execution time (ms)	R-(m,k)-firm
τ_1	8	1	(1,2)
τ_2	10	8	(2,4)
τ_3	16	4	(2,3)
τ_4	20	7	(5,7)
Utilisation		88% with Δ=0	

Table 1: R-(m,k)-firm simulation scenario

It is also easy to discover that a lot of missed deadlines exist during the scheduling trace for τ_3 and τ_4; caused by this, all deadline schemes (DWCS and (m,k)-firm constraint) will be violated. However, R-(m,k)-firm constraint replaced the per-instance deadline by the delay of group of instances, such that a high utilisation is gained (88%). Note that more gain of utilisation can be achieved with the no-zero delay factor.

Moreover, this scenario dealt with the periodic task set, which makes it different from the numerical applications given in section 3. The numerical applications given in section 3 are applications of DLB for the tasks upper bounded by (r;b), while periodic task has less burst. Again, none of the other deadline schemes can achieve the same gain in case of either this scenario or the numerical applications in section 3.

5. Conclusion

Since the current networks often falls into congestion caused by the overload or the over-provisioning is not always possible, graceful degradation of Quality of Service in networks is necessary for efficiently supporting the packet loss tolerant real-time applications such as VoIP, VoD, etc. Selectively discarding packets according to the (m,k)-firm model during overload periods is the key issue of our approach.

In this paper, two main contributions can be found. One is that we proposed a novel real-time QoS constraint, named as Relaxed (m,k)-firm constraint. Under this R-(m,k)-firm constraint, long consecutive loss of packets can be avoided, such that it is suitable for divers multimedia applications. This novel real-time constraint replaces deadlines for each packet by a delay factor of a packet group, moreover, it orients to the more general (b,r)-bounded streams, including periodic and sporadic ones. Another contribution is that one dynamic mechanism, called Double Leaks Bucket, has been proposed to deterministically guarantee the R-(m,k)-firm constraint.

The comparison with other schemes showed the generality of R-(m,k)-firm constraint in contrast with DWCS, Frame-based Model, Pinwheel Model, and Virtual Deadline Scheduling model. Furthermore, Simulation scenarios showed how R-(m,k)-firm constraint increases the resource utilization by replacing deadlines of packets with the delay of group of packets.

References:

[1] El-Gendy, M.A., A. Bose, K.G. Shin, "Evolution of the Internet QoS and support for soft real-time applications", *proceedings of the IEEE*, Vol.91, No.7, pp1086-1104, July 2003.

[2] Floyd, S., and Jacobson, V., "Random Early Detection Gateways for Congestion Avoidance". In *ACM/IEEE Transactions on Networking*g, 3(1), August 1993.

[3] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm deadlines", IEEE Transactions on Computers, 44(4), pp1443–1451, Dec.1995.

[4] F. Wang and P. Mohapatra, "Using differentiated services to support Internet telephony", Computer communications, Vol.24, Issue18, pp1846-1854, Dec. 2001.

[5] Striegel A., G. Manimaran, "Dynamic Class-Based Queue management for scalable media servers", *Journal of systems and software,* vol.66, No.2, pp.119-128, May 2003.

[6] Koubâa, A., Song, Y.Q., "Loss-Tolerant QoS using Firm Constraints in Guaranteed Rate Networks", *10th IEEE Real-Time and Embedded Technology and Applications (RTAS'2004)*, Toronto (Canada) 25-28 May 2004.

[7] G. Quan and X. Hu, "Enhanced Fixed-priority Scheduling with (m, k)-firm Guarantee", Proc. Of 21st IEEE Real-Time Systems Symposium, pp.79-88, Orlando, Florida, (USA), November 27-30, 2000.

[8] Mok, A.K. and W. R. Wang, "Window-Constrained Real-Time Periodic Task Scheduling", *22nd IEEE Real-Time Systems Symposium (RTSS'01)*, pp15-24, London, England, December 03 - 06, 2001.

[9] J. LI. Sufficient Condition for Guaranteeing (m,k)-firm Real-Time Requirement Under NP-DBP-EDF Scheduling. Technical report No. A03-R-452, Stage de DEA, LORIA, Jun, 2003.

[10] Richard West, Yuting Zhang, Karsten Schwan and Christian Poellabauer, "Dynamic Window-Constrained Scheduling of Real-Time Streams in Media Servers", *IEEE Transactions on Computers*, Volume 53, Number 6, pp. 744-759, June 2004

[11] Yuting Zhang, Richard West and Xin Qi, "A Virtual Deadline Scheduler for Window-Constrained Service Guarantees", in *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS)*, December 2004.

[12] Le Boudec, J.Y. and P. Thiran, *Network Calculus: A Theory of Deterministic Queueing Systems For The Internet*, Online Version of the Book of Springer Verlag – LNCS 2050, July 2000

[13] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. "The pinwheel: A real-time scheduling problem". In Proc. of the 22nd Hawaii International Conference on System Science, pages 693 – 702, January 1989.

[14] Frank Liberato, Sylvain Lauzac, Rami Melhem, Daniel Mosse. "Fault Tolerant Real-Time Global Scheduling on Multiprocessors", p.0252, 11th Euromicro Conference on Real-Time Systems, 1999.

[15] A. Koubâa, Y.Q. Song, "Evaluation and improvement of response time bounds for real-time applications under non pre-emptive fixed priority scheduling", *International Journal of*

Production Research, Vol.42, No.14, pp2899-2913, Taylor & Francis group, July 2004.

[16] N. Jia, E. Hyon, Y. Song, "Ordonnancement sous contraintes (m,k)-firm et combinatoire des mots", RTS'2005, Paris (France), April 2005.

SCoCAN: A communication Protocol for Distributed Real Time Systems

J.O. Coronel, P. Pérez, G. Benet, F. Blanes, , J.E. Simó, A. Crespo
Departamento de Informática de Sistemas y Computadores.
Universidad Politécnica de Valencia, Spain.
jacopa1@doctor.upv.es, { pperez, gbenet, pblanes, jsimo, alfons }@disca.upv.es

Abstract

This paper describes a communication protocol, called SCoCAN (for Shared Channels on CAN). This protocol is based on a hybrid communication scheme combining time triggered (TT) and event triggered (ET) paradigms with temporal isolation. Both traffic types are handled by time slots dedicated to each one. SCoCAN has been primarily projected as a communication infrastructure for distributed control applications. Its main goal is to provide determinism in the communication, but without wasting the bus bandwidth. This is achieved by means of a dynamic bandwidth recovery method, based on the recycling of the unused time slots.

1. Introduction[1]

Nowadays, distributed embedded systems are more and more used in complex control architectures with high degree of autonomy. It is common to find them in flight controllers, cars, robots, industrial control, and so on, where fieldbus-based communication systems are frequently found.

Embedded systems generally operate as closed-loop control systems: they sample sensors, calculate appropriate control responses and send those responses to actuators. Hence, to achieve a correct operation of control loops and an appropriate integration of information - both spatial and temporal - real time performance is strongly required. And therefore, the computational resources of each node as well as the temporal characteristics of the communication fieldbus must be taken into account at designing time [9].

Fieldbus designers are typically concerned about the capacity to deliver both time triggered (TT) and event triggered (ET) communication services under timing constraints. Therefore, an adequate scheme seems to be a combination of both TT and ET services, trying to share their respective advantages. This is the approach used in the SCoCAN protocol.

Although there is a great variety of real time buses, CAN[4] (Controler Area Network) is one of the preferred solutions to communicate distributed systems into reduced spaces, such as in mobile robots [17]. But the native CAN protocol does not guarantee a minimum jitter nor the exact moment of transmission due to the high variability of response times in CAN messages, produced by error conditions in the channel and by the traffic overload [18]. This can produce missing deadlines in some control applications. As demonstrated in [15], communication jitter have an adverse effect into many distributed control systems. To avoid these problems, some modifications to the native CAN protocol have been made, leading to new hybrid protocols to appear, such as TTCAN [7], FTTCAN [1] and SCoCAN (our case). These protocols will be introduced in a forthcoming section.

In the next section 2 the main characteristics of the ET and TT communication paradigms have been highlighted, as well as the hybrid ET-TT approach, emphasizing some characteristics such as bandwidth efficient utilization and jitter. Afterwards, in section 3 is presented the SCoCAN protocol, describing its main temporal characteristics, the different slot types and latency specifications. Next, in section 4, several existing protocols on CAN, are compared with SCoCAN. Finally, in section 5 the implementation of SCoCAN in the YAIR robot is used as a case study to test the performance of the protocol.

2. Communication Paradigms

Two different communication approaches for the design of communication infrastructures in distributed applications are mainly used: Event-Triggered (ET) and Time-Triggered(TT) [13], [11]. In the event-triggered (ET) paradigm the system activities, such as the sending of a message, are triggered by the occurrences of events in the environment, whereas in the time-triggered (TT) paradigm the activities are triggered by the progression of the global time. This section will compare those two paradigms, whose emphasis is related to requirements of predictability, resource utilization and efficiency.

2.1 Event-Triggered Paradigm.

In Event Triggered communication systems the temporal control is external to the communication system. That is, only the sender node has knowledge about the point in time when a message has to be transmitted. Therefore, the required amount of resources (i.e.., network bandwidth) for the worst-case communication can become higher when one considers the situation in which all nodes attempt to communicate simultaneously. However, ET paradigm is efficient

[1] This work is being developed under the FEDER-CICYT grant nº DPI2002-04434-C04-03.

concerning average resource utilization, since the nodes generate messages only in response to significant state changes in the environment.

In addition, the ET communication does not explicitly require a global notion of time, and consequently neither synchronization methods. Moreover, the ET communication can support different system configurations that change over time, thus the communication infrastructure is supplied of flexibility [12]. However, the temporal uncertainty of the ET communication protocols can be large and have an adverse effect on control systems [15].

Typically, the ET communication is used to convey alarm conditions, asynchronous non real-time traffic, or sporadic and large data blocks.

2.2 Time-Triggered Paradigm.

In TT communication systems, temporal control resides within the communication system, and is independent of the application in the nodes. All communication occurs at predetermined instants in the time at a rate determined by the dynamics of the environment under control. This allows a relative phase control among the streams of messages to be transmitted over the communication bus, and consequently, a reduction on the number of nodes that attempt to transmit simultaneously. Besides, this feature leads to composability with respect to the temporal behaviour, one of the most important properties of TT communication as is emphasized by Kopetz[12]. On the other hand this paradigm allows an efficient management of resources utilization in communication peak loads. However, if the load condition is low or average, then the resources utilization will be worse than those obtained with a comparable ET approach.

In TT communication, the clocks of all nodes must be synchronized to form a global notion of time. Hence, they require system-wide synchronization mechanisms that can increase the complexity of system management. However, as previously described, the TT approach allows the phase control over the communication, and therefore is possible to eliminate or bound the communication jitter.

Usually, the TT communication is adequate for control applications that require regular transmission (e.g. engine control, motion control, robots control) and it is used to convey data with critical timing, periodic and with long deadlines.

2.3 Joining E-T and T-T paradigms.

Previously, the characteristics of ET and TT paradigms were separately described. However, many practical applications of distributed embedded control, such as automotive systems and robots, require the exchange of information of both sporadic and periodic nature. This last is associated with control loops and the first with alarms, management or code delegation between nodes. Even though these two types of traffic can be conveyed over totally ET systems (as CAN[4] based systems), or totally TT systems (such as TTP[19]), the network efficiency suggests to join both paradigms, sharing their particular advantages, such as intermediate level of flexibility due to the ET part (any node can attempt to transmit) and a predictable temporal behaviour due to the TT part. However, the combination must implement temporal isolation of both types of traffic, to prevent that asynchronous ET traffic can ruin the advantages of the TT paradigm due to mutual interference. A way to achieve this isolation is allocating bandwidth exclusively to each type of traffic.

However, this hybrid scheme is not recent, e.g. in [14] two consecutive phases dedicated to one type of traffic each are defined. Consequently, the bus time turn into an alternate sequence of TT and ET phases. Another typical examples of protocols that using this hybrid scheme are FTTCAN [1], FlexRay[3] and TTCAN[7], moreover, the first and third protocols are one example of a hybrid scheme over fully ET native bus, and the second is one example of transmission over fully TT native bus.

3. Existing Protocols

Nowadays there is a great variety of fieldbus protocols and their election must be made in agreement with the application's requirements and with some relevant characteristics, such as ease of implementation, flexibility, fault tolerance, commercial availability, temporal constraints, required bandwidth, type of transmitted data, etc. For their relevance and interest, some existing protocols will be described in this section, highlighting their main characteristics.

WorldFIP [8] fieldbus uses a centralized MAC protocol (master–slave), first, the master has to poll the nodes for the existence of aperiodic requests to be served, and this is normally carried out using the periodic traffic assigned to each node. Then, when a node indicates that it has pending aperiodic requests, the master has to poll the node for the identification of the individual requests and finally process them one at a time. Therefore, into this protocol the handling of ET traffic is relatively inefficient, requiring a considerable amount of bandwidth to allow the master node to stay alert for aperiodic requests.

Foundation Fieldbus-H1[8] has one communication scheme comparable to WorldFIP. A Link Active Scheduler (LAS) is used for scheduling transmissions of TT messages and authorizing the exchange of data between Link Masters (LMs) devices. Furthermore, this scheduling allows the transmission of event-triggered messages only during precise time windows that do not extend beyond of the time used by the time-triggered messages. The LAS organizes the communication passing a virtual token ring to put order in the access to the network. This token-based method is also relatively inefficient because the token itself consumes bandwidth, and the nodes with pending aperiodic communication requests have to wait for the token even if the remaining nodes in the ring list have no requests.

TTP (Time Triggered Protocol) [19] is based on a pure TDMA (time-division multiple access) approach, with exclusive slots and with static scheduling. The support of time-triggered traffic is obvious whereas the event-triggered traffic can only be supported by pre-allocating a number of slots for the transmission of eventually pending event-triggered messages. However, these slots are dedicated and, at a given instance, if there is not any transmission request for the respective message, the slot remains unused. This time-based polling mechanism for each event-triggered message produces high efficiency under worst case requirements and low efficiency under average-case requirements. Although TTP supports addition of nodes by booking enough time in the TDMA round, this causes extra bandwidth to be allocated, constituting an inefficient bandwidth management.

FlexRay [3] combines a time-triggered along with an event-triggered system. Based on an extended TDMA media access strategy, it has a communication cycle divided into a mandatory static segment, and an optional dynamic segment. In the static segment, requirements such as latency and jitter are handled by means of deterministic communication timing. Into this segment the time slots are equally sized and the point of time when a frame is transmitted on the channel is exactly known. The slot assignment is done off-line during system planning. The dynamic segment consists of one slot of fixed duration and subdivided into mini-slots. A prioritization scheme enables variable bandwidth distribution during runtime. Each sending controller has a mini-slot assigned to a transmit-frame. However, this latter is equivalent to assigning different wait time to asynchronous messages according to their priority. This mechanism can result in substantial bus idle time when there are ready-to-send, yet low-priority messages. Moreover, if a high-priority ET message is generated just later of its assigned mini-slot, this message must wait until next communication cycle, whereas other low-priority messages can be transmitted.

Flexible Time Triggered CAN (FTTCAN)[1] is an extension of CAN, based on a dynamic scheduling TDMA. FTTCAN has a basic cycle divided into two windows, one asynchronous used to transmit ET messages and whose access to the bus is determined by the conventional CAN protocol. The second window is synchronous, being in this window where the TT messages are transmitted. In this window the traffic is dynamically scheduled by a master central node. This feature enables the online admission control of dynamic requests for periodic communication because the respective requirements are held centralised into one local table. In this protocol the handling of ET traffic and network utilization is efficient, and also, it has a flexible handling of TT traffic supporting online admission of change requests for this traffic. However, into synchronous window (TT traffic), lowered and bounded communication jitter might appear due to that within this windows the access to the bus is determined by the conventional CAN protocol.

Time Triggered CAN (TTCAN) [7] is another extension of CAN, based on static schedule TDMA. TTCAN uses a reference message to indicate the beginning of each basic cycle. A basic cycle is divided into three different types of windows: private windows, used to transmit one specific message only, arbitration windows, where the nodes compete by the access to the bus as in a conventional communication of CAN, and free windows, used for future extensions. In this protocol the basic cycles are not always the same. The complete pattern of TTCAN traffic is integrated by a consecutive number of basic cycles that form a matrix. However, there are several practical constraints that must be observed when building the table. For example, all the windows in the same column must be of equal width and type. Moreover, the exclusive windows are dedicated to the transmission of a single time-triggered message. However, the fact that there is a CSMA-based MAC protocol that resolves collisions at bus access during the arbitration windows greatly simplifies the handling of event-triggered traffic.

4. Introduction to SCoCAN.

A first approach for the proposed SCoCAN protocol was presented in [6]. SCoCAN (Shared Channel on Controller Area Network) is a higher layer protocol on top of the CAN data link layer, and follows a hybrid communication scheme, combining time triggered (TT) and event triggered (ET) traffic, but with temporal isolation (achieved by exclusive allocation of bandwidth to each type of traffic in successive time slots).The main goal of SCoCAN is to remove or reduce the jitter, also exploiting the maximum physical bandwidth of the bus. The former is achieved by triggering the time slots sequence, supplying determinism to the communication bus; whereas the latter is achieved by means of dynamic bandwidth recovery by recycling unused time slots. The adoption of CAN bus for SCoCAN protocol has several advantages. It simplifies and makes efficient the handling of event-triggered traffic due to the CAN collision resolution mechanism which utilizes an access method CSMA/CR [4]. Moreover, CAN network controllers and their cabling are relatively inexpensive and the relatively robust physical layer with respect to error detection and tolerance of physical faults enables SCoCAN-based systems operate in harsh environments. And additionally, the CAN controllers have great commercial availability and can be found embedded into several microcontrollers as well as in microprocessors.

The nomenclature used in our protocol follows. The SCoCAN bus time is organized as a sequence of variable duration time-slots. This sequence is called *Basic Cycle* (BC), and the slots size, distribution and assignments are defined at pre-runtime. The BC is organized as a static time table and is distributed to all nodes on network at start-up. In addition, changes between pre-defined and post-defined operational modes are also allowed. The nodes are synchronized by a strictly periodic reference message named Sync Message (SM), which marks the

starting of each *basic cycle*. This message is sent by a master node.

Within each BC are defined several successive time-slots used to convey different types of traffic: *Private slots* are used to convey time-triggered traffic, and are called *private* because in each of these slots only a node can transmit data. *Shared slots* are used to convey exclusively event-triggered traffic, and are called *shared* because all nodes may try to transmit using the native CAN arbitration mechanism. And finally, the *Recycled slots* are the result of a dynamic transformation (at runtime) of *private slots* into *shared slots* when these first remain unused. This transformation achieves high-bandwidth efficiency by means of an active bandwidth recovery and avoiding a waste of bus bandwidth.

To reinforce the temporal isolation between time-triggered and event-triggered traffic and to maintain the temporal properties of TT traffic, such as composability, the transmissions that could not be completed within the *shared slots* must be removed from the network controller transmission buffer, keeping them in the transmission queue. Thus, a short idle time will be defined at the end of each *shared slot* when this is followed by a *private slot* or Sync Message.

4.1. The Basic Cycle and their time slots

The time between two synchronization frames constitutes a *Basic Cycle (BC)* (see Figure 1). And this is formed by adding successive time slots of variable duration, whose length will depend on the transmitted data characteristics. The duration of the whole BC sets the temporal resolution of the communication system. Therefore, the transmission periods of the time-triggered traffic are integer multiples of the BC duration.

As previously described, the time slots of a BC can be used to convey messages with TT or ET characteristics, and any message to be sent has the CAN data format and utilizes a standard CAN message. Moreover, a table that determines the rank of CAN identifiers for each message (depending on traffic type and its priority) must be also defined.

In SCoCAN network, all the nodes will use a transmission time-table to define the BC of protocol, which is previously defined and saved in all nodes during bus start-up routine. Additionally, several operational modes (transmission tables) can be predefined into nodes or dynamically distributed on the network (at runtime).

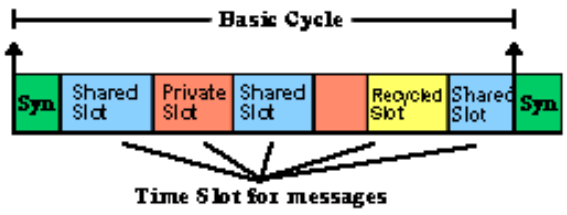


Figure 1. A Basic Cycle (BC) in SCoCAN.

The Figure 1 shows an example of *basic cycle*, with the synchronization message followed by *private slots* and *shared slots*. (In the figure, one *Private Slot* has been recycled to *Shared*).

4.1.1 The Synchronization Message

The synchronization of the modules within the network is done via a periodic Sync Message (SM) which is clocked by one primary or secondary master node. All nodes of the SCoCAN network identify the reference message by its identifier (generally this message has the highest priority). The receipt of SM causes a new reconfiguration of local timers and restarts the cycle time in each node. Additionally, this message may hold additional control information, such as time information or operational modes.

Depending on implementation, the synchronization process may get delays because of differences in SM message reception time in each node (see Figure 2). The synchronization accuracy depends on the physical signal propagation on the bus line and on the processing time of the message. This small delay will cause differences on the starting point of the time slots which must be taken into account. In the Figure 2, nodes 0 and 1 have sensed the reference message in different instants of time, and consequently, in the synchronization process an uncertainty interval may be found.

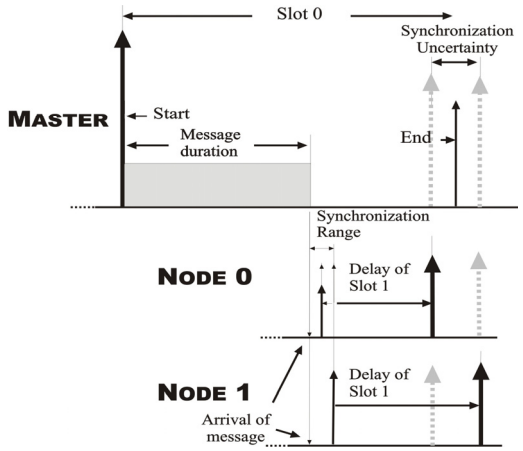


Figure 2. Delays produced during synchronization.

4.1.2 Private Slots.

These slots are used for messages with hard real time constraints, synchronization messages or configuration messages, (i.e., for time-triggered traffic).

In these slots, only one of the nodes is allowed to transmit data (proprietary node) avoiding eventual collisions on bus access. Thus, the communication jitter can be eliminated or bounded. Each node checks when it is allowed to transmit by scanning a local time table containing the identification of the message, type of slot, duration and proprietary node.

The assignation of messages to private time-slots is off-line scheduled, and moreover, to provide flexibility

to scheduling, the same message identifier may be assigned to several time slots into the same BC.

To provide reliable communication, the automatic retransmission caused by transmission errors, (characteristic of the original CAN protocol) is allowed into these slots, but the total retransmission duration is limited to the length of the slot where it takes place.

4.1.3 Shared Slots.

These slots are intended for messages with non-critical timing (soft real time constraints), alarms or messages with large blocks of data, (i.e. for event-triggered traffic).

These slots have not any proprietary. Thus, the nodes use the native priority-based distributed arbitration mechanism of the original CAN protocol and consequently, inheriting its efficiency in handling event-triggered traffic. The scheduling policy is priority driven, with fixed priorities expressed as message identifiers. The automatic retransmission caused by transmission errors of original CAN protocol is also allowed in these slots. But, to maintain a strict temporal isolation between both types of traffic (Time-Triggered and Event Triggered), the *private slots* must be protected from interference of ET traffic. This is achieved by adding a short *idle time* to the end of each *shared slot* (see Figure 3) when this is followed by a *private slot* and to the ending of a BC. All transmission activity is suspended in this *idle time*, including retransmissions and the nodes with pending ET transmission requests are kept on hold until the next *shared slot*.

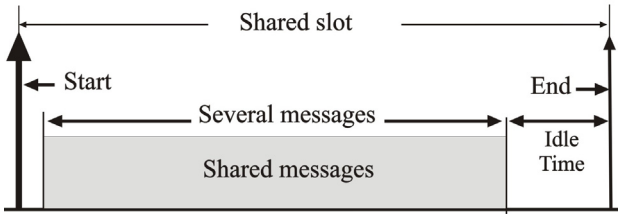


Figure 3. Shared Slot.

4.1.4 Recycled Slots.

An important characteristic of our protocol is that the *private slots* (for TT traffic) can be transformed into *shared slots*, when in the *private slots* there are no data to be transmitted. This transformation is dynamic (during runtime) providing a dynamic bandwidth recovery and avoiding a waste of bus resources.

When inactivity on the bus is detected during an interval of time (inactive time) after the starting point of a *private slot*, other nodes with ET traffic queue, are allowed to start to send their messages (see Figure 4). This inactivity may be discovered directly by sampling the CAN bus for the type A nodes, or by a particular message reception (called Sync1) in the case of type B nodes. The wait time to detect inactivity is named *recycling time*.

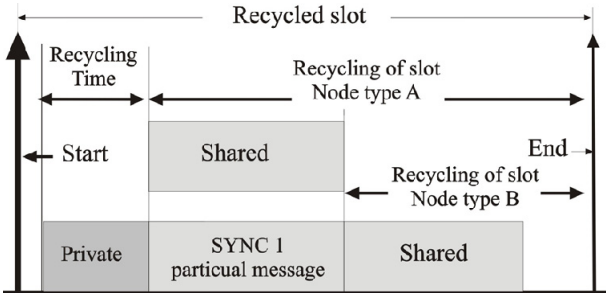


Figure 4. Detection of inactivity in Private Slots. Recycled Slots.

In protocols such as TTCAN or TTP, when a *private slot* is allocated to a node and for any reason this node is turned off, (e.g. HW fault or maintenance), it can cause a waste of bus bandwidth; because these time slots remain allocated and unused. However, in the case of SCoCAN, its recycling mechanism enables private *slots* to be transformed into *shared slots* for ET traffic, thus recovering the bus bandwidth.

4.2 Node types in SCoCAN.

Primary Master Node: its job is to synchronize all nodes on network with high timing accuracy. This node sends SM's to mark the starting of *basic cycle*. Also, the master node must be capable to detect inactivity via hardware in *private slots* and must indicate the unoccupied condition of the slot by sending a particular message (*Sync 1)*. This last feature can be also delegated to other node.

Secondary Master Node: a pre-defined set of nodes (named as Secondary Master Nodes) in the network constantly monitors the activity of the primary master node. If the primary master node fails, any secondary master node immediately handles the synchronization task. Similar criteria as in [10] would be used for the selection of the secondary master node.

Node Type A: Node with an intelligent board, capable of sampling the CAN bus lines, and detect inactivity via hardware. This feature allows these nodes to quickly transmit when there is inactivity during the *private slots*.

Node Type B: Node without capacity of bus sampling. This type of nodes can not recycle the empty *private slots* unless other specialised node (Master node) sends a short message *(Sync1),* signalling the unoccupied condition of the *private slots*.

4.3 Latency Specifications

In this section we present the analysis for the worst-case latency of non-critical messages over *shared slots* on SCoCAN protocol. The analysis has similarities with the analysis described in [17][18], used for computation of maximum transmission latencies on classical CAN networks. The main difference is that the test described in this paper mixes the transmission of TT critical messages on *private slots* with the transmission of non-

critical messages on *shared slots* in a schema equivalent to classical CAN.

Timing analysis makes the following simple assumptions about the characteristics of a message *m*:

- Message *m* has a bounded size,
- Transmission periods T_m are known and constant,
- Jitters to first *shared slot* JS_m are also known,
- The length and identifier of all messages are known.

Timing analysis makes also the following simple assumptions about protocol features:

- The distribution of the shared and *private slots* on each *basic cycle* is static.
- The *private* slots transformation into *shared slots* feature is not applied.
- The exploitation of private messages is not included.
- The transmission is free of errors.

The longest time R_m from task activation until message m gains control of the bus can be calculated using the following extended schedulability test Eq. (1).

$$R_m = JS_m + w_m + C_m \tag{1}$$

Where JS_m is the blocking time due to *private* slots, from the BC start time until the first *shared slot* and it depends on the system configuration. These parameters is used in a similar way with the blocking factor *B*, due to messages of lower priority in the equation described in [18] used for computation of maximum transmission latencies on classical CAN networks.

The C_m factor is the longest time taken to physical transmission of message m defined by:

$$C_m = \left(\left\lfloor \frac{34 + 8s_m}{5} \right\rfloor + 47 + 8s_m \right) t_{bit} \tag{2}$$

Where sm is the number of bytes in the data field of message *m*, and t_{bit} is the time taken to transmit a bit on bus. The term w_{m} (Eq 3) represents the worst-case queuing delay message m. It is similar to longest time between placing the message in CAN controller registers to start of transmission of the message m on a *shared slot* and it is presented in the recursive equation 3:

$$w_m^{n+1} = B_m + IHP_m^n + ISP_m^n \tag{3}$$

Where the term B_m is the worst-case blocking time of message m due to lower priority messages.

$$B_m = \max_{j \in lp(m)} C_j \tag{4}$$

The $I_{hp(m)}^n$ is the delay due to interference between transmission of message *m* and the set of higher priority messages. This is a recursive relation where the factor n defines iteration on the computation. Generally, the initial value is zero, which permits a fast convergence.

$$IHP_m^n = \sum_{j \in hp(m)} \left\lceil \frac{w_m^n}{T_j} \right\rceil C_j \tag{5}$$

Where *hp(m)* is the set of messages with higher priority than message *m*.

$$ISP_m^n = \sum_{\substack{k=0 \\ kT_{CB} < R_i^n \\ R_m^n < \phi_m}}^{\infty} \left(ISPn_m^{k,n} + IS\,\mathrm{Pr}_m^{k,n} \right) \tag{6}$$

The I_{sp}^n (eq. 6) is a blocking time on the message *m* due to *private slots* and it is divided to two factors which depend to allocation of slot into *basic cycle*. Where ϕ_m is signal threshold quality of message *m*. T_{CB} is the rate of *basic cycle*. *k* is the amount of *basic cycle* until the message is transmitted.

$$ISPn_m^{k,n} = \min\left[\left\lfloor \frac{R_i^n}{\left(mT_{CB} + \tau_l^e\right)} \right\rfloor, 1 \right] \cdot \left(T_{CB} - \tau_l^e + \tau_0^s \right) \tag{7}$$

The equation 7 shows the interference produced by private slots at the end of the BC.

The meaning of the terms $\tau_{slot.num}^{position}$ follows: *position* indicates the (s)tarting point or the (e)nding point of the slot, respectively; whereas *slot.num* denotes the order number of the slot into the set of shared slots Ψ_{SS} : {*0, ···, h, ···,l*}. Thus, τ_0^s refers to the time corresponding to the starting of the first slot.

$$ISP\,r_m^{k,n} = \sum_{h \in \Psi_{ss} \wedge \tau_h^s < R_i^n} \min\left[\left\lfloor \frac{R_i^n}{\left(mT_{CB} + \tau_h^e\right)} \right\rfloor, 1 \right] \cdot \left(\tau_{h+1}^s - \tau_h^e \right) \tag{8}$$

This equation show the interference produced by the *private slots* into *basic cycle*, but the interference defined by the equation 7 is not included.

5. CAN protocols comparison

In this section the behaviour of four protocols (TTCAN, FTTCAN, SCoCAN and native CAN) are compared using discrete simulation of several sets of messages by means of a java application tool developed ad-hoc.

The set of messages are generated automatically by the application and clustered in two groups: HRT and SRT (hard real time and soft real time). The HRT are messages with high priority identifier, (in this simulation, any message with an identifier under 1024) and the SRT are the rest of messages (SRT identifiers range between 1024 and 2047). The bus bandwidth reserved by the compared protocols each group is: 60% for SRT and 40% for HRT. (Note that in CAN there are not differences for both traffic types). In FTTCAN,

asynchronous and synchronous windows have reserved the 60% and 40% size of basic cycle respectively. And finally, TTCAN and SCoCAN have defined the 40% of the slots as *private slots*.

To observe the behaviour of each protocol for SRT traffic, the application applies a workload of 100% of bus bandwidth for SRT messages and then simulates this scenario for different levels of HRT messages workload.

The next figures 5, 6 and 7, depict the results of simulations using 0%, 50% and 99% HRT workload levels for reserved *private slots* and show the maximum latencies in microseconds versus the message identifiers. It must be also noted the logarithmic scale for latencies. Finally, in the figure 8 the latencies corresponding to the HRT messages when a 99% of HRT bandwidth is used are shown. (This is the same case as in the previous figure 7).

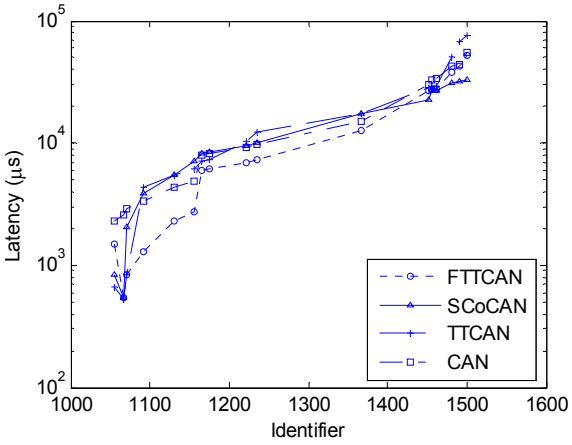


Figure 7. Latencies of SRT messages when a 99% of HRT bandwidth is used.

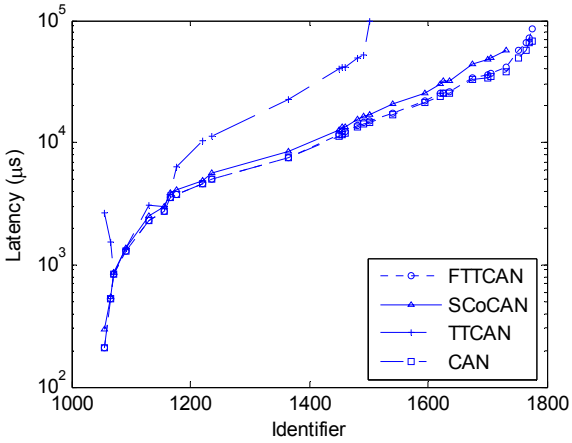


Figure 5. Latencies of SRT messages without HRT workload.

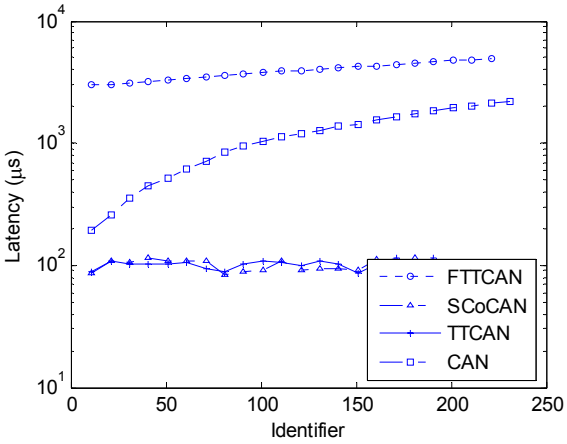


Figure 8. Latencies of HRT messages when a 99% of HRT bandwidth is used.

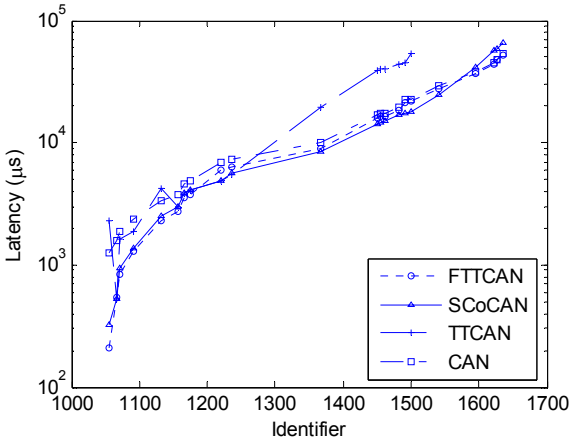


Figure 6. Latencies of SRT messages when a 50% of HRT bandwidth is used.

Examining figures 5 to 7, for each HRT workload, it can be observed that the latencies of SRT messages on TTCAN and FTTCAN are not modified by the variation of the HRT workload percentage. This is a consequence of the complete isolation properties of both traffic types. In the case of TTCAN (see fig.8), the jitter for HRT messages are minimal, whereas in the case of FTTCAN the jitter depends on HRT workload, having a maximum bounded by the length of the synchronous window.

In contrast, the maximum latencies on CAN and SCoCAN depend on the workload of HRT messages. This behaviour is natural on CAN and it follows the Tindell formula [18]. Note that in SCoCAN, for lower HRT workloads, SRT latencies are similar to the CAN case, but for higher HRT workloads, the SRT latencies are similar to those of TTCAN case. This effect is due to the reuse of the available HRT bandwidth.

6. Case study

The SCoCAN protocol has been implemented on the YAIR[2] autonomous mobile robot, whose distributed architecture, temporal sensors and actuators characteristics, and message frames are defined in [5] and [16]. Several field tests and several data acquisition and analysis tools have been implemented, to validate the performance of the protocol. These will be described next.

6.1 Basic Characteristics of the test

The YAIR robot [2] is a distributed system with embedded intelligent modular nodes which manage different subsystems, such as sensors, actuators and control devices. Each node also handles the SCoCAN communication protocol. The nodes involved in the present test are: *infrared node*: it reads a ring with 16 IR sensors; *motor node*: it is used to control the robot's motors and to send speed, acceleration and odometer messages; *odometer node*: it generates high-resolution position data; *ultrasonar node:* it sends for each reading the digitised envelope of an ultrasonic echo; *central node:* it is the main processor of robot. A more detailed description of each node can be read in [5] and [17].

Given the YAIR's architecture and the temporal characteristics of the devices into their distributed system [5][15][16], some basic parameters of the test have been selected: the first one, a *basic cycle* of 10ms and divided into 20 slots of 500 microseconds each has been defined (see Figure 9), (although in the implementation the neighbouring *shared slots* are joined). And second, transmission features of CAN messages such as such as CAN identifier, data size, assigned slot into the BC, slot type, rate and number of CAN messages sent each period are defined in the table 1.

Other SCoCAN features are also defined as follows:

- The *idle time* used is 10*us*
- The *recycled time* used is 20*us*.
- And the CAN transmission rate is established in 1Mbps.

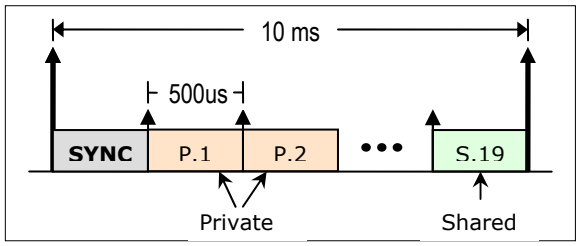


Figure 9. SCoCAN basic cycle for the YAIR robot

In these tests, three basic type of traffic can be found:

- *TT traffic:* this is produced by central, infrared, odometer and motor nodes.
- *Continuous ET traffic:* this is generated by ultra sonar node.
- *Sporadic ET traffic:* this is produced by all the nodes on the network, and this traffic is formed by some alarms and data file transfer. This latter is used to code delegation and distributed file system.

Table 1. Features of the messages used in the SCoCAN tests on YAIR robot.

Specification / Message(msj)	Size Data (byte)	Rate of generation	Nº of msj.	SCoCAN slot	Type of message	Identifier
Infrared	8	20 ms (ring)	4	10,19, 13,16	Private	0x100 – 0x103
Motor-Alarm	1	by event	1	-	Shared	0x010
Motor-Control Mode	1	Generated by PC	1	-	Shared	0x300
Motor-odometer	8	10 ms	1	4	Private	0x10A
Motor-Speed	4	Generated by PC	1	7	Private	0x10B
Extra Odometer	8	10 ms	1	3	Private	0x10C
Ultra sonar	8	200µs/ 80ms	32	-	Shared	0x310 - 0x318

6.2 Data acquisition and analysis tools

To store the transmission and reception time of messages for further analysis and to handle the SCoCAN protocol, a real time communication driver together with a real time CAN monitor were implemented. Additionally, several data analysis tools were developed to obtain the main features of communication process, such as utilization factor of CAN bus, message latencies, jitter, amount of *recycled slots* and accuracy of data transmission into the *private slots*.

6.3 Working modes used in the test

Two working modes of the SCoCAN protocol have been implemented (see Figure 10) on YAIR robot:

- SCoCANv1: only one message can be conveyed into *private slots*. The retransmission by error is disabled which will permit a physical replication of buses. An example of this test is shown at the top of Figure 10.
- SCoCANv2: The automatic retransmission by error is enabled on all the slots. Moreover, several shared messages (ET messages) can be conveyed into the end of each *private slot* even though a TT message is transmitted. However, this transmission of ET messages is possible only when there are not errors on the transmission of the previous TT message. (See Figure 10).

In both tests, the fundamental characteristics of SCoCAN protocol are implemented, such as the recycling of the slots.

[2] YAIR stand for Yet Another Intelligent Robot and is currently developed in our laboratory.

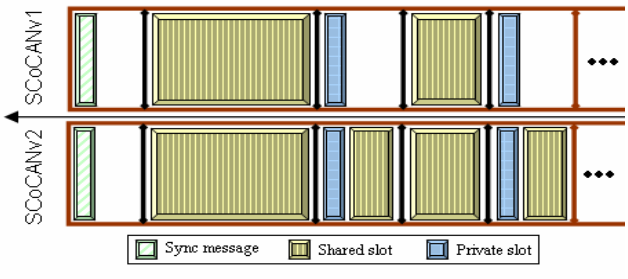


Figure 10. Working modes used in the tests.

6.4 Results of the SCoCAN test

Using the analysis tools, the simulation time was established in 100 *basic cycles*. In the Table 2, the main results of SCoCANv1 test are showed. In this test, the maximum utilization factor (UF) of CAN bus versus *basic cycle*s (BC) was 64.23%, while the minimum UF was 53.83%. Moreover, 35.7% of *private slots* were recycled and hence the bus bandwidth was not wasted.

Table 2: Main features of the SCoCANv1 test

SCoCANv1 TEST			
Simulation time	100 basic cycles		
Maximum UF/BC	64.23%		
Minimum UF/BC	53.83%		
Average UF/sec	58.38%		
Private slots with transmission time deviation <|x| µs	Deviation <|6us|	Deviation <|4us|	Deviation <|2us|
	100%	98.1%	71.0%
Recycled slots	250 recycled / 700 private		

On the other hand, the features of SCoCANv2 test are presented in the Table 3. In this test, the maximum utilization of CAN bus by cycle was 82.56%, whereas the minimum UF was of 59.03% and the amount of *private slots* that were recycled is similar to the SCoCANv1 test.

Table 3: Main features of the SCoCANv2 test

SCoCANv2 TEST			
Simulation time	100 basic cycles		
Maximum UF/BC	82.56%		
Minimum UF/BC	59.03%		
Average UF/sec	66.85%		
Private slots with transmission time deviation <|x| µs	Deviation <|6us|	Deviation <|4us|	Deviation <|2us|
	100%	96.1%	77.7%
Recycled slots	250 recycled / 700 private		

In the Figure 11, the temporal evolution of the UF parameter is shown. In this figure some maximum transmission peaks which correspond with the start of the transmission of data files together with echo data and other TT messages can be distinguished. As expected, in the SCoCANv2 test, UF is higher than obtained in SCoCANv1, since in this latter, the transmission ET messages at the ending of each *private slot* is not allowed.

In both tests, the maximum transmission time deviation (that is, the difference between the instant of transmission of a private message and the real starting time of its slot), produced in messages allocated to *private slots* was always less than 6µs. (This includes the synchronisation uncertainty and clock drifts). Moreover, in more than 70% of the *private slots*, this transmission time deviation is less than 2 µs.

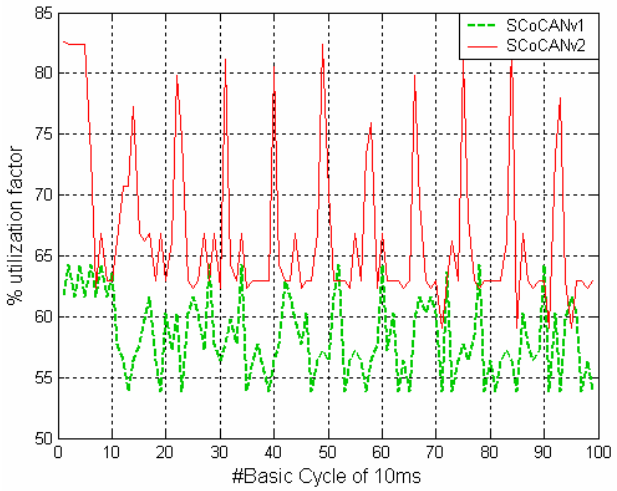


Figure 11. Utilization factor of CAN using SCoCAN protocol in the two tests carried out.

7 Conclusions

In this paper we have discussed the advantages and disadvantages of event-triggered and time-triggered paradigms in fieldbus communication systems as well as of several existing fieldbus protocols. Afterwards, we propose a new protocol, SCoCAN, which is intended as a communication infrastructure for distributed control applications. The main feature of this protocol is that it supports a combination of TT and ET traffic, with temporal isolation between them. And additionally the jitter of messages into the *private slot* is eliminated and the management of network bandwidth and of ET traffic is improved due in part to the dynamic recovery of the *private slots* when these are idle. To appreciate the effect of dynamic BW recovery, a simulation-based comparison between three CAN-based protocols and SCoCAN were carried out with positive results.

The SCoCAN protocol has been implemented and analyzed on the YAIR mobile robot. The analysis of transmitted messages shows a good response that agrees well with our expectations.

References

[1] Almeida, L., Pedreiras, P., Fonseca, J.A., "The FTTCAN protocol: Why and how", IEEE Transactions on Industrial Electronics, Volume 49, Issue 6, pp. 1189-1201. Dec. 2002

[2] Benet, G., Blanes, F., Simó, J.E., Crespo, A., "A Multisensor Robot Architecture with Distributed Behaviour Selection". Proceedings of the 5th IFAC Workshop on Intelligent Manufacturing Systems (IMS'98), Gramado, Brasil, 1998.

[3] Berwanger, J., C. Ebner, et al. (2001). FlexRay--The Communication System for Advanced Automotive Control Systems. SAE World Congress, Detroit, SAE Press paper 2001001-0676. BMW (1999).

[4] CiA (CAN in Automation). "CAN Application Layer for Industrial Applications", Documents No.: DS-201...DS-207, Version 1.1. 1996.

[5] Coronel, J.O, Blanes, F., Pérez, P., Benet,G., Simó, J.E., "Arquitectura de Control distribuida usando nodos empotrados con RT-LINUX sobre el protocolo de comunicaciones SCoCAN", XXV Jornadas de Automática. (Spain). Sep. 2004.

[6] Coronel, J.O, Blanes, F., Benet, G., Pérez, P., Simó, J.E., "CAN-based Distributed Control Architecture using the SCoCAN Communication Protocol", Proc. IEEE Int'l Conf. on Emerging Technologies and Factory Automation, ETFA'2005, vol 1. 2005

[7] Fuhrer, T., Muller, B., Dieterle, W., Hugel, R. "Time Triggerd Communication on CAN". Cia CAN. 7th international CAN Conference 2000.

[8] General Purpose Fieldbus: Vol. 1: P-Net; Vol. 2: PROFIBUS; Vol. 3:WorldFIP, Amend.1: Foundation Fieldbus-H1, European Standard EN50170, 2000.

[9] Heath, Steve, "Embedded Systems Design", Oxford: Newness, 2003.

[10] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. J. ACM, 32(4):841-860, 1985

[11] H. Kopetz, "Should responsive systems be event-triggered or time-triggered?," *IEICE Trans. Inform. Syst.*,vol. E76-D. 1993

[12] H. Kopetz, Real-Time Systems Design Principles for Distributed Embedded Applications Kluwer Academic Publishers, 1997.

[13] J.-P. Thomesse and M. Leon Chavez, "Main paradigms as a basis for current fieldbus concepts," in *Proc. FeT'99 (Int. Conf. Fieldbus Technology)*, Magdeburg, Germany, Sept. 1999.

[14] P. Raja and G. Noubir, "Static and dynamic polling mechanisms for fieldbus networks" ACMOperating Syst. Rev., vol. 27, no. 3, 1993.

[15] Pérez, P., Benet, G., Blanes, F., Simó J. E. "Communications jitter influences on Control loops using Protocols for Distributed Real-Time Systems on CAN bus". 5th IFAC International Symposium. SICICA'03. Aveiro (Portugal). 2003

[16] Pérez, P., Posadas, J.L., Benet, G., Blanes, F., Simó, J.E., "An Intelligent Sensor Architecture for Mobile Robots" 11th International Conference on Advanced Robotics - IEEE ICAR'03. University of Coimbra (Portugal). 2003

[17] Posadas, J.L., Pérez, P., Simó, J.E., Benet, G., Blanes, F., "Communications Structure for Sensory data in mobile robots". Engineering Applications of Artificial Intelligence, vol. 15, no. 3, 341-350. 2002.

[18] Tindell, K., Burns, A. y Wellings, A. J., "Calculating Controller Area Network (CAN) message response time", Control Engineering Practice, Vol. 3(8). 1995.

[19] TTP/C Protocol, version 0.5, TTTech Computertechnik, Vienna, 1999.

Resource and Data Management I

Utility Accrual Real-Time Resource Access Protocols with Assured Individual Activity Timeliness Behavior

Peng Li
Microsoft Corporation
Redmond, WA 98052, USA
pengli@microsoft.com

Binoy Ravindran
ECE Dept., Virginia Tech
Blacksburg, VA 24061, USA
binoy@vt.edu

E. Douglas Jensen
The MITRE Corporation
Bedford, MA 01730, USA
jensen@mitre.org

Abstract

We present a class of utility accrual resource access protocols for real-time embedded systems. The protocols consider application activities that are subject to time/utility function time constraints, and mutual exclusion constraints for concurrently sharing non-CPU resources. We consider the timeliness optimality criteria of probabilistically satisfying individual activity utility lower bounds and maximizing total accrued utility. The protocols allocate CPU bandwidth to satisfy utility lower bounds; activity instances are scheduled to maximize total utility. We establish the conditions under which utility lower bounds are satisfied.

1. Introduction

Many emerging real-time embedded systems such as robotic systems in the space domain (e.g., NASA's Mars Rover [5]) and control systems in the defense domain (e.g., phased array radars [6]) operate in environments with dynamically uncertain properties. These uncertainties include transient and sustained resource overloads (due to context-dependent, activity execution times) and arbitrary, activity arrival patterns. Nevertheless, such systems desire assurances on activity timeliness behavior, whenever possible.

The most distinguishing property of such systems, is that they are subject to "soft" time constraints (besides hard). The time constraints are soft in the sense that completing an activity at any time will result in some (positive or negative) utility to the system, and that utility depends on the activity's completion time. Such soft time-constrained activities are often subject to optimality criteria such as completing all activities as close as possible to their *optimal* completion times—so as to yield maximal collective utility.

Time/utility functions [7] (TUFs) allow the semantics of soft time constraints to be precisely specified. A TUF, which generalizes the deadline constraint, specifies the utility to the system resulting from the completion of an activity as a function of its completion time. A TUF's utility values are derived from application-level QoS metrics. Figures 1(a)–1(b) show some TUF time constraints of two defense applications (see [4] and references therein for application details). Classical deadline is a binary-valued, downward "step" shaped TUF; 1(c) shows examples.

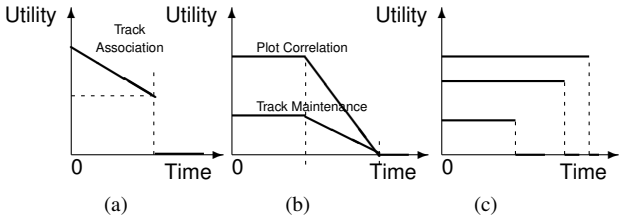


Figure 1. Example TUF Time Constraints. (a): AWACS *association* [4]; (b): Air Defense *correlation* & *maintenance* [4] ; (c): Step TUFs.

When activity time constraints are expressed with TUFs, the timeliness optimality criteria are often based on accrued activity utility, such as maximizing sum of the activities' attained utilities or satisfying lower bounds on activities' maximal utilities. Such criteria are called *Utility Accrual* (or UA) criteria, and scheduling algorithms that consider UA criteria are called UA scheduling algorithms.

UA criteria directly facilitate adaptive behavior during overloads, when (optimally or sub-optimally) completing more important activities, irrespective of activity urgency, is often desirable. UA algorithms that maximize summed utility under downward step TUFs (or deadlines), meet all activity deadlines during under-loads (see algorithms in [9]). When overloads occur, they favor activities that are more important (since more utility can be attained from them), irrespective of urgency. Thus, deadline scheduling's optimal timeliness behavior is a special-case of UA scheduling.

1.1. Contributions

Many embedded real-time systems involve mutually exclusive, concurrent access to shared, non-CPU resources, resulting in contention for the resources. Resolution of the contention directly affects the system's timeliness behavior.

UA algorithms that allow concurrent resource sharing exist (see [9]), but they do not provide any assurances on *individual* activity timeliness behavior—e.g., assured utility lower bounds for each activity. UA algorithms that provide assurances on individual activity timeliness behavior exist [8], but they do not allow concurrent resource sharing. No UA algorithms exist that provide individual activity timeliness assurances under concurrent resource sharing.

We solve this exact problem in this paper. We consider repeatedly occurring application activities that are subject to TUF time constraints. Activities may concurrently, but mutually exclusively, share non-CPU resources. To better account for non-determinism in task execution and inter-arrival times, we stochastically describe those properties. We consider the dual optimality criteria of: (1) probabilistically satisfying lower bounds on each activity's accrued utility, and (2) maximizing total accrued utility, while respecting all mutual exclusion resource constraints.

We present a class of lock-based resource access protocols that optimize this UA criteria. The protocols use the approach in [8] that include off-line CPU bandwidth allocation and run-time scheduling. While bandwidth allocation allocates CPU bandwidth share to tasks, scheduling orders task execution on the CPU. The protocols resolve contention among tasks (at run-time) for accessing shared resources, and bound the time needed for accessing resources.

We present three protocols, which differ in the type of resource sharing that they allow (e.g., direct, nested). We analytically establish upper bounds on the resource access times under the protocols, and establish the conditions for satisfying utility lower bounds.

Thus, the paper's contribution is the class of resource access protocols that we present. We are not aware of any other resource access protocols that solve the UA criteria that are solved by our protocols.

The rest of the paper is organized as follows: Section 2 describes our models. In Section 3, we summarize the bandwidth allocation and scheduling approach in [8] for completeness. Section 4 introduces resource sharing in this approach, and Sections 5, 6, and 7 present the protocols. In Section 8, we show a formal comparison of lock-based versus lock-free resource access protocols. We demonstrate that neither is always better than the other. We conclude in Section 9.

2. Models and Objectives

Tasks and Jobs. We consider the application to consist of a set of tasks, denoted as $\mathbf{T} = \{T_1, T_2, \cdots, T_n\}$. Each instance of a task T_i is called a job, denoted as $J_{i,j}, j \geq 1$. Jobs are assumed to be preemptible at arbitrary times.

We describe task arrivals using the Probabilistic Unimodal Arrival Model (or PUAM) [8]. A PUAM specification is a tuple $\langle p(k), w \rangle, \forall k \geq 0$, where $p(k)$ is the probability of k arrivals during any time interval w. Note that $\sum_{k=0}^{\infty} p(k) = 1$. Poisson distributions $\mathcal{P}(\lambda)$ and Binomial distributions $\mathcal{B}(n, \theta)$ are commonly used arrival distributions. Most traditional arrival models (e.g., frames, periodic, sporadic, unimodal) are PUAM's special cases [8].

We describe task execution times using non-negative random variables—e.g., gamma distributions.

A job's time constraint is specified using a TUF (jobs of a task have the same TUF). A task T_i's TUF is denoted as $U_i(t)$; thus job $J_{i,j}$'s completion at a time t will yield an utility $U_i(t)$. We focus on non-increasing TUFs, as they encompass the majority of time constraints in applications of interest to us (e.g., Figure 1).

Resource Model. Jobs can access non-CPU resources (e.g., disks, NICs, locks), which are serially reusable and are subject to mutual exclusion constraints. Similar to resource access protocols for fixed-priority algorithms [10] and for UA algorithms [9], we consider a single-unit resource model. Thus, only a single instance of a resource is present and a job explicitly specifies the desired resource. The requested time intervals for holding resources may be nested, overlapped or disjoint. Jobs are assumed to explicitly release all granted resources before the end of their execution.

Optimality Criteria. We define a *statistical* timeliness requirement for tasks. For a task T_i, this is expressed as $\langle AU_i, AP_i \rangle$, which means that T_i must accrue at least AU_i percentage of its maximum utility with the probability AP_i. This is also the requirement for each job of T_i. For e.g., if $\{AU_i, AP_i\} = \{0.7, 0.93\}$, then T_i must accrue at least 70% of its maximum utility with a probability no less than 93%. For a task T_i with a step TUF, AU_i is either 0 or 1.

We consider a two-fold optimality criteria: (1) satisfy all $\langle AU_i, AP_i \rangle$, if possible, and (2) maximize the sum of utilities accrued by all tasks. The first criterion is binary in the sense that it is either satisfied or not. The second criterion demands as much accrued utility as possible. Our algorithm first tries to satisfy criterion (1).

3. Bandwidth Allocation and Scheduling

For non-increasing TUFs, satisfying a designated AU_i requires that the task's sojourn time is upper bounded by a "critical time", CT_i. Given a desired utility lower bound

AU_i, $\forall t_1 \leq CT_i, U_i(t_1) \geq AU_i$ and $\forall t_2 > CT_i, U_i(t_2) < AU_i$ holds. To bound task sojourn time by CT_i, we conduct a probabilistic feasibility analysis using the processor demand approach [3]. The key to using the processor demand approach here is allocating a portion of processor bandwidth to each task. We first define *processor bandwidth*:

Definition 3.1. *If a task has a processor bandwidth* ρ*, then it receives at least* ρL *processor time during any time interval of length* L*.*

Once a task is allocated a processor bandwidth, the bandwidth share can be realized and enforced by a *proportional share* (or PS) algorithm (e.g., [11]). A PS algorithm can realize and enforce a desired bandwidth ρ_i for a task T_i with a bounded allocation error, called *maximal lag*, Q, as follows: T_i will receive at least $(\rho_i L - Q)$ processor time during any time interval L. Under a PS scheme, jobs of a task execute on a "virtual CPU" that is not affected by other task behaviors. We focus on bandwidth allocation at an abstract level — using any PS algorithm with a lag Q — hereafter.

Theorem 3.1. *Suppose there are at most* k *arrivals of a task* T *during any time window of length* w *and all jobs of* T *have identical relative critical time* D*. Then, all job critical times can be satisfied if the underlying PS algorithm provides* T *with at least a processor bandwidth of* $\rho = \max\{(C+Q)/D, C/w\}$*, where* C *is the total execution time of* k *jobs released by* T *in a time window of* w*, and* Q *is the maximal lag of the PS algorithm.* □

Proof. Let $C_p(0,L)$ be the processor demand and $S_p(0,L)$ be the available processor time for task T_i on a time interval of $[0,L]$, respectively. The necessary and sufficient condition for satisfying job critical times is:

$$S_p(0,L) \geq C_p(0,L), \forall L > 0 \tag{3.1}$$

Let ρ be the processor bandwidth allocated to T. Thus, $S_p(0,L) = \rho L - Q$. Further, the total amount of processor time demand on $[0,L]$ is $C_p(0,L) = \left(\left\lfloor (L-D)/w \right\rfloor + 1\right) C$. Therefore, Equation 3.1 can be rewritten as:

$$\rho L - Q \geq \left(\left\lfloor (L-D)/w \right\rfloor + 1\right) C, \forall L > 0 \tag{3.2}$$

Since $\left((L-D)/w + 1\right) \geq \left(\lfloor \frac{L-D}{w} \rfloor + 1\right)$, it is sufficient to have $\rho L - Q \geq \left(\frac{L-D}{w} + 1\right) C, \forall L > 0$ so that Equation 3.2 is satisfied. This leads to:

$$\rho \geq \frac{C}{w} + \frac{1}{L}\left(C + Q - C\frac{D}{w}\right), \forall L > 0 \tag{3.3}$$

It is easy to see that ρ is a monotone of L. For a positive $C + Q - C\frac{D}{w}$, the maximal ρ occurs when $L = D$, which yields $\rho = (C+Q)/D$. For a negative $C + Q - C\frac{D}{w}$, the maximal ρ occurs when $L = \infty$. Combining these two cases, the theorem follows. □

For simplicity, we only consider the case $\rho \geq (C+Q)/D$, which implies $D < w$. Note that critical sections in a PS algorithm can be handled by setting Q as the longest critical section of all tasks. Let N_i be the random variable for the number of arrivals during a time window w_i. Then, the processor demand of task T_i during a time window w_i is $C_i = \sum_{j=1}^{N_i} c_{i,j}$, where $c_{i,j}$ is the execution time of job $J_{i,j}$. By Theorem 3.1, $\rho_i \geq (C_i + Q)/CT_i$, where CT_i is T_i's critical time. To satisfy the assurance probability, we require:

$$\Pr\left[\sum_{j=1}^{N_i} c_{i,j} \leq \rho_i CT_i - Q\right] \geq AP_i \tag{3.4}$$

The above condition is the fundamental bandwidth requirement for satisfying a task's critical time. If $N_i = k$, the total processor time demand during a time window becomes $\sum_{j=1}^{k} c_{i,j}$. Therefore, Equation 3.4 can be rewritten as a sum of conditional probabilities:

$$\sum_{k=0}^{\infty}\left(p_i(k) \times \Pr\left[\sum_{j=1}^{k} c_{i,j} \leq \rho_i CT_i - Q\right]\right) \geq AP_i \tag{3.5}$$

3.1. Bandwidth Solutions

Equation 3.4 can be rewritten as:

$$1 - \Pr\left[C_i \geq \rho_i CT_i - Q\right] \geq AP_i \tag{3.6}$$

By Markov's Inequality, $\Pr[X \geq t] \leq E(X)/t$ for any non-negative random variable. Therefore,$1 - \Pr[C_i \geq \rho_i CT_i - Q] \geq 1 - E(C_i)/(\rho_i CT_i - Q)$. If we can determine a ρ_i so that $1 - E(C_i)/(\rho_i CT_i - Q) \geq AP_i$, $\Pr[C_i \leq \rho_i CT_i - Q] \geq AP_i$ is also satisfied. This becomes:

$$\rho_i \geq \frac{E(C_i)}{CT_i(1 - AP_i)} + \frac{Q}{CT_i} \tag{3.7}$$

Note that N_i in Equation 3.4 is a random variable and follows a distribution specified by $p_i(a)$. By Wald's Equation, $E(C_i) = E\left(\sum_{j=1}^{N_i} c_{i,j}\right) = E(c_i)E(N_i)$. Thus,

$$\rho_i \geq \frac{E(c_i)E(N_i)}{CT_i(1 - AP_i)} + \frac{Q}{CT_i} \tag{3.8}$$

This solution is applicable for any distributions of c_i and N_i, and only requires the average number of arrivals and the average execution time.

With minimal assumption regarding task arrivals and execution times, the solution given by Equation 3.8 may be pessimistic for some distributions. Thus, an algorithm that demands and utilizes the information of *full distributions* for task arrivals and execution times is also presented in [8].

For job scheduling, [8] presents a scheduling algorithm called `UJSsched` that uses the Highest Utility Density

First heuristic. UJSsched has the property that if all job critical times can be satisfied by EDF, then UJSsched is also able to do so and accrues at least the same utility as EDF does. Further, if not all job critical times can be satisfied, then UJSsched accrues as much utility as possible.

4. Resource Sharing With Locks

Proportional share uses large time quanta to ensure mutual exclusion. This works well for short critical sections. However, we conjecture that for some cases, a small time quantum combined with lock-based, resource access protocols may yield lower bandwidth requirement. When time quanta are smaller than the length of critical sections, preemptions of a task while it is inside a critical section may happen. Thus, we use locks to ensure mutual exclusion. With locks, three types of blocking can occur:

Direct Blocking. If a job $J_{i,m}$ requests a resource R that is currently held by another job $J_{j,k}$, we say that job $J_{i,m}$ is *directly blocked* by job $J_{j,k}$. Job $J_{j,k}$ is called the blocking job. Because processor bandwidth is allocated on a per task basis, we also say that task T_i is blocked by task T_j.

Transitive Blocking. If a job J_a is blocked by job J_b which in turn is blocked by job J_c, we say that job J_a is *transitively blocked* by J_c.

Queue Blocking. Let a set of tasks $\mathcal{TB} = \{T_{b1}, T_{b2}, \cdots, T_{bk}\}$ be simultaneously blocked on a resource R, held by task T_o. When T_o releases R, one of the blocked tasks, e.g., task T_{bm}, will acquire R and continue execution. Thus, another task T_{bn} will suffer additional blocking due to T_{bm}, besides the blocking due to T_o. We call such an additional blocking *queue blocking*, as it is caused by a queue of blocked tasks. This definition can be expanded to the case of multiple tasks in $\mathcal{TB}$ being granted R before T_{bn}.

The objective of resource access protocols is to effectively bound or reduce task blocking times. We present three protocols, called the Bandwidth Inheritance Protocol (BIP), Resource Level Policy (RLP) and the Early Blocking Protocol (EBP). BIP speeds up the execution of a blocking task and thus reduces direct blocking times. It is inspired by the Priority Inheritance Protocol (PIP) [10] in priority scheduling. RLP bounds the queue blocking time suffered by a task. However, BIP and RLP allows transitive blocking and deadlocks. EBP avoids deadlocks and bounds transitive blocking times.

Recall that UJSsched [8] is used to resolve competition among jobs of the same task. Thus, resource blocking can occur among jobs, which complicates the analysis of the job scheduling algorithm. Note that assurance requirements are at the task level. Thus, we simply disallow preemptions while a job holds a resource. From the perspective of the virtual processor, UJSsched is invoked when a new job arrives and when the currently executing job completes.

Transitive blocking and deadlocks can occur only in the presence of nested critical sections; Lemma 4.1 states this observation. Thus, BIP and RLP disallow nested sections.

Lemma 4.1. *Transitive blocking can occur only in the presence of nested critical sections. That is, if a job J_a is transitively blocked by another job J_c, there must be a job J_b that is currently inside a nested critical section.* □

Proof. By the definition of transitive blocking, there exists a job J_b that blocks J_a and is blocked by J_c. Since J_a is blocked by J_b, J_b must hold a resource, e.g., R_1. Further, the fact that J_b is blocked by J_c implies that J_b requests another resource, e.g., R_2, which is currently held by J_c. Thus, J_b must be inside a nested critical section. □

Besides the property of no transitive blocking, lack of nested critical sections also prevents deadlocks, since *hold-and-wait* — a necessary condition for deadlocks — is disallowed. We now introduce a few notations and assumptions:

- $z_{i,j}$: j^{th} critical section of task T_i;
- $d_{i,j}$: duration of critical section $z_{i,j}$ on a dedicated processor without processor contention;
- $R_{i,j}$: resource associated with critical section $z_{i,j}$;
- d_i^j: duration of task T_i's critical section that accesses resource R_i;
- $z_{i,k} \subset z_{i,m}$: $z_{i,k}$ is entirely contained in $z_{i,m}$;
- All critical sections are "properly" nested, i.e., for any pair of $z_{i,k}$ and $z_{i,m}$, either $z_{i,k} \subset z_{i,m}$, or $z_{i,m} \subset z_{i,k}$, or $z_{i,k} \bigcap z_{i,m} = \emptyset$;
- All critical sections are guarded by binary semaphores.

5. Bandwidth Inheritance Protocol

BIP's key idea is to speed up the execution time of a blocking task T, by transferring all bandwidth of tasks that are blocked by T. Thus, the blocked tasks loose their bandwidth and become stalled. We define BIP as a set of rules:

1. If a task T_i is blocked on a resource R that is currently held by a task T_j, the processor bandwidth of task T_i is inherited by task T_j. That is, the processor bandwidth of task T_j is temporarily increased to $\rho_i + \rho_j$ until T_j releases resource R. In the meanwhile, the bandwidth of task T_i becomes zero. Thus, T_i is stalled even if some jobs of T_i are eligible for execution.
2. Bandwidth inheritance is transitive. That is, if a task T_a is blocked by T_b which in turn is blocked by task T_c, then the bandwidth of T_a is also transferred to T_c.
3. Bandwidth inheritance is additive. Suppose a task T_a holds a resource R, and a set of tasks $\mathcal{TB} = \{T_i, \forall i = 1, ..., k\}$ are all blocked on R. Then, the bandwidth of T_a is increased to $\rho_a + \sum_{i=1}^{k} \rho_i$.

BIP's three rules indicate how the bandwidth of blocked tasks can be transferred to the blocking task for the three

types of blocking. By doing so, we reduce the duration of the blocking task's critical section. Task bandwidth can be transferred through dynamic task *join* and *leave* operations — EEVDF [11] allows this while maintaining a constant lag.

5.1. Blocking Time under BIP

We now upper bound a blocking task's duration of critical section. Assume that the blocking task has a total bandwidth of ρ, possibly through bandwidth inheritance. Then, the duration of the critical section is d_i/ρ. Therefore, the key to bound the duration is to lower bound the processor bandwidth allocated to a blocking task. An arbitrarily small bandwidth essentially yields an unbounded blocking time.

Section 3 presented methods to determine the minimal bandwidth needed to satisfy task utility bounds, without resource blocking. We now establish the relationship between the bandwidth requirements with and without blocking.

Theorem 5.1. *In Theorem 3.1's task model, if a task is blocked on resource access, the minimal required bandwidth is* $\rho = (B+C+Q)/D$*, where* B *is the total blocking time of jobs of the task during a time window* W*.* □

Proof. The proof is similar to that of Theorem 3.1 [8]. To satisfy job critical times, the available processor time during any time interval $[0,L]$, excluding the blocking time, should be greater than or equal to job processor demand:

$$S_p(0,L) - Q - \left(\left\lfloor \frac{L-D}{W} \right\rfloor + 1\right) B \geq \left(\left\lfloor \frac{L-D}{W} \right\rfloor + 1\right) C, \forall L > 0 \tag{5.1}$$

This leads to:

$$\rho L \geq \left(\left\lfloor \frac{L-D}{W} \right\rfloor + 1\right)(B+C) - Q, \forall L > 0 \tag{5.2}$$

By the same argument as in the proof of Theorem 3.1, we have $\rho \geq (B+C+Q)/D$. □

Thus, if $\rho_i^{min} = (C_i+Q)/D_i$ is T_i's processor bandwidth by assuming no resource blocking, it is safe to use ρ_i^{min} as the lower bound on T_i's bandwidth even in the presence of resource blocking. Also, observe that if T_i is a blocking task, it must inherit the bandwidth of at least one blocked task. Let $\mathcal{TR}$ be the set of tasks that may be blocked by T_i. T_i's total bandwidth while it is inside the critical section (of using resource R) is at least $\rho_i^{min} + \min\{\rho_j^{min} | j \neq i \bigwedge T_j \in \mathcal{TR}\}$. The direct blocking time caused by T_i is upper bounded by $(d_i + Q)\Big/\left(\rho_i^{min} + \min\{\rho_j^{min} | j \neq i, T_j \in \mathcal{TR}\}\right)$, where d_i is the duration of T_i's critical section for R. This blocking time calculation is repeated for all critical sections of a task, and for all jobs of a task in a time window.

5.2. Bandwidth Allocation under BIP

Let each task T_i access n_i resources, denoted $R_{i,j}, j = 1,\ldots,n_i$. Let $d_{R_{i,j}}$ denote the maximal length of the critical section for accessing resource $R_{i,j}$, and $\rho_{R_{i,j}}^{min}$ denote the smallest ρ^{min} among all tasks that may access $R_{i,j}$. T_i's direct blocking time for accessing $R_{i,j}$ is $B_{R_{i,j}} = d_{R_{i,j}}\Big/\left(\rho_{R_{i,j}}^{min} + \rho_i^{min}\right)$. A job of T_i's direct blocking time is:

$$B_D = \sum_{j=1}^{n_i} B_{R_{i,j}} = \sum_{j=1}^{n_i} \frac{d_{R_{i,j}} + Q}{\rho_{R_{i,j}}^{min} + \rho_i^{min}}, \tag{5.3}$$

where n_i is the number of critical sections of T_i. By Theorem 5.1, we require that the probability of satisfying task critical time is at least AP_i. This leads to:

$$\begin{aligned} &\sum_{k=0}^{\infty} p_i(k) \Pr[B + C + Q \leq \rho_i C T_i] \geq AP_i \Rightarrow \\ &\sum_{k=0}^{\infty} p_i(k) \Pr\left[k \sum_{j=1}^{n_i} \frac{d_{R_{i,j}}+Q}{\rho_{R_{i,j}}^{min} + \rho_i^{min}} + \sum_{j=1}^{k} c_{i,j} + Q \leq \rho_i C T_i\right] \geq AP_i \end{aligned} \tag{5.4}$$

For all tasks, we first calculate the minimal bandwidth requirements without resource blocking, i.e., ρ_i^{min}, using the techniques in Section 3. The direct blocking time for each job of T_i, namely B_D is then calculated. Observe that the net effect of resource blocking is an increase in task execution time. In the case of direct blocking, the execution time of a job is increased by B_D, which has been calculated. Once the blocking time is calculated, the bandwidth requirement under BIP can be computed from Equation 5.4. Solutions in Section 3 can be applied to solve Equation 5.4 for ρ_i.

6. Resource Level Policy

RLP's idea is to associate a static numerical value with each task, called a task's Resource Level (or RL). A task's RL is static in the sense that it is assigned when the task is created, is maintained intact during the task's life time, and is the same for all jobs of the task. By using static RLs, we aim to produce a predictable order for accessing a shared resource, in case a queue of tasks are blocked on the same resource. Thus, queue blocking times can be bounded.

If there are n tasks in a system, the RLs of tasks are integers from 1 to n. We assume that a larger numeric value means higher RL. There are different ways for assigning static RLs. In general, static RLs must be assigned reflecting our objective of maximizing summed utility. Here, we propose several alternatives for assigning static RLs:

(1) **Maximal Height of TUF.** For any pair of tasks, if $maxU_i > maxU_j$, then $RL_i > RL_j$. $maxU_i$ is the maximal height of a TUF, i.e., $maxU = \{U_i(t) | I_i \leq t \leq X_i\}$. I_i and X_i are the first and last time instances on which

$U_i(t)$ is defined. The approach is easy to implement and works well for step TUFs. However, it ignores task execution time information. Further, for non-step TUFs, the maximal TUF height may be much higher than task accrued utility.

(2) **Pseudo Slope.** For a task T_i, this is defined as: $pSlope_i = U_i(I_i)/(X_i - I_i)$. Pseudo Slope seeks to capture a TUF's shape, but it ignores task execution times.

(3) **Pseudo Utility Density.** For a task T_i, this measures the utility that can be accrued, by average, per unit execution time: $pUD_i = U_i(\rho_i^{min} E(c_i)) \Big/ \rho_i^{min} E(c_i)$.

Using static RLs, the task with the highest RL will be granted a resource R if there is a queue of tasks blocked on R. Thus, when calculating the queue blocking time for task T_i, we only need to consider tasks with RLs higher than that of T_i—e.g., if $RL_i = i$, then T_i only suffers queue blocking due to tasks $T_j, j = i+1, ..., n$.

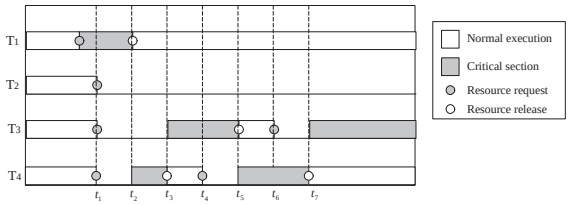


Figure 2. An Example of Using Static Resource Levels

Unfortunately, this scheme of using static RLs may yield unbounded queue blocking times for low RL tasks. Figure 2 shows an example. In Figure 2, task T_2 is blocked on a resource request and is later starved.

To overcome the difficulty with static RLs, we introduce the concept of Effective Resource Level (or ERL). Besides RL, each task is associated with an ERL, which may increase over time. The idea is to use ERL to prevent a few high RL tasks from dominating the usage of shared resources. With ERLs, RLP works as follows:

1. If a task is not blocked on any resource, its ERL is the same as its static RL.
2. Whenever a resource R is released, the ERL's of all tasks that are currently blocked on R are increased by n, where n is the number of tasks in the system.
3. When a resource R becomes free, one of the blocked tasks with the highest ERL is granted resource access. If a tie among the highest ERL tasks occurs, the task with the longest blocking time wins.
4. When a task acquires the resource on which it was blocked, its ERL returns to its static RL.

Theorem 6.1. *Under RLP, a task T_k can be queue blocked on a resource R for at most $(m-2)$ critical sections, where m is the number of tasks that may access R.* □

Proof. Consider a set of tasks $\mathcal{TB}$, including task T_k, that are blocked on a resource R. Obviously, $|\mathcal{TB}| \leq m-1$, because one task must be holding the resource. At time instant t_0, let R be released by the current blocking task. Thus T_k's ERL is increased to $RL_k + n$, which is higher than $RL_i, \forall i$. This high ERL effectively ensures that no tasks that are blocked on R after t_0 can queue block T_k. Therefore, T_k can only suffer additional queue blocking from existing blocked tasks, which are at most $(m-3)$ critical sections. Note that at t_0, one of the tasks from $\mathcal{TB}$ namely task T_r, is granted resource R. Therefore, the number of the remaining blocked tasks, excluding T_k, is $|\mathcal{TB} - T_k| - 1 \leq (m-3)$. The theorem follows by summing up queue blocking times before and after instant t_0, i.e., $1 + (m-3) = (m-2)$. □

Theorem6.1 leads to the following corollary:

Corollary 6.2. *The ERL of a task T_i is within the range of $[RL_i, (m-1)n + RL_i]$, where m is defined in Theorem 6.1 and n is the number of tasks in the system.* □

Proof. By Theorem 6.1, a task can suffer a queue blocking time of at most $(m-2)$ critical sections. In addition, it suffers one direct blocking. Upon releasing a shared resource, these blocking tasks increase the ERL of a task $(m-2)+1 = m-1$ times. Since each increase is n, the ERL of T_i is bounded by $(m-1)n + RL_i$. □

Theorem 6.3. *Let $\mathcal{T}_R$ be the set of tasks that may access resource R. Theorem 6.1's queue blocking time bound is tight for any $T_i \in \mathcal{T}_R$, except the highest RL task in $\mathcal{T}_R$.* □

Proof. Without loss of generality, let $\mathcal{T}_R = \{T_1, T_2, ..., T_m\}$ and $RL_i = i$. We prove this theorem by showing that there *always* exists a resource access pattern so that any task $T_i \in \mathcal{T}_R, i < m$ suffers a queue blocking time of $(m-2)$ critical sections. The resource access pattern can be constructed as follows: Let t_i be a time stamp and satisfies $t_{i+1} > t_i$. Now:

- t_0: Task T_{i+1} is holding resource R and tasks $\mathcal{TB} = \{T_k | T_k \in \mathcal{T}_R, k \neq i \bigwedge k \neq i+1\}$ are blocked on R. $|\mathcal{TB}| = (m-2)$.
- t_1: Task T_{i+1} releases R. A task in $\mathcal{TB}$, say T_r is granted resource R. ERL's of remaining tasks in $\mathcal{TB}$ are increased by n.
- t_2: Task T_{i+1} requests R and is blocked on R.
- t_3: Task T_i requests R and is blocked on R.

Now, at time t_3, the ERL of task T_i is lower than those of all other tasks in the blocked task queue, which includes $(m-2)$ tasks. Therefore, T_i will suffer a queue blocking time of $(m-2)$ critical sections. □

We now revisit the example in Figure 2. In Figure 3, we show the behavior of tasks by using the dynamic resource level adjustment rules. Note that the numbers on each timeline of a task indicates the ERL of that task. In this case,

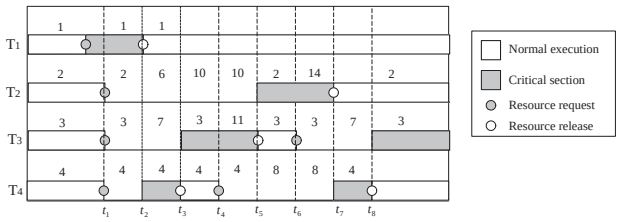


Figure 3. Dynamic Resource Levels

$m = 4$. Thus, task queue blocking times should be bounded by $m-2 = 2$ critical sections, which is consistent with Figure 3. Observe that task T_2 is queue blocked for exactly two critical sections (of T_3 and T_4, respectively). On the other hand, task T_3 suffers one critical section of queue blocking for its resource requests; task T_4 only incurs one critical section of queue blocking during its second resource request.

We consider a task T_b, along with a queue of k tasks, that are blocked by a task T_a. Figure 4 shows this scenario.

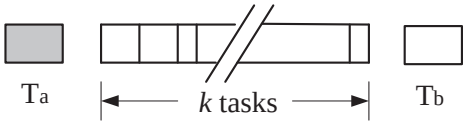


Figure 4. An Example of Queueing Blocking

To determine T_b's queue blocking time, we examine the blocking time due to each task in the $k-task$ queue. Observe that the qi^{th} task in the $k-task$ queue executes with a CPU bandwidth of at least $\rho_{qi}^{min} + \left(\sum_{j=i+1}^{k} \rho_{qj}^{min}\right) + \rho_b^{min} = \left(\sum_{j=i}^{k} \rho_{qj}^{min}\right) + \rho_b^{min}$ due to bandwidth inheritance. Thus, the total queue blocking time resulting from the k tasks is:

$$B_Q[k] = \sum_{i=1}^{k} \frac{d_{qi}+Q}{\left(\sum_{j=i}^{k} \rho_{qj}^{min}\right) + \rho_b^{min}} \tag{6.1}$$

Let $d_q = \max\{d_{qi} | i = 1,...,m-2\}$ and $\rho_q^{min} = \min\{\rho_{qj}^{min} | j = 1,...,m-2\}$. Then, $B_Q[k]$ is bounded by:

$$B_Q^m[k] = \sum_{i=1}^{k} \frac{d_q+Q}{\left(\sum_{j=i}^{k} \rho_q^{min}\right) + \rho_b^{min}}$$

$$= \sum_{i=1}^{k} \frac{d_q+Q}{(k-i+1)\rho_q^{min} + \rho_b^{min}} = \sum_{i=1}^{k} \frac{d_q+Q}{i\rho_q^{min} + \rho_b^{min}} \tag{6.2}$$

We need to determine a k such that $B_Q^m[k]$ achieves its maximal value and thus bounds T_b's queue blocking time. We show that the maximal queue blocking time occurs with maximal number of tasks in the queue, i.e., $k = (m-2)$.

Lemma 6.4. *The $B_Q^m[k]$ function defined in Equation 6.2 monotonically increases with k.* □

Proof. We define two auxiliary functions $B_Q^-[k]$ and $B_Q^+[k]$. $B_Q^-[k]$ is the amount of blocking time that may be reduced if a $(k+1)^{th}$ blocked task is added into the existing $k-task$ queue. $B_Q^+[k]$ is the additional queue blocking time due to the $(k+1)^{th}$ blocked task. That is, $B_Q^-[k] = \sum_{i=1}^{k} \frac{d_q+Q}{i\rho_q^{\min}+\rho_b^{\min}} - \sum_{i=1}^{k} \frac{d_q+Q}{(i+1)\rho_q^{\min}+\rho_b^{\min}}$ and $B_Q^+[k] = \frac{d_q+Q}{\rho_q^{\min}+\rho_b^{\min}} = B_Q^+$.

Now, the relationship between $B_Q^m[k+1]$ and $B_Q^m[k]$ can be derived as: $B_Q^m[k+1] = B_Q^m[k] + B_Q^+ - B_Q^-[k]$. It follows that: $B_Q^-(k)/(d_q+Q) = \sum_{i=1}^{k} \frac{1}{i\rho_q^{\min}+\rho_b^{\min}} - \sum_{i=1}^{k} \frac{1}{(i+1)\rho_q^{\min}+\rho_b^{\min}}$

$$= \sum_{i=1}^{k} \left(\frac{1}{i\rho_q^{\min}+\rho_b^{\min}} - \frac{1}{(i+1)\rho_q^{\min}+\rho_b^{\min}} \right)$$
$$= \frac{1}{\rho_q^{\min}+\rho_b^{\min}} - \frac{1}{2\rho_q^{\min}+\rho_b^{\min}} + \frac{1}{2\rho_q^{\min}+\rho_b^{\min}} - \frac{1}{3\rho_q^{\min}+\rho_b^{\min}} + \cdots + \frac{1}{k\rho_q^{\min}+\rho_b^{\min}} - \frac{1}{(k+1)\rho_q^{\min}+\rho_b^{\min}}$$
$$= \frac{1}{\rho_q^{\min}+\rho_b^{\min}} - \frac{1}{(k+1)\rho_q^{\min}+\rho_b^{\min}}$$
$$= \frac{1}{\rho_q^{\min}+\rho_b^{\min}} \frac{k\rho_q^{\min}}{(k+1)\rho_q^{\min}+\rho_b^{\min}}$$
$$= \frac{1}{\rho_q^{\min}+\rho_b^{\min}} \frac{k\rho_q^{\min}}{k\rho_q^{\min}+\rho_q^{\min}+\rho_b^{\min}} < \frac{1}{\rho_q^{\min}+\rho_b^{\min}}$$
$$= B_Q^+/(d_q+Q)$$

Therefore, $B_Q^m[k+1] = B_Q^m[k] + B_Q^+ - B_Q^-[k] > B_Q^m[k]$. □

By Lemma 6.4, a task T_i's queue blocking time is $B_Q = \sum_{j=1}^{n_i} B_{Q_j}^m[m_j-2]$, where $B_{Q_j}^m[m_j-2]$ is the maximal queue blocking time for accessing resource $R_{i,j}$. Now,

$$B_{Q_j}^m[m_j-2] = \sum_{l=1}^{m_j-2} \left((d_{qj}+Q) \Big/ \left(l\rho_{qj}^{min} + \rho_i^{min} \right) \right) \tag{6.3}$$

Using a technique similar to that in Equation 5.4, the bandwidth requirement under RLP is:

$$\sum_{k=0}^{\infty} p_i(k) \Pr[B_D + B_Q + C + Q \le \rho_i CT_i] \ge AP_i$$
$$\Rightarrow \sum_{k=0}^{\infty} p_i(k) \Pr\left[k \sum_{j=1}^{n_i} \frac{d_{R_{i,j}}+Q}{\rho_{R_{i,j}}^{min}+\rho_i^{min}} + k\sum_{j=1}^{n_i} B_{Q_j}^m(m_j-2) + \sum_{j=1}^{k} c_{i,j} + Q \le \rho_i CT_i\right] \ge AP_i \tag{6.4}$$

7. The Early Blocking Protocol

We design EBP to deal with nested critical sections. Nested sections may create deadlocks and transitive blocking. EBP's basic idea is to block an "unsafe" resource request even if the requested resource is free. An unsafe resource request is one that may cause deadlocks. Meanwhile, a safe request is granted. [2, 10] uses a similar scheme.

Let a task T invoke *nest_req_res*(R', RV) to enter a nested critical section. In their order of access, RV, called a "resource vector," is a list of resources that T may access while it is inside nested critical sections. R' is RV's first element.

For single-unit resources, a deadlock occurs if and only if there is a cycle in the resource graph. A cycle can only be formed by at least two tasks inside nested critical sections. Further, there must be at least one resource R that is requested by one task T_i and which is held by another task T_j, both of which are inside nested critical sections—i.e., the resource vectors of T_i and T_j overlap. Thus, EBP compares the resource vector of a requesting task with those of the existing tasks. If any resource vectors overlap, there is a deadlock possibility, and the requesting task is blocked.

We formulate EBP as follows: Let a task T invoke *nest_req_res*(R', RV).

1. If R' is held by another task, then T is blocked.
2. If R' is free, then *nest_req_res*(R', RV) may or may not be granted, per the following:
   (a) Let $\mathcal{T}_{nest}$ be the set of tasks that are currently inside nested sections. For any task $T_i \in \mathcal{T}_{nest}$, let RV_i be T_i's current resource vector.
   (b) If for any task $T_i \in \mathcal{T}_{nest}$, $RV \bigcap RV_i = \emptyset$, then *nest_req_res*$(R', RV)$ is granted; the request is blocked otherwise.
3. When a task exits a nested critical section, RLP checks if granting any pending *nest_req_res*(R', RV) is safe. If more than one pending *nest_req_res*(R', RV) is safe, then RLP is invoked.

We now establish that EBP is deadlock-free and can bound transitive blocking times.

Lemma 7.1. *Under EBP, for any pair of tasks that are currently inside nested critical sections, their resource vectors do not have common elements.* □

Proof. Let tasks T_1 and T_2 enter nested critical sections at instants $t_1 < t_2$, respectively. If $RV_1 \bigcap RV_2 \neq \emptyset$, then T_2 cannot enter its nested section. Thus, the resource vectors of T_1 and T_2 do not have common elements. □

Lemma 7.1 leads to Theorem 7.2 and Corollary 7.3:

Theorem 7.2. *EBP avoids deadlock.* □

Corollary 7.3. *Under EBP, if a task T_1 is blocked by a task T_2 while T_1 is inside nested critical sections, then T_2 is not inside nested critical sections.* □

Proof. Suppose T_2 is inside nested critical sections. If T_1 is blocked by T_2, then T_1 needs a resource R that is currently held by T_2. Thus, R is a common element in T_1 and T_2's resource vectors. This violates Lemma 7.1. □

Theorem 7.4. *Under EBP, a chain of transitive blocking includes three tasks.* □

Proof. We use $T_i \rightarrow R_i$ to denote that task T_i needs resource R_i. Similarly, $R_i \rightarrow T_i$ means that resource R_i is currently held by task T_i. Thus, a chain of transitive blocking has the form $T_1 \rightarrow R_1 \rightarrow T_2 \rightarrow R_2 \rightarrow T_3 \rightarrow \ldots \rightarrow T_n$. Since there is a chain of transitive blocking, $n \geq 3$. It is easy to see that any task $T_i, i \neq 1 \bigwedge i \neq n$ must be inside nested critical sections. By Corollary 7.3, if T_2 is inside nested critical sections, T_3 cannot be inside nested critical sections. Therefore, T_3 must be at the end of the chain. Thus, $n = 3$. □

Theorem 7.5. *Let a task T requests resource R_i. Let $\mathcal{T}_{i,j}$ be the set of tasks that have a resource vector $RV = \{\ldots, R_i, \ldots, R_j, \ldots\}$ and let $\mathcal{T}_j$ be the set of tasks that may access resource R_j. T's transitive blocking time for R_i is bounded by $(d_{max} + Q) \Big/ (\rho^{min} + \rho^{min}_{R_{i,j}} + \rho^{min}_{R_j})$. ρ^{min} is T's minimal bandwidth, $d_{max} = \max\{d^j_k | T_k \in \mathcal{T}_j\}$, $\rho^{min}_{R_{i,j}} = \min\{\rho^{min}_k | T_k \in \mathcal{T}_{i,j}\}$, and $\rho^{min}_{R_j} = \min\{\rho^{min}_k | T_k \in \mathcal{T}_j\}$.* □

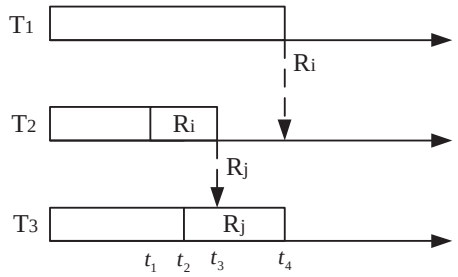


Figure 5. Illustration of Transitive Blocking

Proof. Consider a chain of transitive blocking as in Figure 5. Task T_1 is transitively blocked by task T_3 when it requests resource R_i. By Theorem 7.4, the scenario illustrated in Figure 5 is the only possible scenario.

Further, task T_3 has a bandwidth of at least $\rho^{min}_1 + \rho^{min}_2 + \rho^{min}_3$ due to bandwidth inheritance. We consider the worst case where the most pessimistic bounds are assumed. That is, $\rho^{min}_2 = \rho^{min}_{R_{i,j}} = \min\{\rho^{min}_k | T_k \in \mathcal{T}_{i,j}\}$ and $\rho^{min}_3 = \rho^{min}_{R_j} = \min\{\rho^{min}_k | T_k \in \mathcal{T}_j\}$. The theorem follows. □

8. Lock-Based versus Lock-Free

As discussed earlier, our conjecture is that for some cases, our lock-based, resource access protocols may work well. For other cases, the lock-free scheme—i.e., setting quantum size as the longest critical section in the system [1], may perform better. We now explore the conditions under which resource access protocols may be beneficial, and the reverse conditions as well.

The discussion focuses on two aspects: (1) bandwidth requirement for a given task; and (2) feasibility of a task set. Given a set of n tasks and their allocated bandwidth,

if $\sum_{i=1}^{n} \rho_i \leq 1$, we say that the task set is feasible for the particular bandwidth allocation. Otherwise, the task set is said infeasible for the particular allocation.

We first introduce some notations:

- ρ_i^p: bandwidth requirement of task T_i under lock-based resource access protocols;
- ρ_i^{np}: bandwidth requirement of task T_i under the lock-free scheme (also called *non-preemptive* scheme as there will be at most one preemption while a task tries to access a resource [1]);
- Q_p: quantum size under the lock-based resource access protocols
- Q_{np}: quantum size under the lock-free scheme.

Lemma 8.1. *Suppose Q_{np} equals to the length of a critical section of task T_m (accessing resource R_m). If a task T_i may be blocked on R_m, then $\rho_i^p > \rho_i^{np}$.*

Proof. Let $d_R = Q_{np}$ be the length of the critical section. If task T_i may be blocked on R, it suffers at least one direct blocking due to access to R. The direct blocking time is calculated as:

$$B_D = k \sum_{j=1}^{n_i} \frac{d_{R_{i,j}} + Q_p}{\rho_{R_{i,j}}^{min} + \rho_i^{min}} \geq \frac{d_R + Q_p}{\rho_R^{min} + \rho_i^{min}} \geq d_R + Q_p > d_R \tag{8.1}$$

The total blocking time is $B = B_D + B_Q + B_T \geq B_D > d_R$. Given the total execution time of C during a time window, we have:

$$B + C + Q_P > d_R + C + Q_p = Q_{np} + C + Q_p > Q_{np} + C \tag{8.2}$$

Recall that the fundamental bandwidth requirement under resource access protocols is:

$$\sum_{k=0}^{\infty} p_i(k) \Pr\left[B_k + C_k + Q_p \leq \rho_i^p CT_i\right] \geq AP_i \tag{8.3}$$

and under the lock-free scheme is:

$$\sum_{k=0}^{\infty} p_i(k) \Pr\left[C_k + Q_{np} \leq \rho_i^{np} CT_i\right] \geq AP_i \tag{8.4}$$

where C_k is the sum of k job execution times, B_k is the total blocking time of k jobs. Since $C_k + Q_{np} < B_k + C_k + Q_p, \forall k$, $\rho_i^{np} < \rho_i^p$. □

Lemma 8.2. *Suppose Q_{np} equals to the length of a critical section of task T_m (accessing resource R_m). If a task T_i may not be blocked on R_m, then ρ_i^p can be smaller than ρ_i^{np}.*

Proof. We prove this lemma by considering an extreme case where resource R_m is only accessed by task T_m and another task T_k. All other tasks in the system do not use any shared resources. For any task that does not use any shared resource, its blocking time is zero. Further, Q_p can be smaller than Q_{np}. Therefore,

$$B + C + Q_p = C + Q_p < C + Q_{np} \tag{8.5}$$

If that is the case, ρ_i^p is smaller than ρ_i^{np}. □

Theorem 8.3. *If a task set is feasible under the lock-free scheme, it can be infeasible under resource access protocols, and vice versa.*

Proof. We prove this theorem by examples.

1. A task set is feasible under the lock-free scheme, but infeasible using resource access protocols.

Suppose all tasks access a single resource R in a system. By Lemma 8.1, $\rho_i^{np} < \rho_i^p, \forall i = 1, ..., n$. Thus,

$$\sum_{i=1}^{n} \rho_i^{np} < \sum_{i=1}^{n} \rho_i^p \tag{8.6}$$

Also assume $\sum_{i=1}^{n} \rho_i^{np} = 1$ for this particular task set. Then, $\sum_{i=1}^{n} \rho_i^p > 1$, and hence the task set if infeasible under resource access protocols.

2. A task set is feasible under resource access protocols, but infeasible under the lock-free scheme.

Consider a system where only two tasks, T_1 and T_2 need to access a resource R. Other tasks do not need to access any shared resources. Let:

$$\begin{aligned} U_p &= \sum_{i=1}^{n} \rho_i^p = (\rho_1^p + \rho_2^p) + \sum_{i=3}^{n} \rho_i^p \\ U_{np} &= \sum_{i=1}^{n} \rho_i^{np} = (\rho_1^{np} + \rho_2^{np}) + \sum_{i=3}^{n} \rho_i^{np} \end{aligned} \tag{8.7}$$

By Lemma 8.1, $\rho_i^{np} < \rho_i^p, i = 1, 2$. However, if $\rho_1^p + \rho_2^p$ is small enough, we have:

$$\begin{aligned} U_p &\approx \sum_{i=3}^{n} \rho_i^p \\ U_{np} &\approx \sum_{i=3}^{n} \rho_i^{np} \end{aligned} \tag{8.8}$$

By Lemma 8.2, $\rho_i^p < \rho_i^{np}, i = 3, ..., n$. Therefore, $U_p < U_{np}$. If $U_p = 1$ for this particular task set, then the task set if infeasible under the lock-free scheme. □

Through Lemmas 8.1 and 8.2 and Theorem 8.3, we demonstrate that neither the lock-free scheme, nor the resource access protocols are *always* better than the other. Specifically, if only a small number of tasks share a few resources, then using resource access protocols is beneficial. If resources are shared by most of the tasks in the system, then the lock-free scheme is more suitable in terms of bandwidth requirement.

Another hybrid case is that tasks can be partitioned into logical groups. Tasks in each logic group closely interact with each other and share resources. In addition, resource sharing across group boundaries is rare. For example, in a networked computer, device drivers may share the protocol input/output queues with the network protocol stack. On the contrary, a word processor is very unlikely to access the protocol queues. For this hybrid case, if the critical sections in a logic group are considerably longer than those in other groups, resource access protocols may still help to reduce bandwidth requirement. If all critical sections are on the same magnitude, little can be gained by using resource access protocols. Resource access protocols may even adversely affect system performance, because smaller time quanta result in higher overhead.

9. Conclusions

We present three UA resource access protocols. The protocols consider activities that are subject to TUF time constraints, and mutual exclusion constraints on sharing non-CPU resources. We consider the timeliness objective of probabilistically satisfying lower bounds on the utility accrued by each activity, and maximizing the total accrued utility. The protocols allocate CPU bandwidth to activities to satisfy utility lower bounds, while activity instances are scheduled to maximize total utility. We analytically establish the conditions under which utility bounds are satisfied.

The protocols presented here have been folded into a timing analysis software tool, in corporation with an industrial vendor. The tool is currently being used in US DoD programs. Future work includes studying the sensitivity of the protocols to the accuracy of the required scheduling parameters, and extending them to multiprocessors.

References

[1] J. H. Anderson, R. Jain, and K. Jeffay. Efficient object sharing in quantum-based real-time systems. In *IEEE RTSS*, pages 346–355, December 1998.

[2] T. P. Baker. Stack-based scheduling of real-time processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[3] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2(4):301–324, Nov. 1990.

[4] R. Clark, E. D. Jensen, A. Kanevsky, J. Maurer, et al. An adaptive, distributed airborne tracking system. In *IEEE WPDRTS*, pages 353–362, April 1999.

[5] R. K. Clark, E. D. Jensen, and N. F. Rouquette. Software organization to facilitate dynamic processor scheduling. In *IEEE WPDRTS*, April 2004.

[6] GlobalSecurity.org. Multi-platform radar technology insertion program. `http://www.globalsecurity.org`.

[7] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time systems. In *IEEE RTSS*, pages 112–122, December 1985.

[8] P. Li, B. Ravindran, and E. D. Jensen. Utility accrual real-time scheduling with probabilistically assured timeliness performance. In *PARTES Workshop, ACM EMSOFT*, Sept. 2004.

[9] B. Ravindran, E. D. Jensen, and P. Li. On recent advances in time/utility function real-time scheduling and resource management. In *IEEE ISORC*, pages 55 – 60, May 2005.

[10] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Computers*, 39(9):1175–1185, Sept. 1990.

[11] I. Stoica, H. A.-Wahab, K. Jeffay, et al. A proportional share resource allocation algorithm for real-time, time-shared systems. In *IEEE RTSS*, pages 288–299, December 1996.

Improvement of QoD and QoS in RTDBSs*

Emna Bouazizi, Claude Duvallet and Bruno Sadeg
LIH, Université du Havre, 25 rue Philippe Lebon
BP 540, F-76058 LE HAVRE Cedex
{Emna.Bouazizi,Claude.Duvallet,Bruno.Sadeg}@univ-lehavre.fr

Abstract

In current research toward the design of more powerful behavior of RTDBS under unpredictable workloads, different research groups focus their work on QoS (Quality of Service) guarantee. Their research is often based on feedback control real-time scheduling theory. However, due to the high service demand, and even with guarantee, some transactions may miss their deadlines. In this paper, we propose a technique which allows to execute transactions on time using fresh and precise data while taking into account the global size of the database. We have extended the feedback-based miss ratio control by both using multi-versions data, and proposing a data management policy combining (1) limitation of the versions number and (2) dynamic adjustment of this limit according to a maximum database size parameter. Simulation results have shown that the proposed approach successfully provides tight miss ratio guarantees and hight quality of data freshness.

1 Introduction

In previous years, a lot of work has been done on RTDBSs [12][13], which are systems designed to manage applications where it is desirable to execute transactions timely using fresh and precise data [1]. Since the workload in this systems is unpredictable, the system may become quickly overloaded, leading to the decrease of the well-known RTDBS performance criterion: the number of transactions that complete before their deadlines.

To support these applications, some techniques based on Quality of Service (QoS) guarantee have been proposed to control the transient overshoot. They are often based on feedback control real-time scheduling theory [10][11]. Up to now, the major drawback is that in case of conflicts between transactions, some transactions are blocked, or aborted and restarted. This may lead transactions to miss deadlines. To address this problem, we have extended the feedback-based miss ratio control by using a multi-versions data architecture. This limits data access conflicts between transactions, enhancing then the concurrency and limits the deadline miss ratio.

The main objective of our approach is to maximize the number of transactions which meet their deadlines. In addition, our work aims to support a certain freshness for the data accessed by time-constrained transactions under a condition: the fixed maximum size of the database. To this purpose, we merge two previously approaches proposed in [6] and [7]. In the new mixed approach, the number of versions is dynamically adjusted, but does not have to exceed a threshold which consists in a maximum data versions number, and also does not have to exceed a fixed threshold which consists in the maximum database size.

The remaining of the paper is organized as follows. Section 2 describes the real-time database model. In Section 3, our proposed model is described. Some simulation results are given and commented in Section 4. In Section 5, we conclude the paper and give some perspectives.

2 Real-Time Database model

We consider firm RTDBS model, in which late transactions are aborted because they are useless after their deadline, and we consider a main memory database model.

This work on QoS guarantees is guided by the following premises:

1. Transactions are executed according to

*Real-Time Database Systems

their priority and they are classified into two categories: update transactions and user transactions (see section 2.2).

2. We keep different versions for each data item. These versions are dynamically adjusted by verifying the data freshness and considering the Data Error (DE). Data Error is computed by comparing the data version stored in the database with the corresponding value of the data in the real world. DE must respect an upper bound given by the Maximum Data Error (MDE) [2]. In our real-time database, validity intervals are used to maintain the temporal consistency between the real world and the sensor data stored in the database [12]. A data version d_i is considered temporally inconsistent (not fresh or stale) if the current time is later than the timestamp of d_i followed by its absolute validity interval (denoted AVI_i), i.e $CurrentTime > Timestamp_i + AVI_i$ (see section 2.3).

2.1 Data model

Data objects are classified into either real-time or non real-time data. A non real-time data is a classical data found in conventional databases, whereas a real-time data has a validity interval beyond which it becomes useless. These data may change continuously to reflect the real world state (for example, the current temperature value). Each real-time data has a timestamp indicating the last observation of the real world state. In our model, we consider only real-time data.

Many versions of a real-time data item may be stored in the database and the number of versions considered may be either fixed or dynamically adjusted. To store a version, data freshness and the MDE parameters are taken into account.

2.2 Transaction model

Transactions are classified into two classes: update transactions and user transactions. Update transactions are used to update the values of real-time data in order to reflect the state of real world. Update transactions are executed periodically and have only to write sensor data. User transactions, representing user requests, arrive aperiodically and may read real-time data, and read or write non real-time data.

2.3 Performance metrics

Three main performance metrics are considered: $MissRatio$ (MR), $DataFreshness$ (DF), and $DataError$ (DE) [3].

1. $MissRatio$: the transactions miss ratio is defined as follows:

$$MR = 100 \times \frac{\#Late}{\#Terminated}(\%) \qquad (1)$$

where $\#Late$ denotes the number of transactions that have missed their deadline, and $\#Terminated$ is the number of terminated transactions.

2. $DataFreshness$: in RTDBS, data can become outdated. To measure the freshness of a data item d_i in an RTDB, the notion of absolute validity interval (AVI) is used. A data version is related to a timestamp indicating the latest observation of this data item in the real world. d_i is considered temporally consistent (or fresh) if $(CurrentTime - Timestamp(d_i) \leqslant AVI(d_i))$. The database freshness can also be measured. It represents the ratio between fresh data and all the data in the database.

3. $DataError$: it represents the deviation between the current data value ($CV(d_i)$) and the updated value ($UV(d_i)$) when $d_i \neq 0$. The upper bound of the error is given by the maximum data error (denoted MDE). DE of data version d_i is defined as:

$$DE_i = 100 \times \frac{CV(d_i) - UV(d_i)}{CV(d_i)}(\%) \qquad (2)$$

We note that the quality of data (QoD) depends on its freshness and on DE, whereas the quality of transaction (QoT) depends on the Miss Ratio.

3 Multi-Versions Data-Feedback Control Scheduling Architecture

3.1 Introduction

It is well known that feedback control is very effective in management of QoS in RTDBS, under unpredictable workloads [4]. The goal is to control the system performances, defined by a set of controlled variables in order to satisfy a

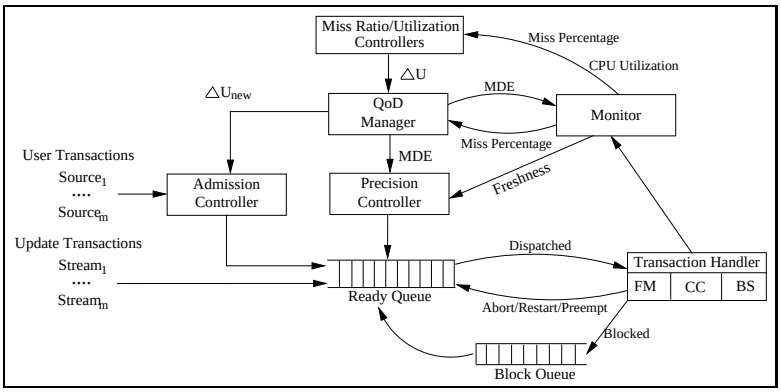


Figure 1. A Feedback Control Scheduling Architecture (FCSA) [2].

given QoS specification. The general outline of the feedback control scheduling architecture is given in Figure 1. An RTDBS consists of several components. In our study, the component we are interested in are: the admission controller, the ready queue, the blocked queue, and the transaction handler.

For the QoS management, a monitor, a miss ratio controller, an utilization controller and a QoD manager are added to the system in order to adjust its performances and to control the information flows. An Admission Controller (AC) is used to avoid system overload by rejecting some user transactions. Transactions handler provides a platform for managing transactions. It consists of a Concurrency Controller (CC), a Freshness Manager (FM) and a basic scheduler (BS). Transactions are scheduled by a Basic Scheduler in the ready queue using, for example, the EDF scheduling policy [9]. The FM checks the freshness before a transaction accesses a data item. It blocks a user transaction if the target data item is stale. Based on the two phase locking protocol, the CC ensures the concurrent transactions serializability. In case of conflict between transactions, when a higher priority transaction uses the data item, transactions with lower priority will be blocked. At each sampling period, the monitor samples the system performance data from the transaction manager and sends them to the controller. The miss ratio and utilization controller generates signals based on the sampled miss ratio and utilization data.

Feedback control has been proven to be very effective in supporting a required performance specification [3].

The base of our work are the articles of Amirijoo et al. [2][5]. We have extended the FCS architecture by exploiting several versions of real-time data, and then proposed the Multi-Versions Data-Feedback Control Scheduling (MVD-FCS) approach [6][7]. In this section, we present a new approach which enhances MVD-FCSA depicted in Figure 2 where the solid arrows represent the transaction flows and the dotted arrows represent the real-time data flows.

3.2 Motivation for MVD-FCSA

In RTDBS, an update transaction always writes a real-time data item while a user transaction reads real-time data items. In general, only update transactions modify the real-time data. Most conflict cases come from incompatible access patterns when an update transaction wants to modify a data item that is accessed by user transactions. One of these transactions must be aborted and restarted according to the used concurrency control protocol. Furthermore, when the accessed data item are stale, the FM blocks the user transaction. This increases the risk that transactions miss their deadline. MVD notion is used to alleviate this risk. When an update transaction wants to modify a real-time data, a new data version is created. We consider all data values that correspond to different versions of the same data item.

To summarize, in the FCSA, two actions are considered important for improving the MVD-FCSA service:

- In case of conflict between transactions, when a higher priority transaction uses the data item, transactions with lower priority will be blocked.

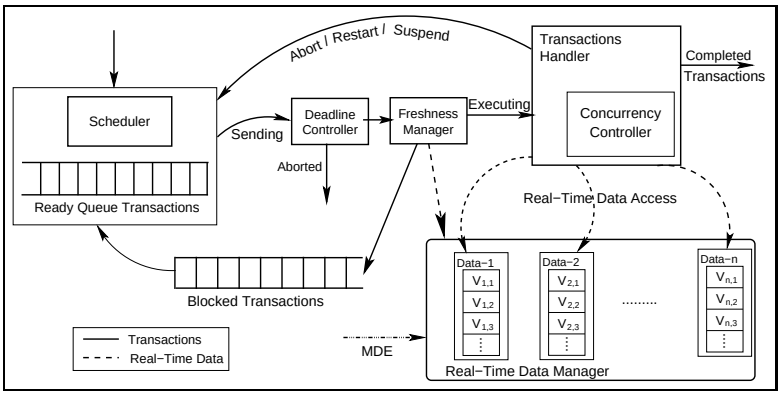


Figure 2. Multi-Version Data - Feedback Control Scheduling Architecture.

- FM blocks user transactions if the acceded data are stale.

To enhance this protocol and to minimize the transaction miss ratio, we have proposed the MVD-FCSA, which consists of the creation of data versions as soon as conflicts (read-write) occur between transactions. This approach limits the data access conflicts between transactions, and then enhances the concurrency. When an update transaction wants to modify a real-time data, a new data version is created.

3.3 MVD-FCSA components

The majority of MVD-FCSA components exist in the classical feedback control scheduling architecture [2], but they are adapted as follows.

3.3.1 Real-time transactions Scheduler

Transactions are scheduled according to their priority. The priority of a transaction depends on both its deadline and its type (update or user transaction). Hence, we merge EDF policy with respect to transaction type and priority. A lower priority transaction can be scheduled if there are no ready transactions with higher priority to schedule.

3.3.2 Deadline Controller

To control transaction validity [9][8], the Deadline Controller (denoted DC) uses three controlled variables: transaction deadline, current time and minimal execution time. If the current time is greater than transaction deadline, the transaction will be aborted. Otherwise, DC makes a second test, where a transaction is accepted only if the sum of its minimal execution time and the current time is lower than transaction deadline. Otherwise the transaction will be aborted. If the two verifications steps succeed, then the transaction is transferred to the data Freshness Manager (FM).

3.3.3 Freshness Manager

The Freshness Manager checks if transactions issued from the deadline controller access fresh data. Freshness Manager (FM) is used to provide better QoS in RTDBS where several transactions access to the same fresh data. It checks the freshness of acceded data just before a transaction commits. This way, the data accessed by committed transactions are always fresh at commit time. If the accessed data is fresh, transactions can be executed and then sent to the transactions handler. Otherwise, if the accessed data item is currently stale or if its validity is estimated to be expired before the deadline of the transaction, then FM blocks the user transaction. The blocked transaction will be transferred from the blocked queue to the ready queue as soon as the corresponding update has committed. In our architecture, the FM checks the freshness of accessed data, i.e., AVI is not reached and data remains fresh until the end of the transaction execution. Therefore, we verify that AVI of accessed data is greater than the transaction deadline. To this purpose, we use MVD. So, the freshness condition must be considered by checking the freshness of the most recent data version that is not write-locked by other transaction(s).

3.3.4 Real-Time Data Manager

The main objective of this component is to guarantee the data freshness and to enhance the deadline miss ratio even in the presence of conflicts and unpredictable workloads.

To achieve this goals, in [7] we have used a MVD with a fixed number of data versions. This number is fixed in advance by the DBA according to QoS requirement level, and it is the same for each real-time data. In [6], we have enhanced this approach by allowing the dynamic adjustment of the version number. For each data, we have a version queue. The queue is continuously updated in order to limit the number of data versions by supressing/adding versions, based on both the data freshness and MDE criterion. The size of each version queue, denoted SVQ, is dynamically adjusted according to the following formula:

$$SVQ_{i,j} = \lfloor \frac{AVI_j}{Period_i} \rfloor \quad (3)$$

where $Period_i$ is the period of $transaction_i$, and AVI_j is the absolute validity interval of d_j.

RTDBSs usually monitor the current real-world state using periodic updates. In this paper, we investigate several important problems to guarantee the desirable quality of real-time data services in terms of timeliness, freshness, precision, the decreasing of transaction miss deadline and the database size constraint. In a recent paper [6], we have shown that MVD-FCSA is a good solution to alleviate the risk that transactions miss their deadlines compared to the classical feedback control scheduling approaches. However, in [6] the proposed approach does not take into account the database size.

In this paper, we have extended the last approach (MVD-FCSA with dynamically adjusted number of data versions) by taking into account the database size constraint. We merge the two approaches described in [7] and [6], i.e. the number of data versions is dynamically adjusted and does not have to exceed the fixed threshold representing the number of data versions, and we have considered in the same time a threshold representing the database size. In this new approach, a data item will be accessed only if its version number is lower than the maximum database size. This way, RTDB size constraints are respected. The respect of the threshold of the RTDB size is a practical factor for RTDBS specification.

3.3.5 Concurrency Control with MVD

One of the most important issues in the design of RTDBS is the concurrency control component. Its objectives are (i) to control the interaction between concurrently transactions and (ii) to maintain the database consistency. In this paper, we focus on the interaction between update and user transactions.

The database consistency can be maintained using concurrency control protocols. We use 2PL-HP (Two Phase Locking-High Priority Protocol) where lower priority transactions will be blocked or aborted if a higher priority transaction accesses a data item. Otherwise, the transaction is aborted and restarted. Consequently, the 2PL-HP may increase the execution time of transactions. This leads transactions to miss their deadlines. To address this problem, i.e. to alleviate this risk, we propose (1) the MVD technique that allows user transactions to not wait for the last version if data is current updating, and (2) an adapted 2PL-HP when the maximum number of versions is reached. The priority is applied on transactions group that accessed to the same data version [6]. The priority of transactions group corresponds to the highest transaction priority among all transactions in this group.

4 Simulations and results

4.1 Simulations

We have studied and evaluated the behavior of an RTDBS according to a set of performance metrics. The performance evaluation is undertaken by a set of simulation experiments, where a set of parameters have been varied. Table 1 summarizes these parameters.

The simulated workload consists of update and user transactions. Update transactions occupy approximately 50% of the workload. The period of update transaction ($Period_i$) is uniformly distributed and estimated execution time is given by: $ExecutionTime_i = NbOfOperation_i \times OpExecTime$, where $NbOfOperation_i$ and the $OpExecTime$ represent respectively the number of operations in the transaction T_i and the execution time of an operation.

The model consists of eight components. We have used two transaction generators: an update transaction generator (UpdateTransGen) which generates update transactions and an user transaction generator (UserTransGen)

Parameter	Meaning	Value
$NbOfOperations$	Number of operations in an user transaction	[1, 5]
$OpExecTime$	Execution time of an operation	1s
$Period_i$	Periodicity of update transaction	[1000ms, 5000ms]

Table 1. Parameters of Simulation.

which generates user transactions. The workload model characterizes transaction in terms of the number of read/write operations. Update transactions can only write one data. Transactions are scheduled by a scheduler (Scheduler) in ready queue according to their priority. The priority assignment formula is given by P(T_i)= 1/deadline (T_i). Deadline controller (DC) uses three controlled variables: transaction deadline (deadline), current time (StartTime) and minimal execution time (ExecutionTime). The deadline formula is calculated as follows: $deadline(T_i) = StartTime + ExecutionTime \times (1 + SlakTime)$ where $SlakTime$ is a constant that provides control over tightness/slackness of transaction deadlines. To check the freshness of accessed data, the freshness manager uses the AVI parameter. Transactions handler controls the execution of transactions. Concurrency controller use the adapted 2PL-HP protocol to control the interaction between transactions [6]. The real-time data management (RTDM) is the most important component of our model. In the sets of experiments, we have varied the database size and the maximum number of data versions for each data item. The database size represents the number of versions.

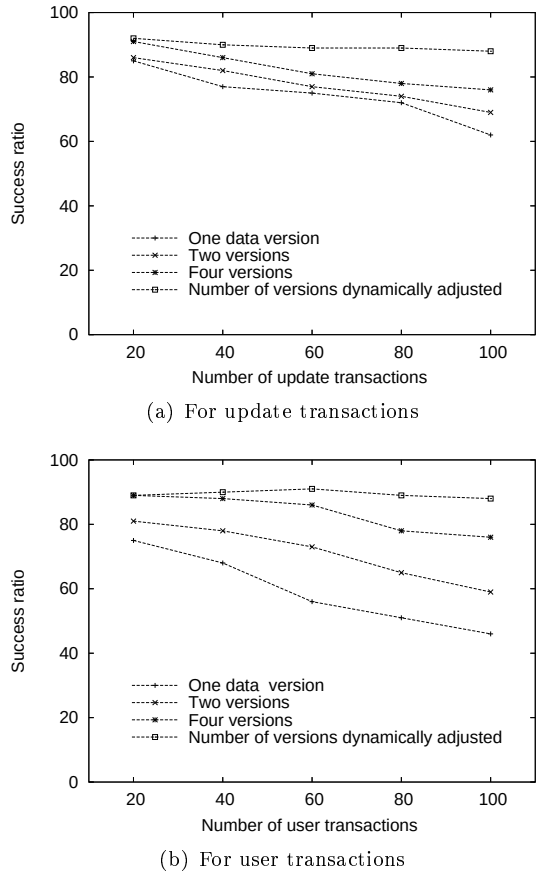
4.2 Results

Since in our approach we have only extended the transactions flows of the classical FCSA, the performance metric in our experiments is the success ratio. The graphical results show the miss ratio of transactions when using MVD-FCSA. We have evaluated the behavior of the system by varying a set of parameters:

1. The threshold of data versions number
2. The threshold of database size
3. The number of transactions

4.2.1 Experiment 1: Results of MVD-FCSA

As shown in Figure 3, when we use the classical FCSA (with one version), the MVD-FCSA with two versions and the MVD-FCSA with four versions, the resulting success ratio increases as soon as the number of versions increases.



(a) For update transactions

(b) For user transactions

Figure 3. Simulation results for the MVD-FCSA.

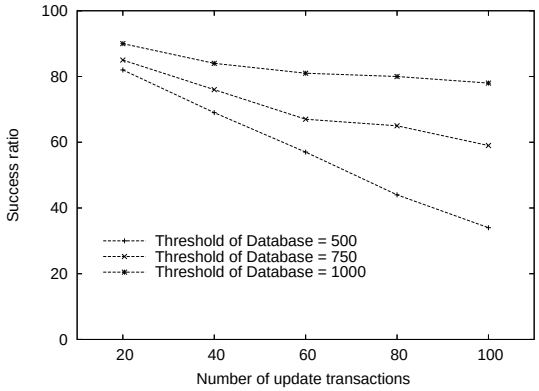
Compared to the effect of using MVD-FCSA with a fixed number of versions, using MVD-FCSA with dynamically adjusted number of data versions shows a relatively high success ratio, as shown in Figure 3.

4.2.2 Experiment 2: Varying the threshold of database size using the mixed approach of MVD-FCSA

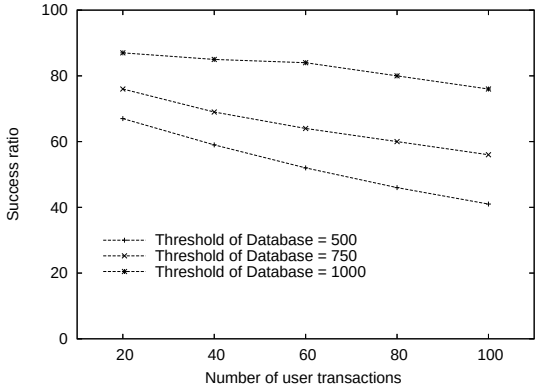
We use our mixed approach (dynamic adjustment of data versions with maximum fixed

number). In Figure 4, we have fixed a threshold of data versions number (equal to 4 versions) and we have varied the database size (500, 750, 1000). In Figure 5, we have also varied the database size, while the threshold of data versions number is equal to 6.

Figures 4 et 5 show the effect of varying the database size. The resulting success ratio increases according to the increase of the database size.



(a) Update transactions

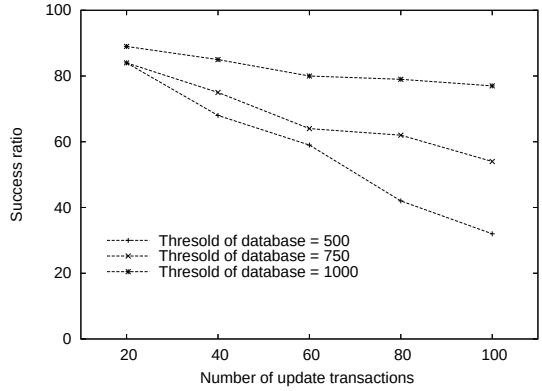


(b) For user transactions

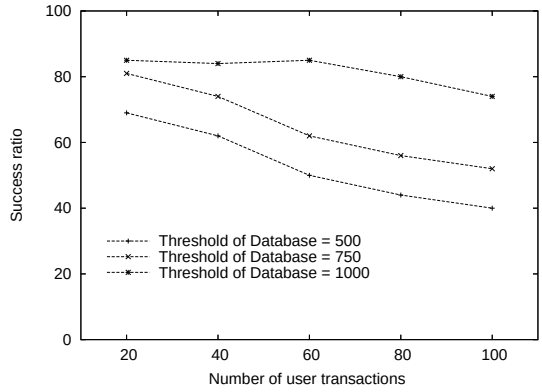
Figure 4. Simulation results when using the mixed approach of MVD-FCSA (maximum number of versions = 4) and varying the threshold of database size.

4.2.3 Experiment 3: Varying the threshold of data versions number

We have also used the mixed approach of MVD-FCSA. We have fixed the database size and we have varied the threshold of data versions number. Compared to the effect of using a maximum of six versions, the use of four versions shows a relatively high success ratio, as shown in Figures 6 and 7. This may be explained by the complexity of the versions management.



(a) For update transactions



(b) For user transactions

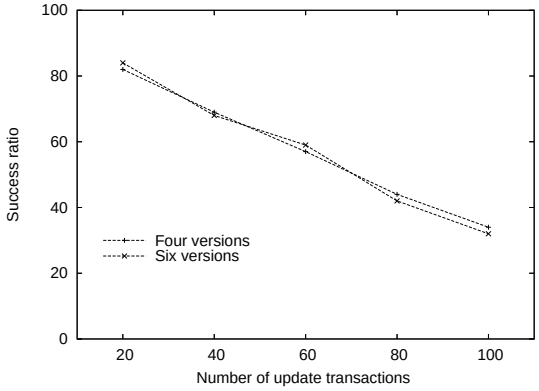
Figure 5. Simulation results when using the mixed approach of MVD-FCSA (maximum number of versions = 6) and varying the threshold of database size.

4.3 Summary of results and discussions

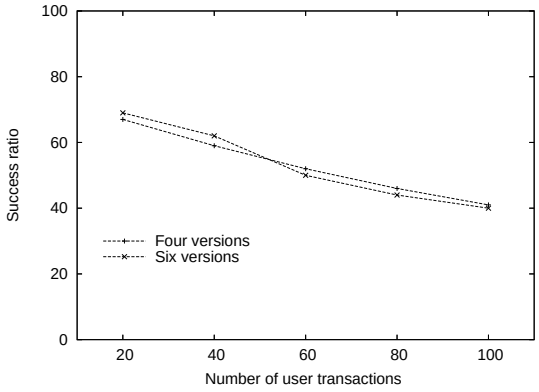
We have compared the system performances, in terms of miss ratio, by varying the database size and by varying the maximum number of data versions. All experiments simulation show that:

1. MVD-FCSA minimizes transactions miss deadline (compared to the classical FCSA).

2. Generaly, the success ratio increases according to the increase of the number of versions.

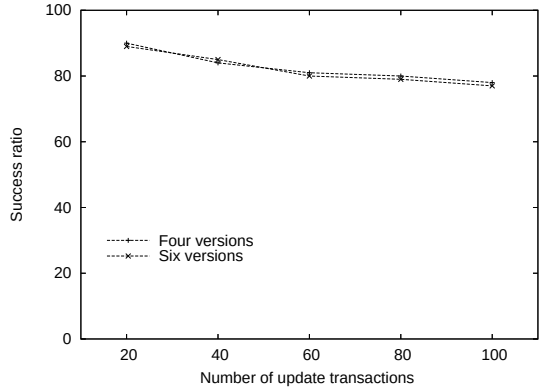


(a) For update transactions

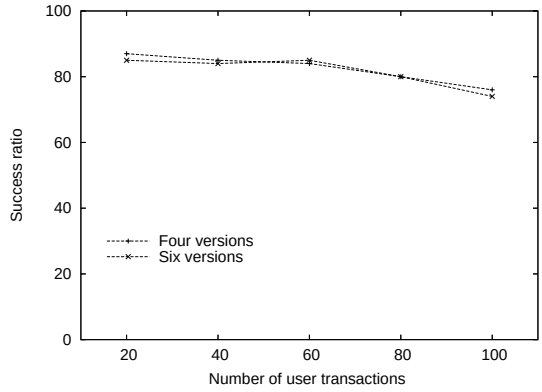


(b) For user transactions

Figure 6. Simulation results of using the mixed approach of MVD-FCSA: varying the number of versions and the threshold of database size 500.



(a) For update transactions



(b) For user transactions

Figure 7. Simulation results of using the mixed approach of MVD-FCSA: varying the number of versions and the threshold of database size 1000.

3. The success ratio increases according to the increase of the database size.

4. When the size of the database is fixed, the performances are not enhanced beyond a certain threshold which represents a maximum number of versions for each data.

5 Conclusion and future work

In this paper, we have presented the multi-versions data-feedback control scheduling architecture for quality of service management. We have used multi-versions data with dynamically adjusted number of versions while taking into account the RTDB size constraint. This improvement consists in minimizing the number of conflicts by minimizing the number of aborted transactions when using an adapted 2PL-HP concurrency control protocol.

Simulation results show that MVD-FCSA with dynamically adjusted number of data versions may be applied efficiently in RTDBS, i.e more transactions meet their deadlines. We note that the respect of the threshold of the RTDB size might be a practical factor for RTDBS specification.

We plan to extend this work in several ways. We will take into account the data *importance*. Indeed, in case of a small threshold of the RTDB size, all data beyond the threshold value are not accessed whatever their importance. The importance of the data item may be modeled by assigning to each data item a weight according to its importance. Further, we also plan to extend our work to manage derived data and to consider other aspects to study different components of the feedback control scheduling architecture for quality of service management in RTDBS. Among them, we will deal with imprecise computing [2], applied to video contents.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling Real-Time Transactions: A Performance Evaluation. *International Journal of Distributed and Parallel Databases*, 1(2), 1988.

[2] M. Amirijoo, J. Hansson, and S. H. Son. Algorithms for Managing Real-time Data Services Using Imprecise Computation. In *Proceedings of International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, Taiwan, 2003.

[3] M. Amirijoo, J. Hansson, and S. H. Son. Error-Driven QoS Management in Imprecise Real-Time Databases. In *Proceedings of* 15^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, Portugal, 2003.

[4] M. Amirijoo, J. Hansson, and S. H. Son. Specification and Management of QoS in Imprecise Real-Time Databases. In *Proceedings of International Database Engineering and Applications Symposium (IDEAS)*, Hong Kong, 2003.

[5] M. Amirijoo, J. Hansson, and S. H. Son. Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations. *IEEE Transaction Knowledge and Data Engineering*, 55(3):304–319, March 2006.

[6] E. Bouazizi, C. Duvallet, and B. Sadeg. Management of QoS and Data Freshness in RTDBSs using Feedback Control Scheduling and Data Versions. In 8^{th} *IEEE International Symposium on Object-oriented Real-time distributed Computing (IEEE-ISORC'05)*, Washington, 2005.

[7] E. Bouazizi, C. Duvallet, and B. Sadeg. Using Feedback Control Scheduling and Data Versions to enhance Quality of Data in RTDBSs. In *IEEE International Computer System and Information Technology (ICSIT'2005)*, Alger, Algerie, 2005.

[8] C.S. Date. *An Introduction to Database Systems*. Addison-Wesley, 1985.

[9] C. Liu and J. Leyland. Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[10] C. Lu. *Feedback Control Real-Time Scheduling*. PhD thesis, University of Virginia, May 2001.

[11] C. Lu, J.A. Stankovich, G. Tao, and S.H. Son. Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms. *Real-Time Systems*, 23(1/2):85–126, 2002.

[12] K. Ramamritham. Real-Time Databases. *Journal of Distributed and Parallel Databases*, 1(2):199–226, 1993.

[13] K. Ramamritham, S.H. Son, and L.C. DiPippo. Real-Time Databases and Data Services. *Real-Time Systems*, 28:179–215, 2004.

Multiprocessor Scheduling

The Partitioned Multiprocessor Scheduling of Non-preemptive Sporadic Task Systems*

Nathan Fisher Sanjoy Baruah
The University of North Carolina at Chapel Hill

Abstract

We consider polynomial-time algorithms for partitioning a collection of non-preemptive or restricted-preemption tasks among the processors of an identical multiprocessor platform. Since the problem of partitioning tasks among processors (even with unlimited preemption) is NP-hard in the strong sense, these algorithms are unlikely to be optimal. For task systems where the ratio between the largest execution time and the smallest relative deadline is small, we provide a sufficient condition for feasibility. The application of this algorithm to preemptive quantum-based systems is also discussed. For all other task systems, we experimentally evaluate different variants of our heuristic over sets of randomly generated tasks.

1 Introduction

In many real-time systems, complete *a priori* knowledge of job release times is either impractical or impossible. The **sporadic task model** [16] provides a characterization of real-time computation of such task systems by allowing time between the release of successive jobs of a task to vary. For a sporadic task τ_i, a *minimum inter-arrival separation* parameter (historically called the *period*) describes the minimum time interval between successive jobs of a task. A collection of jobs generated by the sporadic task system is called *legal* if the minimum inter-arrival separation is respected for each task. A *relative deadline* parameter identifies the time interval from a job's release time to its absolute deadline during which execution of the job must complete. A collection of sporadic tasks is called a *sporadic task system.*

The advantage of preemptive scheduling on uniprocessors has been known since Lui and Layland [14] showed that total utilization of a uniprocessor is achievable under preemptive EDF scheduling. In preemptive scheduling, a job may be halted before the completion of its execution and resumed at a later time. However, in many systems, preemption is impractical or undesirable due to the high overhead involved in context switching between different tasks. In *non-preemptive scheduling*, once a job begins execution it executes continuously on the processor until its completion. Non-preemptive scheduling for these systems can reduce scheduling overhead, and have the following additional benefits [12]: elimination of the need for complex resource sharing protocols for resources or critical sections that are local to a processor; reduction in the implementation complexity of scheduling protocols; and estimates of worst-case execution time for tasks may be more accurate in the non-preemptive model.

In addition to pure non-preemptive tasks, it is sometimes useful to model both preemptive and non-preemptive behavior in the same system. The *restricted-preemption model* allows tasks to execute non-preemptively in short intervals, and be preempted in-between the non-preemptive intervals. Each task specifies a *non-preemption parameter* which indicates the maximum length of time a task may non-preemptively execute. Notice that a non-preemptive system can be represented in the restricted preemption model by setting the non-preemption parameter for each task equal to its execution requirement. A preemptive system can be represented by setting the non-preemption parameter for each task to zero. The restricted-preemption model is also useful for characterizing the behavior of *quantum-based* scheduling systems.

An important endeavor in real-time scheduling theory is determining whether a task system is *feasible* on a given processing platform. A task system is feasible if there exists, for each legal collection of job releases, a schedule on the processing platform in which no task misses a deadline. For non-preemptive and restricted-preemption systems, a more restrictive notion of feasibility can be defined: *feasibility without inserted idle times (IIT)*. A task system is feasible without IIT if there exists a schedule for every legal collection of jobs in which a processor is never idle while there are jobs awaiting execution. A scheduling algorithm is *optimal* in this model if it can schedule every task system that is feasible without IIT.

Uniprocessor Scheduling. The non-preemptive real-time scheduling of sporadic tasks has been studied extensively for uniprocessor platforms. For sporadic task systems where each task's relative deadline is equal to its period, Jeffay et al. [12] proved that the non-preemptive *Earliest Deadline First* algorithm (EDF) [14] is optimal on uniprocessors with respect to scheduling without IIT, and

*This research has been supported in part by the National Science Foundation (Grant Nos. ITR-0082866, CCR-0204312, and CCR-0309825).

they provided necessary and sufficient conditions for EDF-schedulability. Non-preemptive EDF schedules at each idle instant the job with the nearest deadline (from the set of jobs awaiting execution). George et al. removed the restriction on a task's relative deadline, showed that EDF is optimal without IIT [10], and provided modified necessary and sufficient conditions for EDF-schedulability [11]. Researchers have also focused different techniques and models [2, 9, 6] for limiting preemptions in an attempt to obtain the benefits of both preemptive and non-preemptive scheduling.

Multiprocessor Scheduling. For the multiprocessor scheduling of non-preemptive and restricted-preemption sporadic tasks, two alternative paradigms exist: *global* and *partitioned* scheduling. For restricted-preemption and non-preemptive global scheduling, a job executing a non-preemptive code section will execute continuously on the same processor; a job executing a preemptive code-section can be halted and can resume execution on a different processor. For non-preemptive partitioned scheduling, a task is assigned to a processor, and all jobs of the task are always executed on that processor.

Non-preemptive and restricted-preemption multiprocessor scheduling of sporadic tasks has received much less attention. Baruah [5] considered the non-preemptive global scheduling of periodic and sporadic tasks on an identical multiprocessor platform. To the best of our knowledge, there has been no work done on the partitioned scheduling of non-preemptive sporadic task systems.

This research. We consider the non-preemptive and restricted-preemption multiprocessor scheduling of sporadic task systems under the partitioned paradigm. Since partitioning tasks among processors reduces the multiprocessor scheduling algorithm to a series of uniprocessor scheduling problems (one to each processor), the optimality (without IIT) of non-preemptive EDF [12, 10] makes EDF a reasonable algorithm to use as the run-time scheduler on each processor. Therefore, we henceforth make the assumption that each processor, and the tasks assigned to it by the partitioning algorithm, are scheduled during runtime according to non-preemptive EDF and focus on the partitioning algorithm.

Recently, Albers and Slomka [1] developed a fully polynomial-time approximation scheme for determining the feasibility of preemptive sporadic task systems on a uniprocessor. Their results also extend to non-preemptive sporadic task systems. In [7], we non-trivially applied the results of Albers and Slomka to obtain a polynomial-time partitioning algorithm for preemptive sporadic task systems. In this paper, we extend and generalize the results in [7] to be applicable for non-preemptive and restricted-preemption systems. To address some of the drawbacks of the simple restricted-preemption partitioning algorithm that we derive, we also consider a family of heuristics which we evaluate experimentally.

Organization. In Section 2, we present background on the non-preemptive and restricted-preemption sporadic task model. We also discuss uniprocessor feasibility tests for restricted-preemption sporadic tasks. In Section 3, we design a simple algorithm for partitioning restricted-preemption systems, and evaluate this algorithm theoretically. The application of this algorithm to preemptive quantum-based systems is explored as well. In Section 4, we consider some more pragmatic heuristics for partitioning and evaluate their performance empirically. In Section 5, we draw together the conclusions of this paper.

2 Task and machine model

A *restricted-preemption sporadic task* [6] $\tau_i = (e_i, q_i, d_i, p_i,)$ is characterized by a *worst-case execution requirement* e_i, a *non-preemption parameter* q_i, a *(relative) deadline* d_i, and a *minimum inter-arrival separation* p_i. In general, a restricted-preemption sporadic task is subject to the trivial constraints that $q_i \leq e_i$, $e_i \leq p_i$, and $e_i \leq d_i$. The *utilization* of task τ_i represents the amount computational capacity required by τ_i on a single processor and is denoted by $u_i \stackrel{\text{def}}{=} e_i/p_i$. The non-preemption parameter q_i specifies the maximum length of time at which τ_i may execute non-preemptively on a single processor. Observe that this parameter specifies only the interval length during which τ_i executes non-preemptively, not the start and end times of the non-preemptive interval. In our model, τ_i may execute non-preemptively for up to q_i time units starting at any point in time, and may do so arbitrarily often (subject to the constraints of the other task parameters). We may model a pure non-preemptive sporadic task by setting $q_i = e_i$.

We will assume that we are given a multiprocessor Π comprised of m identical processors, $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$. Let τ be a system of n restricted-preemption sporadic tasks where $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and $\tau_i = (e_i, d_i, p_i, q_i)$ for all i, $1 \leq i \leq n$.

We may categorize sporadic task systems based on the relationship between the values of p_i and d_i for each $\tau_i \in \tau$. For the purposes of this paper, we consider three subclasses based on this relationship:

- **Implicit-deadline**: Each sporadic task $\tau_i \in \tau$ satisfies the constraint that $d_i = p_i$.
- **Constrained**: Each sporadic task $\tau_i \in \tau$ satisfies the constraint that $d_i \leq p_i$.
- **Arbitrary**: There is no restriction placed on the relationship between d_i and p_i.

2.1 The Demand-Bound Function

For any sporadic task τ_i and any real number $t \geq 0$, the *demand bound function* $\text{DBF}(\tau_i, t)$ is the largest cumulative execution requirement of all jobs that can be generated by τ_i to have both their arrival times and their deadlines within a contiguous interval of length t. It has been shown [8] that the cumulative execution requirement of jobs of τ_i over an interval $[t_o, t_o + t)$ is maximized if one job arrives at the start of the interval – i.e., at time-instant

t_o – and subsequent jobs arrive as rapidly as permitted — i.e., at instants $t_o + p_i$, $t_o + 2p_i$, $t_o + 3p_i$, ... Equation (1) below follows directly [8]:

$$\text{DBF}(\tau_i, t) \stackrel{\text{def}}{=} \max\left(0, \left(\left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1\right) \times e_i\right) \quad (1)$$

Albers and Slomka [1] have proposed a technique for *approximating* the DBF; the following approximation to DBF is obtained by applying their technique:

$$\text{DBF}^*(\tau_i, t) = \begin{cases} 0, & \text{if } t < d_i \\ e_i + u_i \times (t - d_i), & \text{otherwise} \end{cases} \quad (2)$$

As stated earlier, it has been shown that the cumulative execution requirement of jobs of τ_i over an interval is maximized if one job arrives at the start of the interval, and subsequent jobs arrive as rapidly as permitted. Intuitively, approximation DBF* (Equation 2 above) models this job-arrival sequence by requiring that the first job's deadline be met explicitly by being assigned e_i units of execution between its arrival-time and its deadline, and that τ_i be assigned $u_i \times \Delta t$ of execution over time-interval $[t, t + \Delta t)$, for all instants t after the deadline of the first job, and for arbitrarily small positive Δt.

Observe that the following inequalities hold for all τ_i and for all $t \geq 0$:

$$\text{DBF}(\tau_i, t) \leq \text{DBF}^*(\tau_i, t) < 2 \cdot \text{DBF}(\tau_i, t)\ . \quad (3)$$

2.2 Uniprocessor Feasibility

It has been shown [12] that the response time of a task τ_i (i.e. the time from a job's release to its completion) is maximized in the following scenario: Let τ_k $(k > i)$ be the task with the largest non-preemptive execution requirement. The worst-case sequence is if a job of τ_k is released just prior to the release of a job of τ_i, and all other tasks τ_j $(j \neq i, k)$ release jobs simultaneously with τ_i; in addition, successive jobs of τ_j are released as soon as legally possible. Any feasibility test for non-preemptive sporadic systems must determine whether a deadline miss will occur in this scenario. The feasibility test we use for non-preemptive and restricted-preemptions uniprocessor systems is from [6]:

Theorem 1 (from [6]) *A restricted-preemption sporadic task system $\tau = \{\tau_1, \ldots, \tau_n\}$ is feasible (without IIT) if and only if*

$$\forall t : 0 \leq t : \sum_{i=1}^{n} \text{DBF}(\tau_i, t) \leq t, \quad (4)$$

and for all $\tau_j = (e_j, q_j, d_j, p_j)$ where $1 \leq j \leq n$

$$\forall t : 0 \leq t \leq d_j : q_j + \sum_{\substack{i=1 \\ i \neq j}}^{n} \text{DBF}(\tau_i, t) \leq t \quad (5)$$

Throughout the remainder of this paper, we will adopt the convention that "feasibility" is with respect to restricted-preemption model without IIT.

3 A Partitioning Algorithm

Given a system of sporadic tasks, the problem of determining whether it is possible for the task system to always satisfy all timing constraints is called *feasibility-analysis*. In this paper, we are interested in partitioned feasibility-analysis. Even for implicit-deadline sporadic task systems under the preemptive model, partitioned feasibility-analysis is NP-hard in the strong sense (by transformation from the bin-packing problem [13]). Unfortunately, the complexity results extend to partitioned feasibility-analysis of all restricted-preemption (or non-preemptive) sporadic task systems.

For the preemptive model, several bin-packing heuristics have been studied [15]. Typically, each of the bin-packing heuristics adheres to the following pattern:

1. Tasks of the task system are sorted by some criteria.
2. Tasks are assigned (in order) to a processor upon which they "fit" according to a sufficient (and sometimes necessary) condition.

In Section 3.1, we describe such a partitioning algorithm. Section 3.2 derives sufficient conditions for schedulability of a restricted-preemption task system using this algorithm. The algorithm and the derivation of the sufficient conditions generalizes work done for partitioning of preemptive sporadic task systems in [7]. Section 3.3 describes an application of this algorithm for preemptive quantum-based scheduling.

3.1 Algorithm NP-PARTITION

We now describe a simple partitioning algorithm called NP-PARTITION. Given a restricted-preemption sporadic task system τ comprised of n sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$, and a processing platform Π comprised of m unit-capacity processors $\pi_1, \pi_2, \ldots, \pi_m$, NP-PARTITION will attempt to partition τ among the processors of Π. The NP-PARTITION algorithm is a variant of a bin-packing heuristic known as *first-fit-decreasing*. For this section, we will assume the tasks of τ_i are indexed in non-decreasing order of their relative deadline (i.e. $d_i \leq d_{i+1}$, for $1 \leq i < n$). Let $q_{\max}(\tau) \stackrel{\text{def}}{=} max_{i=1}^{n}\{q_i\}$ denote the maximum non-preemption parameter of any task in τ.

The NP-PARTITION algorithm considers the tasks in increasing index order (i.e. $\tau_1, \tau_2, \ldots$). We will now describe how to assign task τ_i assuming that tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ have already successfully been allocated among the m processors. Let $\tau(\pi_\ell)$ denote the set of tasks already assigned to processor π_ℓ where $1 \leq \ell \leq m$. Considering the processors $\pi_1, \pi_2, \ldots, \pi_m$ in any order, we will assign task τ_i to the first processor π_k, $1 \leq k \leq m$, that satisfies the following two conditions:

$$\left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)\right) \geq e_i + q_{\max}(\tau) \quad (6)$$

and

$$\left(1 - \sum_{\tau_j \in \tau(\pi_k)} u_j\right) \geq u_i \; ; \tag{7}$$

If no such π_k exists, then Algorithm NP-PARTITION returns PARTITIONING FAILED: it is unable to conclude that sporadic task system τ is feasible upon the m-processor platform. Otherwise, NP-PARTITION returns PARTITIONING SUCCEEDED.

If τ is a constrained, sporadic task system, then it suffices to check only Equation 6:

Lemma 1 *For* **constrained** *sporadic task systems, any τ_i and π_k satisfying Equation 6 during execution of the* NP-PARTITION *algorithm will also satisfy Equation 7*

Proof: Observe that for any constrained task τ_i, Equation 2 implies that for all $t \geq d_i$,

$$\text{DBF}^*(\tau_i, t) = u_i \times (t + p_i - d_i) \geq u_i \times t.$$

Hence, Equation 6

$$\begin{array}{rcl} \equiv & \left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \geq e_i + q_{\max}(\tau)\right) \\ \Rightarrow & d_i - \sum_{\tau_j \in \tau(\pi_k)} (u_j \times d_i) \geq e_i + q_{\max}(\tau) \\ \Rightarrow & 1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq \frac{e_i}{d_i} + \frac{q_{\max}(\tau)}{d_i} \\ \Rightarrow & 1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq u_i + \frac{q_{\max}(\tau)}{d_i} \end{array} \tag{8}$$

The last inequality implies Equation 7. ■

We must now show that by assigning task τ_i to processor π_k we have not adversely affected the feasibility of tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ previously assigned to the processors. The next lemma shows that the system remains EDF-feasible if we assign tasks according to Equations 6 and 7.

Lemma 2 *If the tasks previously assigned to each processor were* EDF*-feasible (with respect to restricted-preemption) on that processor and Algorithm* NP-PARTITION *assigns task τ_i to processor π_k, then the tasks assigned to each processor (including processor π_k) remain* EDF*-feasible on that processor.*

Proof: Observe that assigning τ_i to processor π_k does not affect the tasks previously assigned to other processors. Therefore, we focus our attention only on π_k, and show that if $\tau(\pi_k)$ was EDF-feasible prior to the addition of τ_i, and Equation 6 and 7 are satisfied, then $\tau(\pi_k) \cup \{\tau_i\}$ remains EDF-feasible after the addition of τ_i.

For the sake of contradiction, assume that $\tau(\pi_k)$ and τ_i satisfies Equation 6 and 7 of NP-PARTITION, but that EDF misses a deadline when scheduling the tasks $\tau(\pi_k) \cup \{\tau_i\}$ on processor π_k. Let t_f be the time that processor π_k misses a deadline. Observe that $t_f > d_i$ since $\tau(\pi_k)$ is EDF-feasible before the addition of τ_i.

By Theorem 1, either

$$\text{DBF}(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}(\tau_j, t_f) > t_f \tag{9}$$

or there exists $\tau_\ell \in \tau(\pi_k) \cup \{\tau_i\}$ such that

$$q_\ell + \sum_{\substack{\tau_j \in \tau(\pi_k) \cup \{\tau_i\} \\ j \neq \ell}} \text{DBF}(\tau_j, t_f) > t_f. \tag{10}$$

We will show that if either Equation 9 or 10 is true, then a contradiction is reached. Assume that Equation 9 is true. Then, since DBF* is an upper bound on DBF,

$$\text{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t_f) > t_f \tag{11}$$

Since tasks are considered in order of non-decreasing relative deadline, it must be the case that all tasks $\tau_j \in \tau(\pi_k)$ have $d_j \leq d_i$. We therefore have, for each $\tau_j \in \tau(\pi_k)$,

$$\begin{array}{rcll} \text{DBF}^*(\tau_j, t_f) & = & e_j + u_j(t_f - d_j) & \text{(By definition)} \\ & = & e_j + u_j(d_i - d_j) + u_j(t_f - d_i) & \\ & = & \text{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i) & \end{array} \tag{12}$$

Furthermore,

$$\begin{array}{rcl} \text{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t_f) & & \\ \equiv \quad (e_i + u_i(t_f - d_i)) + & & \textit{(By Equation 12 above)} \\ \left(\sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i)\right) & & \\ \equiv \quad \left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)\right) & & \\ +(t_f - d_i)\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right) & & \end{array}$$

Consequently, Inequality 11 above can be rewritten as follows:

$$\left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)\right) + (t_f - d_i)\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right) > (t_f - d_i) + d_i \tag{13}$$

However by Condition 6, $(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)) \leq d_i - q_{\max}(\tau) \leq d_i)$; Inequality 13 therefore implies

$$(t_f - d_i)\left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j\right) > (t_f - d_i)$$

which in turn implies that

$$(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j) > 1$$

which contradicts Condition 7.

Now, assume that Equation 10 is true. Similar to Equation 11, we obtain

$$q_{\max}(\tau) + \sum_{\substack{\tau_j \in \tau(\pi_k) \cup \{\tau_i\} \\ j \neq \ell}} \text{DBF}^*(\tau_j, t_f) > t_f. \qquad (14)$$

Following similar logical steps of Equations 12 through 13, we would obtain from Equation 14:

$$(u_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \cup \{\tau_i\} \\ j \neq \ell}} u_j) > 1.$$

which also contradicts Equation 7.

Since both Equations 9 and 10 lead to a contradiction, our supposition that processor π_k missed a deadline at time t_f is false. Thus, $\tau(\pi_k) \cup \{\tau_i\}$ will always meet all deadlines on processor π_k. ■

The correctness of Algorithm NP-PARTITION follows, by repeated applications of Lemma 2:

Theorem 2 *If Algorithm* NP-PARTITION *returns* PARTITIONING SUCCEEDED *on task system τ, then the resulting partitioning is* EDF*-feasible.*

Run-time complexity. In attempting to map task τ_i, observe that Algorithm NP-PARTITION essentially evaluates, in Equations 6 and 7, the workload generated by the previously-mapped $(i-1)$ tasks on each of the m processors. Since $\text{DBF}^*(\tau_j, t)$ can be evaluated in constant time (see Equation 2), a straightforward computation of this workload would require $\mathcal{O}(i+m)$ time. Hence the runtime of the algorithm in mapping all n tasks is no more than $\sum_{i=1}^{n} \mathcal{O}(i+m)$, which is $\mathcal{O}(n^2)$ under the reasonable assumption that $m \leq n$.

3.2 Theoretical Evaluation

In this section, we derive a set of sufficient conditions for the success of NP-PARTITION. In particular, for each subclass of restricted-preemption sporadic tasks we derive a different sufficient condition: Theorem 3 corresponds to implicit-deadline systems, Theorem 4 for constrained systems, and Theorem 5 for arbitrary systems.

Given a task system τ, the following notation and terminology will be useful for our analysis.

$d_{\min}(\tau)$	$\stackrel{\text{def}}{=} \min_{i=1}^{n}\{d_i\}$	
$\rho(\tau)$	$\stackrel{\text{def}}{=} q_{\max}(\tau)/d_{\min}(\tau)$	(*max. blocking ratio*)
$u_{\max}(\tau)$	$\stackrel{\text{def}}{=} \max_{i=1}^{n}\{u_i\}$	(*max. utilization*)
$u_{\text{sum}}(\tau)$	$\stackrel{\text{def}}{=} \sum_{i=1}^{n} u_i$	(*system utilization*)
δ_i	$\stackrel{\text{def}}{=} e_i/d_i$	(*task load ratio*)
$\delta_{\max}(\tau)$	$\stackrel{\text{def}}{=} \max_{i=1}^{n}\{\delta_i\}$	(*max. load ratio*)
$\delta_{\text{sum}}(\tau)$	$\stackrel{\text{def}}{=} \max_{t>0}\left(\frac{\sum_{i=1}^{n} \text{DBF}(\tau_i,t)}{t}\right)$	(*system load*)

The following lemma describes subcases where either Equation 6 or 7 is trivially satisfied. The corollary immediately following shows that certain combination of subcases imply that τ is trivially restricted-preemption feasible on a single processor. The lemma and corollary will be useful in proving the main results of Theorems 3, 4, and 5.

Lemma 3 *Given a restricted-preemption sporadic task system τ and an m unit-capacity processor system* Π*,* NP-PARTITION *has the following properties:*

P1: *If $u_{\text{sum}}(\tau) \leq 1$, Equation 7 is always satisfied.*

P2: *If $\delta_{\text{sum}}(\tau) \leq \frac{1-\rho(\tau)}{2}$, then Equation 6 is always satisfied.*

P3: *Let τ be an implicit-deadline system. If $u_{\text{sum}}(\tau) \leq 1 - \rho(\tau)$, then both Equations 6 and 7 are always satisfied.*

Proof: P1 is trivially true, since violating Equation 7 requires that $(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j)$ exceed 1.

To see P2, observe that $\delta_{\text{sum}}(\tau) \leq \frac{1-\rho(\tau)}{2}$ implies that $\sum_{\tau_j \in \tau} \text{DBF}(\tau_j, t_0) \leq \frac{t_0(1-\rho(\tau))}{2}$ for all $t_0 \geq 0$. By Inequality 3, this in turn implies that

$$\sum_{\tau_j \in \tau} \text{DBF}^*(\tau_j, t_0) \leq t_0(1-\rho(\tau)) \qquad (15)$$

for all $t_0 \geq 0$; specifically at $t_0 = d_i$ when evaluating Equation 6 for τ_i. Evaluating Equation 15 at $t_0 = d_i$ implies that $e_i + \sum_{j=1}^{i-1} \text{DBF}^*(\tau_j, d_i) \leq d_i - q_{\max}(\tau)$. Since $\tau(\pi_k) \subseteq \{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ when attempting to add τ_i to processor π_k in NP-PARTITION, $e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \leq d_i - q_{\max}(\tau)$. This implies Equation 6.

To see P3, observe that

$$u_{\text{sum}}(\tau) \leq 1 - \rho(\tau) \qquad (16)$$

trivially implies Equation 7. It remains to show that Equation 6 is satisfied. In an implicit-deadline system, it follows from Equation 2 that DBF^* can be rewritten as:

$$\text{DBF}^*(\tau_i, t) = \begin{cases} 0, & \text{if } t < d_i \\ u_i t, & \text{otherwise} \end{cases} \qquad (17)$$

By multiplying both sides of Equation 16 by d_i, we obtain

$$\begin{aligned} & \textstyle\sum_{j=1}^{n} u_j d_i \leq d_i - q_{\max}(\tau) \\ \Rightarrow \quad & e_i + \textstyle\sum_{\substack{j=1 \\ j \neq i}}^{n} u_j d_i \leq d_i - q_{\max}(\tau) \\ \Rightarrow \quad & e_i + \textstyle\sum_{\substack{j=1 \\ j \neq i}}^{n} \text{DBF}^*(\tau_j, d_i) \leq d_i - q_{\max}(\tau) \\ \Rightarrow \quad & e_i + \textstyle\sum_{j=1}^{i-1} \text{DBF}^*(\tau_j, d_i) \leq d_i - q_{\max}(\tau). \end{aligned}$$

Since $\tau(\pi_k) \subseteq \{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ when attempting to add τ_i to processor π_k in NP-PARTITION, the last inequality implies Equation 6. ■

Corollary 1

1. *Any restricted-preemption sporadic task system τ satisfying $(u_{\text{sum}}(\tau) \leq 1 \bigwedge \delta_{\text{sum}}(\tau) \leq \frac{1-\rho(\tau)}{2})$ is successfully partitioned on any number of processors ≥ 1 by* NP-PARTITION.

2. *Any constrained, restricted-preemption sporadic task system τ satisfying $(\delta_{\text{sum}}(\tau) \leq \frac{1-\rho(\tau)}{2})$ is successfully partitioned on any number of processors ≥ 1 by* NP-PARTITION.

3. *Any implicit-deadline, restricted-preemption sporadic task system τ satisfying $(u_{\text{sum}}(\tau) \leq 1-\rho(\tau))$ is successfully partitioned on any number of processors ≥ 1 by* NP-PARTITION.

Proof: Part 1 and 3 follow directly from Lemma 3, Equations 6 and 7 of NP-PARTITION will always evaluate to "true."

Part 2 follows directly from Lemmas 3 and 1. By Lemma 1, we need only determine that Equation 6 is satisfied. By Property P2 of Lemma 3, this is ensured by having $\delta_{\text{sum}}(\tau) \leq \frac{1-\rho(\tau)}{2}$.

■

We are now prepared to prove sufficient conditions for NP-PARTITION successfully partitioning an implicit-deadline, restricted-preemption sporadic task system.

Theorem 3 *Any implicit-deadline, restricted-preemption sporadic task system τ where $\rho(\tau) < 1 - u_{\text{sum}}(\tau)$ is successfully scheduled by* NP-PARTITION *on m unit-capacity processors, for any*

$$m \geq \frac{u_{\text{sum}}(\tau) - u_{\max}(\tau)}{1 - \rho(\tau) - u_{\max}(\tau)} \tag{18}$$

Proof Sketch: The proof is by contradiction. Assume that Equation 18 is true, but NP-PARTITION fails to partition τ. Then there exists a task $\tau_i \in \tau$ such that when attempting to add τ_i to each processor $\pi_k \in \Pi$, either Equation 6 or 7 is violated. Corollary 1 implies for each $\pi_k \in \Pi$

$$u_{\text{sum}}(\tau(\pi_k) \cup \{\tau_i\}) > 1 - \rho(\tau) \tag{19}$$

because otherwise when attempting to assign τ_i to processor π_k, NP-PARTITION would be able to "fit" $\tau(\pi_k)$ and τ_i on the same processor.

Summing Inequality 19 over all $\pi_k \in \Pi$, and noting that the tasks on these processors is a subset of τ, we obtain

$$\begin{aligned} & u_{\text{sum}}(\tau) + (m-1)u_i > m(1-\rho(\tau)) \\ \Rightarrow \quad & m(1-\rho(\tau)-u_i) < u_{\text{sum}}(\tau) - u_i \\ \Rightarrow \quad & m < \tfrac{u_{\text{sum}}(\tau)-u_i}{1-\rho(\tau)-u_i} \end{aligned}$$

Observe that $u_{\text{sum}}(\tau) > 1 - \rho(\tau)$; otherwise, τ would be trivially feasible according to Corollary 1. Therefore, the left-hand side of the above inequality is maximized when u_i is as large as possible. This implies,

$$m < \frac{u_{\text{sum}}(\tau) - u_{\max}(\tau)}{1 - \rho(\tau) - u_{\max}(\tau)}$$

which contradicts our assumption, thereby proving the lemma. ■

In the next lemmas, we describe the necessary conditions for algorithm NP-PARTITION failing to assign τ_i to a processor in Π. If τ_i was not assigned to a processor then either Equation 6 or 7 evaluated to false for each $\pi_k \in \Pi$. Lemma 4 quantifies the maximum number of processors for which Equation 6 is false; Lemma 5 quantifies the maximum number of processors for which Equation 7 is false.

Lemma 4 *Let m_1 denote the number of processors, $0 \leq m_1 \leq m$, on which Equation 6 fails when the partitioning algorithm is attempting to map τ_i. Assuming that $\rho(\tau) < 1 - \delta_i$,it must be the case that*

$$m_1 < \frac{2\delta_{\text{sum}}(\tau) - \delta_i}{1 - \rho(\tau) - \delta_i} \tag{20}$$

Proof: Let Π_1 be the set of m_1 processor for which Equation 6 evaluates to false when attempting to add task τ_i. Then, for each $\pi_k \in \Pi_1$,

$$\sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) > d_i - e_i - q_{\max}(\tau)$$

Summing over all $\pi_k \in \Pi_1$, and noting that $\bigcup_{\pi_k \in \Pi_1} \tau(\pi_k) \subseteq \tau$, we obtain

$$\begin{aligned} & \sum_{j=1}^{n} \text{DBF}^*(\tau_j, d_i) > (d_i - e_i - q_{\max}(\tau))m_1 + e_i \\ \Rightarrow \quad & \text{(by Inequality 3)} \\ & 2\textstyle\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) > (d_i - e_i - q_{\max}(\tau))m_1 + e_i \\ \Rightarrow \quad & \tfrac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} > \tfrac{m_1}{2}(1 - \delta_i - \tfrac{q_{\max}(\tau)}{d_i}) + \tfrac{e_i}{2d_i} \end{aligned} \tag{21}$$

By definition of $\delta_{\text{sum}}(\tau)$

$$\frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} \leq \delta_{\text{sum}}(\tau). \tag{22}$$

Chaining Inequalities 21 and 22, and observing that $\rho(\tau) \geq \frac{q_{\max}(\tau)}{d_i}$, we obtain

$$\begin{aligned} & \tfrac{m_1}{2}(1 - \delta_i - \rho(\tau)) + \tfrac{e_i}{2d_i} < \delta_{\text{sum}}(\tau) \\ \Rightarrow \quad & m_1 < \tfrac{2\delta_{\text{sum}}(\tau)-\delta_i}{1-\rho(\tau)-\delta_i} \end{aligned}$$

which is claimed by the lemma. ■

Lemma 5 *Let m_2 denote the number of processors, $0 \leq m_2 \leq m$, on which Equation 7 fails when the partitioning algorithm is attempting to map τ_i. It must be the case that*

$$m_2 < \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i} \tag{23}$$

Proof: Let Π_2 be the set of m_2 processor for which Equation 7 evaluates to false when attempting to add task τ_i. Then, for each $\pi_k \in \Pi_2$,

$$1 - u_i < \sum_{\tau_j \in \tau(\pi_k)} u_j$$

Summing over all $\pi_k \in \Pi_2$, and noting that $\bigcup_{\pi_k \in \Pi_2} \tau(\pi_k) \subseteq \tau$, we obtain

$$\begin{aligned} & (1-u_i)m_2 + u_i < \textstyle\sum_{j=1}^{n} u_j \\ \Rightarrow \quad & m_2 < \tfrac{u_{\text{sum}}(\tau)-u_i}{1-u_i} \end{aligned}$$

which is asserted by the lemma. ■

We are now prepared to prove sufficient conditions for success of NP-PARTITION on constrained and arbitrary task systems.

Theorem 4 *Any constrained, restricted-preemption sporadic task system τ where $\rho(\tau) < 1 - \delta_{\max}(\tau)$ is successfully scheduled by* NP-PARTITION *on m unit-capacity processors, for any*

$$m \geq \frac{2\delta_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \rho(\tau) - \delta_{\max}(\tau)} \tag{24}$$

Proof: The proof is by contradiction. Assume that task system τ and platform Π satisfy Inequality 24, but that NP-PARTITION fails to assign some task $\tau_i \in \tau$ to any processor. By Lemma 1, it must be the case that Equation 6 fails for task τ_i on each of the m processors (i.e. m equals m_1 from Lemma 4). Consequently, Lemma 4 implies

$$m < \frac{2\delta_{\text{sum}}(\tau) - \delta_i}{1 - \rho(\tau) - \delta_i}$$

By the second part of Corollary 1, $\delta_{\text{sum}}(\tau) > \frac{1-\rho(\tau)}{2}$; otherwise, τ can trivially be partitioned by NP-PARTITION. In this case, the right-hand side of the above inequality is maximized when δ_i is as large as possible. Thus,

$$m < \frac{2\delta_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \rho(\tau) - \delta_{\max}(\tau)}$$

which contradicts Inequality 24. ■

Theorem 5 *Any restricted-preemption sporadic task system τ where $\rho(\tau) < 1 - \delta_{\max}(\tau)$ is successfully scheduled by* NP-PARTITION *on m unit-capacity processors, for any*

$$m \geq \frac{2\delta_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \rho(\tau) - \delta_{\max}(\tau)} + \frac{u_{\text{sum}}(\tau) - u_{\max}(\tau)}{1 - u_{\max}(\tau)} \tag{25}$$

Proof Sketch: The proof is by contradiction. Assume that task system τ and platform Π satisfy Inequality 25, but that NP-PARTITION fails to assign some task $\tau_i \in \tau$ to any processor. Let Π_1 be the set of m_1 processor on which Equation 6 evaluates to false while attempting to assign τ_i. Let Π_2 be the remaining m_2 processors (i.e. $m_2 \stackrel{\text{def}}{=} m - m_1$) on which Equation 7 evaluates to false.

According to Part 1 of Corollary 1, $(u_{\text{sum}}(\tau) > 1)$ or $(\delta_{\text{sum}}(\tau) > \frac{1-\rho(\tau)}{2})$; otherwise, NP-PARTITION could trivially partition τ on a single processor. We will consider three separate cases and show that in each case a contradiction will arise. Due to space requirements we only fully show the first case.

Case(i): ($\delta_{\text{sum}}(\tau) > \frac{1-\rho(\tau)}{2}$ and $u_{\text{sum}}(\tau) > 1$).: In this case, both m_1 and m_2 are non-zero. Summing Inequalities 20 and 23 of Lemmas 4 and 5 (respectively), we obtain

$$m = m_1 + m_2 < \frac{2\delta_{\text{sum}}(\tau) - \delta_i}{1 - \rho(\tau) - \delta_i} + \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i}$$

Because ($\delta_{\text{sum}}(\tau) > \frac{1-\rho(\tau)}{2}$), ($u_{\text{sum}}(\tau) > 1$), and ($\rho(\tau) < 1 - \delta_{\max}(\tau)$), the right-hand side of the above inequality is maximized when both δ_i and u_i are as large as possible. Therefore,

$$m < \frac{2\delta_{\text{sum}}(\tau) - \delta_{\max}(\tau)}{1 - \rho(\tau) - \delta_{\max}(\tau)} + \frac{u_{\text{sum}}(\tau) - u_{\max}(\tau)}{1 - u_{\max}(\tau)}$$

which contradicts Inequality 25.

Case(ii): ($\delta_{\text{sum}}(\tau) > \frac{1-\rho(\tau)}{2}$ and $u_{\text{sum}}(\tau) \leq 1$). Similar to Case (i) via a simple application of Lemma 4.

Case(iii): ($\delta_{\text{sum}}(\tau) \leq \frac{1-\rho(\tau)}{2}$ and $u_{\text{sum}}(\tau) > 1$). Similar to Case (i) via a simple application of Lemma 5.

■

3.3 Quantum-based Preemptive Scheduling

A *quantum-based* scheduler allocates a processor to tasks in "blocks" of time called *quantums*. During a quantum a task may execute non-preemptively until the earliest of the following two events occurs: the task completes execution or its quantum expires. In the event of task completion, the scheduler is invoked, and the next available task (according to the scheduling algorithm) with remaining execution is assigned the newly idle processor. If the quantum expires prior to the completion of the task, the task is preempted and the scheduling algorithm makes a decision about what task to execute in the next quantum (possibly the same task). In this model, a job of a task can be "blocked" by a lower-priority (assuming a priority-based scheduling algorithm) only if it arrives in the middle of a quantum. Therefore, the maximum blocking time for any task in the system is equal to one quantum. We will make the following simplifying assumptions about the system:

1. The scheduler is invoked only at the completion of the execution of a job, or at the end of the quantum.
2. The scheduler's execution requirement is negligible.
3. The quantum-size is less than the execution requirement of each task in the system.
4. The quantum sizes are fixed and identical on each processor.

NP-PARTITION is a highly applicable partitioning algorithm for these quantum-based systems. We can model a quantum-based system by setting, for each task $\tau_i \in \tau$, q_i equal to the quantum-size. Since NP-PARTITION reserves $q_{\max}(\tau)$ units of "slack" at every instance of time on each processor, this guarantees that each task assigned to a processor can still meet its deadline despite being blocked for a quantum's duration of time. In fact, even an optimal partitioning algorithm for a multiprocessor quantum-based must ensure that a job can be delayed

for one quantum. Therefore, we can evaluate the performance of NP-PARTITION in quantum-based systems using a technique know as *resource augmentation* [17]. Resource algorithm compares a given algorithm against a hypothetical optimal algorithm and determines the factor by which we augment the processing platform for the given algorithm to match the performance of the optimal. For the purpose of this paper, the resource augmentation technique is as follows: given a task system that is known to be feasible upon a particular platform (with respect to quantum-based scheduling), we determine the multiplicative factor of speed by which we must augment our platform in order for NP-PARTITION to return PARTITIONING SUCCEEDED (shown in Theorem 6).

First, we need a technical lemma that characterizes the necessary conditions for feasibility. In [7], Baruah and Fisher showed for preemptive systems that the larger of $\delta_{\max}(\tau)$ and $u_{\max}(\tau)$ represents the maximum computational demand of any task, and the larger of $\delta_{\text{sum}}(\tau)$ and $u_{\text{sum}}(\tau)$ represents the maximum computational demand of a preemptive sporadic task system τ. This result trivially extends to restricted-preemption systems.

Lemma 6 (from [7]) *If task system τ is feasible (under either the partitioned or the global paradigm) on an identical multiprocessor platform comprised of m processors of computing capacity ξ each, it must be the case that*

$$\xi \geq \max(\delta_{\max}, u_{\max}) \ ,$$

and

$$m \cdot \xi \geq \max(\delta_{\text{sum}}, u_{\text{sum}}) \ .$$

Theorem 6 *Given an identical multiprocessor platform Π with m processors and a restricted-preemption sporadic task system τ feasible on Π without IIT, the* NP-PARTITION *algorithm has the following performance guarantees for a quantum-based system with quantum-size equal to $q_{\max}$ (i.e. $\rho(\tau) = \frac{q_{\max}}{d_{\min}(\tau)}$):*

1. *if τ is an implicit-deadline system, then* NP-PARTITION *will successfully partition τ upon a platform comprised of m processors that are each $\left(\frac{2-\frac{1}{m}}{1-\rho(\tau)}\right)$ times as fast as the processors of Π.*
2. *if τ is a constrained system, then* NP-PARTITION *will successfully partition τ upon a platform comprised of m processors that are each $\left(\frac{3-\frac{1}{m}}{1-\rho(\tau)}\right)$ times as fast as the processors of Π.*
3. *if τ is an arbitrary system, then* NP-PARTITION *will successfully partition τ upon a platform comprised of m processors that are each $\left(\frac{4-\frac{2}{m}}{1-\rho(\tau)}\right)$ times as fast as the processors of Π.*

Proof Sketch: Due to space considerations, we will only show the proof of the third claim. The other claims can be proven similarly.

Assume that we are given arbitrary task system τ feasible (without IIT) on m processors each of speed ξ, it follows from Lemma 6 that τ must satisfy the following properties:

$$\begin{array}{ll} \delta_{\text{sum}}(\tau) \leq m \cdot \xi & u_{\text{sum}}(\tau) \leq m \cdot \xi \\ \delta_{\max}(\tau) \leq \xi & u_{\max}(\tau) \leq \xi \end{array}$$

Suppose that τ is successfully scheduled on m unit-capacity processor by NP-PARTITION. By substituting the inequalities above in Equation 25 of Theorem 5, we get

$$\begin{array}{rcl} m \geq \frac{2\delta_{\text{sum}}(\tau)-\delta_{\max}(\tau)}{1-\rho(\tau)-\delta_{\max}(\tau)} + \frac{u_{\text{sum}}(\tau)-u_{\max}(\tau)}{1-u_{\max}(\tau)} & & \\ \Leftarrow & m \geq \frac{2m\xi-\xi}{1-\rho(\tau)-\xi} + \frac{m\xi-\xi}{1-\xi} & \\ \Leftarrow & m \geq \frac{2m\xi-\xi}{1-\rho(\tau)-\xi} + \frac{m\xi-\xi}{1-\rho(\tau)-\xi} & \\ \equiv & \xi \leq \frac{m(1-\rho(\tau))}{4m-2} & \\ \equiv & \frac{1}{\xi} \geq \frac{4-\frac{2}{m}}{1-\rho(\tau)} & \end{array}$$

which is claimed in the third part of the theorem. ■

4 Heuristics

Though NP-PARTITION is useful for theoretically evaluating preemptive quantum-based systems, it pessimistically assumes that any task in the system could potentially blocked for $q_{\max}(\tau)$ time units (Equation 6). Therefore, the algorithm is impractical for all but small values of $\rho(\tau)$ and entirely unusable for $\rho(\tau) > 1$. In a general restricted-preemption system, we may not need assume the same maximum non-preemption parameter each processor. In this section, we will consider a family of polynomial-time heuristics based on the first-fit-decreasing bin-packing heuristic. Section 4.1 will describe each of the heuristics we consider. Section 4.2 will present our empirical-evaluation methodology and results. We will also discuss potential theoretic justifications for the experimental results.

4.1 Heuristic Descriptions

As mentioned in the beginning of Section 3, typical partitioning heuristics will first sort the tasks according to some criteria, and then assign the tasks to a processor according to a sufficient condition for "fitting". The NP-PARTITION algorithm matches this heuristic pattern by sorting tasks in (non-decreasing) order of relative deadline and adding tasks to processors (in order) according to Equations 6 and 7. In this section, we consider a family of algorithms that each sorts tasks in non-increasing order according to different criteria and assigns a task to the first processor upon which it fits (each algorithm is a variant of first-fit-decreasing). The conditions for fitting are the same for each heuristic considered and are a more optimistic sufficient conditions than used for NP-PARTITION (i.e. a task is more likely to be assigned to a processor). The following are the eleven different sorting criteria: $\frac{1}{d_i}$ (same as NP-PARTITION), $\frac{1}{p_i}$, e_i, q_i, u_i, $\lambda_i \stackrel{\text{def}}{=} \frac{e_i}{\min(d_i,p_i)}$,

δ_i, $\widehat{u}_i \stackrel{\text{def}}{=} \frac{q_i}{p_i}$, $\widehat{\lambda}_i \stackrel{\text{def}}{=} \frac{q_i}{\min(d_i,p_i)}$, $\widehat{\delta}_i \stackrel{\text{def}}{=} \frac{q_i}{d_i}$, and *random* order. Each heuristic is denoted by FFD-NP-$\langle$*sorting-criteria*$\rangle$.

For each the above heuristics, we will attempt to add tasks $\tau_i \in \tau$ to processors of Π in the order specified. Since we have potentially changed the order that tasks are assigned, Lemma 2 is not necessarily valid. Therefore, every time we assign a task τ_i to a processor π_k, we must check that each task in $\tau(\pi_k)$ meets its deadline after the addition of τ_i. We will add a task τ_i to processor π_k only if it does not affect the EDF-feasibility of the tasks of $\tau(\pi_k)$.

Equations 6 and 7 of the NP-PARTITION algorithm ensured that Equations 4 and 5 of Theorem 1 are not violated, by leaving enough slack on a processor to fit the job with the largest non-preemption parameter. We can consider a more optimistic test by making the following observation: *since we are considering* EDF*-schedulability, a task τ_j on a processor may only be blocked (i.e. a priority inversion occurs) if a task with a larger relative deadline is executing when a job of τ_j arrives.* Therefore, when checking if task τ_i fits on a processor π_k, we need only check if all tasks $\tau_j \in \tau(\pi_k)$ with $d_j < d_i$ have enough slack to accommodate being blocked by τ_i for q_i time units. We must also ensure that the added demand placed on π_k by τ_i after time d_i does not affect the slack necessary to accommodate jobs of $\tau_j \in \tau(\pi_k)$ with $d_j > d_i$. Therefore, we replace Equation 6 with the following conditions: for each $\tau_\ell \in \{\tau\} \cup \tau(\pi_k)$,

$$\left(d_\ell - \sum_{\substack{\tau_j \in \tau(\pi_k) \\ d_j < d_\ell}} \text{DBF}^*(\tau_j, d_\ell) \right) \geq e_\ell + \max_{\substack{\tau_j \in \{\tau\} \cup \tau(\pi_k) \\ d_j > d_\ell}} \{q_j\}. \quad (26)$$

It turns out that if Equations 26 and 7 are true, then we can safely add τ_i to processor π_k. The full proof of correctness is omitted due to space considerations.

Run-time Complexity. For each $\pi_k \in \Pi$, let i_k be the number of tasks assigned to processor π_k at the time we are attempting to assign task τ_i. For each $\tau_\ell \in \{\tau_i\} \cup \tau(\pi_k)$, it takes $\mathcal{O}(i_k)$ time to evaluate the $\sum \text{DBF}^*(\tau_j, d_\ell)$ term and $\mathcal{O}(1)$ time to evaluate $\max\{q_j\}$ term in Equation 26. Therefore, the time complexity for testing if τ_i fits on processor π_k is $\mathcal{O}(i_k^2)$. This implies the overall time-complexity of each of the heuristics is $\mathcal{O}(n^4)$.

4.2 Empirical Evaluation

Methodology. For empirically evaluating each of the heuristics, we generated a set of pseudo-random tasks sets in a manner similar to Baker [3]. We pseudo-randomly generate a partition of $m+1$ preemptive tasks. That is, each processor is assigned a set of randomly generated tasks which are preemptively feasible on the processor, and the total number of tasks in this initial set is $m+1$. The method of generating an individual task $\tau_i = (e_i, q_i, d_i, p_i)$ is as follows: we generate a random utilization parameter u_i for the task according to the exponential distribution with mean individual task utilization of 0.25. We then randomly generate the period p_i from a uniform distribution with values ranging from 1 to 1000. The execution requirement e_i is computed from the task utilization and period. The relative deadline d_i of a task is an integer value chosen from the uniform distribution with range $[e_i, 2p_i]$.

A partition of these $m+1$ tasks is randomly generated such that each task is preemptively feasible. If $\tau(\pi_k)$ is the set of tasks from the initial task set assigned to processor π_k, the condition of feasibility is $\sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t) \leq t$ for all $t \geq 0$. After determining an initial partition, we compute a maximum non-preemption parameter $\hat{q}_i$ for each task in this partition, and assign to q_i a value chosen from the exponential distribution with range $[0, \min(e_i, \hat{q}_i)]$. To generate subsequent task sets, we add a randomly generated task to the current partition and attempt to fit it on any processor. If the task fits on the processor, we add the resulting task set to our sample; otherwise, we start over with a newly generated set of $m+1$ tasks.

We generated several different sets of task systems according to the methodology described. However, due to space consideration, we discuss only one set of 1,000,000 randomly generated task systems. Each of the task systems generated are feasible on a four processor platform. Figure 1 shows the distribution of generated task systems with respect to a task system's δ_{sum} value ([4] gives justification for using δ_{sum} as metric of comparison).

Results and Discussion. The results of running each of the eleven heuristics over the sample set are shown in Figure 2. The values are shown for $\delta_{\text{sum}} \geq 2.5$ only, as the smaller δ_{sum}-valued task systems are partitionable by all heuristics. The major trend we have observed from this experiment and other experiments (omitted for space) are that heuristics that have the non-preemption parameter q_i in the sorting criteria (i.e. FFD-NP-q_i, FFD-NP-$\widehat{u}_i$, FFD-NP-$\widehat{\lambda}_i$, and FFD-NP-$\widehat{\delta}_i$) dominate heuristics that exclude q_i for $\delta_{\text{sum}} \leq 3.8$. Indeed, FFD-NP-$q_i$ dominates all heuristics for $\delta_{\text{sum}} \leq 3.8$. A potential reason for the domination of the heuristics using q_i is that by assigning tasks with larger non-preemption parameter first, we more effectively utilize the slack on each processor. If we ignored the q_i parameter in ordering tasks, we may not leave enough room for future tasks with large q_i. However, according to Figure 2, heuristics based on the task load or density (FFD-NP-λ_i and FFD-NP-δ_i) may be more effective than q_i-based heuristics for task systems with high δ_{sum} values. Not surprisingly, FF-NP-RANDOM does the worst and is dominated by all other heuristics since it does not exploit any information about the task system.

5 Conclusions

Non-preemptive scheduling of tasks has the advantage of decreasing system complexity and scheduling overheads. However, little work has been done on studying non-preemption for partitioned multiprocessor systems. In this paper, we considered the parititoned multiprocessor scheduling of restricted-preemption and non-preemptive sporadic task systems. We introduced a simple partitioning algorithm NP-PARTITION for which we

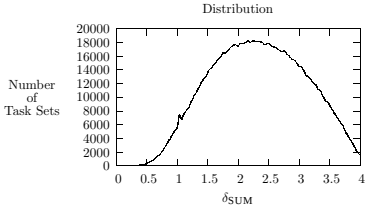


Figure 1. Distribution of δ_{sum} values for randomly generated task sets

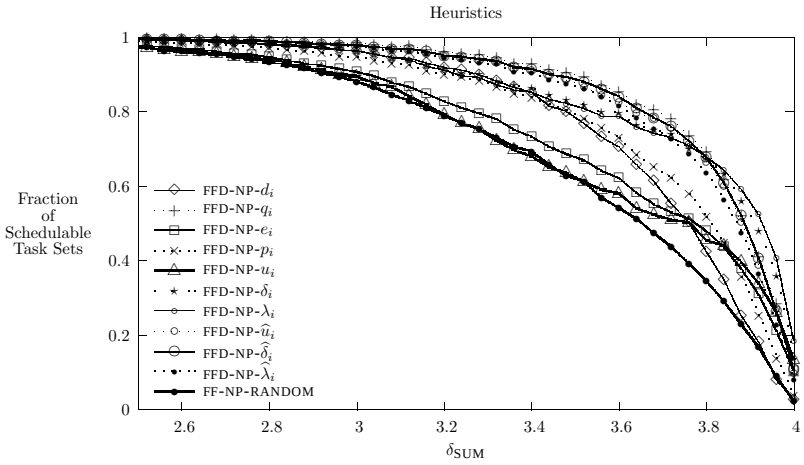


Figure 2. Comparison of partitioning heuristics

derived sufficient conditions for success (Theorems 3, 4, and 5). NP-PARTITION can be applied to partitioning preemptive sporadic tasks in a multiprocessor system that uses quantum-based scheduling. For quantum-based systems, we were able to characterize the effectiveness of NP-PARTITION in terms of resource augmentation bounds (Theorem 6).

To address the drawbacks of the pessimistic behavior of NP-PARTITION, we considered eleven different partitioning heuristics for restricted-preemption sporadic task systems. We characterized the performance of these heuristics empirically over a set of randomly generated task systems, and observed that heuristics which use the non-preemption or load information of the task system outperform heuristics that ignore this information. In the future, we would like to obtain better theoretical bounds for the heuristics presented in Section 4.

References

[1] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 187–195, Catania, Sicily, July 2004. IEEE Computer Society Press.

[2] T. P. Baker. Stack-based scheduling of real-time processes. *Real-Time Systems: The International Journal of Time-Critical Computing*, 3, 1991.

[3] T. P. Baker. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Technical Report TR-050601, Department of Computer Science, Florida State University, 2005.

[4] T. P. Baker, N. Fisher, and S. Baruah. Algorithms for determining the load of a sporadic task system. Technical report, Department of Computer Science, Florida State University, 2005.

[5] S. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. *Real-Time Systems: The International Journal of Time-Critical Computing*. Accepted for publication.

[6] S. Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 137–144, Palma de Mallorca, Balearic Islands, Spain, July 2005. IEEE Computer Society Press.

[7] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Miami, Florida, December 2005. IEEE Computer Society Press.

[8] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.

[9] P. Gai, G. Lipari, and M. di Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, December 2001.

[10] L. George, P. Muhlethaler, and N. Rivierre. Optimality and non-preemptive real-time scheduling revisited. Technical Report RR-2516, INRIA: Institut National de Recherche en Informatique et en Automatique, 1995.

[11] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Technical Report RR-2966, INRIA: Institut National de Recherche en Informatique et en Automatique, 1996.

[12] K. Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the 12th Real-Time Systems Symposium*, pages 129–139, San Antonio, Texas, December 1991. IEEE Computer Society Press.

[13] D. S. Johnson. *Near-optimal Bin Packing Algorithms*. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, 1973.

[14] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[15] J. M. Lopez, J. L. Diaz, and D. F. Garcia. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems: The International Journal of Time-Critical Computing*, 28(1):39–68, 2004.

[16] A. K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[17] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, Texas, 4–6 May 1997.

Probabilistic QoS Assessment of Tasks with Uncertain Parameters in Preemptive Multi–Processor Scheduling

Amare Leulseged and Nimal Nissanke
Institute for Computuing Research, Faculty of BCIM
London South Bank University, London, UK

Abstract

This paper presents a probabilistic approach for Quality of Service (QoS) assessment of periodic tasks with uncertainties in computation times and deadlines in priority–driven multi–processor execution environments. The approach is a generally applicable one but is aimed at novel non–critical real–time applications (e.g. multimedia). Uncertainties bear a truly random character at task request times, though in between arrival times they can be worked out by tracing the execution process in time. Considering each task in turn, its possible execution scenarios are systematically worked out in relation to other tasks in the system. This permits the computation of the probability of task execution at each instant of time and, hence, the probability distribution of each task in the next instant in time, leading in turn to the determination of QoS indicators such as success rate, expectation of response time latency and jitter between consecutive task instances in a straightforward manner. A stochastic simulation demonstrates the validity of the approach.

1 Introduction

The objective of this paper is to develop a sufficiently general probabilistic framework for the analysis of multiprocessor schedulability of tasks with uncertain parameters. Though no specific assumptions are made to that effect, the approach is considered particularly suitable for assessing the Quality of Service (QoS) indicators of individual tasks with uncertainties in their resource requirements. As is well known, QoS criteria are becoming imperative in the design of non–critical real-time applications such as video conferencing and other multimedia applications, where traditional techniques based on worst case execution times invariably result in highly conservative uneconomic designs.

Among the works devoted to uncertainties in real-time tasks and their implications on scheduling is the Statistical Rate Monotonic Scheduling approach [1, 2] dealing with periodic tasks with variable execution times expressed through a probability density function and a QoS measure defined in terms of the failure probability of tasks. An important part of its strategy, however, is a job admission controller intended to safeguard the QoS of tasks already admitted. The work by Manolache et al. [11] deals with performance analysis of periodic non–preemptable tasks with the uncertainties in their execution times specified in the form of a continuous probability distribution function. A motivation for the latter is to avoid the complexity associated with discrete states, though an important source of the efficiency of this technique appears to lie in the analysis conducted alongside the construction of the underlying stochastic process. Based on Markov process modelling and sharing the same aspirations as this paper, the work [5] proposes a stochastic approach to fixed priority and dynamic scheduling of periodic tasks with uncertainties in their computation times. However, the approach is limited to uni–processor scheduling. Gardner [7] proposes Stochastic Time Demand Analysis for determining the lower bound on execution rates in uni–processor fixed priority context. This is based on an extension of the periodic task model to include both a guaranteed execution time ranging from zero to the maximum execution time and a guaranteed inter–release time above the required minimum inter–release time, depending on the real-time nature of the tasks concerned. Zhou et al. [14] propose a modified rate monotonic schedulability analysis, incorporating two experimentally determined parameters to account for uncertainties in operating system overheads: a constant representing the CPU utilisation of operating system activities and a worst–case timer delay factor. Edgar et al. [6] propose the use of Gumbel distribution for estimating the worst–case execution time as an extreme value. Another important contribution is the Real–Time Queueing Theory [9], though it is yet to demonstrate its application in relation to any practically significant problems.

Based on a discrete time model, our work deals with dynamic multi-processor real-time scheduling [8, 10, 13] and with static and dynamic uni-processor real-time scheduling [12]. Though the discrete time model generally raises the complexity of computations involved, it offers the capability to analyse, in principle, problems of any arbitrary nature and not necessarily those which fit in with standard mathematical models. The work [13] deals with a probabilistic analysis of dynamic multi–processor scheduling with emphasis on the overall performance of the scheduling environment as a whole. However, lacking the means to identify individual tasks, the latter could not

address the QoS performance of individual tasks except in a limited sense. In contrast, the works [8, 10, 12] shift the focus to scheduling of individual tasks, though the approaches taken in [8, 10] for multi-processor case and [12] for uni-processor case are completely different. The approach in [12] uses 'timed sequences of probabilities' for working out the execution patterns and termination times of tasks, allowing the calculation of a range of QoS indicators. Demonstrative examples in [5] and [7] being limited to uni-processor scheduling, it is easier to compare, amongst our work, [12] against [5, 7]. In this respect, the results in [12] are broadly in line with the simulation in [7] and the analytical results in [5], as well as our own simulation results.

This paper and the thesis [8], [10] being a preliminary version of the same, propose a new framework for probabilistic analysis of tasks with uncertainties in multi–processor scheduling and, hence, their QoS issues on an individual basis. Uncertainties concern only computation times and deadlines expressed indirectly through laxities (a measure of urgency). These are represented in terms of probability mass functions (PMF) and are of truly random nature at the request times of tasks. The essence of the approach is in how to work out their PMFs in between request times as the tasks are executed using any priority–driven scheduling algorithm, including algorithms such as the Least Laxity First (LLF) and the Earliest Deadline First (EDF). This forms the basis for arriving at several QoS indicators of practical interest, e.g. rate of successful execution, response time latency, jitter and so forth. Extending the work [10], this paper deals with several important issues, in particular, a verification of the approach using a stochastic simulation, an assessment of the complexity of the approach as presented here, a more accurate way of calculating jitter and a study of the effect of tie–breaking on probabilistic predictions.

Section 2 presents the core ideas of the proposed framework. Section 3 examines practically useful performance and QoS indicators. Section 4 sets out the relationship of the above with a stochastic simulation. Section 5 illustrates the use of the approach and demonstrates its validity using a stochastic simulation. Section 6 concludes the paper with a summary of achievements.

2 Analytical Framework

2.1 Representation of Tasks

Considering a set Ω of n tasks τ_i, $i = 1, 2, \cdots, n$, let each task τ_i arrive at fixed time intervals separated by a period T_i but with an unpredictable computation time C_i. For generality, as justified in [8], the deadline of τ_i is also considered unpredictable and is specified in terms of laxity (urgency) L_i, that is, the length of time from the current time to the deadline minus the computation time. Thus, L_i and C_i are both treated as random variables at the time of τ_i's each arrival. Their sampling spaces are the sets of integers $0 \,..\, l_{max}$ and $1 \,..\, c_{max}$ respectively, the notation $x \,..\, y$ denoting the set of integers from x to y inclusively, and c_{max} and l_{max} representing, respectively, the maximum envisaged values of the computation time and the laxity.

A convenient visualisation of tasks with particular values of l and c is to view them as points on a two–dimensional orthogonal (discrete) coordinate system with l (horizontal) and c (vertical) as its axes; see Figure 1(a). This system is referred to as the l–c space. The area enclosed within $0 \leqslant l \leqslant l_{max}$ and $1 \leqslant c \leqslant c_{max}$ is referred to as the *scheduling domain* and is denoted by S. By including $l =$ -1 in this representation, we use the term *exit domain*, and the symbol E, to refer to the points on $l =$ -1, or on $c = 0$, but excluding the point (-1, 0). Given a task at a particular point (l, c), if it were to be executed at time t, its position at time $(t + 1)$ would be at $(l, c - 1)$, otherwise at $(l - 1, c)$. In the case of tasks which have exited the system, any tasks located on $l =$ -1 are failed tasks, while those on $c = 0$ are successfully executed ones. The scheduling and exit domains together are referred to as the *task domain* and is denoted by $\bar{S}$. Uncertainties in task parameters are taken into account in a general form by treating L_i and C_i as jointly discrete random variables with a joint probability mass function (PMF) $p_{L_i,C_i}(l, c)$ or, for brevity, by $p_i(l, c)$.

$$p_i(l,c) \quad = \quad \mathcal{P}_{L_i,C_i}[\,\{\,(l',c') : l' = l, c' = c\,\}\,] \qquad (1)$$

As a function of two random variables, the above denotes the probability of τ_i having a laxity l and computation time c at the time of τ_i's arrival or, referring to the two–dimensional representation, the probability of τ_i arriving at a point (l, c). Thus, referring to the illustration in Figure 1(a), τ_2 has a probability of 0.6 having a laxity 1 and a computation time 2.

As $p_i(l, c)$ in (1) keeps changing with time, the notation $p_i^t(l, c)$ denotes the probability of τ_i having the pair of values (l, c) at time t. $p_i^t(l, c)$ acquires its random value only at times when τ_i is freshly requested; i.e. when $t \bmod T_i$ is equal to zero. At other times, $p_i^t(l, c)$ is determined by $p_i^{t-1}(l, c)$ and the manner in which τ_i has been executed at time $(t - 1)$. In essence, the determination of $p_i^t(l, c)$ for each task τ_i and for each time value t over the scheduling domain lies at the heart of the given scheduling problem.

2.2 Scheduling Algorithms and Task Execution

The scheduling model consists of a multi-processor system with M identical processors, such that any task can be executed on any processor, and with no explicit consideration of preemption and task migration costs. Given an arbitrary priority–driven scheduling algorithm ϕ, a function $f_\phi(l, c)$ returns the priority level assigned by ϕ to any task located at (l, c) as a natural number K ranging from K_{min} to K_{max}. Priorities are assumed to decrease with increasing K. For example, in EDF $f_\phi(l, c)$ returns $(l + c)$ (i.e. relative deadline) as K and in LLF just l (i.e. laxity). At any particular instant in time, associated with any given value K, $K = f_\phi(l, c)$ for some (l, c), are three subsets of the set Ω of n tasks. These are:

1) $\Omega_1^K(l, c)$ – The set of all tasks having a non–zero probability of being at a point (x, y) in S such that $f_\phi(x, y) < K$; i.e. of being of a higher priority than any task at (l, c).

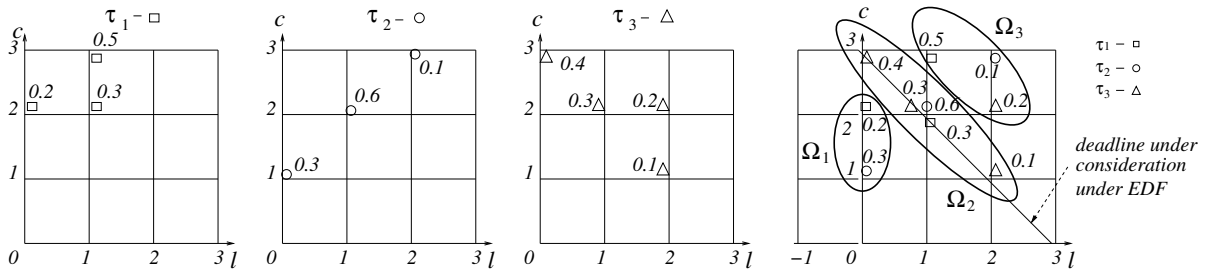


Figure 1. a) Distributions p_i for each $\tau_i, i = 1, 2, 3$ and b) task sets Ω_k, $k = 1, 2, 3$, in the task domain

2) $\Omega_2^K(l,c)$ – The set of all tasks having a non–zero probability of being at a point (x,y) in S such that $f_\phi(x,y) = K$; i.e. of being of an identical priority to any task at (l,c).
3) $\Omega_3^K(l,c)$ – The set of all tasks having a non–zero probability of being at a point (x,y) in S such that $f_\phi(x,y) > K$ or of being in the exit domain E; i.e. of being of a lower priority than any task at (l,c) or having exited the system.

The above sets are not necessarily pairwise disjoint, since a task may belong to more than one set at the same time. For brevity, from now on we use the simpler notation Ω_j^K, $j = 1, 2, 3$, for referring to the above sets. Figure 1(b) shows the task sets, $\Omega_j^3, j = 1, 2, 3$, for the set of tasks comprising $\tau_i, i = 1, 2, 3$, given in Figure 1(a), under EDF scheduling at a relative deadline of 3 units of time; i.e. with $\phi =$ EDF in $f_\phi(x,y)$ and $K = 3$. Thus, in Figure 1(b), $\Omega_1^3 = \{\tau_1, \tau_2\}$ and $\Omega_2^3 = \Omega_3^3 = \{\tau_1, \tau_2, \tau_3\}$.

Considering a particular task τ_i in Ω_2^K for scheduling, let us now select three sets ω_j, for $j = 1, 2, 3$, with p_j elements in each, such that $\tau_i \in \omega_2$ and

a) $\omega_j \subseteq \Omega_j^K$ (i.e., each ω_j is a subset of the corresponding Ω_j^K). As a result, $0 \leqslant p_j \leqslant \mid \Omega_j^K \mid$.
b) $\omega_1 \cup \omega_2 \cup \omega_3 = \Omega$, Ω being the set of all tasks in the task domain (in the system).
c) $\omega_j \cap \omega_k = \emptyset$ for $j, k = 1, 2, 3$ and $j \neq k$ (i.e., they are pairwise mutually disjoint).

Note that $p_1 + p_2 + p_3 = n$. A 3-tuple of the form $(\omega_1, \omega_2, \omega_3)$ represents a particular *scheduling scenario* of τ_i. Note that each task appears exactly in one, but some, ω_j. All tasks in ω_1 enjoy higher priority than τ_i, those in ω_2 enjoy the same priority as τ_i, and those in ω_3 enjoy lower priority than τ_i. For example, in Figure 1(b), when scheduling τ_2 at $(1,2)$ under EDF, two possible scenarios are: $(\{\tau_1\}, \{\tau_2, \tau_3\}, \{\})$ and $(\{\tau_1\}, \{\tau_2\}, \{\tau_3\})$ with $\tau_2 \in \omega_2$. Let R_i^ω denote the set of all such possible scenarios. The size of R_i^ω for a given task τ_i is necessary in the assessment of complexity of the approach as presented here; see Section 2.3. Obviously, it depends on how the tasks are spread over the task domain and can be found as

$$\mid R_i^\omega \mid = \prod_{k=1}^{n} N_{\tau_k}^\Omega \qquad (2)$$

where $N_{\tau_k}^\Omega$ denotes the number of sets Ω_j^K, $j = 1, 2, 3$, of which τ_k is an element, taking values 1, 2 or 3 only. In other words, each τ_k appears at least in one of the Ω_j^K sets but, in general, appears in $N_{\tau_k}^\Omega$ number of Ω_j^K sets. In the case of τ_i, however, $N_{\tau_i}^\Omega = 1$ since τ_i is considered to be only in ω_2. (2) can be proven by mathematical induction.

The execution probability of a particular task τ_i in Ω_2^K depends on two factors: firstly, on the realisation of its possible execution scenarios and, secondly, on its choice for execution by the scheduling algorithm concerned, both expressed in probabilistic terms. In connection with the first, given that $r \in R_i^\omega$, let $P_{i,r}$ denote the probability of the realisation of a particular scenario r. Furthermore, for any scenario $r = (\omega_1, \omega_2, \omega_3)$ and any $\tau \in \omega_k$, $k = 1, 2, 3$, let $p_\tau(\omega_k)$ denote the sum of probabilities of a task τ being anywhere in the region covered by the corresponding set Ω_k^K. Obviously, $p_\tau(\omega_k)$ can be determined knowing $p_\tau^t(l,c)$, introduced in Section 2.1. Note that consideration of Ω_k^K as a whole in working out $p_\tau(\omega_k)$ helps us to avoid point–wise enumeration of task scenarios in the l–c space. This could result in some reduction in the complexity – an issue dealt with in Section 2.3 later. $P_{i,r}$ can be determined as the product of the probabilities of each of the tasks participating in the scenario r. That is

$$P_{i,r} = \prod_{k=1,2,3} \Big(\prod_{\tau_j \in \omega_k} p_{\tau_j}(\omega_k) \Big) \qquad (3)$$

Turning to the second factor, there are two possibilities that need to be taken into consideration, namely, that, given that the scenario r is realised, the task under consideration, τ_i, is definitely executed (when $p_1 + p_2 \leqslant M$, M being the number of available processors), or may encounter a tie with other tasks at the same priority level (when $p_1 < M \wedge p_1 + p_2 > M$). In the latter case, there are basically two different ways of breaking the tie, namely, completely non–deterministically, that is, with a uniform probability among the tasks involved in the tie, or according to some secondary priority criterion. The former suits better the level of abstraction maintained in this work and, hence, has already been considered in our previous work [10]. However, a simulation used for the verification of the approach (to be discussed in Section 5) has revealed that the manner of tie breaking could have a noticeable effect on the results. Any simulation being an implementation, often involving some specific design choices, verification of theoretical predictions by simulation invariably requires consideration of how the ties are broken in the simulation. This explains the reason for considering in this work both the above approaches to the resolution of ties.

Scenario $r = (\omega_1, \omega_2, \omega_3)$	Deadline (Probability: $p_{\tau_j}(\omega_k)$)			$P_{i,r}$ - prob. of scenario realisation; eq. (3)	$\rho_{2,r}(1,2)$ eq. (4)	$P_{2,r} \times \rho_{2,r}$ see eq. (6)
	τ_1	τ_2	τ_3			
$(\{\tau_1\}, \{\tau_2, \tau_3\}, \{\})$	2 (0.2)	3 (0.6)	3 (0.8)	$0.2 \times 0.6 \times 0.8 = 0.096$	$\frac{1}{2}$	0.048
$(\{\}, \{\tau_2, \tau_3\}, \{\tau_1\})$	4 (0.5)	3 (0.6)	3 (0.8)	$0.5 \times 0.6 \times 0.8 = 0.24$	$\frac{2}{2} = 1$	0.24
$(\{\}, \{\tau_1, \tau_2\}, \{\tau_3\})$	3 (0.3)	3 (0.6)	4 (0.2)	$0.3 \times 0.6 \times 0.2 = 0.09$	$\frac{1}{2} = 1$	0.048

Table 1. Calculations in (6) illustrated

Let $\rho_{i,r}$ be the probability of execution of τ_i by any one of the available M processors in scenario r. Pursuing the non–deterministic option to tie–breaking first, $\rho_{i,r}$ can be found as

$$\rho_{i,r} = \begin{cases} 1 & \text{if } p_1 + p_2 \leqslant M \\ \frac{M-p_1}{p_2} & \text{if } p_1 \leqslant M-1 \wedge p_1 + p_2 > M \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

The second condition of the above has a non-zero probability only if at least one of the M processors can effectively be reserved for tasks operating at the same priority level as τ_i. These conditions constrain the manner in which the sets ω_k, $k = 1, 2, 3$, can be chosen for execution and, thus, limits the number of scenarios eligible for execution. Turning now to the second option to tie–breaking, whereby a tie is always broken in one way or another according to some secondary priority (or preference) criterion, let *eligible* be a predicate expressing such a criterion. In this respect, let *eligible*(τ_i) be true if and only if τ_i is eligible for execution according to the secondary priority criterion. In this case

$$\rho_{i,r} = \begin{cases} 1 & \text{if } (p_1 + p_2 \leqslant M) \vee \\ & (p_1 \leqslant M-1 \wedge p_1 + p_2 > M \wedge \textit{eligible}(\tau_i)) \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

In computing τ_is overall conditional probability of execution over the points (l, c) such that $f_\phi(l, c) = K$, that is $v_{i,K}$, all possible scenarios in $R_i^\omega(l, c)$ must be taken into consideration. That is

$$v_{i,K} = \sum_{r \in R_i^\omega} P_{i,r} \times \rho_{i,r} \quad (6)$$

As an illustration, considering τ_2 for scheduling in the context depicted in Figure 1(b), Table 1 shows how to calculate the terms in the summation (6) for three specific scenarios. [Note that in order to reduce the clutter we have deliberately omitted writing the temporal index t in various quantities introduced above, though they are really functions of time. As is done below, it is time to restore the temporal indexation with t.] Having obtained the conditional probability $v_{i,K}^t$ of τ_i, as $v_{i,K}$ in (6), it is possible to derive the actual execution probability of τ_i at all the points (l, c) where $f_\phi(l, c) = K$ at time t as

$$ex_i^t(l, c) = p_i^t(l, c) \times v_{i,K}^t \quad (7)$$

as the joint probability of the event captured in (6) in conjunction with the event that τ_i is actually at (l, c). The corresponding probability of τ_i missing execution (non-execution) at (l, c) at time t follows from the above as

$$ms_i^t(l, c) = p_i^t(l, c) - ex_i^t(l, c) \quad (8)$$

Consequently, for points (l, c) in the task domain the probability of τ_i being at (l, c) at the next time unit $(t + 1)$ depends on the probability of τ_i having been executed at $(l, c + 1)$ at time t and the probability of τ_i having missed execution at $(l + 1, c)$ at time t. This results in

$$p_i^{t+1}(l, c) = ex_i^t(l, c + 1) + ms_i^t(l + 1, c) \text{ if } (l, c) \in S \quad (9)$$

assuming that each of the functions $ex_i^t(l, c)$ and $ms_i^t(l, c)$ returns zero for any $(l, c) \notin S$. However, if at time t τ_i lies on the line $c = 1$, or on the line $l = 0$, then it has a probability of exiting the system by successfully completing, or by failing to meet its deadline, respectively. Therefore

$$p_i^{t+1}(l, c) = p_i^t(l, c) + ex_i^t(l, c + 1) \text{ if } l \geqslant 0 \wedge c = 0 \quad (10)$$

$$p_i^{t+1}(l, c) = p_i^t(l, c) + ms_i^t(l+1, c) \text{ if } l = \text{-}1 \wedge c > 0 \quad (11)$$

requiring the points in the exit domain to 'accumulate' the probabilities of any task exiting the system successfully, or unsuccessfully. Equations (9)–(11) in effect describe how to compute the probability of any task in the scheduling domain at time t finding itself, in general, in the task domain at time $(t + 1)$. Obviously, this process of computation can be performed for all n tasks in the l–c space and for all time values of interest. This results in a complete characterisation of the evolution of $p_i^t(l, c)$ over time which, as was suggested at the outset in Section 2.1, forms the basis of subsequent computation QoS indicators of individual tasks.

2.3 Computational Issues

Implicit in the analytical framework introduced above is a way to compute the distribution of tasks over the l–c space. Its focus has been to gain an insight into the problem of probabilistic scheduling rather than an efficient way to computing. Nevertheless, an assessment of the algorithmic complexity is necessary and it requires making explicit the computations involved in some detail. This is the purpose of the algorithm *execute*^{t} given below. It is a recursive algorithm with respect to time and, each time t, returns the task PMFs in the form of a three dimensional matrix *result*$[i, l, c]$ (lines 2 and 29). Indices i, l, c of *result* signify respectively the task indices, the laxities and the computation times and range $i \in 1 \,..\, n$, $l \in$ -1 $..\, l_{max}$ and $c \in 0 \,..\, c_{max}$ (line 2). The matrix $p^t[i, l, c]$ (line 2) is of an identical form to *result* but, with its additional index (superscript) t, stands essentially for $p_i^t(l, c)$. The matrix p^t consists of the task PMFs as applicable at time t, whereas *result* consists of the task PMFs computed by the algorithm to be used at time $t + 1$ in the case of tasks not requested at time $t + 1$.

The iteration in lines 4–7 over the tasks in Ω initialises the matrix $p_i^t(l, c)$, assuming any of its elements initially unspecified to be zero. The case of freshly requested tasks at time t is dealt with in line 5. The matrix $p^{init}[i, l, c]$ (line

5) is identical in form to p^t and consists of the task PMFs to be used at each request time of the tasks concerned. The iteration between lines 8–22 computes $ex_i^t(l,c)$ and $ms_i^t(l,c)$ of (7)–(8), represented here respectively as matrices $ex[i,l,c]$ and $ms[i,l,c]$ (lines 19 and 20), for K varying from K_{min} to K_{max}. The function *comp*-Ω (line 9) first computes the sets $\Omega_j^K, j = 1,2,3$. In lines 10–22 is an iteration over each task in Ω_2^K. The function *comp*-R_i^ω (line 11) computes all possible execution scenarios. In lines 13–17, the algorithm computes the conditional probability of execution of the task concerned assuming it to be there with the relevant priority. In lines 18–21, the algorithm calculates the actual probability of execution, as well as that of missing execution, of each task at points having the same K value. The probability of each task being at specific points in task domain, which is to be used in the next time unit, is calculated in lines 24–28 following a literal translation of (9)–(11). In line 29, *execute*^{t} returns the matrix $result[i,l,c]$ which is to be used in the next time unit.

```
1.  algorithm execute^t
2.  var result[i,l,c], p^t[i,l,c], ex[i,l,c], ms[i,l,c], – local variables
        Ω_1^K, Ω_2^K, Ω_3^K, R_i^ω, P_{i,r}, ρ_{i,r}, v_{i,K};
3.  begin
4.      for i ∈ 1 .. n do
5.        if t mod T_i = 0 then p^t[i,l,c] := p^{init}[i,l,c] for (l,c) ∈ S̄;
6.        else p^t[i,l,c] := execute^{t-1};
                – initialises or updates the PMFs
7.      end for
8.      for K from K_min to K_max do
9.          (Ω_1^K, Ω_2^K, Ω_3^K) := comp-Ω;
                – function for computing Ω_j^K for j from 1 to 3
10.         for i ∈ Ω_2^K do
                – for each task on the line
11.             R_i^ω := comp-R_i^ω;
                – function for computing all scenarios of task τ_i
12.             v_{i,K} := 0.0
                – initialise v_{i,K} to zero
13.             for each r ∈ R_i^ω do
                – for each scenario containing τ_i
14.                 P_{i,r} := comp-P_{i,r};
                – function for computing P_{i,r}; eqn (3)
15.                 ρ_{i,r} := comp-ρ_{i,r};
                – function for computing ρ_{i,r}; eqns (4) or (5)
16.                 v_{i,K} := v_{i,K} + P_{i,r} × ρ_{i,r};
                – conditional prob. of τ_i; eqn (6)
17.             end for
18.             for (l,c) such that f_φ(l,c) = K do
                – for each scheduling point on line f_φ(l,c) = K
19.                 ex[i,l,c] := p[i,l,c] × v_{i,K};
                – prob. of τ_i being executed at (l,c); eqn (7)
20.                 ms[i,l,c] := p[i,l,c] − ex[i,l,c];
                – prob. of τ_i missing execution at (l,c); eqn (8)
21.             end for
22.         end for
23.     end for
24.     for i ∈ 1 .. n do
25.         for (l,c) ∈ S̄ do                    – eqns (9)–(11)
26.             result[i,l,c] := ex[i,l,c+1] + ms[i,l+1,c];
                                 if (l,c) ∈ S
                result[i,l,c] := p^t[i,l,c] + ms[i,l+1,c];
                                 if (l = −1) and (c > 0)
                result[i,l,c] := p^t[i,l,c] + ex[i,l,c+1];
                                 if (l ⩾ 0) and (c = 0)
27.         end for
28.     end for
29.     return result
30. end algorithm excecute^t
```

Turning to the complexity of the above algorithm, it is sufficient here to focus on the running time of the segment of statements between lines 8–23, bearing in mind the smaller additive contribution of the remaining segments of the algorithm. The running time is estimated per every time unit of scheduling under EDF, scheduling under LLF being only marginally different. Omitting the derivation here for reasons of space, the cost of executing the statements concerned is found be of $O(n^2 3^{n-1})$. Despite the reduction in complexity achieved earlier by avoiding point–wise enumeration of task scenarios, this reflects the inherently combinatorial nature of the scenario space. Though it is manageable for moderately sized problems, this is an issue that requires further research.

3 Quality of Service Indicators

Knowing the time history of task PMFs, that is, $p_i^t(l,c)$ for $\tau_i, i \in 1 \,..\, n$ and for all points in the task domain and all time values of interest, it is possible to compute various Quality of Service (QoS) indicators of individual tasks. These include the rates (probabilities) of successful execution or failure of each task, latency in response time, jitter (irregularity in termination times between successive task instances), etc. For the time values of interest, let us consider the Least Common Multiple (LCM) L of the periods of tasks under consideration, i.e., LCM of T_i for $i \in 1 \,..\, n$. All tasks are assumed to arrive initially at time zero and thereafter each task τ_i to arrive at every T_i time units. The request time for the jth instance of τ_i, denoted as $\tau_{i,j}$, coincides with the end of the period of $(j-1)$th instance and, as a consequence, the jth instance terminates only from next time unit onward since it must last at least one time unit.

Turning to QoS indicators, let $S_{i,j}$ (or $F_{i,j}$), $j = 1..L/T_i$, denote the overall probability of successful execution (or failure) of $\tau_{i,j}$ anywhere within its period. It follows from (7) and (8) that

$$\begin{aligned} S_{i,j} &= \sum_{k=(j-1)T_i+1}^{jT_i} \left(\sum_{l=0}^{l_{max}} ex_i^k(l,1) \right) \\ F_{i,j} &= \sum_{k=(j-1)T_i+1}^{jT_i} \left(\sum_{c=1}^{c_{max}} ms_i^k(0,c) \right) \end{aligned} \qquad (12)$$

The expressions within parentheses in the above, denoted below by $S_{i,j}^k$ and $F_{i,j}^k$, represent the probability of successful execution and the probability of failure of $\tau_{i,j}$ at the kth time unit. Let $R_{i,j}$ represents the *success response time* of $\tau_{i,j}$, the probability of it successfully terminating within the first d_i time units of its period, $0 < d_i \leqslant T_i$, can be found as

$$\mathcal{P}(R_{i,j} \leqslant d) = \sum_{k=1}^{d} S_{i,j}^{(j-1)T_i+k} \qquad (13)$$

Jitter can be defined in several ways, for example, as a measure of irregularity, relative to task periods, of time between termination times of consecutive task instances, or that between the start and completion times of individual task instances. The latter gives input/output jitter as applicable to control problems and can be dealt with by making use of $R_{i,j}$ in (13). Dealing only with the former, let G_i be a

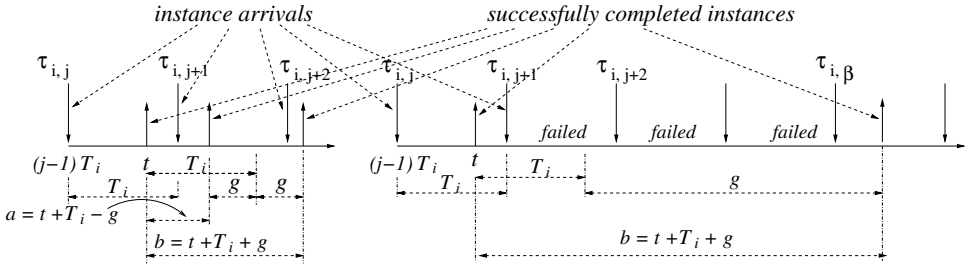


Figure 2. Jitter relative to successfully completed instances

random variable giving the length of time between the termination times of two consecutive instances of τ_i and let us define the regularity jitter of τ_i as $J_i = \mid G_i - T_i \mid$, J_i being a random variable ranging over $0 \,..\, T_i$. As a measure, jitter can be obtained in a variety of forms: as a probability $\mathcal{P}(J_i = g)$ of τ_i experiencing a particular value of regularity jitter g, or as an expectation.

Dealing with the so-called *success regularity jitter*, let us use $\bar{J}_{i,j,next}$ to refer to the regularity jitter between the successfully executed instance $\tau_{i,j}$ and τ_i's next successfully executed instance. As shown in Figure 2, there are two possibilities: Figure 2(a) showing the case of consecutive instances successfully completing, and Figure 2(b) the case of there being intervening failures between successfully completed instances. Thus, the probability of $\bar{J}_{i,j,next}$ taking a value g can be defined as

$$\mathcal{P}(\bar{J}_{i,j,next} = g) = \sum_{t=(j-1)T_i+1}^{j\,T_i} S_{i,j}^{t}\,(W_1 + W_2\,) \qquad (14)$$

where W_1 and W_2 are given by

$$W_1 \;=\; \begin{cases} S_{i,\alpha}^{a} & \text{if } j\,T_i < a \\ 0 & \text{otherwise} \end{cases}$$

$$W_2 \;=\; \begin{cases} S_{i,\beta}^{b} \displaystyle\prod_{k=j+1}^{\beta-1} F_{i,k} & \text{if } j+1 < \beta \\ S_{i,\beta}^{b} & \text{if } j+1 = \beta \wedge a < b \\ 0 & \text{otherwise} \end{cases} \qquad (15)$$

and $S_{i,j}^{t}$ denotes, as defined earlier, the probability of $\tau_{i,j}$ successfully completing at the tth time unit, $a = (t + T_i - g)$, $b = (t + T_i + g)$, $\alpha = \lceil (a/T_i) \rceil$ and $\beta = \lceil (b/T_i) \rceil$; $\lceil x \rceil$ denoting the ceiling of x. Based on the definition of regularity jitter in [3], termination of an instance of a task g time units before, or after, its period, as counted from the time of its last termination, has the same effect. In this respect, W_1 in (14) represents the probability of the next instance, that is, $\tau_{i,j+1}$, successfully terminating $T_i - g$ time units after $\tau_{i,j}$; see Figure 2(a). W_2 in (14), however, represents the probability of the next successful instance following $\tau_{i,j}$ successfully terminating $g + T_i$ time units after $\tau_{i,j}$; see Figures 2(a) and 2(b). This allows for potential failures of one or more instances of τ_i following the successful termination of $\tau_{i,j}$.

Let $\bar{J}_i$ represents the success regularity jitter of τ_i. With $u_i = \frac{L}{T_i}$, the probability of $\bar{J}_i$ having a value g is defined as

$$\mathcal{P}(\bar{J}_i = g) = \frac{1}{u_i} \sum_{j=1}^{u_i+1} \mathcal{P}(\bar{J}_{i,j,next} = g) \qquad (16)$$

The probability of successful execution of a task, or a set of tasks, in a given environment is another important QoS attribute guaranteeing a required level of service. If $S_i(m,n)$ denotes the probability of successful execution of task τ_i in an environment with m processors and n tasks, then an increase in m is generally expected to raise $S_i(m,n)$, while an increase in n to lower it. Though the interplay of these effects, as well as that of other factors such as individual task parameters, has not been a subject addressed here, an example in [10] illustrates the effect of the number of processors on successful task executions.

4 Verification by Stochastic Simulation

This section presents the formulae to be used in the stochastic simulation, presented in Section 5, for verifying the probabilistic approach described above. The stochastic simulation is based on a large sample of instances of each task τ_i generated according to its PMF $p_i(l,c)$ used at request times, introduced in Section 2.1, with a sufficiently closely matching relative frequency distribution (histogram). Letting L denote the LCM of task periods, the simulation period is assumed to cover q LCM cycles, or qL time units in total. Thus, each task τ_i is requested $Q_i = q \times \frac{L}{T_i}$ number of times over the whole simulation period. Based on the results of the stochastic simulation, various QoS indicators such as probability of successful execution, probability of failure and response time latency can be obtained and compared against analogous results established by the probabilistic analysis.

When dealing with various quantities encountered in simulation, let us distinguish them notationally from their equivalent counterparts in the probabilistic analysis by the use of an overbar. Thus, $\bar{\tau}_{i,j}^{k}$ represents the jth instance of τ_i in the kth simulation cycle, for $k \in 1 \,..\, q$ and $\bar{S}_{i,j}$ and $\bar{F}_{i,j}$ be the rates of successful execution and failure of the jth instances of τ_i respectively, for $j = 1, 2, ..L/T_i$. The latter are defined as the ratios of total number of successful executions, and likewise failures, of $\bar{\tau}_{i,j}^{k}$ to the total number of cycles. In other words

$$\bar{S}_{i,j} = \frac{1}{q} \sum_{k=1}^{q} \sum_{t=1}^{T_i} Y_{i,j}^{k,t} \;\text{ and }\; \bar{F}_{i,j} = 1 - \bar{S}_{i,j} \qquad (17)$$

where $Y_{i,j}^{k,t} = \begin{cases} 1 & \text{if } \bar{\tau}_{i,j}^{k} \text{ is executed successfully} \\ & \text{at time } t \text{ within its period} \\ 0 & \text{otherwise} \end{cases}$

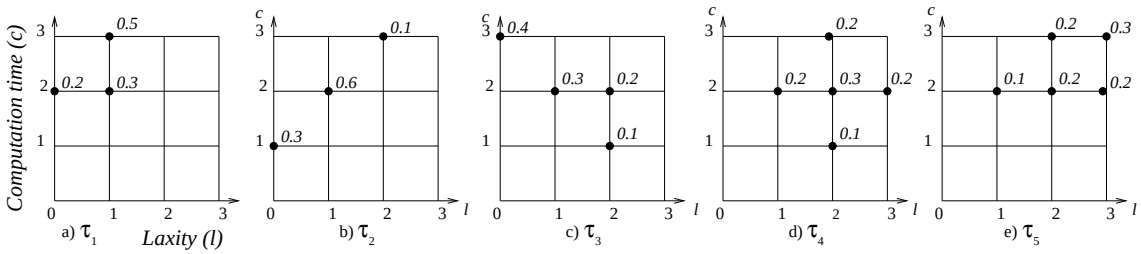


Figure 3. Characteristics L_i and C_i of tasks τ_i, $i = 1, \cdots, 5$, at request times.

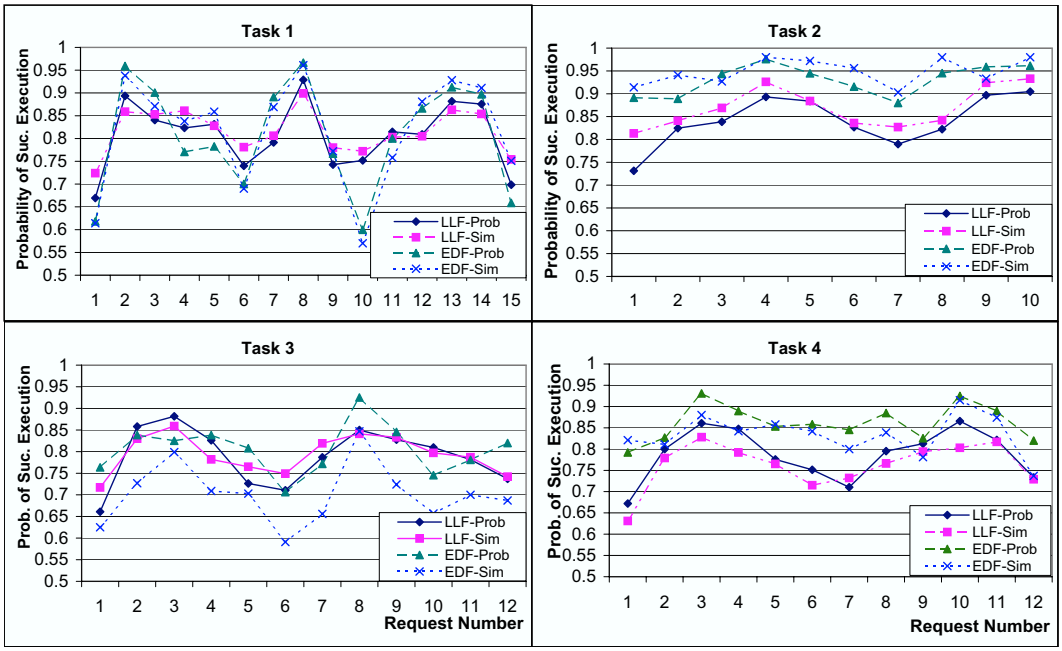


Figure 4. Probabilistic and simulation results for tasks τ_1–τ_4 in Task Set 3 under LLF and EDF scheduling with random tie–breaking.

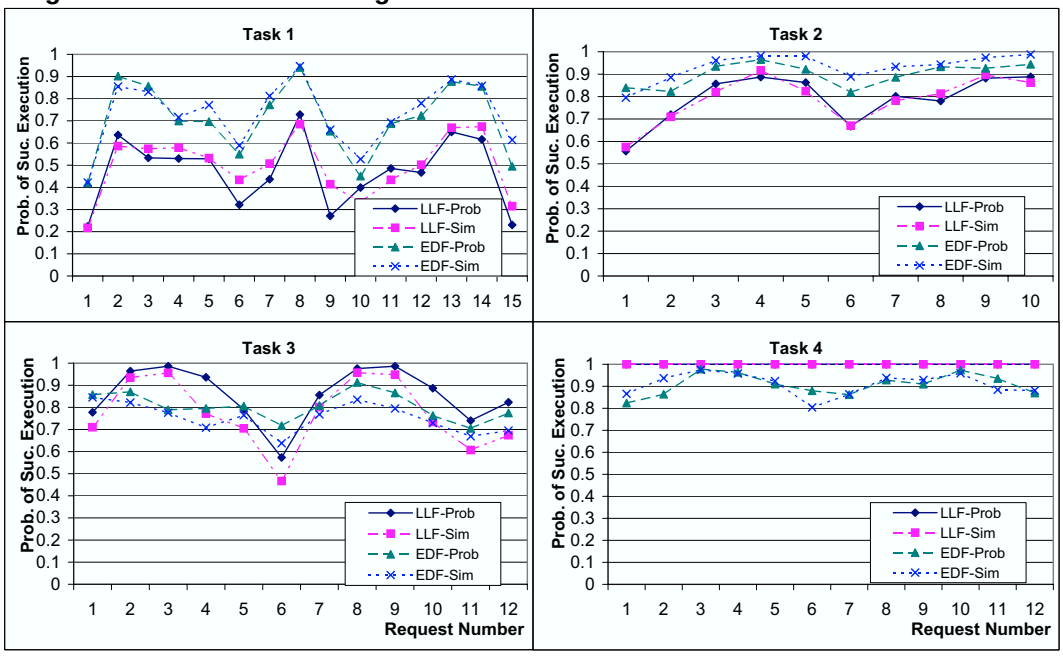


Figure 5. Probabilistic and simulation results for tasks τ_1–τ_4 in Task Set 3 under LLF and EDF scheduling with highest–indexed–task–first tie–breaking.

τ_1			τ_2			τ_3			τ_4			τ_5		
(l,c)	$\mathcal{P}$	Hist	(l,c)	$\mathcal{P}$	Hist	(l,c)	$\mathcal{P}$	Hist	(l,c)	$\mathcal{P}$	Hist	(l,c)	$\mathcal{P}$	Hist
(0,2)	0.2	0.202	(0,1)	0.3	0.302	(0,3)	0.4	0.404	(1,2)	0.2	0.202	(1,2)	0.1	0.101
(1,2)	0.3	0.299	(1,2)	0.6	0.599	(1,2)	0.3	0.300	(2,1)	0.1	0.100	(2,2)	0.2	0.201
(1,3)	0.5	0.498	(2,3)	0.1	0.099	(2,1)	0.1	0.100	(2,2)	0.3	0.300	(2,3)	0.2	0.199
						(2,2)	0.2	0.196	(2,3)	0.2	0.202	(3,2)	0.2	0.202
									(3,2)	0.2	0.196	(3,3)	0.3	0.296

Table 2. Comparison between the frequencies of the generated data and the probability values

Sched.	Task	1			2			3		
regime	Task Set	1	2	3	1	2	3	1	2	3
EDF	Mean	0.9738	0.9048	0.8060	0.9935	0.9650	0.9306	0.9570	0.8972	0.8057
	St. Dev.	0.0608	0.1018	0.1162	0.0115	0.0294	0.0324	0.0615	0.0624	0.0545
LLF	Mean	0.9917	0.9918	0.8062	0.9876	0.9389	0.8412	0.9897	0.9417	0.7880
	St. Dev.	0.0181	0.0501	0.0717	0.0212	0.0515	0.0522	0.0198	0.0554	0.0642

Table 3. Mean and Standard Deviation for the $S_{i,j}$ of τ_1, τ_2 and τ_3 in Task Sets 1, 2 and 3.

Using the probabilistic notions for comparative purposes, the probability of $\bar{\tau}_{i,j}$ having a *success response time t* is defined as the ratio of the total number of $\tau^k_{i,j}$ instances successfully executed at time *t* within their periods to the total number of cycles. Letting $\bar{R}_{i,j}$ denote the success response time of instances $\tau^k_{i,j}$, for $t = 1, 2, ..T_i$, the probabilities of $\tau^k_{i,j}$s successfully terminating at time *t*, and the probabilities of them successfully terminating within *t* time units in their periods, can be defined respectively as

$$\mathcal{P}(\bar{R}_{i,j} = t) = \frac{1}{q}\sum_{k=1}^{q} Y^{k,t}_{i,j},\ \mathcal{P}(\bar{R}_{i,j} \leqslant t) = \frac{1}{q}\sum_{k=1}^{q}\sum_{s=1}^{t} Y^{k,s}_{i,j} \quad (18)$$

with $Y^{k,t}_{i,j}$ and $Y^{k,s}_{i,j}$ remaining as defined in (17).

Considering *success regularity jitter* of tasks, let $\bar{J}_{i,g}$ represents the total count of measurements g observed over the simulation time in relation to τ_i, where $g = \mid y - T_i \mid$ and y is the length of time between termination times of two consecutive successfully executed instances of τ_i. The probability of τ_i having a success regularity jitter g is defined as the ratio of total count to the total number of instances

$$\mathcal{P}(\bar{J}_i = g) = \frac{\bar{J}_{i,g}}{Q_i} \quad (19)$$

5 An Illustrative Example

Our previous work [10] illustrates in some detail the kind of results that can be obtained using the proposed approach, in particular, a comparison of the probability of successful execution of individual task instances over the LCM of task periods under EDF and LLF scheduling, the same on average as a function of the number of available processors and a measure of jitter experienced by individual tasks. These can be important indicators of performance of individual tasks and the system as a whole. In this paper, we extend this example to include other aspects, in particular, a verification of the approach by a stochastic simulation (described in Section 4), the effect of tie–breaking on scheduling (described in Section 2.2) and additional QoS indicators (described in Section 3).

The illustration consists of three related examples involving three separate sets of tasks: $\{\tau_1, \tau_2, \tau_3\}$, $\{\tau_1, \tau_2, \tau_3, \tau_4\}$ and $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$; these are referred to as Task Sets 1, 2 and 3 respectively. The latter two are intended to show the adverse effect of the increased workload due to τ_4 and τ_5 on scheduling the tasks in Set 1. The PMF-values $p_i(l,c)$ of individual tasks at their request times, used in probabilistic analysis, are shown in Figure 3. Interpretation of these data is such that, according to Figure 3(a), τ_1 has a probability of 0.3 arriving with a laxity 1 and a computation time 2 and a probability of 0.5 arriving with the same laxity but a computation time 3. Task periods are 4, 6, 5, 5 and 6 respectively. (N.B. these values are not related to any particular application.) The above tasks are scheduled using LLF and EDF algorithms on two processors. The data used in the stochastic simulation have been generated randomly from the data in Figure 3 covering a simulation period consisting of 1000 LCM cycles. Table 2 compares the PMF-values (under $\mathcal{P}$) used in probabilistic analysis and the relative frequency histograms (under *Hist*) of the dada used in simulation.

Turning to the results, Table 3 presents the cumulative probabilities of successful execution of tasks τ_1, τ_2 and τ_3 for all three Task Sets in the EDF and LLF regimes over their respective periods over the LCM in terms of the mean and the standard deviation. The table shows the adverse effect of the increased workload due to τ_4 on its own (in Task Set 2), and τ_4 and τ_5 together (in Task Set 3) on the execution of τ_1, τ_2 and τ_3 (Task Set 1). It also shows how different tasks tend to benefit differently from the two scheduling algorithms, τ_1 and τ_3 favourably from LLF and τ_2 from EDF. Despite their inadequacy for drawing any general conclusions, these examples tend to suggest a link between the algorithms and the more dominant task characteristic on the probability of successful execution.

Figure 4 presents the probability of successful execution of individual task instances over the LCM due to both probabilistic predictions and simulation outcomes. Though in the case of tasks τ_1 and τ_2 the results due to two approaches are fairly close to each other, there is a marked difference in the case of τ_3 under EDF scheduling, probabilistic results even indicating a better performance under EDF than under LLF and, to an extent, a reversal of performance in the simulation. In this respect, it is worth

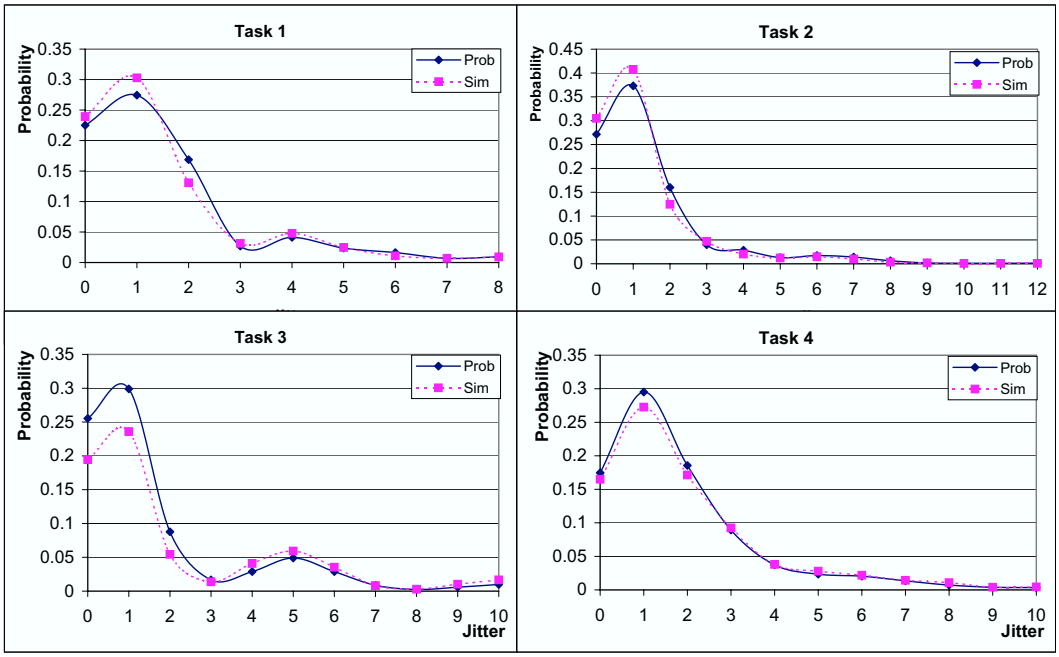


Figure 6. Probability of success regularity jitter experienced by tasks τ_1–τ_4 in Task Set 3.

noting that the above set of results in Figure 4 is based on non–deterministic tie-breaking strategy discussed in Section 2.2.

In contrast, Figure 5 shows analogous results using a highest–indexed–task–first strategy to tie breaking, consistently offering greatest advantage to τ_5 and least advantage to τ_1. It has also been observed that EDF exhibited less number of ties compared to LLF. As a result, tasks such as τ_1 and τ_2, which are relatively less advantaged, are less affected by this strategy under EDF in comparison to LLF. In the case of τ_3, with reference to the observation made above on non–deterministic tie-breaking, not only has there been an improvement in its performance but also an apparent narrowing down of the gap between probabilistic and simulation results under EDF. Enjoying higher priority over other tasks whenever a tie is encountered, τ_4 and τ_5 exhibit a high performance under both algorithms, both tasks showing a probability of 1.0 in successful execution in LLF and τ_4 a probability of over 0.8 in successful execution in EDF. An analogous analysis has been conducted for tie–breaking based on lowest–indexed–task–first strategy. Though the results of this analysis are not given here for reasons of space, the outcome is a reversal of the above, letting τ_1 enjoy the best performance and τ_5 the worst. In this case, τ_1 and τ_2 achieve a probability of 1.0 in successful execution in LLF in both probabilistic analysis and simulation. The probability of successful execution of τ_3 under both highest–indexed–task–first and lowest–indexed–task–first strategies is higher than that under non–deterministic tie-breaking strategy.

Illustrating the effect of the scheduling algorithms EDF and LLF on success regularity jitter, i.e. the jitter experienced by successive successful instances of the tasks concerned, Figure 6 shows a close correlation between the probabilistic predictions and the simulation results for tasks τ_1–τ_4 in Task Set 3 under non–deterministic tie–breaking. Figure 7 makes a similar comparison of the expectation of (average) response times of the same tasks but under highest–indexed–task–first approach to tie breaking. In this case too there appears to be a close correlation between the probabilistic predictions and the simulation results, but except for τ_3.

6 Conclusions

This paper presents an analytical approach for computing QoS indicators of tasks with uncertainties executed by a multi-processor system. Tasks are periodic but are characterised by uncertainties in computation times and deadlines (expressed indirectly through laxity, or urgency), described in terms of jointly discrete probability mass functions. At the request times, task computation times and laxities are truly random in character, though subsequently their variation is dependent on the manner of execution. Though only EDF and LLF algorithms are considered here, the approach is applicable to the study of execution under any priority driven algorithm. By examining possible computation times and laxities of each task in turn, a systematic approach is developed for enumerating its execution scenarios involving other competing tasks. Knowing the probability of realisation of each scenario and the manner in which the task under consideration is treated in the face of competition by other tasks at the same priority level, its execution probability is computed for each time value. Essentially, this allows a way to work out the probability distribution of each task at every instant of time from the probability distributions of all tasks in the previous time instant. These probability distributions form the basis of QoS assessment of indi-

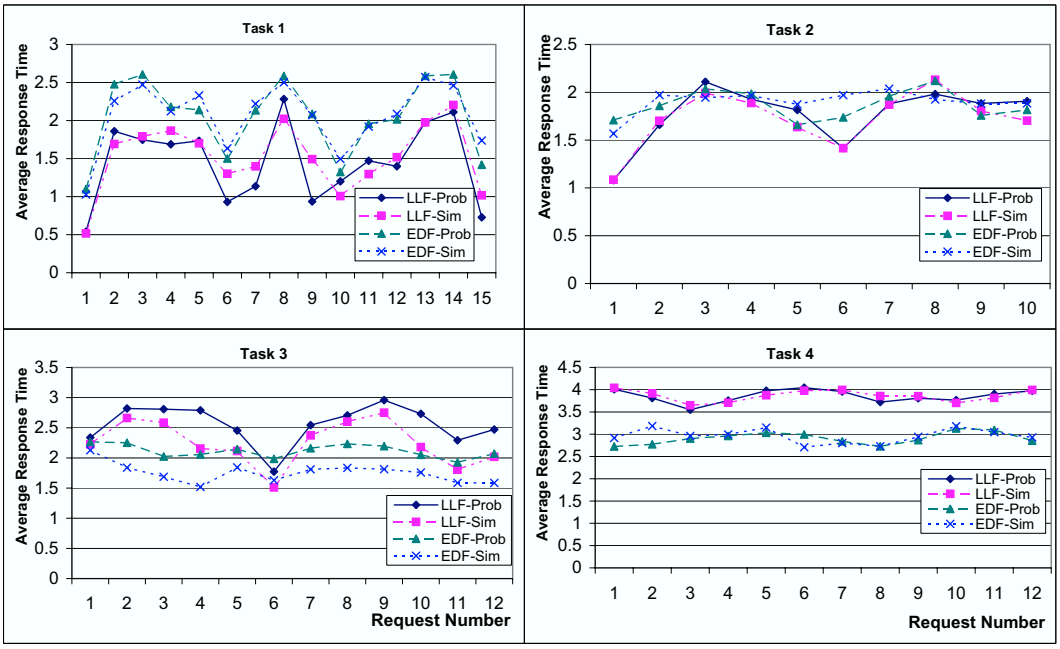


Figure 7. Expectation of response times of tasks τ_1–τ_4 in Task Set 3.

vidual tasks. In this connection, the paper presents how to compute QoS indicators such as the rate of successful execution, latency in response time and jitter.

An example illustrates the approach in some detail with particular reference to the benefits to be gained and the capabilities of the approach, with a further stochastic simulation demonstrating the general validity of the approach. These results indicate some sensitivity of the analysis to the manner in which any ties are broken among tasks competing at the same priority level, thus highlighting the practical significance of the manner of tie–breaking at the implementation level.

The combinatorial nature of the enumeration of execution scenarios inevitably results in an $O(n^2 3^{n-1})$ complexity, making the direct application of the approach as proposed to scheduling large numbers of tasks infeasible. Therefore, the approach should be regarded as applicable only to moderately sized problems. As is the case with most applications, limitation of uncertainty only to computation times would help to reduce the complexity. In addition to providing a general solution, an important contribution of this work is the comprehensive insight it has given to understanding various computational aspects of probabilistic scheduling. The result is a sound foundation for undertaking research into more efficient scheduling algorithms, possibly, employing heuristic means to reduce the complexity. Another important issue that needs addressing is the establishment of the validity of the approach using a more comprehensive range of simulations and, if possible, the bounds of this validity.

References

[1] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. *19th IEEE Real-Time Systems Symposium*. 1998.

[2] A. Atlas and A. Bestavros. Design and Implementation of Statistical Rate Monotonic Scheduling in KURT Linux. *IEEE Real-Time Systems Symposium*. 1999.

[3] L. David, F. Cottet and N. Nissanke. Jitter Control in On-line Scheduling of Dependent Real-time Tasks. *22nd IEEE Real-Time Systems Symposium*, London, 2001.

[4] M. L. Dertouzos and A.K. Mok. Multi-Processor On-line Scheduling of Hard Real-time Systems. *IEEE Trans. on Software Engineering*, 15(12), December 1989.

[5] J. L. Diaz, D. F. Garcia, et al. Stochastic Analysis of Periodic Real-time Systems. *23rd IEEE Real-Time Systems Symposium*, Austin, Texas, 2002.

[6] S. Edgar and A. Burns Statistical Analysis of WCET for Scheduling. *22nd IEEE Real-Time Systems Symposium*. London, UK. 2001.

[7] M. Gardner. Probabilistic Analysis and Scheduling of Critical Soft Real–Time Systems. Ph.D. Thesis. Univ. of Illinois at Urbana–Champagn. 1999.

[8] A. Leulseged. Probabilistic Analysis of Real-Time Multi-Processor Scheduling. PhD Thesis, London South Bank University; November 2005.

[9] J. P. Lehoczky, "Real–time Queueing Theory", *17th Real-time System Symposium*, December 1996.

[10] A. Leulseged and N. Nissanke. Probabilistic Analysis of Multi–processor Scheduling of Tasks with Uncertain Parameters. *9th International Conference on Real–Time Embedded Computing Systems and Applications*. Taiwan, R.O.C. 2003.

[11] S. Manolache, P. Eles and Z. Peng. Memory and Time–efficient Schedulability Analysis of Task Sets with Stochastic Execution Time. *13th Euromicro Conference on Real–Time Systems*. 2001, Pages 19–26.

[12] N. Nissanke, L. David and F. Cottet. Probabilistic Preemptive Schedulability Analysis. Int. Workshop on Probabilistic Analysis Techniques for Real Time and Embedded Systems (PARTES 2004), Pisa, September 2004.

[13] N. Nissanke, A. Leulseged and S. Chillara. Probabilistic Performance Analysis in Multiprocessor Scheduling. *Computing and Control Engineering Journal*. 13(4), August 2002.

[14] L. Zhou, K. G. Shin and E. A. Rundensteiner. Rate-monotonic scheduling in the presence of timing unpredictability. *IEEE Real–Time Technology and Applications Symposium*. 1998.

A Comparison of Global and Partitioned EDF Schedulability Tests for Multiprocessors

Theodore P. Baker*
Florida State University
Dept. of Computer Science
Tallahassee, FL 32306 USA
baker@cs.fsu.edu

Abstract

This paper compares the performance of several variations on EDF-based global and partitioned multiprocessor scheduling algorithms, together with their associated feasibility tests, on a variety of pseudo-randomly chosen sets of sporadic tasks. A new hybrid EDF-based scheme is shown to perform better than previously studied priority-based global scheduling schemes, though not as well as EDF-based first-fit partitioned scheduling.

1 Introduction

Recent trends in microprocessor design have drawn interest to multi-core and multiprocessor designs for high performance embedded real-time systems. The predominant approach to scheduling multiprocessor hard-real-time systems has been partitioned, in which each task is assigned statically (more or less) to one processor. Partitioned scheduling has the virtue of permitting schedulability to be verified using well-understood single-processor analysis techniques.

The alternative to partitioned scheduling is global scheduling, in which there is a single job queue from which jobs are dispatched to any available processor according to a global priority scheme. Until recently, it was believed that global scheduling policies with fixed job priorities[1], such as Rate Monotonic and Earliest-Deadline-First (EDF), could not even guarantee schedulability for systems of hard-deadline tasks whose total processor demand exceeded the capacity of a single processor. However, there have been several recent improvements in the worst-case analysis of global hard-deadline multiprocessor scheduling [1, 5, 12, 13, 2, 8, 9, 11, 4]. Among other developments, the EDF-US[1/2] scheduling policy[18], in which a few high-utilization tasks are scheduled at top priority and other tasks are scheduled according to deadlines, has been shown to guarantee worst-case schedulability up to the same processor utilization level as partitioned EDF scheduling.

Global scheduling remains controversial. There are individuals who believe strongly that global scheduling is impractical, because of the overhead of synchronizing schedulers between processors and the lost performance due to translation look-aside buffer and memory cache misses following the migration of a tasks between processors. On the other hand, the concept of global scheduling is appealing, especially in systems where average as well as worst-case response time is important. It is a well-known result of queueing theory that single-queue scheduling produces better average response times than queue-per-processor scheduling [15].

This paper attempts to compare the present state of the art for global EDF scheduling against the state of the art for partitioned EDF scheduling. Because the worst-case performance of both approaches has been shown to be the same, at least for the case where deadline equals period, the comparison is of empirical performance. That is, what are the odds that a randomly chosen set of periodic or sporadic tasks can be guaranteed schedulable by a given combination of scheduling policy and feasibility test? As a further contribution, the paper introduces a previously unstudied hybrid global scheduling algorithm, called EDF-LM.

2 Prior Work

For the review of prior work, some terminology is needed.

2.1 Terminology

A *task set* τ is a collection of sporadic tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task generates a potentially infinite sequence of *jobs*, and is characterized by a triple $\tau_i = (c_i, d_i, T_i)$. The parameter c_i is the *worst-case execution time requirement* of each job of τ_i and d_i is the *deadline* of each job relative to its release time. If the task is *periodic*, T_i is the separation between the release

[1] Here, fixed-*job*-priority scheduling is distinguished from fixed-*task*-priority, where a task is as a sequence of jobs. That is, with EDF scheduling the priority of each task varies from one job to another, but the priority of each job is fixed at the time the job is released.

times of the jobs, and called the *period* of the task. If the task is *sporadic*, T_i is interpreted instead as just the *minimum separation* between the release times of the jobs. It follows that a *periodic task set* is a restricted form of *sporadic task set*.

The *utilization* of a task τ_i is denoted by $u_i \stackrel{\text{def}}{=} c_i/T_i$, and the *density* of τ_i is denoted by $\lambda_i \stackrel{\text{def}}{=} c_i/min(d_i, T_i)$.

2.2 General Limitation

Andersson, Baruah, and Jonsson [1] showed that the utilization guarantee for EDF or any other fixed-job-priority multiprocessor scheduling algorithm – whether partitioned or global – cannot be higher than $(m+1)/2$ on an m-processor platform. This result holds for independent periodic task sets with deadline equal to period, and generalizes directly to the sporadic case.

2.3 Partitioned Scheduling

The optimal partitioning of tasks among processors is reducible to the bin packing and integer partition problems, which are known to be NP complete. Therefore, research on partitioned multiprocessor scheduling has focused on the analysis of heuristic algorithms for the assignment of tasks to processors, and on bounding how badly they can do compared to an optimal algorithm. Some of this research has looked at average-case performance. Other research has attempted to find tight bounds on the worst-case performance of heuristic partitioning algorithms.

Lopez *et al.* [16] showed that it is possible to schedule on m processors any system of n independent periodic tasks with maximum individual utilization $= u_{\max}$ and total utilization $< \frac{m\beta_{EDF}+1}{\beta_{EDF}+1}$ where $\beta_{EDF} = \lfloor 1/u_{\max} \rfloor$. For the unrestricted case, where $u_{\max} = 1$ and $\beta_{EDF} = 1$, this says the guaranteed utilization bound is $(m+1)/2$. It follows from Andersson, Baruah, and Jonsson [1] that this result is tight.

Baruah and Fisher [4] studied a partitioning algorithm that assigns tasks to processors by a first-fit algorithm in deadline-monotonic order (that is, sorted by increasing deadline). The single-processor test for fit is based analysis of a demand-bound function, as follows:

Theorem 1 (BF) *A set of independent sporadic tasks τ_1,...,τ_n is EDF schedulable on one processor if both of the following hold for each task τ_i:*

$$d_i - \sum_{j=1}^{n} DBF^*(j, d_i) \geq c_i \quad (1)$$

$$1 - \sum_{j=1}^{n} u_j \geq u_i \quad (2)$$

where $u_i = c_i/T_i$ and

$$DBF^*(i,t) = \begin{cases} 0, & \text{if } t < d_i \\ c_i + (t-d_i)u_i, & \text{otherwise} \end{cases}$$

	$\frac{c_i}{T_i} \leq \lambda$	$\frac{c_i}{T_i} > \lambda$
$d_i \leq T_i$	$\frac{c_i}{T_i}(1 + \frac{T_i - d_i}{d_k})$	$\frac{c_i}{T_i}(1 + \frac{T_i}{d_k}) - \lambda\frac{d_i}{d_k}$
$d_i > T_i$	$\frac{c_i}{T_i}$	$\frac{c_i}{T_i}(1 + \frac{T_i}{d_k})$

Table 1. Definition of $\beta_k(i)$.

2.4 Global Scheduling

Goossens, Funk, and Baruah [13] showed that a system of independent periodic tasks can be scheduled successfully on m processors by EDF scheduling if the total utilization is at most $m(1-u_{\max})+u_{\max}$, where $u_{\max}$ is the maximum utilization of any individual task. They also showed that this utilization bound is tight, in the sense that there is no utilization bound $\hat{U} > m(1-u_{\max})+u_{\max}+\epsilon$, where $\epsilon > 0$, for which $U \leq \hat{U}$ guarantees EDF schedulability. Srinivasan and Baruah [18] also examined the global EDF scheduling of periodic tasks on multiprocessors, and showed that any system of independent periodic tasks for which the utilization of every individual task is at most $m/(2m-1)$ can be scheduled successfully on m processors if the total utilization is at most $m^2/(2m-1)$.

In 2002, Srinivasan and Baruah [18] proposed a method for dealing with a few heavy tasks, using a *hybrid* scheduling policy. Their idea is to give highest (fixed) priority to to tasks of utilization greater than some constant ζ, and schedule the other tasks according to the basic EDF algorithm. This algorithm is called EDF-US[ζ]. Algorithm EDF-US[$m/(2m-2)$] was shown to correctly schedule on m processors any periodic task system with total utilization $U \leq m^2/(2m-2)$.

In 2003, Goossens, Funk and Barush [13] introduced another hybrid method, called PriD, for periodic task systems. The idea is to choose the k tasks ($0 \leq k < m$) with highest utilization (u_i) and give those special tasks top priority; the remaining $n-k$ tasks are scheduled according to the EDF policy. The value k is chosen to be the minimum such that the remaining $n-k$ tasks satisfy a utilization-based schedulability test for $m-k$ processors.

Baker [2, 3] derived several sufficient feasibility tests for m-processor preemptive EDF scheduling of sets of periodic and sporadic tasks with arbitrary deadlines, including the following.

Theorem 2 (BAK) *A set of independent sporadic tasks $\tau_1, \ldots, \tau_n$ is EDF-schedulable on m identical processors if, for every task τ_k, there exists a positive value $\mu \leq m - (m-1)\lambda_k$ such that*

$$\sum_{i=1}^{N} \min(\beta_k(i), 1) \leq \mu$$

where $\lambda = \frac{m-\mu}{m-1}$ and $\beta_k(i)$ is as defined in Table 1.

Baker also showed that the optimal value of ζ in EDF-US[ζ] with respect to maximizing the worst-case guaranteed schedulable utilization is $\zeta = 1/2$, for which the utilization bound is $(m+1)/2$. It follows from the argument in [1] that this bound is tight, and it is identical to

the worst-case utilization bound for EDF-based first-fit-decreasing (FFD) partitioned scheduling.

Bertogna, Cirinei and Lipari [8] made further improvements in global EDF schedulability tests. First, they observed that the proof of the utilization bound test of [13] extends naturally to cover pre-period deadlines if the utilization $u_i = c_i/T_i$ is replaced by c_i/d_i. As observed by Sanjoy Baruah[2], the same proof extends to the case of post-period deadlines if c_i/d_i is replaced by density ($\lambda_i = c_i/\min(d_i, T_i)$).

Theorem 3 (GFB) *A set of independent sporadic tasks $\tau_1,\ldots,\tau_n$ is EDF schedulable on m identical processors if*

$$\sum_{i=1}^{n} \lambda_i \leq m - \lambda_{\max}(m-1)$$

where $\lambda_{\max} = \max\{\lambda_i | i = 1,\ldots,n\}$.

Bertogna *et al.* also developed the following new schedulability test.

Theorem 4 (BCL) *A set of independent sporadic tasks $\tau_1,\ldots,\tau_n$ with constraint $d_i \leq T_i$ is EDF schedulable on m identical processors if for each task τ_k one of the following is true:*

$$\sum_{i \neq k} \min\{\beta_i, 1-\lambda_k\} < m(1-\lambda_k) \tag{3}$$

$$\sum_{i \neq k} \min\{\beta_i, 1-\lambda_k\} = m(1-\lambda_k) \quad \textit{and} \quad \exists i \neq k : 0 < \beta_i \leq 1-\lambda_k \tag{4}$$

where

$$\beta_i = \frac{N_i c_i + \min\{c_i, \max\{0, d_k - N_i T_i\}\}}{d_k}$$

and

$$N_i = \left\lfloor \frac{d_k - d_i}{T_i} \right\rfloor + 1$$

Bertogna *et al.* demonstrated that the BCL, GFB, and BAK tests are generally incomparable, but observed that the BCL test seemed to do better than the rest on task sets with a few "heavy" (high utilization) tasks. They reported simulations on collections of such pseudo-randomly generated tasks sets, for which the BCL was able to discover significantly more schedulable task sets than either of the other two tests. However, they did not compare these results against the EDF-US[ζ] hybrid method of handling heavy tasks, or any other hybrid method. Since such hybrid methods are much better at handling a few heavy tasks than pure EDF scheduling, it is more important how a schedulability test for such task systems performs in the hybrid environment than in the pure EDF environment.

[2] personal communication

3 Empirical Comparisons

To evaluate the overall efficacy of the known schedulability tests for global EDF scheduling, and to compare the efficacy of global *versus* partitioned scheduling, a series of experiments were conducted on pseudo-randomly generated sets of periodic tasks.

Of course, the usual disclaimers for such simulations apply. The performance of a scheduling policy and schedulability test on generated task sets is only suggestive of the performance that can be expected in practice. The schedulability tests do not take implementation overheads into account. The distribution of test cases considered in any such experiment may bias the outcome. In fact, it can be argued that for each specific application the consideration of any more cases than the one task set at hand should be biased, since the only important question is whether that one task set can be successfully scheduled. Perhaps as the system evolves the task set may change and the question will be asked again, but for each application one is still interested a very small and very specific collection of task sets.

So, what good are simulation results, such as those reported below? Ideally, for each application one should experiment with different scheduling policies and tests on the specific task set of interest, and perhaps also with a range of variations that anticipate future evolution of the system. However, when one has a large number of choices of scheduling policies and schedulability tests, each of which could work better on some cases, such exhaustive experimentation is not be practical for every application. How does one narrow the range of choices? The same need for narrowing the range of scheduling policy choices arises when one is deciding which policies too support in a generic real-time operating system kernel. In this context, the statistical trends over large numbers of task sets should be a better predictor of comparative performance than pure intuition.

For these experiments, 48 different datasets were considered, each containing 1,000,000 task sets. The datasets were generated in several different ways, with the hope of discovering some correlation between the way the tasks were generated and which combination of scheduling policy and schedulability test did better. However, the trends across all the experiments were quite similar, and the space here is limited, so only the results of a few representative experiments are reported.

3.1 Task Set Generation Methodology

These methods by which the datasets were generated were based on the following goals:

1. Focus on cases where scheduling performance is most likely to matter, excluding task sets that are clearly so easy that it does not matter how they are scheduled (trivially schedulable), or that are clearly impossible to schedule by any method (infeasible).

2. For each model, generate as large a sample of task sets as is practical.
3. Cover a range of multiprocessor sizes.
4. Cover a variety of deadline models.
5. Cover a variety of distributions of task utilizations, including for comparison the bimodal kind of distribution of Bertogna, Cirinei, and Lipari [8].
6. Keep the number of cases small enough to allow running a complete battery of tests in less than a day.

Task periods T_i were chosen pseudo-randomly from the integer interval $[1, 1000]$. Task utilization factors (and, implicitly, the compute times) were chosen according to the following distributions, truncated to bound the individual task utilizations between 0.001 and 0.999:

1. uniformly chosen from $[1/T_i, 1]$
2. bimodal distribution: heavy tasks uniformly chosen from $[0.5, 1]$; light tasks uniformly chosen from $[1/T_i, 0.5]$; probability of being heavy = 1/3
3. exponential distribution with mean 0.25
4. exponential distribution with mean 0.50

The deadlines were chosen in several different ways:

1. period: $d_i = T_i$
2. constrained: d_i uniformly chosen from $[c_i, T_i]$.
3. unconstrained: d_i uniformly chosen from $[c_i, 4T_i]$
4. superperiod: d_i uniformly chosen from $\{T_i, 2T_i, 3T_i, 4T_i\}$

Datasets were generated for three different numbers of processors ($m = 2, 4, 8$), as follows: An initial set of $m + 1$ tasks was generated, and tested. Then another task was generated and added to the previous set, and all the schedulability tests were run on the new set. This process of adding tasks was repeated until the total processor utilization exceeded m. The whole procedure was then repeated, starting with a new initial set of $m + 1$ tasks.

Note that the above method of generating task sets already eliminates tasks sets that can be trivially scheduled by assigning one task per processor, or that are clearly infeasible because they have utilization greater than m. Additional screening was performed, to remove task sets with total density $\sum_{i=1}^{n} \lambda_i \leq 1.0$ (schedulable by EDF on one processor) or total load $\delta_{\text{sum}} \geq m$ (infeasible on m processors) [4, 10]. The load-bound function is defined by

$$\delta_{\text{sum}} \stackrel{\text{def}}{=} \sum_{i=1}^{n} \frac{\text{DBF}(\tau_i, t)}{t}$$

and

$$\text{DBF}(\tau_i, t) \stackrel{\text{def}}{=} \max(0, (\lfloor \frac{t - d_i}{p_i} \rfloor + 1)e_i).$$

Many infeasible task sets were still included in the experiments, because the only necessary and sufficient test for global EDF schedulability of n tasks on m processors known to this author has worst-case execution time of the order $O(mn \cdot \Pi_{i=1}^{n} T_i c_i)$. The author implemented and tested that algorithm, but running it on datasets of the size considered here was not practical. Reporting the performance of the efficient sufficient tests of feasibility against one another on large numbers of tasks sets seemed more useful than comparing them against a perfect but computationally impractical test on a much smaller number of task sets, with smaller periods.

Note that [8] implies that "simulation of the schedule up to the hyper-period checking for missed deadlines" is a necessary and sufficient test for schedulability. This author is not aware of any proof that such a simulation is a sufficient test for feasibility of sporadic task sets, or even of periodic tasks sets with arbitrary initial release time offsets. Even under the assumption of strictly periodic tasks and simultaneous start times, if periods can exceed deadlines simulation to the hyper-period is not sufficient.

The results of the experiments are displayed as histograms. For example, see Figure 1. *For all the histograms in this paper* the horizontal axis represents values of a task set's total density, and each bucket corresponds to a range of values $[i \cdot 0.01, (i + 1) \cdot 0.01)$. The vertical axis indicates a number of task sets. The plotted lines with datapoint symbols ("X", asterisk, square, *etc.*) show how many task sets were verifiably schedulable according to one pair of a scheduling algorithm and a schedulability test. The legend shows the meaning of each datapoint symbol. There is also a solid upper line, with legend "N", which shows the total number of task sets of the given density in the dataset, including both feasible and infeasible task sets.

3.2 Representative of Global EDF

To choose a representative for global EDF scheduling, simulations were run comparing the performance of several schedulability tests, for both pure EDF scheduling and some hybrids. The following sufficient tests for feasibility under pure global EDF scheduling were considered:

BAK Baker's test as stated in Theorem 2 above.
GFB Goossens, Funk and Baruah's test, extended to arbitrary deadlines by Bertogna, Cirinei and Lipari, as stated Theorem 3 above.
BCL Bertogna, Cirinei, and Lipari's test, as stated in Theorem 4 above.

Figures 1-2 compare the performance of these tests on two datasets, with pure global EDF scheduling. Figure 1 is for one of the datasets were the BCL test excelled. Figure 2 is for one of the datasets where the GFB excelled. Through a detailed analysis of specific cases one can verify that each of the three tests is able to verify schedulability for some task sets that are not verifiable by the other tests. However, the histograms show some clear global patterns: (1) the overall performances of the generalized GFB test and BAK test are similar, with the GFB test generally taking the lead; (2) as reported in [8], BCL does better for task

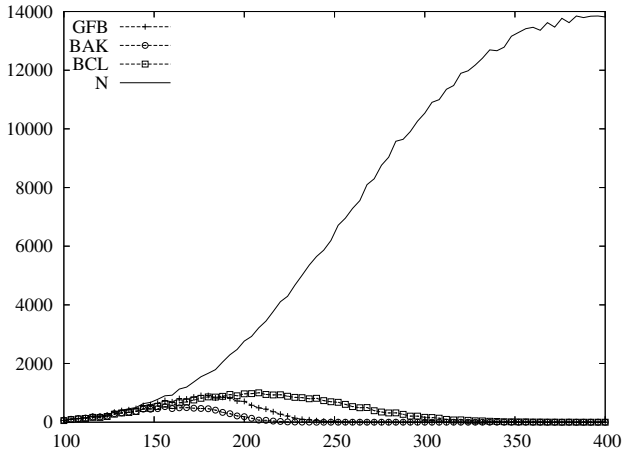


Figure 1. Constrained deadlines, bimodal utilization distribution, four processors

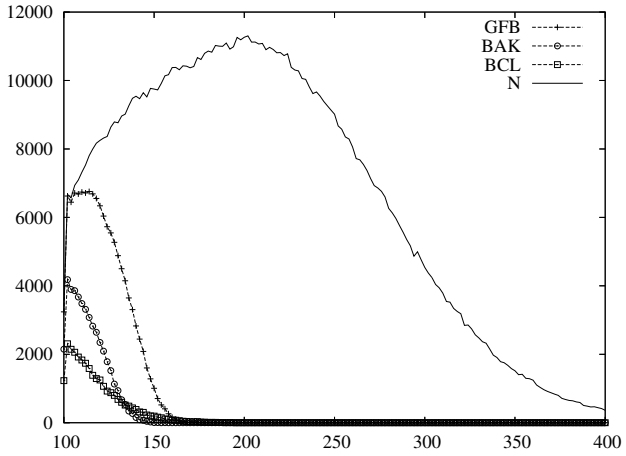


Figure 2. Constrained deadlines, exponential utilizations with mean 0.25, two processors

sets with a few heavy tasks; (3) none of the tests is able to show that very many of the task sets are schedulable. Lacking a practical necessary and sufficient test of global EDF schedulability, one cannot say for certain whether the low rate of success is a property of the global EDF scheduling policy or of the schedulability tests. However, the much better performance of the hybrid scheduling policies reported below (using the same schedulability tests) gives some evidence that the main weakness is with the pure EDF policy.

3.3 Hybrid Global Schemes

As mentioned in Section 2.4, the performance of global scheduling for systems with a few heavy tasks can be improved by giving special treatment to the heavy tasks. As the EDF-US[ζ] algorithm was originally proposed in [18], the heaviness criterion was $u_i > \zeta$. However, this idea can be generalized to other schedulability tests and the need for a fixed cut-off value ζ can be eliminated. The fundamental idea is a generalization of the PriD scheme of Goossens, Funk and Barush [13]: the k most problematic tasks ($0 \leq k < m$) for the given schedulability test are chosen to receive top priority; the rest of the tasks are scheduled according to the EDF policy. The k special tasks will certainly meet their deadlines. Schedulability of the $n - k$ remaining tasks can be verified using the pure global EDF schedulability test under the (very conservative) assumption that they need to run on the $m - k$ remaining processors. Successively larger values of k are tried, until one is found for which the system is schedulable, or until all $m - 1$ values have been tried without success.

The performance of the following hybrids of EDF and highest-utilization-first scheduling were tested, along with several other variations:

1. EDF-US[1/2]: give special priority to all the tasks of utilization greater than 1/2;
2. EDF-LM[3]: give special priority to the k tasks with highest density value (λ_i), where k is the smallest value between 0 and m for which the system can be verified as schedulable by some test for global EDF (*e.g.*, GFB, BCL, BAK).

Note that EDF-LM is actually a family of algorithms, since the choice of k is dependent on which global schedulability test is applied. Figure 3 shows the results of applying these two hybrid EDF scheduling policies, for both the GFB and the BCL tests, on the same datasets reported in Figure 1 and Figure 2. The performance of the two tests with pure global EDF scheduling is also included, for comparison. These results are typical of what was observed on all of the datasets, *i.e.*, The EDF-LM hybrid schemes clearly find a much higher number of verifiably schedulable task sets at every total utilization level than the GFB test alone.

These figures also show the comparative effectiveness of the GFB and BCL tests in the hybrid context. It can be seen that the GFB performs consistently better. Observe that Figure 1 is the same dataset, with a few heavy tasks, for which BCL seemed to have an advantage in Figure 1. Over all the tests run, the EDF-LM hybrid scheduling with the GFB test was able to verifiably schedule significantly more task sets than any of the other combinations.

3.4 Representative of Partitioned EDF

To select a representative for partitioned scheduling, several EDF-based partitioning schemes were evaluated. In each case the tasks were assigned to processors according to the first-fit algorithm in order of some metric, such as relative deadline (d_i) or density (λ_i). Two tests for fit were evaluated: (BF) the sufficient test of [4]; (BHR) the necessary and sufficient test of Baruah *et al.* [7]. The strength of the BF test is its low complexity, which is $\mathcal{O}(n)$. In contrast, the worst-case upper bound on the complexity of the BHR test is the LCM of the task periods.

[3]The "LM" in EDF-LM stands for "lambda-monotonic", since the tasks are considered in increasing order of λ_i.

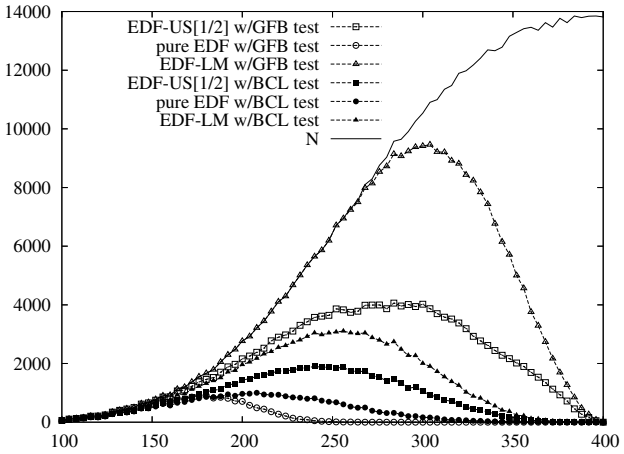


Figure 3. Pure *vs.* hybrid scheduling with GFB and BCL tests, constrained deadline, bimodal utilization, 4 CPUs

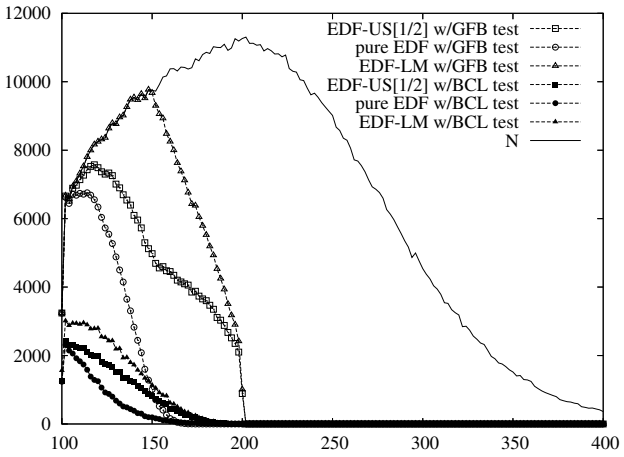


Figure 4. Pure *vs.* hybrid scheduling with GFB and BCL tests, constrained deadline, exponential utilization with mean 0.25, 2 processors

Figure 5 compares the success rates of the first-fit-decreasing algorithm with three different ordering heuristics: decreasing utilization (FFD-U); decreasing density (FFD-L); increasing relative deadline (FFD-D). With this and other datasets, the performances of FFD-L and FFD-U were very close. On some datases FFD-L dominated by a small margin, and and others the results were indistinguishable. Figure 6 compares the success rates of the FFD-L algorithm with the BHR (exact) and the BF (approximate) tests for single-processor EDF schedulability. The BF FFD-L scheme does not do quite as well as the BHR FFD-L scheme, but it is more efficiently computable and provides performance that is fairly close to the exact test.

3.5 Partitioned *versus* Global

The final set of experiments compared the performance of global EDF-based scheduling against partitioned EDF scheduling. Since the global approach already appeared

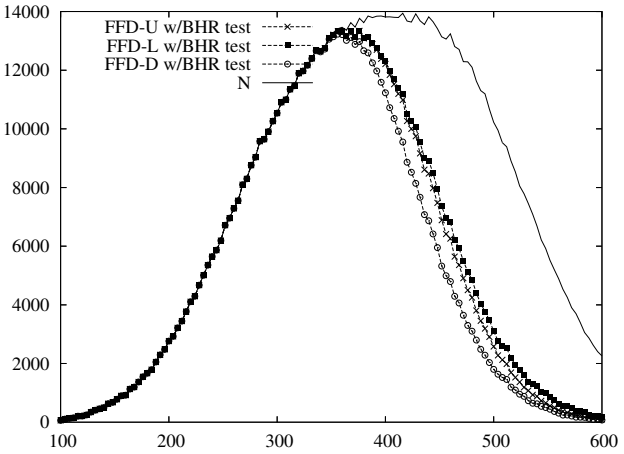


Figure 5. Performance of ordering heuristics compared, constrained deadline, bimodal utilization, 4 CPUs

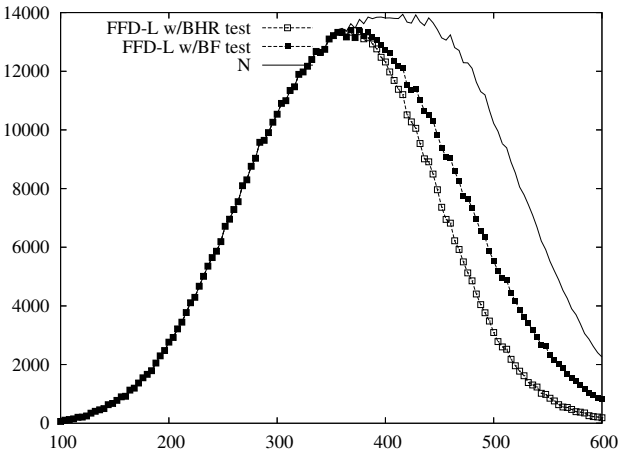


Figure 6. Performance of BHR vs BF tests compared, constrained deadline, bimodal utilization, 4 CPUs

to be the underdog, it was given every advantage: (1) besides pure global EDF, EDF-LM (the EDF hybrid based on giving extra priority to a few high-density tasks) was included in the comparison; (2) instead of applying just one schedulability test, if a task set failed the GFB test the BCL test was applied, and if it failed both of those the BAK test was applied; (3) for the partitioned approach the FFD-L (first-fit in order of decreasing density) was applied using the BF test for single-processor schedulability. The results for several datasets are shown Figures 7-8. The pattern exhibited in these examples persisted over all of the datasets tested. In all cases the hybrid global scheduling scheme improved the success rate significantly over pure EDF, but it still fell short of the success rate with partitioned scheduling.

The GFB, BAK, and BCL family of tests all seem to have an inherent limitation of density $\lambda_i = m$, because they are conceptually based on bounding density. This limitation is especially apparent for the case $m = 2$, where the drop-off is very sudden. It is clear that the partitioned methods do not have this limitation. What is not

clear is whether the limitation is a property of global EDF scheduling or just a limitation of the current generation of global schedulability tests (which seems more likely).

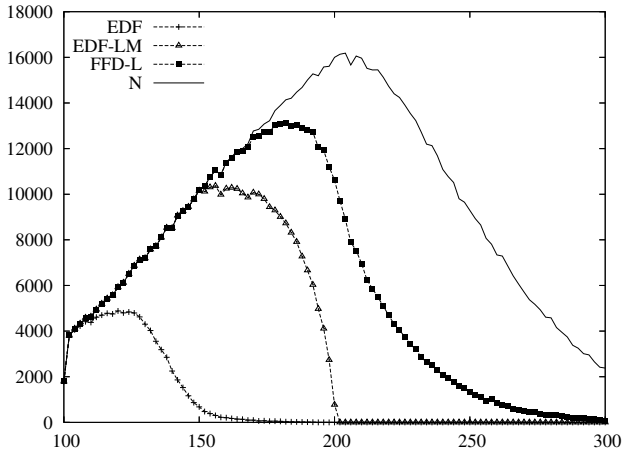


Figure 7. Constrained deadline, bimodal utilization, 2 CPUs

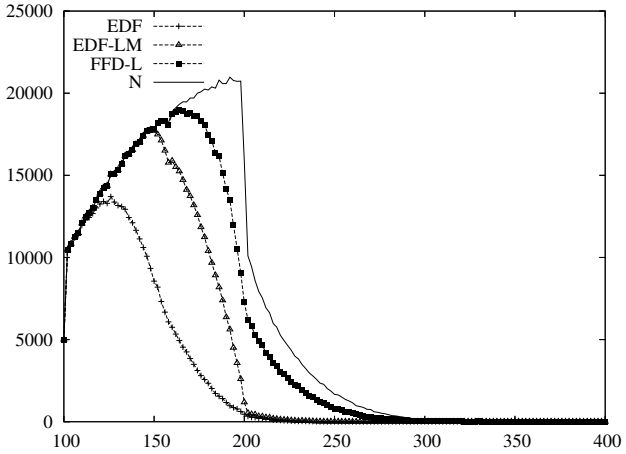


Figure 8. Unconstrained deadline, bimodal utilization, 2 CPUs

4 Conclusions and Future Work

The experiments reported here indicate that the available schedulability tests for global EDF scheduling have improved significantly. However, the global approach has not yet pulled ahead. Partitioned scheduling still appears to have an advantage over the best feasibility tests for global scheduling, with respect to the statistical chance of being able to schedule an arbitrary hard-deadline task set. If one also takes into consideration the fact that static task assignment has lower runtime overhead, partitioned scheduling looks even stronger.

This is not the end of the global *vs.* partitioned scheduling question. Further progress in the analysis of global EDF scheduling appear possible. Even if global EDF

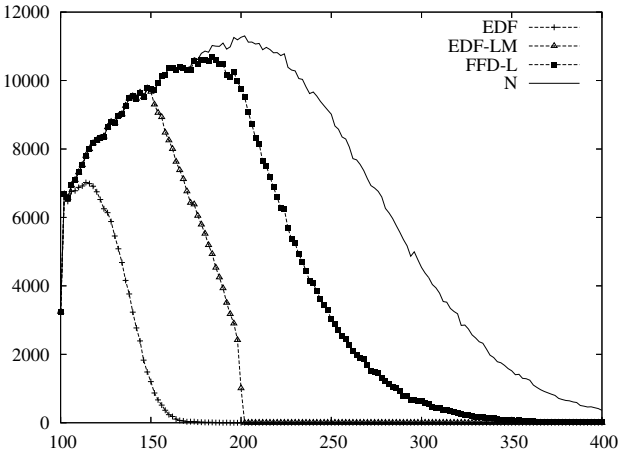


Figure 9. Constrained deadline, exponential utilization w/mean 0.25, 2 CPUs

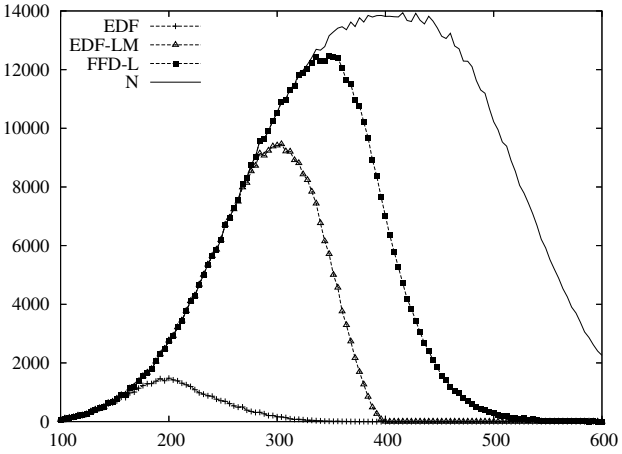


Figure 10. Constrained deadline, bimodal utilization, 4 CPUs

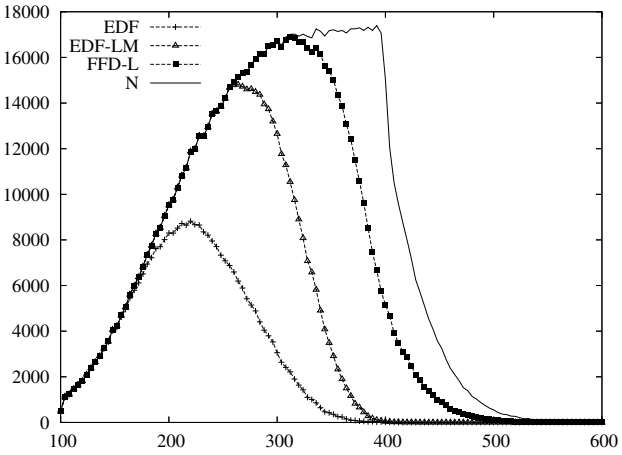


Figure 11. Unconstrained deadline, bimodal utilization, 4 CPUs

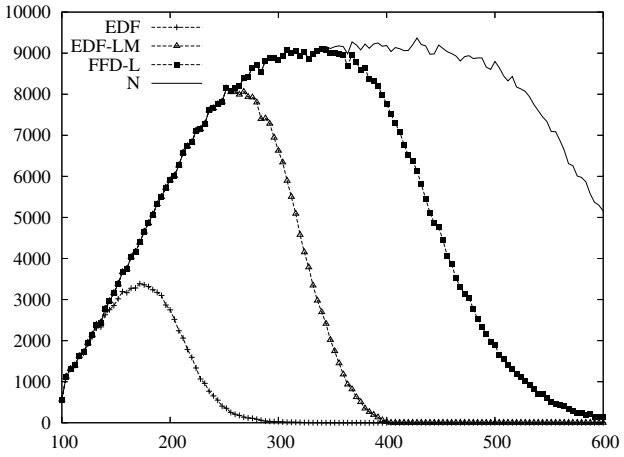


Figure 12. Constrained deadline, exponential utilization w/mean 0.25, 4 CPUs

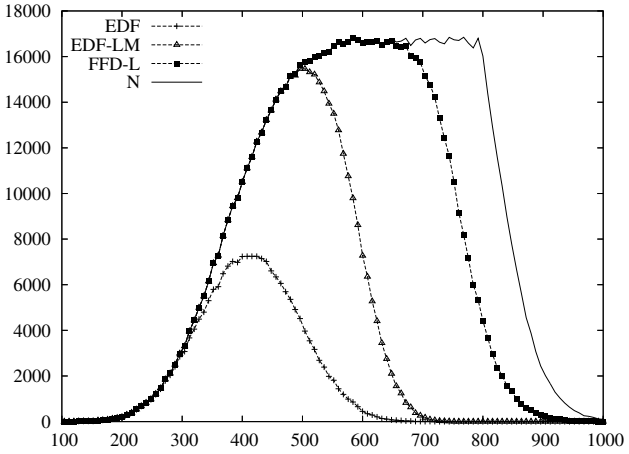


Figure 13. Unconstrained deadline, bimodal utilization, 8 CPUs

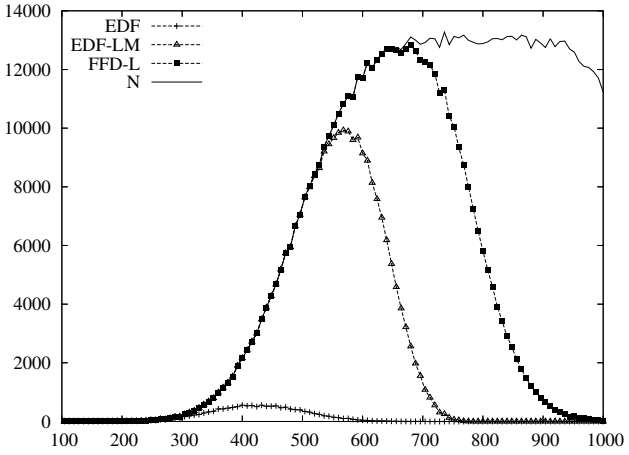


Figure 14. Constrained deadline, bimodal utilization, 8 CPUs

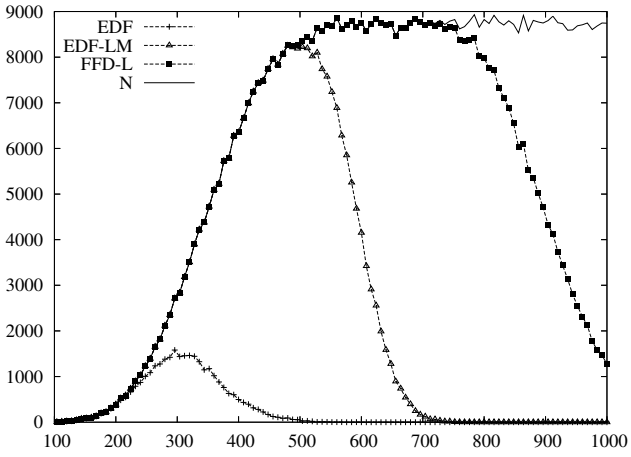


Figure 15. Constrained deadline, exponential utilization w/mean 0.25, 8 CPUs

does not ultimately prove to be competitive with partitioned EDF scheduling, there are other global scheduling schemes to be considered. Some of these can even guarantee worst-case schedulability at higher processor utilization levels than the $(m+1)/2$ bound for job-static priority scheduling.

There are several variants of the PFAIR concept. Baruah, Cohen, Plaxton and Varvel [6] showed that PFAIR scheduling is optimal for scheduling periodic tasks on a multiprocessor, has a linear-time necessary and sufficient schedulability test, and for sufficiently small quantum size can guarantee schedulability at processor utilization levels arbitrarily close to m. Srinivasan and Anderson showed that the PFAIR approach is also optimal for multiprocessor scheduling of sporadic and rate-based tasks [17], and there have been many more variations and extensions to the PFAIR theory made since that. The main problem with PFAIR scheduling is the need to slice time into small quanta, and the consequently high implementation overhead. In this regard, the fixed-job-priority algorithms, like those considered in this paper have an advantage, whether applied globally or partitioned.

Is there another algorithm that can break the $(m+1)/2$ bound but does not require such frequent time slicing as the PFAIR approach? One possibility is suggested by the work on "throw-forward" scheduling, shown by Johnson and Maddison [14] to be optimal for scheduling batches of independent jobs on a multiprocessor system. It will be interesting to see whether the idea of throw-forward scheduling (which is to combine consideration of deadline and laxity) can be extended to periodic and sporadic tasks systems and a sufficient test for schedulability found.

Of course there are also some remaining questions about the comparative implementation overhead of the global vs. partitioned approaches. Global scheduling can have higher overhead in at least two respects: the contention delay and the synchronization overhead for a single dispatching queue is higher than for per-processor queues; the cost of resuming a task may be higher if it is on

a different processor (due to interprocessor interrupt handling and cache reloading) than on the processor where it last executed. The latter cost can be quite variable, since it depends on the actual portion of a task's memory that remains in cache when the task resumes execution, and how much of that remnant will be referenced again before it is overwritten. These issues are discussed at some length by Srinivasan *et al.* in [19], which includes some simulation results comparing the overhead of global EDF and PD^2 scheduling, a PFAIR variant. It seems that only experimentation with actual implementations can make a conclusive case as to how serious are these overheads, and how they balance against any advantages global scheduling may have for on-time completion of tasks in real applications.

Acknowledgments

The author thanks the referees for their constructive criticisms, including the advice to omit some parts of the original submission, including the description and evaluation of an improved version of the BAK global EDF schedulability test whose proof of correctness would not fit, figures showing the obvious fact that combining the GFB, BAK, and BCL tests results in improved accuracy as compared to the individual tests. The author is also grateful to Michele Cirinei for proof-reading the final copy of this paper and catching several errors.

References

[1] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proc. 22nd IEEE Real-Time Systems Symposium*, pages 193–202, London, UK, Dec. 2001.

[2] T. P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proc. 24th IEEE Real-Time Systems Symposium*, pages 120–129, 2003.

[3] T. P. Baker. An analysis of EDF scheduling on a multiprocessor. *IEEE Trans. on Parallel and Distributed Systems*, 15(8):760–768, Aug. 2005.

[4] S. Baruah and N. Fisher. Partitioned multiprocessor scheduling of sporadic task systems. In *Proc. of the 26th IEEE Real-Time Systems Symposium*, Miami, Florida, Dec. 2005.

[5] S. Baruah and J. Goossens. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Trans. Computers*, 52(7):966–970, July 2003.

[6] S. K. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel. Proportionate progress: a notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.

[7] S. K. Baruah, R. R. Howell, and L. E. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *Real-Time Systems*, 2, 1990.

[8] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proc. 17th Euromicro Conference on Real-Time Systems*, pages 209–218, Palma de Mallorca, Spain, July 2005.

[9] M. Bertogna, M. Cirinei, and G. Lipari. New schedulability tests for real-time tasks sets scheduled by deadline monotonic on multiprocessors. In *Proc. of the 9th International Conf. on Principles of Distributed Systems*, Pisa, Italy, Dec. 2005.

[10] N. Fisher, T. P. Baker, and S. Baruah. Algorithms for determining the demand-based load of a sporadic task system. Submitted for Publication, 2006.

[11] N. Fisher and S. Baruah. The partitioned, static-priority scheduling of sporadic real-time tasks with constrained deadlines on multiprocessor platforms. In *Proc. of the 9th International Conf. on Principles of Distributed Systems*, Pisa, Italy, Dec. 2005.

[12] S. Funk, J. Goossens, and S. Baruah. On-line scheduling on uniform multiprocessors. In *Proc. 22nd IEEE Real-Time Systems Symposium*, pages 183–192, London, UK, Dec. 2001. IEEE Computer Society.

[13] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real Time Systems*, 25(2–3):187–205, 2003.

[14] H. H. Johnson and M. S. Maddison. Deadline scheduling for a real-time multiprocessor. In *Proc. Eurocomp Conference*, pages 139–153, 1974.

[15] L. Kleinrock. *Queueing Systems - Volume 2: Computer Applications*. Wiley Interscience, 1976.

[16] J. M. Lopez, M. Garcia, J. L. Diaz, and D. F. Garcia. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proc. 12th Euromicro Conf. Real-Time Systems*, pages 25–33, 2000.

[17] A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. In *Proc. 34th ACM Symposium on Theory of Computing*, pages 189–198. ACM, May 2002.

[18] A. Srinivasan and S. Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84:93–98, 2002.

[19] A. Srinivasan, P. Holman, J. H. Anderson, and S. Baruah. The case for fair multiprocessor scheduling. In *Proc. 11th International Workshop on Parallel and Distributed Real-time Systems*, Apr. 2003.

Resource and Data Management II

Solving Allocation Problems of Hard Real-Time Systems with Dynamic Constraint Programming

Pierre-Emmanuel Hladik[1], Hadrien Cambazard[2], Anne-Marie Déplanche[1], Narendra Jussien[2]
[1] IRCCyN, UMR CNRS 6597
1 rue de la Noë – BP 9210
44321 Nantes Cedex 3, France
{hladik,deplanche}@irccyn.ec-nantes.fr
[2] École des Mines de Nantes, LINA CNRS
4 rue Alfred Kastler – BP 20722
44307 Nantes Cedex 3, France
{hcambaza,jussien}@emn.fr

Abstract

In this paper, we present an original approach (CPRTA for "Constraint Programming for solving Real-Time Allocation") based on constraint programming to solve an allocation problem of hard real-time tasks in the context of fixed priority preemptive scheduling. CPRTA is built on dynamic constraint programming together with a learning method to find a feasible processor allocation under constraints. It is a new approach which produce in its current version as acceptable performances as classical algorithms do. Some experimental results are given to show it. Moreover, CPRTA shows very interesting properties. It is complete —i.e., if a problem has no solution, the algorithm is able to prove it—, and it is non-parametric —i.e., it does not require specific initializations—. Thanks to its capacity to explain failures, it offers attractive perspectives for guiding the architectural design process.

1. Introduction

Real-time systems have applications in many industrial areas: telecommunication systems, automotive, aircraft, robotics, etc. Today's applications are becoming more and more complex, as much in their software part (an increasing number of concurrent tasks with various interaction schemes), as in their execution platform (many distributed processing units interconnected through specialized network(s)), and in their numerous functional and non-functional requirements too (timing, resource, power, etc. constraints). One of the main issues in the architectural design of such complex distributed applications is to define an allocation of tasks onto processors so as to meet all the specified requirements. In general, it is a difficult constraint satisfaction problem. Even if it has to be solved off-line most of the time, it needs efficient and adaptable search techniques which are able to be integrated into a more global design process. Furthermore, it is desirable that those techniques return relevant information intended to help the designer who is faced with architectural choices. The "binary" result, in particular, (has a feasible allocation been found?: yes and here it is, or no, and that's all) which is usually returned by the search algorithm is not satisfactory in failure situations. The designer would expect some explanations justifying the failure and enabling him to revisit his design. Therefore, more sophisticated search techniques that would be able to collect some knowledge about the problem they solve are required. Here are the general objectives of the work we are conducting.

More precisely, the problem we are concerned with consists in assigning a set of periodic, preemptive tasks to distributed processors in the context of fixed priority scheduling, to respect schedulability but also to account for requirements related to memory capacity, coresidence, redundancy, and so on. We assume that the characteristics of tasks (execution time, priority, etc.) and the ones of the physical architecture (processors and network) are all known a priori —Only static real-time systems are here considered—.

Assigning a set of hard preemptive real-time tasks in a distributed system under allocation and resource constraints is known to be an NP-Hard problem [14]. Up to now, it has been massively tackled with heuristic methods [18], simulated annealing [21] and genetic algorithms [16]. Recently, Szymanek et al. [20] and especially Ekelin [7] have used constraint programming to produce an assignment and a pre-runtime scheduling of distributed systems under optimization criteria. Even if their context is different from ours, their results have shown the ability of such an innovative approach to solve an allocation problem for embedded systems and have encouraged us to go further.

Like numerous hybridation schemes [9, 4], the way we are investigating uses the complementary strengths of constraint programming and optimization methods from operational research. In this paper, we present its principle and study its performances. It is a decomposition-based method (related to logic Benders-based decomposition [9]) which separates the allocation problem from the scheduling one: the allocation problem is solved by means of *dynamic constraint programming* tools, whereas

the scheduling problem is treated with specific real-time schedulability analysis. The main idea is to "learn" from the schedulability analysis to re-model the allocation problem so as to reduce the search space. In that sense, we can compare this approach to a form of *learning from mistakes*. Lastly we underline that a fundamental property of this method is the completeness : when a problem has no solution, it is able to prove it (contrary to heuristic methods that are unable to decide).

The remainder of this paper is organized as follows. In section 2, we describe the problem. Section 3 is dedicated to the description of the master- and sub-problems, and the relations between them. The logical Benders decomposition scheme is briefly introduced and the links with our approach are put forward. In Section 4 the method is applied to a case study. Some experimental results are presented in Section 5. Section 6 shows how it is possible to set up a failure analysis able to aid the designer to review his plans. It is a first attempt that proves its feasibility and will need to go deeper. The paper ends with concluding remarks in Section 7.

2 The problem description

2.1 The real-time system architecture

The hard real-time system we consider can be modeled by a software architecture: the set of tasks, and a hardware architecture: the execution platform for the tasks, as represented in Fig. 1.

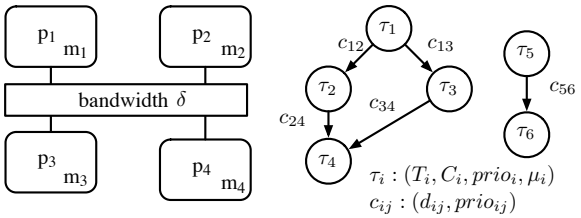


Figure 1. An example of hardware (left) and software (right) architecture.

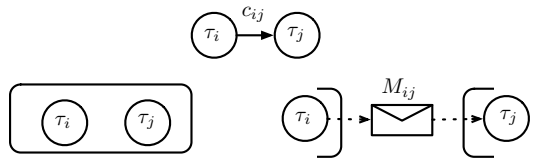
By *hardware architecture* we mean a set $\mathcal{P} = \{p_1, \ldots, p_k, \ldots, p_m\}$ of m processors with fixed memory capacity m_k and identical processing speed. Each processor schedules tasks assigned to it with a fixed priority strategy. It is a simple rule : a static priority is given to each task and at run-time, the ready task with the highest priority is put in the running state, preempting eventually a lower priority task. Those processors are fully connected through a communication medium with a bandwidth δ. In this paper, we look at a communication medium called a *CAN bus* which is currently used in a wide spectrum of real-time embedded systems. However any other communication network could be considered as far as its timing behaviour (including its protocol rules) is predictable. Thus the first experiments we have conducted addressed a token ring network.

CAN (Controller Area Network) [5] is both a protocol and physical network. CAN works as a broadcast bus meaning that all connected nodes will be able to read all messages sent on the bus. Each message has a unique identifier which is also used as the message priority. On each node waiting messages are queued. The bus makes sure that when a new message gets selected to transfer, the message with the highest priority, waiting on any connected node, will get transmitted first. When at least one bit of a message has started to be transfered it can't get preempted even though higher priority messages arrive. As a result, the CAN's behaviour will be seen subsequently as the one of a non preemptive fixed priority message scheduling.

The *software architecture* is modeled as a valued, oriented and acyclic graph $(\mathcal{T}, \mathcal{C})$. The set of nodes $\mathcal{T} = \{\tau_1, ..., \tau_n\}$ represents the tasks. A task in turn is a set of instructions which must be executed sequentially in the same processor. The set of edges $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T}$ refers to the data sent between tasks.

A task τ_i is defined through timing characteristics and resource needs: its period T_i (as a task is periodically activated ; the date of its first activation is free), its worst-case execution time without preemption C_i and its memory need μ_i. A priority $prio_i$ is given to each task. Task τ_j has priority over τ_i if and only if $prio_i < prio_j$. Edges $c_{ij} = (\tau_i, \tau_j) \in \mathcal{C}$ are weighted with its tramission time C_{ij} (the time it takes to transfer the message on the bus) together with a priority value $prio_{ij}$ (useful in the CAN context). Task priorities are assumed to be different. The same assumption is made on message priorities. In this model, we assume that communicating tasks have the same activation period. However, we don't consider any precedence constraint between them : they are periodically activated in an independent way, and they read input data and write output data at the beginning and the end of their execution.

The underlying communication model is inspired from OSEK-COM specifications [17]. OSEK-COM is an uniform communication environment for automotive control unit application software. It defines common software communication interface and behaviour for internal communications (within an electronic control unit) and external ones (between networked vehicle nodes) which is independent of the communication protocol used. It is the following. Tasks that are located on the same processor communicate through local memory sharing. Such a local communication cost is assumed to be zero. On the other hand, when two communicating tasks are assigned to two distinct processors, the data exchange needs the transmission of a message on the network. Here we are interested with the *periodic transmission mode* of OSEK-COM. In this mode data production and message transmission aren't synchronised : a producer task writes its output data into a local unqueued buffer from where a pe-



(a) tasks are allocated on the same processor

(b) tasks are allocated on different processors

Figure 2. Depending of the task allocation, a message exists, or not.

riodic protocol service reads it and sends it into a message. The building of protocol data units considered here is very simple : each data that has to be sent from a producer task τ_i to a consumer task τ_j in a distant way gives rise to its proper message M_{ij}. Moreover in this paper, for a sake of simplicity, the *asynchronous receiving mode* is preferred. It means that the release of a consumer task τ_j is strictly periodic and unrelated with the M_{ij} message arrival : when a node receives a message from the bus, its protocol records its data into a local unqueued buffer from where it can be read by the task τ_j. In [8] an extension of this work to a *synchronous receiving mode* is proposed in which a message reception notification activates the consumer task.

As a result, depending on the task allocation, an edge c_{ij} of the software architecture may give rise to two different equivalent schemes as illustrated in Fig. 2. In Fig. 2(b), M_{ij} inherits its period T_i from τ_i and its priority $prio_{ij}$ from c_{ij}.

Therefore from a scheduling point of view, messages on the bus are very similar to tasks on a processor. Like for tasks, each message M_{ij} is "activated" every T_i units of time; its (bus) priority is $prio_{ij}$; and it has a transmission time C_{ij}.

2.2 The allocation problem

An allocation is a mapping $A : \mathcal{T} \rightarrow \mathcal{P}$ such that:

$$\tau_i \mapsto A(\tau_i) = p_k \tag{1}$$

The allocation problem consists in finding the mapping A which respects the whole set of constraints described in the immediate below.

Timing constraints. They are expressed by the means of relative deadlines for the tasks. A timing constraint enforces the duration between the activation date of any instance of the task τ_i and its completion time to be bounded by its relative deadline D_i. Depending on the task allocation, such timing constraints may concern the instanciated messages too. For tasks as well as messages, their relative deadline is hereafter assumed equal to their activation period.

Resource constraints. Three kinds of constraints are considered —precise units aren't specified but obviously they have to be consistent with the given expressions—:

- **Memory capacity**: The memory use of a processor p_k cannot not exceed its capacity (m_k):

$$\forall k = 1..m, \sum_{A(\tau_i)=p_k} \mu_i \leq m_k \tag{2}$$

- **Utilization factor**: The utilization factor of a processor cannot exceed its processing capacity. The following inequality is a necessary schedulability condition :

$$\forall k = 1..m, \sum_{A(\tau_i)=p_k} \frac{C_i}{T_i} \leq 1 \tag{3}$$

- **Network use**: To avoid overload, the messages carried along the network per unit of time cannot exceed the network capacity:

$$\sum_{\substack{c_{ij}=(\tau_i,\tau_j) \\ A(\tau_i) \neq A(\tau_j)}} \frac{C_{ij}}{T_i} \leq 1 \tag{4}$$

Allocation constraints. Allocation constraints are due to the system architecture. We distinguish three kinds of constraints.

- **Residence**: a task may need a specific hardware or software resource which is only available on specific processors (*e.g.* a task monitoring a sensor has to run on a processor connected to the input peripheral). This constraint is expressed as a couple (τ_i, α) where $\tau_i \in \mathcal{T}$ is a task and $\alpha \subseteq \mathcal{P}$ is the set of available host processors for the task. A given allocation A must respect:

$$A(\tau_i) \in \alpha \tag{5}$$

- **Co-residence**: This constraint enforces several tasks to be assigned to the same processor (they share a common resource). Such a constraint is defined by a set of tasks $\beta \subseteq \mathcal{T}$ and any allocation A has to fulfil:

$$\forall (\tau_i, \tau_j) \in \beta^2, A(\tau_i) = A(\tau_j) \tag{6}$$

- **Exclusion**: Some tasks may be replicated for some fault-tolerance objectives and therefore cannot be assigned to the same processor. It corresponds to a set $\gamma \subseteq \mathcal{T}$ of tasks which cannot be placed together. An allocation A must satisfy:

$$\forall (\tau_i, \tau_j) \in \gamma^2, A(\tau_i) \neq A(\tau_j) \tag{7}$$

An allocation A is said to be *valid* if it satisfies allocation and resource constraints. It is *schedulable* if it satisfies timing constraints. Finally, a solution to our problem is a valid and schedulable allocation of the tasks.

3 Solving the problem

Constraint programming (CP) techniques have been widely used to solve a large range of combinatorial problems. They have proved quite effective in a wide range of applications (from planning and scheduling to finance – portfolio optimization – through biology) thanks to main advantages: declarativity (the variables, domains, constraints description), genericity (it is not a problem dependent technique) and adaptability (to unexpected side constraints).

A *constraint satisfaction problem* (CSP) consists of a set V of variables defined by a corresponding set D of possible values (the so-called *domain*) and a set C of constraints. A solution to the problem is an assignment of a value in D to each variable in V such that all constraints are satisfied. This mechanism coupled with a backtracking scheme allows the search space to be explored in a *complete way*. For a deeper introduction to CP, we refer to [2].

3.1 Solving strategy : Logic-based Benders decomposition in CP

Due to space limitation, we only give the basic principles of this technique. Our approach is based on an extension of a Benders scheme. A Benders decomposition [3] is a solving strategy of linear problems that uses a partition of the problem among its variables: x, y. A master problem considers only x, whereas a subproblem tries to complete the assignment on y and produces a Benders cut added to the master. This cut is the central element of the technique, it is usually a linear constraint on x inferred by the dual of the subproblem. Benders decomposition can therefore be seen as a form of *learning from mistakes*.

For a discrete satisfaction problem, the resolution of the dual consists in computing the infeasibility proof of the subproblem (in this case, the dual is called an *inference dual*) and determining under what conditions the proof remains valid to infer valid cuts. The Benders cut can be seen in this context as an explanation of failure which is learnt by the master. We refer here to a more general Benders scheme called *logic Benders decomposition* [9] where any kind of subproblems can be used as long as the inference dual of the subproblem can be solved.

We propose an approach inspired from methods used to integrate constraint programming into a logic-based Benders decomposition [4]. The allocation and resource constraints are considered on one side, and schedulability on the other (see Fig. 3). The master problem solved with constraint programming yields a valid allocation. The subproblem checks the schedulability of this allocation, eventually finds out why it is unschedulable and designs a set of constraints, named *nogoods* which rules out all the assignments which are unschedulable for the same reason.

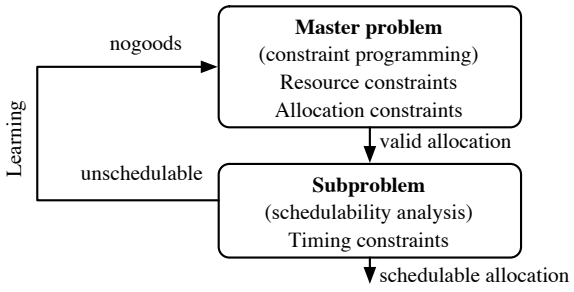


Figure 3. Logic-based Benders decomposition to solve an allocation problem

3.2 Master problem

As the master problem is solved using constraint programming techniques, we need first to translate our problem into CSP. The model is based on a redundant formulation using three kinds of variables: x, y, w.

Let us first consider n integer-valued variables x which are decision variables and correspond to each task, representing the processor selected to process the task: $\forall i \in \{1..n\},\ x_i \in \{1,\ldots,m\}$. Then, boolean variables y indicate the presence of a task on a processor: $\forall i \in \{1..n\}, \forall p \in \{1..m\},\ y_{ip} \in \{0,1\}$. Finally, boolean variables w are introduced to express whether a pair of tasks exchanging data are located on the same processor or not: $\forall c_{ij} = (\tau_i, \tau_j) \in \mathcal{C},\ w_{ij} \in \{0,1\}$. Integrity constraints are used to enforce the consistency of the redundant model.

One of the main objectives of the master problem is to solve efficiently the assignment part. It handles two kinds of constraints: allocation and resource.

- **Residence:** (*cf.* Eq. (5)) it consists of forbidden values for x. A constraint is added for each forbidden processor p of τ_i: $x_i \neq p$
- **Co-residence:** (*cf.* Eq. (6)) $\forall (\tau_i, \tau_j) \in \beta^2, x_i = x_j$
- **Exclusion:** (*cf.* Eq. (7)) *AllDifferent*$(x_i | \tau_i \in \gamma)$. An *AllDifferent* constraint on a set V of variables ensures that all variables among V are different.
- **Memory capacity:** (*cf.* Eq. (2)) $\forall p \in \{1..m\},\ \sum_{i \in \{1..n\}} y_{ip}\mu_i \leq \mu_p$
- **Utilization factor:** (*cf.* Eq. (3)) Let $\mathrm{lcm}(T)$ be the least common multiple of periods of the tasks — utilization factor and network use are reformulated with the lcm of task periods because our constraint solver cannot currently handle constraints with both real coefficients and integer variables—. The constraint can be written as follows:

$$\forall p \in \{1..m\}, \qquad \sum_{i \in \{1..n\}} \frac{y_{ip}\,\mathrm{lcm}(T) C_i}{T_i} \leq \mathrm{lcm}(T)$$

- **Network use:** (*cf.* Eq. (4)) The network capacity is bound by δ. Therefore, the size of the set of messages carried on the network cannot exceed this limit:

$$\sum_{i\in\{1..n\}j\in\{1..n\}} \frac{w_{ij}\,\mathrm{lcm}(T)C_{ij}}{T_i} \leq \mathrm{lcm}(T)$$

3.3 Subproblem

The subproblem we consider here is to check whether a valid solution produced by the master problem is schedulable or not. A widely chosen approach for the schedulability analysis of a task set S is based on the following necessary and sufficient condition [15] : S is schedulable if and only if, for each task of S, its worst-case response time is less or equal to its relative deadline. Thus the subproblem solving leads us to compute worst-case response times for tasks on processors and for messages on the bus. According to the features of the considered task and message models, as well as the processor and bus scheduling algorithms, a "classical" computation can be used and its main results are given in the immediate following.

Task worst-case response time. For independent and periodic tasks with a preemptive fixed priority scheduling algorithm, it has been proven that the worst execution scenario for a task τ_i happens when it is released simultaneously with all the tasks which have a priority higher than $prio_i$. When D_i is (less or) equal to T_i, the worst-case response time for τ_i is given by [15]:

$$R_i = C_i + \sum_{\tau_j\in hp_i(A)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \qquad (8)$$

where $hp_i(A)$ is the set of tasks with a priority higher than $prio_i$ and located on the processor $A(\tau_i)$ for a given allocation A, and $\lceil x \rceil$ calculates the smallest integer $\geq x$. The summation gives us the number of times tasks with higher priority will execute before τ_i has completed. The worst-case response time R_i can be easily solved by looking for the fix-point of Eq. (8) in an iterative way.

Message worst-case response time. As mentioned earlier, message scheduling on the CAN bus can be viewed as a non-preemptive fixed priority scheduling strategy. Thus when doing a worst-case response time equation for a message, Eq. (8) has to be reused with some modifications. First it has to be changed so that a message only can be preempted during its first transmitted bit instead of its whole execution time. Second a blocking time, i.e. the largest time the message might be blocked by a lower priority message, must be added. The resulting worst-case response time equation for the CAN message M_{ij} is [22]:

$$R_{ij} = C_{ij} + L_{ij} \qquad (9)$$

with

$$L_{ij} = \sum_{M'\in hp_{ij}(A)} \left\lceil \frac{L_{ij}+\tau_{bit}}{T'} \right\rceil C' + \max_{M'\in lp_{ij}(A)} \{C' - \tau_{bit}\} \qquad (10)$$

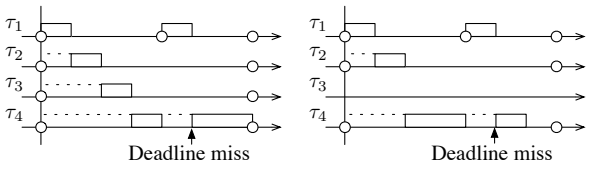


Figure 4. Illustration of a schedulability analysis. The task τ_4 does not meet its deadline. The subset $\{\tau_1, \tau_2, \tau_4\}$ is identified to explain the unschedulability of the system.

where $hp_{ij}(A)$ (respectively $lp_{ij}(A)$) is the set of messages derived from the allocation A with a priority higher (respectively lower) than $prio_{ij}$; τ_{bit} is the transmission time for one bit (τ_{bit} is in relation with the bus bandwidth δ, $\tau_{bit} = 1/\delta$) ; C' is the worst-case transmission time for the message M'.

Here as well the computation of Eq. (10) can be solved iteratively.

3.4 Cooperation between master and subproblem(s)

We now consider a valid allocation (as the one the constraint programming solver may propose) in which some tasks are not schedulable. Our purpose is to explain why this allocation is unschedulable, and to translate this into a new constraint for the master problem.

Tasks. The explanation for the unschedulability of a task τ_i is the presence of tasks with higher priority on the same processor that interfere with τ_i. For any other allocation with τ_i and $hp_i(A)$ on the same processor, it is sure that τ_i will still be detected unschedulable. Therefore, the master problem must be constrained so that all solutions where τ_i and $hp_i(A)$ are together are not considered any further. This constraint corresponds to a *NotAllEqual* on x —A *NotAllEqual* on a set V of variables ensures that at least two variables among V take distinct values—:

$$\textit{NotAllEqual}\,(x_j | \tau_j \in S_i(A) = hp_i(A) \cup \{\tau_i\})$$

It is worth noticing that this constraint could be expressed as a linear combination of variables y. However, *NotAllEqual(x_1,x_3,x_4)* excludes the solutions that contain the tasks τ_1, τ_3, τ_4 gathered on *any* processor.

It is easy to see that this constraint is not totally relevant. For example, in Fig. 4, τ_4 that shares a processor with τ_1,τ_2 and τ_3 misses its deadline. Actually the set $S_4(A) = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ explains the unschedulability but it is not minimal in the sense that if we remove one task from it, the set is still unschedulable. Here, the set $S_4(A)' = \{\tau_1, \tau_2, \tau_4\}$ is sufficient to justify the unschedulability.

In order to derive more precise explanations (to achieve a more relevant learning), a conflict detection algorithm,

namely *QuickXplain* [10] (see algorithm 1), has been used to determine a minimal (*w.r.t.* inclusion) set of involved tasks $S_i(A)'$. A new function is defined, $R_i(X)$, as the worst-case response time of τ_i as if it was scheduled with those tasks belonging to the set X that have priority over it:

$$R_i(X) = C_i + \sum_{\tau_j \in hp_i(A) \cap X} \left\lceil \frac{R_i(X)}{T_j} \right\rceil C_j \qquad (11)$$

Algorithm 1 Minimal task set

QUICKXPLAINTASK(τ_i, A, D_i)
  $X := \emptyset$
  $\sigma_1, ..., \sigma_{\#hp_i(A)}$ {an enumeration of $hp_i(A)$. The enumeration order of $hp_i(A)$ may have an effect on the content of the returned minimal task set}
  while $R_i(X) \leq D_i$ **do**
    $k := 0$
    $Y := X$
    while $R_i(Y) \leq D_i$ $k < \#hp_i(A)$ **do**
      $k := k + 1$
      $Y := Y \cup \{\sigma_k\}$ {according to the enumeration order}
    end while
    $X := X \cup \{\sigma_k\}$
  end while
  return $X \cup \{\tau_i\}$

Messages. The reasoning is quite similar. If a message M_{ij} is found unschedulable, it is because of the messages in $hp_{ij}(A)$ and the longest message in $lp_{ij}(A)$. We denote $M_{ij}(A)$ their union together with $\{M_{ij}\}$. The translation of this information in term of constraint yields to:

$$\sum_{M_{ab} \in M_{ij}(A)} w_{ab} < \#M_{ij}(A)$$

where $\#X$ stands for the cardinality of X.

It is equivalent to a *NotAllEqual* constraint on a set of messages since to be met it requires that at least one message of $M_{ij}(A)$ "disappear" ($w_{ab} = 0$).

Like for tasks, so as to reduce the set of involved messages, QUICKXPLAIN has been implemented, using a similar adaptation of Eq. (9) and (10). It returns a minimal set of messages $M_{ij}(A)'$.

Integration of nogoods in constraint programming solver. Dynamic integration of nogoods at any step of the search performed by the MAC (Maintaining arc consistency) algorithm of the constraint solver is based on the use of explanations. Explanations consist of a set of constraints and record enough information to justify any decision of the solver such as a reduction of domain or a contradiction. Dynamic addition/retraction of constraints are possible when explanations are maintained [12].

For example, the addition of a constraint at a leaf of the tree search will not lead to a standard backtracking from that leaf (which could be very inefficient as a wrong choice may exist at the beginning of the search because the constraint was not known at that time). Instead, the solver will jump (MAC-CBJ for conflict directed backjumping) to a node appearing in the explanation and therefore responsible for the contradiction raised by the new constraint. More complex and more efficient techniques such as MAC-DBT (for dynamic backtracking) exist to perform intelligent repair of the solution after the addition or retraction of a constraint.

4 Applying the method to an example

An example to illustrate the theory is developed hereafter. It will show how the cooperation between master- and sub-problems is performed. Table 1 shows the characteristics of the considered hardware architecture (with 4 processors) and Table 2 those of the software architecture (with 20 tasks). The entry "$x, y \rightarrow$ j" for the task τ_i indicates an edge c_{ij} with $C_{ij} = x$ and $prio_{ij} = y$.

p_i	p_0	p_1	p_2	p_3
m_i	102001	280295	360241	41617

Table 1. Processor characteristics

τ_i	T_i	C_i	μ_i	$prio_i$	Message
τ_0	36000	2190	21243	1	600,1 $\rightarrow$ 13
τ_1	2000	563	5855	6	500,3 $\rightarrow$ 8
τ_2	3000	207	2152	15	600,7 $\rightarrow$ 7
τ_3	8000	2187	21213	3	
τ_4	72000	17690	168055	7	300,4 $\rightarrow$ 9
τ_5	4000	667	6670	8	800,5 $\rightarrow$ 19
τ_6	12000	3662	36253	14	
τ_7	3000	269	2743	16	
τ_8	2000	231	2263	12	100,6 $\rightarrow$ 18
τ_9	72000	6161	59761	9	
τ_{10}	12000	846	8206	4	200,2 $\rightarrow$ 15
τ_{11}	36000	5836	60694	20	
τ_{12}	9000	2103	20399	10	
τ_{13}	36000	5535	54243	13	
τ_{14}	18000	3905	41002	18	
τ_{15}	12000	1412	14402	5	
τ_{16}	6000	1416	14301	17	700,8 $\rightarrow$ 17
τ_{17}	6000	752	7369	19	
τ_{18}	2000	538	5487	11	
τ_{19}	4000	1281	12425	2	

Table 2. Task and message characteristics

The problem is constrained by :

- residence constraints:
  - CC_1 : τ_0 must be allocated to p_0 or p_1 or p_2.
  - CC_2 : τ_{16} must be allocated to p_1 or p_2.
  - CC_3 : τ_{17} must be allocated to p_0 or p_3.

- co-residence constraint:
  - CC_4 : τ_7, τ_{17} and τ_{19} must be on the same processor.
- exclusion constraints:
  - CC_5 : τ_3, τ_{11} and τ_{12} must be on different processors.

To start the resolution process, the solver for the master problem finds a valid solution in accordance with CC_1, CC_2, CC_3, CC_4 and CC_5. How the constraint programming solver finds such a solution is here out of our purpose. The valid solution it returns is:

- processor p_0: τ_2, τ_5, τ_7, τ_8, τ_9, τ_{17}, τ_{19}.
- processor p_1: τ_4, τ_6, τ_{12}, τ_{13}.
- processor p_2: τ_0, τ_{11}, τ_{14}, τ_{15}, τ_{16}.
- processor p_3: τ_1, τ_3, τ_{10}, τ_{18}.

One deduces that messages are $M_{0,13}$, $M_{1,8}$, $M_{4,9}$, $M_{8,18}$, $M_{10,15}$, and $M_{16,17}$.

It is easy to check it is a valid solution by considering allocation and resource constraints:

- $\mu_2+\mu_5+\mu_7+\mu_8+\mu_9+\mu_{17}+\mu_{19} = 93383 \leq m_0$;
- $\mu_4+\mu_6+\mu_{12}+\mu_{13} = 278950 \leq m_1$;
- $\mu_0+\mu_{11}+\mu_{14}+\mu_{15}+\mu_{16} = 151642 \leq m_2$;
- $\mu_1+\mu_3+\mu_{10}+\mu_{18} = 40761 \leq m_3$;
- $\frac{C_2}{T_2}+\frac{C_5}{T_5}+\frac{C_7}{T_7}+\frac{C_8}{T_8}+\frac{C_9}{T_9}+\frac{C_{17}}{T_{17}}+\frac{C_{19}}{T_{19}} = 0.972 \leq 1$;
- $\frac{C_4}{T_4}+\frac{C_6}{T_6}+\frac{C_{12}}{T_{12}}+\frac{C_{13}}{T_{13}} = 0.938 \leq 1$;
- $\frac{C_0}{T_0}+\frac{C_{11}}{T_{11}}+\frac{C_{14}}{T_{14}}+\frac{C_{15}}{T_{15}}+\frac{C_{16}}{T_{16}} = 0.794 \leq 1$;
- $\frac{C_1}{T_1}+\frac{C_3}{T_3}+\frac{C_{10}}{T_{10}}+\frac{C_{18}}{T_{18}} = 0.894 \leq 1$.
- $\frac{C_{0,13}}{T_0}+\frac{C_{1,8}}{T_1}+\frac{C_{4,9}}{T_4}+\frac{C_{8,18}}{T_8}+\frac{C_{10,15}}{T_{10}}+\frac{C_{16,17}}{T_{16}} = 0.454 \leq 1$.

The subproblem checks now the schedulability of the valid solution. The schedulability analysis proceeds in three steps.

First step: analysing the schedulability of tasks. The worst-case response time for each task is obtained by application of Eq. (8) and it is compared with its relative deadline. Here τ_5, τ_{12}, τ_{16} and τ_{19} are found unschedulable.

Second step: analysing the schedulability of messages. The worst-case response time for each message is obtained by application of Eq. (9) and Eq. (10) and it is compared with its relative deadline. Here $M_{1,8}$ is found unschedulable.

Third step: explaining why this allocation is not schedulable. The unschedulability of τ_5 is due to the interference of higher priority tasks on the same processor: $hp_5 = \{\tau_2, \tau_7, \tau_8, \tau_9, \tau_{17}\}$. By applying QUICKXPLAINTASK (see algorithm 1) with hp_5 ordered by increasing index, we find $S_5(A)' = \{\tau_5, \tau_9\}$ as minimal set. Consequently, the explanation of the unschedulability is translated into the new constraint:

$$CC_6 : \textit{NotAllEqual}\{x_5, x_9\}$$

In the same way, by applying QUICKXPLAINTASK:

- for τ_{12}: CC_7 : *NotAllEqual*$\{x_6, x_{12}, x_{13}\}$,
- for τ_{16}: CC_8 : *NotAllEqual*$\{x_{11}, x_{16}\}$,
- for τ_{19}: CC_9 : *NotAllEqual*$\{x_9, x_{19}\}$

For $M_{1,8}$, we have:

$$M_{1,8}(A) = \{M_{0,13}, M_{1,8}, M_{4,9}, M_{8,18}, M_{16,17}\}.$$

QUICKXPLAIN returns $\{M_{0,13}, M_{1,8}, M_{4,9}, M_{16,17}\}$ as $M_{1,8}(A)'$ the minimal set. An other constraint is created:

$$CC_{10} : w_{0,13} + w_{1,8} + w_{4,9} + w_{16,17} < 4$$

These new constraints CC_6, CC_7, CC_8, CC_9 and CC_{10} are added to the master problem. They define a new problem for which it has to search for a valid solution and so on.

After 20 iterations between the master problem and the subproblem, this allocation problem is proven without solution. This results from 78 constraints learnt all along the solving process. This example has been solved using ŒDIPE (see Section 5). On a computer with a G4 processor (800MHz), its computing time was 10.3 seconds.

5 Experimental results

We have developed a dedicated tool named ŒDIPE [6] that implements our solving approach (CPRTA). It is based on the CHOCO [13] constraint programming system and PALM [11], an explanation-based constraint programming system.

For the allocation problem, no specific benchmarks are available as a point of reference in the real-time community. Experiments are usually done on didactic examples [21, 1] or randomly generated configurations [18, 16]. We opted for this last solution. Our generator takes several parameters into account:

- n, m, mes: the number of tasks, processors (experiments have been done on fixed sizes: $n = 40$ and $m = 7$) and edges;
- $\%_{global}$: the global utilization factor of processors;
- $\%_{mem}$: the memory over-capacity, *i.e.* the amount of additionnal memory available on processors with respect to the memory needs of all tasks;

Mem.	$\%_{mem}$	Alloc.	$\%_{res}$	$\%_{co-res}$	$\%_{exc}$	Sched.	$\%_{global}$	Mes.	mes/n	$\%_{msize}$
1	60	1	0	0	0	1	40	1	0	0
2	30	2	15	15	15	2	60	2	0.5	70
3	10	3	33	33	33	3	90	3	0.875	150

Table 3. Details on difficulty classes

- $\%_{res}$: the percentage of tasks included in residence constraints;
- $\%_{co-res}$: the percentage of tasks included in co-residence constraints;
- $\%_{exc}$: the percentage of tasks included in exclusion constraints;
- $\%_{msize}$: the size of a data is evaluated as a percentage of the period of the tasks exchanging it.

Task periods and priorities are randomly generated. Worst-case execution times are initially randomly chosen and evaluated again so as: $\sum_{i=1}^{n} C_i/T_i = m\%_{global}$. The memory need of a task is proportional to its worst-case execution time. Memory capacities are randomly generated while satisfying: $\sum_{k=1}^{m} m_k = (1 + \%_{mem}) \sum_{i=1}^{n} \mu_i$. For a sake of simplicity, only linear data communications between tasks are considered and the priority of an edge is inherited from the task producing it.

The number of tasks involved in allocation constraints is given by the parameters $\%_{res}$, $\%_{co-res}$, $\%_{exc}$. Tasks are randomly chosen and their number (involved in co-residence and exclusion constraints) can be set through specific levels. Several classes of problems have been defined depending on the difficulty of both allocation and schedulability problems. The difficulty of schedulability is evaluated using the global utilization factor $\%_{global}$ which varies from 40 to 90 %. Allocation difficulty is based on the number of tasks included in residence, co-residence and exclusion constraints ($\%_{res}$, $\%_{co-res}$, $\%_{exc}$). Moreover, the memory over-capacity, $\%_{mem}$ has a significant impact (a very low capacity can lead to solve a $packing$ problem, sometimes very difficult). The presence of data exchanges impacts on both problems and the difficulty has been characterized by the ratios mes/n and $\%_{msize}$. $\%_{msize}$ expresses the impact of data exchanges on schedulability analysis by linking periods and message sizes.

Table 3 describes the parameters of each basic difficulty class. By combining them, categories of problems can be specified. For instance, a W-X-Y-Z category corresponds to problems with a memory difficulty in class W, an allocation difficulty in class X, a schedulability difficulty in class Y and a network difficulty in class Z.

5.1 Results

Table 4 summarizes some of the results of experiments with CPRTA. We do not give the results for all the intermediate classes of problems (like 1-1-1-1, 2-1-1-1, etc.) because they are easily solved and they don not exhibit a specific behaviour. $\%_{\text{RES}}$ gives the number of problem instances successfully solved (a schedulable solution has been found or it has been proven that none exists) within the time limit of 10 minutes per instance. $\%_{\text{VAL}}$ gives the percentage of schedulable solutions found (thus $\%_{\text{RES}} - \%_{\text{VAL}}$ gives the percentage of inconsistent problems). ITER is the number of iterations between the master problem and the subproblem. CPU is the mean computation time in seconds. NOG is the number of nogoods inferred from the subproblem. The data are obtained in average (on instances solved within the required time) on 100 instances (40 tasks, 7 processors) per class of difficulty with a Pentium 4 (3 GHz).

First, by examining the CPU column, we notice that CPRTA still remains very efficient in spite of its seeming complexity. Moreover as measured by ITER and NOG, the cooperation between master and sub-problems is quite significant and the learning is of some importance.

The lines 1 to 5 in Table 4 show results for high difficulty classes without communications between tasks. The results in lines 1 to 3 are very good. They illustrate the basic ability of constraint programming to consider memory and allocation constraints. Lines 4 and 5 display some performances that are going down when the schedulability difficulty increases. Indeed, the schedulability constraints set is empty at the beginning of the search. Therefore, all the knowledge dealing with schedulability has to be learnt from the subproblem. Furthermore, learning is only effective when a valid solution is produced by the master problem solver and as a consequence it is not really integrated into the constraint programming algorithm. To improve CPRTA performances from this point of view, a new approach is now being developed that integrates schedulability analysis into the constraint programming algorithm so as not "to delay" its taking into account —it is not a Benders decompostion, it is a new constraint defined from schedulability properties—.

The lines 6 to 8 deal with allocation problems where tasks may communicate. Once more, one can notice that when data exchanges increase (and thus message exchanges on the bus too), the CPRTA performances decrease. Reasons are the same as those of task schedulability: the more the messages are on the bus, the more their scheduling becomes difficult. Moreover, we have observed that nogoods inferred from message unschedulability are usually "weaker" (the search space cut is smaller) than the ones inferred from task unschedulability. Learning is then less efficient for this kind of problems. As for

tasks, we hope to improve CPRTA by integrating the network schedulability as a global constraint into the master problem.

	cat.	$\%_{RES}$	$\%_{VAL}$	ITER	CPU	NOG
1	2-2-2-1	100.0	56.0	13.5	1.6	95.2
2	3-2-2-1	98.0	57.0	31.0	10.4	133.2
3	2-3-2-1	99.0	19.0	6.6	1.4	43.5
4	1-1-3-1	74.0	74.0	95.7	115.7	471.6
5	2-2-3-1	67.0	12.0	8.3	33.2	59.7
6	2-2-2-2	98.0	69.0	21.1	7.5	69.9
7	1-2-2-3	66.0	43.0	188.3	70.5	110.7
8	2-2-2-3	47.0	30.0	137.7	66.8	117.2

Table 4. Average results on 100 instances randomly generated into classes of problems

5.2 Comparison with simulated annealing

As to get comparative performances for CPRTA, a simulated annealing (SA) algorithm, inspired from [21], has been implemented. In [21] the energy function takes into account residence, exclusion and memory constraints as well as task deadline constraints. To be consistent with the CPRTA model, the schedulability of messages on the CAN bus and co-residence constraints have been integrated too. The implementation has been optimized so as to reduce computation times of this energy function.

SA is a heuristic method. As a consequence, in our case, it can only conclude on problems with a solution. Therefore, in Table 5 only CPRTA results for such problems are compared to SA. As seen on Table 5, except for problems for which CPRTA must be improved (see Section 5.1), CPRTA produces as satisfactory results as SA does, but with better computation times. Introduction of schedulability as a constraint into the master problem should improve CPRTA, and certainly increases its efficiency in a significant manner. Moreover, it should be pointed out that even if CPRTA is sometimes less efficient than SA, CPRTA solves on average more problems than SA does if we take into account problems without solution.

	SA		CPRTA	
cat.	$\%_{VAL}$	CPU	$\%_{VAL}$	CPU
2-2-2-1	56.0	4.7	56.0	2.4
3-2-2-1	53.0	50.8	57.0	17.4
2-3-2-1	16.0	35.5	19.0	4.1
1-1-3-1	99.0	3.2	74.0	115.7
2-2-3-1	20.0	113.9	12.0	60.82
2-2-2-2	68.0	18.1	69.0	10.0
1-2-2-3	64.0	52.0	43.0	27.4
2-2-2-3	62.0	59.1	30.0	58.6

Table 5. Comparison between CPRTA and SA

6 Explanations

In comparison with other search methods, using a constraint solver may help "intrinsically" to answer some classical queries when a problem is proven without solution such as: why does my problem have no solution ? Usually, when the domain of a variable of a CSP becomes empty (no value exists that will respect all the constraints on that variable), basic constraint programming systems notify the user that there is no solution. Nevertheless, thanks to the versatility of the explanation-based constraint approach we use, those relevant constraints, which explain the failure, are made available in addition [11].

Thus in the case of an allocation problem for which no solution has been found, we analyse the set of constraints that is returned to explain the problem inconsistency. There can be many reasons to explain inconsistency. At the design level, we would like to be able to incriminate high level characteristics of the system such as : allocation constraints, schedulability requirements of tasks, processors or network limitation. However, two points of view, based on the software or hardware architecture, can be adopted. We will first focus on the characteristics of the software architecture by analysing how each task is "responsible" for the failure. We will give there some insight on the way a critical task from the schedulability point of view can be identified. Each failure of the search process due to schedulability is analysed and transformed into a constraint criterion that encapsulates an accurate reason for this failure. The study of those criteria may lead to the guilty tasks. The rationale of this evaluation is based on the following remarks:

- The more a task appears within a nogood, the more this task has an impact on the schedulability inconsistency.
- The level of propagation performed by a nogood (either *NotAllEqual*(x_i) or $\sum w_{ij} < B$), i.e its impact within the proof is strongly related to its size (the number of tasks it involves). "Small" *NotAllEqual* have stronger impact.

In its general form, a constraint (learnt from a nogood) is defined by *NotAllEqual*(x_i) or $\sum w_{ij} < B$ (see Section 3.4). We denote NAE the set of constraints in the *NotAllEqual* form and SUM the set of constraints in the second form. For a task τ_i a constraint criterion $\mathcal{C}_i$ is evaluated:

$$\mathcal{C}_i = \sum_{\substack{c \in \mathsf{NAE} \\ x_i \in c}} \frac{1}{\#c} + \sum_{\substack{c \in \mathsf{SUM} \\ \exists j, w_{ij} \in c \vee w_{ji} \in c}} \frac{1}{\#c}$$

This criterion considers the presence of a task in each constraint and its impact. Bigger $\mathcal{C}_i$ is, bigger the impact of τ_i is on the inconsistency. By studying tasks with high $\mathcal{C}_i$ and understanding why they have such an impact on the

inconsistency (*e.g.* low priority allocation, too large processor utilization), it is possible to change some requirements (*e.g.* by adapting priorities, or choosing a different version for a task with an other period) and so to obtain a solution for the problem.

Table 6 gives $\mathcal{C}_i$ obtained on the example of the Section 4 with ŒDIPE [6]. Task τ_{19} has the biggest $\mathcal{C}_i$. This task has a low priority together with a high processor utilization ($C_{19}/T_{19} = 0.32$). By just changing its priority to the highest one, and reusing CPRTA, we found a solution for this problem.

Notice that this process consists in analysing the final set of constraints with a heuristic based on the information gathered during the search. This process can be generalized to memory and allocation constraints by the use of a specific search technique [19] even if explicit reasons for failure on memory or allocation are not kept in memory in our current approach (contrary to schedulability one).

τ_i	$\mathcal{C}_i$	τ_i	$\mathcal{C}_i$	τ_i	$\mathcal{C}_i$	τ_i	$\mathcal{C}_i$
τ_{19}	6.33	τ_{13}	4.78	τ_2	3.22	τ_3	2.53
τ_{14}	5.98	τ_9	3.95	τ_1	2.85	τ_{16}	2.25
τ_{11}	5.98	τ_6	3.83	τ_{10}	2.77	τ_{18}	1.97
τ_5	5.42	τ_7	3.45	τ_4	2.65	τ_8	1.73
τ_{12}	5.42	τ_{15}	3.32	τ_{17}	2.55	τ_0	1.15

Table 6. Constraint criterions computed on example

7 Conclusion and future work

In this paper, we present an original and complete approach (CPRTA) to solve a hard real-time allocation problem. We use a decomposition method which is built on a logic Benders decomposition scheme. The whole problem is split into a master problem handling allocation and resource constraints and a subproblem for timing constraints. A rich interaction between master and subproblems is performed with the computation of minimal sets of unschedulable tasks and messages. It implements a kind of learning technique in an effort to combine the various issues into a solution that satisfies all constraints.

One important specificity of CPRTA is its completeness, *i.e.*, if a problem has no solution, the search algorithm is able to prove it. Moreover it offers good potential means for building an analysis able to give an aid to the user in case of failure.

The results produced by our experiments encourage us to go a step further. Further works concern the inclusion of (task and message) schedulability analysis into the search process of the CP algorithms in the form of a global constraint. This should improve efficiency of CPRTA for hard-schedulability-constrained problems. Another interesting work deals with the explanation of failure. Our aim is to integrate into the design process an intelligent tool based on CPRTA ables to return pertinent explanations justifying the failure. We need to go deeper in that way and to try it out on some concrete cases.

References

[1] P. Altenbernd and H. Hansson. The slack method: A new method for static allocation of hard real-time tasks. *Real-Time Systems*, 15(2):103–130, 1998.

[2] R. Barták. Constraint programming: In pursuit of the holy grail. In *Proc. of the Week of Doctoral Students (WDS99)*, 1999.

[3] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[4] T. Benoist, E. Gaudin, and B. Rottembourg. Constraint programming contribution to benders decomposition: a case study. *Lecture notes in Computer Science*, 2470:603–617, 2002.

[5] Bosch. *CAN Specification version 2.0*, 1991.

[6] H. Cambazard and P. Hladik. ŒDIPE. http://oedipe.rts-software.org/.

[7] C. Ekelin. *An Optimization Framework for Scheduling of Embedded Real-Time Systems*. PhD thesis, Chalmers University of Technology, 2004.

[8] P.-E. Hladik and A.-M. Déplanche. Extension au réseau can des problèmes de placement. Technical Report 4, IRCCyN, 2005.

[9] J. N. Hooker and G. Ottoson. Logic-based benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[10] U. Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *Proc. of IJCAI 01*, 2001.

[11] N. Jussien. The versatility of using explanations within constraint programming. Technical Report RR 03-04-INFO, École des Mines de Nantes, 2003.

[12] N. Jussien, R. Debruyne, and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *CP 2000*, number 1894 in Lecture Notes in Computer Science, pages 249–261, Singapore, Sept. 2000. Springer-Verlag.

[13] CHOCO. http://choco.sourceforge.net/.

[14] E. Lawler. Recent results in the theory of machine scheduling. *Mathematical Programming: The State of the Art*, 1983.

[15] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS 1990)*, pages 201–209, 1990.

[16] Y. Monnier, J.-P. Beauvais, and A.-M. Déplanche. A genetic algorithm for scheduling tasks in a real-time distributed system. In *ECRTS'98*, 1998.

[17] OSEK Group. *OSEK/VDX Communication version 3.0.2*.

[18] K. Ramamritham. Allocation and scheduling of complex periodic tasks. In *Proc. of ICDCS 1990*, 1990.

[19] P. Refalo. Impact-based search strategies for constraint programming. In *Proc. of CP 2004*, 2004.

[20] R. Szymanek, F. Gruian, and K. Kuchcinski. Digital systems design using constraint logic programming. In *Proc. of PACLP 2000*, 2000.

[21] K. W. Tindell, A. Burns, and A. Wellings. Allocating hard real-time tasks: An np-hard problem made easy. *Real-Time Systems*, 4(2):145–165, 1992.

[22] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysis real-time communications: controller area network (can). In *Proc. of RTSS 1994*, pages 259–265, 1994.

Schedulability Analysis of Serial Transactions

Karim TRAORE
{karim.traore@ensma.fr}

GROLLEAU Emmanuel
{grolleau@ensma.fr}

COTTET Francis
{cottet@ensma.fr}

LISI/ENSMA
Laboratoire d'Informatique Scientifique et Industrielle
École Nationale de Mécanique et d'Aérotechnique
Téléport 2 – BP 40109 F-86961 Chasseneuil Futuroscope Cedex, France

Abstract[1]

On the basis of a concrete real-time application, we present in this article a new task model called "serial transaction". This model is a particular instance of the task model with offsets defined by Tindell and Palencia and al.. A serial transaction is typically a task reading serial information (RS232, CAN,…): several instances are identical and read a unitary part of a serial packet, these tasks have the same WCET, offset shifting, priority and relative deadline. In addition, the last task of a transaction has to deal with the packet, and is typically longer, but has a longer relative deadline, and a lower priority. The need for this task model appeared in a real application, that couldn't be validated using known methods on transactions, so we present a less pessimistic real-time evaluation method dedicated on to this new model.

1. Introduction

Several laboratories of Poitiers (ENSMA and University) are developing together a mini UAV (Unmanned Air Vehicle) (see Figure 1). The LISI is in charge of developing and validating the system (embedded and ground station). The embedded processing unit is a microcontroller (Freescale/Motorola MPC555) connected via serial port to a GPS receiver and a modem used in order to communicate with the ground station. The measurement of the attitude of the UAV is done by an IMU (Inertial Measurement Unit) connected to the microcontroller via a CAN network.

In the development of a real-time application like this one, two techniques of scheduling can be used : the on-line scheduling, with a fixed [LL73, LW82, Aud91] or variable allocation of priorities of the tasks in the tasks set [Der74, Lab74, DM89] and off-line techniques which use a sequence whose correctness was proved [XP92, Gro99]. The real-time RTOS (Real-Time Operational System) OSEKTurbo OS/MPC5xx [OSM1, OSM2], in conformity with standard OSEK/VDX [Osek1, Osek2], selected for this application, allows only fixed priorities. We thus used an on-line approach with fixed priority technique.

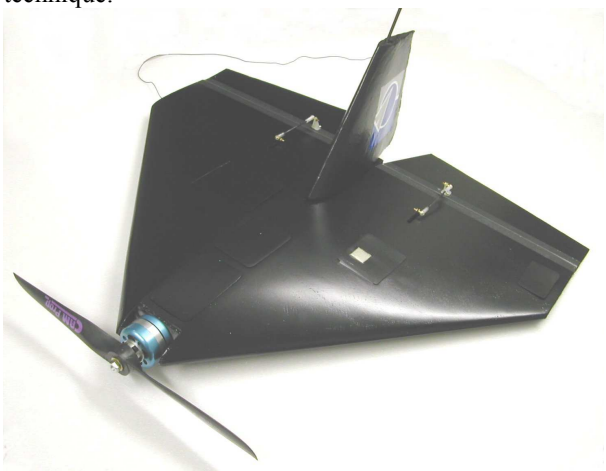


Figure 1: the AMADO

After the definition of the software architecture and the temporal parameters of the various tasks, one of the most important phases is the temporal validation which consists in proving that whatever happens, all the tasks meet their temporal constraints. RTA (Response Time analysis) methods are used to bound the worst case response time of the tasks of an application. Tindell [Tin94] proposed a method for calculating an upper bound of the worst-case response time which is less pessimistic than classic RTA (considering that a critical instant consists in a simultaneous release of all the tasks) in a context of tasks with offsets.

Palencia and Harbour [PG98] extended Tindell's work with dynamic offsets, and formalized his work as transactions. Lastly, [TN04b][MS03] introduced the concept of "imposed" interference differing from "released for execution" interference used by Tindell. However, for now the exact calculation methods used to determinate the exact worst-case response time rely on calculating every combination of the tasks of the transactions; it thus remains exponential in time.

[1] This work was supported by ONERA/DGA

In order to validate the control system of the UAV, we had to deal with tasks with offset which are particular instances of transactions: these tasks are activated by peripherals connected on serial and CAN ports. Section 2 presents the case study. Section 3 recalls some general results about transactions. Section 4 presents some new results obtained, allowing us to analyse the interference of a serial transaction in a pseudo polynomial time for a subset of the tasks of the task system. Section 5 applies these new results in order to validate our case study.

2. Presentation of the Application

The project, named AMADO, is a UAV with a wingspread of 55 cm, using a delta shaped wing with two symmetrical drifts for a total weight (including the control system) of 930 grams. The main objective is to create an autonomous plane embedding a camera, and to be able to follow dynamically defined waypoints. The UAV is connected to a ground station thanks to a wireless modem, allowing it to receive high level orders during a mission. The critical parts of the flight control are embedded.

2.1 Hardware architecture

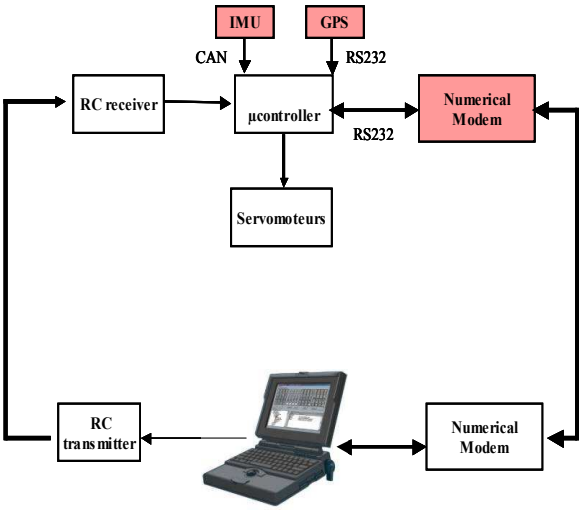


Figure 2: main architecture of the AMADO

The Figure 2 shows two parts: the ground station, and the embedded station. The ground station can communicate thanks to a half duplex modem with the embedded system, and the traditional radio emitter is kept as an emergency control in case of general failure of the embedded system. The main role of the ground station is video displaying/recording, flight instruments, and high level commands (either waypoints flight, or assisted flight).

The embedded system heart is a Freescale/Motorola MPC555 [MPC1] connected to the actuators (3 servo-commands and the speed-variator, refreshed every 20 ms), an IMU [IMU1], a GPS receiver [GPS1], a traditional radio receiver and a modem. The MPC555 is a 32 bits PowerPC with a frequency of 40MHZ, 448KB of flash memory and 26KB of RAM.

Two sensors are used in order to calculate the position and attitude of the UAV: the GPS receiver and the IMU. The Inertial Measurement Unit sends information about angular speed and accelerations, which, once treated, give the roll and the pitch of the UAV. This IMU is connected on a CAN port and delivers information at a frequency of 50Hz and a throughput of 1Mbps. A frame of the IMU is compound of 3 blocks of 6 bytes. In order for the system to get a complete frame, since there is no possible memorisation of the blocks, each block must be read before the next arrives. Once the system has 3 blocks, it can constitute the frame, and handle it to calculate the roll and the pitch.

The GPS receiver is used to get the speed (direction and module) and the absolute 3Dimensional position of the UAV. The GPS Receiver sends data to the controller at a frequency of 4Hz and delivers information with a throughput of 57600bps. As a RS232 communication, the information is sent byte after byte; the number of bytes sent during one period (frame) of the GPS can reach 120 bytes. As in the case of the IMU, the system must recover each byte and arrange it before the arrival of the next byte, under penalty of losing the complete frame.

Finally the modem connected to the microcontroller on the serial port is bi-directional and communicates with the microcontroller at a throughput of 115kbps. The length of the frame transmitted to the microcontroller by the modem can reach 10 bytes. The requirements are the same as in the case of the GPS receiver. In the presentation of this architecture, we omitted voluntarily the video circuit that does not have any impact on the real-time aspects of this application.

2.2 Software architecture of the application

We have chosen the real-time executive OSEKTurbo OS/MPC5xx of Metrowerks for our application. This RTOS is conforming to the standard OSEK/VDX; standard defined for applications with limited resources [OSM3]. The OSEK/VDX executives are light because they are based on a static description of all the system using the OIL (OSEK Implementation Language).

Apart the initialisation task, there are 12 tasks in the control system (see Table 1). The priorities of the tasks have been assigned following a Deadline Monotonic policy [LL73]. Note that the value L=120 (resp. L=3, L=10) corresponds to the number of times the task has to be activated in order to acquire a frame.

This kind of application can't be validated easily if the offsets are not taken into account. Indeed, it appears clearly that task TreatGPS is released when the whole GPS frame has been received; it cannot thus be released at the same time as the task Acq GPS; it is the same case for task TreatIMU and the task Acq IMU; the same situation occurs for the task TreatInstruction and the task Acq Instruction.

Tasks	Period	WCET		deadline	Priority
	(in microsecond)				
Monitoring (1)	200000	60		200000	1
Acq PWM (2)	20000	24		10000	7
Transmit Grd (3)	50000	3360		30000	5
Deliver Cmd (4)	20000	40		10000	6
Navigation (5)	250000	560		140000	2
ReguleAttitude (6)	60000	32400		60000	4
Acq GPS (7)	250000	100	L=120	160	11
Acq IMU (8)	20000	96	L=3	720	10
Acq Instruction(9)	100000	12	L=10	80	12
TreatGPS (10)	250000	3000		5000	9
TreatIMU (11)	20000	900		7500	8
TreatInstruction (12)	100000	900		70000	3

Table1: task system of the UAV

The Figure 3 presents a model of a serial transaction, L_i instances of the acquisition of a part of a frame are separated by a duration corresponding to the arrival rate of the packets (Acq GPS, Acq IMU, Acq Instruction), and a longer task is used to handle the whole frame (TreatGPS, TreatIMU, TreatInstruction). In a serial transaction, the acquisition tasks are usually short, because they only have to bufferize the packets until the whole frame is built, while the treatment tasks are longer since they have to deal with the full frame. Moreover, the first release of the serial transaction is not known precisely because serial transaction is activated by an external peripheral.

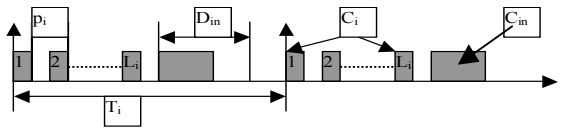


Figure 3 : pattern of serial transaction

In order to define a serial transaction as a particular case of a transaction, let us first give a survey of definitions and results found in [Tin94][TN04a][PG98].

3. Transactions

The model of tasks with offsets was proposed by Tindell in order to reduce existing pessimism of the schedulability analysis when the critical instant for a task occurs when it is released at the same time as all the tasks of higher priority. Indeed, certain tasks can for example have the same period and be bound by relations of offsets i.e. they can never be released at the same time. A set of tasks of the same period bounded by offset is called a transaction. A task system is compound of a set of transactions [PG98][TN04a]:

$\Gamma := \{\Gamma_1, \Gamma_2, \ldots \Gamma_k\}$

A transaction (see Figure 4) Γ_i contains $|\Gamma_i|$ tasks having the same period T_i : $\Gamma_i := <\{\tau_{i1},\ldots, \tau_{i|\Gamma i|}\}, T_i>$.

A task is defined by $\tau_{ij} := <C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij}>$ where C_{ij} is the worst-case execution time (WCET), O_{ij} is the offset (minimal time between the release of the transaction and the release of the task), in order to simplify the analysis, we will consider a reduced task offset Φ_{ij} which is always within 0 and T_i: $\Phi_{ij} = O_{ij}\%T_i$. D_{ij} is the relative deadline, J_{ij} the maximum jitter (giving t_0 the release date of an instance of the transaction Γ_i, then the task τ_{ij} is released between t_0+O_{ij} and $t_0+O_{ij}+J_{ij}$), B_{ij} maximum blocking due to lower priority tasks, and P_{ij} the priority. Without loss of generality, we consider that the tasks are ordered by increasing offsets Φ_{ij}; in our case, we define the response time as being the time between the release of the task and the completion of the task. In the table 3, we have represented all the transactions of the UAV application.

Let us note also $hp_i(\tau_{ua})$ the set of indices of the tasks of Γ_i with a priority higher than the priority of a task τ_{ua} i.e. $j \in hp_i(\tau_{ua})$ if and only if $P_{ij} > P_{ua}$.

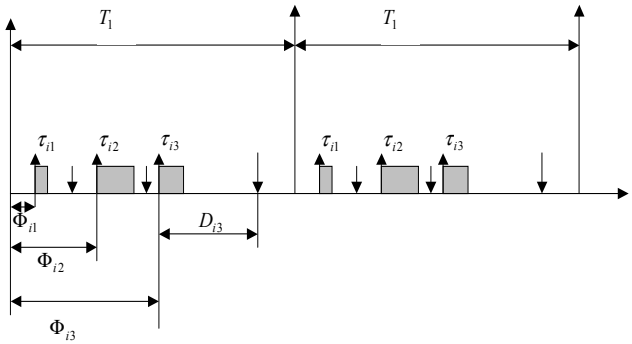


Figure 4: model of tasks with offsets

The RTA method is to be applied on each task of the transactions. The task under analysis is usually noted τ_{ua}. Tindell showed that the critical instant of τ_{ua} is a particular instant when it is released at the same time as one task of higher priority in each transaction (its own transaction being handled separately). The main difficulty is to determine what is the critical instant candidate τ_{ic} of a transaction Γ_i that initiates the critical instant of τ_{ua}. An exact calculation method would require to evaluate the response time obtained by

Transactions	Period	tasks	WCET	Offset	deadline	Priority	"Release for execution" worst-case response time
1	200000	11	200000	0	200000	1	56156
2	20000	21	10000	0	10000	7	**11332**
3	50000	31	30000	0	30000	5	23784
4	20000	41	10000	0	10000	6	**11672**
5	250000	51	140000	0	140000	2	56096
6	60000	61	60000	0	60000	4	54636
7	250000	7i(i=1,…,120)	100	$O_{7i} = (i-1)*160$	160	11	124
		7121	3000	120*160	5000	9	3408
8	20000	81 - 82 - 83	96	0 – 720 - 2*720	720	10	468
		84	900	3*720	7500	8	**10720**
9	100000	9i(i=1..10)	12	$O_{9i} = (i-1)*80$	80	12	12
		911	900	10*80	70000	3	55416

Table 2 : representation of all the tasks of the configuration using the symbolism of transaction and values of worst-case response time with "release for execution" method

carrying out all the possible combinations of the tasks of priority higher than τ_{ua} in each transaction and to choose the task τ_{ic} in each transaction that leads to the worst response time. This exhaustive method has an exponential complexity and is intractable for realistic task systems; several approximation methods giving an upper bound of the worst-case response time have been proposed.

Upper bound method based on the interference "released for execution"

[Tin94][PG98] Let us note τ_{ic} the task of Γ_i that coincides with the critical instant of τ_{ua}. Let us note $W_{ic}(\tau_{ua}, t)$ the interference of Γ_i on the response time of τ_{ua} during a time interval of length t.

$$W_{ic}(\tau_{ua}, t) = \sum_{j \in hp_i(\tau_{ua})} \left(\left\lceil \frac{t^*}{T_i} \right\rceil \cdot C_{ij} \right)$$

$$t^* = t - phase(\tau_{ij}, \tau_{ic})$$

$$phase(\tau_{ij}, \tau_{ic}) = (O_{ij} - O_{ic}) \bmod T_i$$

t^* represents the time during which τ_{ij} can interfere with τ_{ua}.

Let us note $A(\tau_{ua}, \Gamma_i, t) = \max_{c \in \Gamma_i} W_{ic}(\tau_{ua}, t)$

The upper bound of the response time is obtained by iteration : $R_{ua}^0 = C_{ua}$

$$R_{ua}^{(n+1)} = C_{ua} + \sum_{\Gamma_i \in \Gamma} A(\tau_k, \Gamma_i, R_{ua}^n).$$

The value of R_{ua} is thus obtained by a classic fix-point iteration lookup.

The interference that a transaction imposes on a task can be represented by a periodic and static pattern. [TN04a] proposed an optimisation of the computation of the interference. This technique consists in storing in a table the parameters of the interference function of a transaction on a task of lower priority. This approach reduces the computation time but this method does not reduce the difference between the real worst-case response time and the upper bound obtained. Therefore, we couldn't validate our system with the general method because the tasks (2), (4) and (11) have a worst-case response time greater than their relative deadline; while the real worst-case response time of all the tasks of the set could in fact be lower than their deadline. (see Table2).

We thus present a method given in [TN04b] giving a tighter upper bound.

Upper bound method based on the "imposed" interference

This method has been proposed in [TN04b]. It removes the unnecessary overestimation taken into account in the computation of the interference created by a task on a lower priority one. This overestimation does not have any impact in the case of tasks without offset but has a considerable effect in the approximation of the worst-case response time when we are in the presence of tasks with offsets. This method consists in calculating the interference effectively imposed by a task τ_j on a task τ_{ua} with a lower priority during a time interval of length t; the idea is that the interference cannot exceed the interval of time t.

$$\frac{dInterférence_j(t)}{dt} \leq \frac{dt}{dt}$$

In order to calculate this "imposed" interference, [TN04b] subtracts a parameter x (see Figure 5) from the original interference formula:

$$W_{ic}(\tau_{ua}, t) = \sum_{j \in hp_i(\tau_{ua})} \left(\left(\left\lfloor \frac{t^*}{T_i} \right\rfloor + 1 \right) * C_{ij} - x_{icj}(t) \right)$$

$$t^* = t - phase(\tau_{ij}, \tau_{ic})$$

$$phase(\tau_{ij}, \tau_{ic}) = (O_{ij} - O_{ic}) \bmod T_i$$

$$x_{icj}(t) = \begin{cases} 0 \\ \max\left(0, C_{ij} - (t^* \bmod T_i)\right) \end{cases} \quad \text{for } t^* \geq 0$$

$x_{icj}(t)$ corresponds to the part of the task τ_{ij} that cannot be executed in the time interval of length t; since this interference is not effectively imposed in this interval, it is not taken into account.

Example: this transaction has 4 tasks with period $T_i = 50$

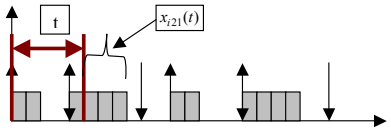


Figure 5: "imposed" interference

$$\Gamma_i := < \{\tau_{i1}, \tau_{i2}, \tau_{i3}, \tau_{i4}\}, 50 >$$

$$\tau_{i1} := < 2, 0, 4, 0, 0, 4 >$$

$$\tau_{i2} := < 4, 4, 8, 0, 0, 2 >$$

$$\tau_{i3} := < 2, 11, 5, 0, 0, 3 >$$

$$\tau_{i4} := < 4, 16, 15, 0, 0, 1 >$$

$$W_{i1}(\tau_{ua}, 5) = (2-0) + (4-3) + (0-0) + (0-0) = 3$$

For determining the upper bound of the response-time, we use this function :

$$W_i(\tau_{ua}, t) = \max_{c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, t)$$

With the value of each $W_i(\tau_{ua}, t)$, the response time R_{ua} of τ_{ua} can be calculated. $R_{ua}^{(n+1)} = C_{ua} + \sum_{\Gamma_i \in \Gamma} W_i(\tau_{ua}, R_{ua}^n)$. R_{ua} is obtained by fix-point iteration starting with $R_{ua}^0 = C_{ua}$. Let us execute this method on the example (Figure6)

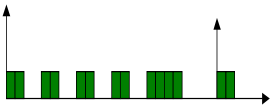


Figure 6. Example for imposed interference

In the transaction Γ_i, we have five tasks. Let us consider a lower priority task τ_{ua} with $C_{ua} = 5$. Let us calculate the response-time. We present at first the details of iteration number 2:

Iteration 2:

$$W_{i1}(\tau_{ua}, 5) = (2-0) + (2-1) + (0-0) + (0-0) + (0-0) = 3$$

$$W_{i2}(\tau_{ua}, 5) = (0-0) + (2-0) + (2-1) + (0-0) + (0-0) = 3$$

$$W_{i3}(\tau_{ua}, 5) = (0-0) + (0-0) + (2-0) + (2-1) + (0-0) = 3$$

$$W_{i4}(\tau_{ua}, 5) = (0-0) + (0-0) + (0-0) + (2-0) + (4-3) = 3$$

$$W_{i5}(\tau_{ua}, 5) = (0-0) + (0-0) + (0-0) + (0-0) + (4-0) = 4$$

$$W_i(\tau_{ua}, 0) = 4 \qquad R_{ua} = 9$$

We give the values obtained in the different iterations :

Iteration	t	W_{i1}	W_{i2}	W_{i3}	W_{i4}	W_{i5}	W_i	R_{ua}
1	0	0	0	0	0	0	0	5
2	5	3	3	3	3	4	4	9
3	9	5	5	5	6	5	6	11
4	11	6	6	7	6	6	7	12
5	12	6	6	8	6	6	8	13
6	13	7	7	8	7	7	8	13

Consequently, the value of R_{ua} is equal to 13.

4- Contribution to RTA of transactions

4.1 Transactions without jitters

In this section, we first simplify the way to compute the interference [PG 98] for general transactions with no jitter.

according to [PG98] the interference of a transaction for a task τ_{ic} candidate to coincide with the critical instant is given by:

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t) \right) \quad \text{with}$$

$$I_{ijc}^{Set1} = \left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor C_{ij}, \quad I_{ijc}^{Set2} = \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil C_{ij}, \text{ and}$$

$$\Phi_{ijc} = (T_i + O_{ij} - (O_{ic} + J_{ic}))\% T_i$$

By assumption, the jitter is null, so the interference is written :

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(\left\lfloor \frac{(T_i + O_{ij} - O_{ic})\% T_i}{T_i} \right\rfloor C_{ij} + \left\lceil \frac{t - (T_i + O_{ij} - O_{ic})\% T_i}{T_i} \right\rceil C_{ij} \right)$$

By definition, $(T_i + O_{ij} - O_{ic})\% T_i < T_i$ therefore

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left\lceil \frac{t - (T_i + O_{ij} - O_{ic})\% T_i}{T_i} \right\rceil C_{ij}$$

Which is equivalent to

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left\lceil \frac{t - (T_i + \Phi_{ij} - \Phi_{ic})\% T_i}{T_i} \right\rceil C_{ij}$$

Let us note $k_1, k_2, \ldots, k_{|hpi(\tau ua)|}$ the indices ordered by offset of $hp_i(\tau_{ua})$ (i.e. $p<q \Rightarrow \Phi_{ikp} \leq \Phi_{ikq}$). Since the offsets are assumed to be lower than the period, $(T_i+\Phi_{ij}-\Phi_{ic})\%T_i$ correspond to $\Phi_{ij}-\Phi_{ic}$ if $\Phi_{ic} \leq \Phi_{ij}$ and $(T_i+\Phi_{ij}-\Phi_{ic})$ if $\Phi_{ij} < \Phi_{ic}$. Hence, separating the formula between tasks

released before and after the critical instant candidate τ_{ikp}, we have :

$$W_{ik_p}(\tau_{ua},t)=\sum_{\substack{k_j\in hp_i(\tau_{ua})\\ k_j<k_p}}\left\lceil\frac{t-(T_i+\Phi_{ik_j}-\Phi_{ik_p})}{T_i}\right\rceil C_{ik_j}+\sum_{\substack{k_j\in hp_i(\tau_{ua})\\ k_j\geq k_p}}\left\lceil\frac{t-(\Phi_{ik_j}-\Phi_{ik_p})}{T_i}\right\rceil C_{ik_j}$$

so $W_{ik_1}(\tau_{ua},t)=\sum_{k_j\in hp_i(\tau_{ua})}\left\lceil\frac{t-(\Phi_{ik_j}-\Phi_{ik_1})}{T_i}\right\rceil C_{ik_j}$

$$W_{ik_2}(\tau_{ua},t)=\left\lceil\frac{t-(T_i+\Phi_{ik_1}-\Phi_{ik_2})}{T_i}\right\rceil C_{ik_1}+\sum_{\substack{k_j\in hp_i(\tau_{ua})\\ k_j\geq k_2}}\left\lceil\frac{t-(\Phi_{ik_j}-\Phi_{ik_2})}{T_i}\right\rceil C_{ik_j}$$

And so on. Therefore

$$W_{ik_1}(\tau_{ua},t)-W_{ik_2}(\tau_{ua},t)=\left(\left\lceil\frac{t-(\Phi_{ik_1}-\Phi_{ik_1})}{T_i}\right\rceil-\left\lceil\frac{t-(T_i+\Phi_{ik_1}-\Phi_{ik_2})}{T_i}\right\rceil\right)C_{ik_1}+\sum_{\substack{k_j\in hp_i(\tau_{ua})\\ k_j\geq k_2}}\left(\left\lceil\frac{t-(\Phi_{ik_j}-\Phi_{ik_1})}{T_i}\right\rceil-\left\lceil\frac{t-(\Phi_{ik_j}-\Phi_{ik_2})}{T_i}\right\rceil\right)C_{ik_j}$$

Let us analyze now, how we can determine efficiently the differences between the interference function when comparing the first task as the critical instant candidate comparing to another task :

$\left(\left\lceil\frac{t}{T_i}\right\rceil-\left\lceil\frac{t-(T_i+\Phi_{ik_1}-\Phi_{ik_2})}{T_i}\right\rceil\right)C_{ik_1}$ is always equal to 0 or C_{ik1} because $\Phi_{ij}<T_i$.

The difference is C_{ik1} if and only if :

$t\%T_i>0 \; and \; t\%T_i-(T_i+\Phi_{ik_1}-\Phi_{ik_2})\leq 0$,

equivalently $t\%T_i\in]0..T_i+\Phi_{ik_1}-\Phi_{ik_2}]$

For the other tasks interference (i.e. other part of the sum) :

$\left(\left\lceil\frac{t-(\Phi_{ik_j}-\Phi_{ik_1})}{T_i}\right\rceil-\left\lceil\frac{t-(\Phi_{ik_j}-\Phi_{ik_2})}{T_i}\right\rceil\right)C_{ik_j}$ is always equal to 0 or $-C_{ikj}$ because $\Phi_{ij}<T_i$.

The difference is equal to $-C_{ikj}$ if and only if :

$t\%T_i-(\Phi_{ik_j}-\Phi_{ik_1})\leq 0 \; and \; t\%T_i-(\Phi_{ik_j}-\Phi_{ik_2})>0$

equivalently if :

$t\%T_i\in]\Phi_{ik_j}-\Phi_{ik_2}..\Phi_{ik_j}-\Phi_{ik_1}]$

We can thus calculate $W_{ik_1}(\tau_{ua},t)-W_{ik_2}(\tau_{ua},t)$ testing $|hp_i(\tau_{ua})|$ intervals.

We will now calculate the difference $\forall k_p\in hp_i(\tau_{ua}), k_p\neq k_1, W_{ik_1}(\tau_{ua},t)-W_{ik_p}(\tau_{ua},t)$:

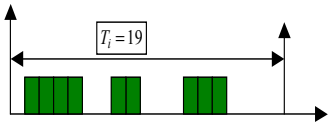
$$W_{ik_1}(\tau_{ua},t)-W_{ik_p}(\tau_{ua},t)=\sum_{\substack{k_j\in hp_i(\tau_{ua})\\ k_j<k_p}}\left(\left\lceil\frac{t-(\Phi_{ik_j}-\Phi_{ik_1})}{T_i}\right\rceil-\left\lceil\frac{t-(T_i+\Phi_{ik_j}-\Phi_{ik_p})}{T_i}\right\rceil\right)C_{ik_j}+\sum_{\substack{k_j\in hp_i(\tau_{ua})\\ k_j\geq k_p}}\left(\left\lceil\frac{t-(\Phi_{ik_j}-\Phi_{ik_1})}{T_i}\right\rceil-\left\lceil\frac{t-(\Phi_{ik_j}-\Phi_{ik_p})}{T_i}\right\rceil\right)C_{ik_j}$$

The first sum has a value ≥ 0 whereas the second has a value ≤ 0. We have :

Difference of $+C_{ikj}$ for $k_j<k_p$ if $t\%T_i\in]\Phi_{ik_j}-\Phi_{ik_1}..T_i+\Phi_{ik_j}-\Phi_{ik_p}]$ (1)

Difference of $-C_{ikj}$ for $k_j\geq k_p$ if $t\%T_i\in]\Phi_{ik_j}-\Phi_{ik_p}..\Phi_{ik_j}-\Phi_{ik_1}]$ (2)

Example : transaction of period=19 with 3 tasks (Fig. 7)



$\tau_{i1}=<4,1,6,0,0,1>$
$\tau_{i2}=<2,7,5,0,0,2>$
$\tau_{i3}=<3,12,6,0,0,3>$

Figure 7.calculation with intervals

	$W_{i1}-W_{i2}$			$W_{i1}-W_{i3}$		
	$]0;13]$	$]0;6]$	$]6;11]$	$]0;8]$	$]6;14]$	$]0;11]$
If $t\in I$	C_{i1}	$-C_{i2}$	$-C_{i3}$	C_{i1}	C_{i2}	$-C_{i3}$
If $t\notin I$	0	0	0	0	0	0

Evaluation of $W_i(\tau_{ua},t)$ with t=14

$W_{i1}(\tau_{ua},14)-W_{i2}(\tau_{ua},14)=0+0+0=0$

$W_{i1}(\tau_{ua},14)-W_{i3}(\tau_{ua},14)=0+C_{i2}+0=2$

Thus $W_i(\tau_{ua},14)=W_{i1}(\tau_{ua},14)=9$

With this method, it is sufficient to evaluate only one value of $W_{ic}(\tau_{ua},t)$

4.2 Serial transaction

Let us introduce the definition of a serial transaction:

Definition1: A serial transaction is a transaction with the following constraints:

Let Γ_I be a serial transaction,

- null jitter: $\forall i/\tau_{ij}\in\Gamma_i, J_{ij}=0$
- regular arrival pattern p_i: $\forall j\in[1..|\Gamma_i|], \Phi_{ij}=(j-1)p_i$.
- there are two kinds of tasks :
  - the $L_i=|\Gamma_i|-1$ acquisition tasks such that : $\tau_{ij,j\in[1..Li]}:=<C_i,(j-1)p_i,p_i,0,B_{ij},P_i>$;
  - the treatment task $\tau_{i|\Gamma i|}:=<C_{in},L_ip_i,D_{in},0,B_{ij},P_{in}>$

- with $C_{in}>C_i$, $D_{in}>p_i$, $P_{in}<P_i$ and $T_i-(L_i\cdot p_i+C_{in})>p_i-C_{in}$. This means that the treatment task is longer than the acquisition tasks, but is provided a longer deadline and a lower priority.

Example of serial transaction : (Figure 6)

Definition2 : a task τ_{ua} is an intermediate priority task for a serial transaction Γ_i if the priority of τ_{ua} is lower than acquisition tasks of Γ_i but higher than the treatment task of Γ_i.

Definition3 : a task τ_{ua} is a lower priority task for a serial transaction Γ_i if the priority of τ_{ua} is lower than all the tasks of Γ_i.

The next result relies on the intervals defined in section 4.1, let us define E_{ikj} as the shift between two successive tasks of higher priority than the task under analysis (Figure 8). Let $k_1,k_2,.....,k_{|hp_i(\tau_{ua})|}$ the elements of $hp_i(\tau_{ua})$. We assume that $hp_i(\tau_{ua})$ is ordered by offsets values increasing i.e $\Phi_{ik_j}\le\Phi_{ik_{(j+1)}}$ for $j<|hp_i(\tau_{ua})|$

$E_{ij}=\Phi_{ik_{(j+1)}}-\Phi_{ik_j}$ for $j<|hp_i(\tau_{ua})|$ and

$E_{i|hp_i(\tau_{ua})|}=T_i+\Phi_{ik_1}-\Phi_{ik_{|hp_i(\tau_{ua})|}}$

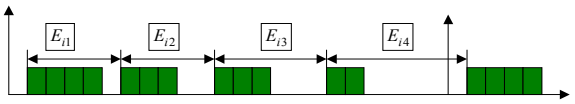


Figure 8. Illustration of E_{ij} and theorem 1

Theorem 1 shows that for specific patterns of transactions without jitters where the WCET of tasks are decreasing and the shifts between successive offsets are increasing, the critical instant of a task always coincides to the first instance of the transaction. The acquisition tasks of a serial transaction follow this kind of pattern, therefore the critical instant of a task of an intermediate priority (lower than acquisition tasks but higher than treatment task) always coincides with the first acquisition task.

Theorem 1 : let Γ_i be a transaction, τ_{ua} a task under analysis. If the jitters are null and if the tasks of Γ_i are such that their WCET are decreasing, i.e. $C_{ij}\ge C_{ik}$ $\forall(j\le k)\in hp_i(\tau_{ua})$, and offset shifting are increasing i.e. $E_{ij}\le E_{i(j+1)}$ for $j<|hp_i(\tau_{ua})|$, then the critical instant of τ_{ua} coincide with the release of the first task of $hp_i(\tau_{ua})$.

Proof : the proof is based on the interferences. According to the definition of E_{ij}, $\sum E_{ij}=T_i$. For this proof, we use the method of calculation presented in section 4.1. In this section we have shown that the difference of interference between a candidate k_p and the candidate k_1 was obtained for every $k_j\in hp_i(\tau_{ua})$ by :

Difference of $+C_{ikj}$ for $k_j<k_p$ if

$t\%T_i\in]\Phi_{ik_j}-\Phi_{ik_1}..T_i+\Phi_{ik_j}-\Phi_{ik_p}]$ (1)

Difference $-C_{ikj}$ for $k_j\ge k_p$ if

$t\%T_i\in]\Phi_{ik_j}-\Phi_{ik_p}..\Phi_{ik_j}-\Phi_{ik_1}]$ (2)

Let us analyze these intervals in the context C_{ij} decreasing and E_{ij} increasing ; let us compare the candidates k_1 and k_2 :

$k_j=k_1$	Difference $+C_{ik1}$ for $t\%T_i\in]0..T_i+\Phi_{ik_1}-\Phi_{ik_2}]$ i.e. for $t\%T_i\in]0..T_i-E_{ik_1}]$, let us note I_{ik2k1} this interval
$k_j=k_2$	Difference of $-C_{ik2}$ for $t\%T_i\in]0..\Phi_{ik_2}-\Phi_{ik_1}]$ i.e. for $t\%T_i\in]0..E_{ik_1}]$, let us note I_{ik2k2} this interval
$k_j=k_3$	Difference of $-C_{ik3}$ for $t\%T_i\in]\Phi_{ik_3}-\Phi_{ik_2}..\Phi_{ik_3}-\Phi_{ik_1}]$, $t\%T_i\in]E_{ik_2}..E_{ik_1}+E_{ik_2}]$, let us note I_{ik2k3} this interval
$k_j=k_n$	Difference of $-C_{ikn}$ for $t\%T_i\in]\Phi_{ik_n}-\Phi_{ik_2}..\Phi_{ik_n}-\Phi_{ik_1}]$ i.e. for $t\%T_i\in]E_{ik_2}+E_{ik_3}+...+E_{ik_{n-1}}..E_{ik_1}+E_{ik_2}+E_{ik_3}+...+E_{ik_{n-1}}]$, let us note I_{ik2kn} this interval

We will prove now that with our constraints, the intersection of the intervals giving a negative difference is empty, i.e. there is at most one negative value for any value of $t\%T_i$; and then if $t\%T_i$ is in an interval giving a negative value, in such a case we are in an interval giving a positive value. Therefore, we will prove that either there is not any difference of interference (neither negative nor positive) or there is at most one negative value but in this case there is a positive difference that is greater or equal to the negative difference (since its value is C_{ik1}). In the proof, an interval I is < (lower) than an interval J if any value of I is lower than any value of J.

$I_{ik2k2}< I_{ik2k3}$ because $E_{ik1}\le E_{ik2}$

$I_{ik2k3}<I_{ik2k4}$ because $E_{ik1}+E_{ik2}\le E_{ik2}+E_{ik3}$ because $E_{ik1}\le E_{ik3}$

…

$I_{ik2kn-1}<I_{ik2kn}$ because $E_{ik1}+E_{ik2}+\ldots+E_{ikn-2}\le E_{ik2}+E_{ik3}+\ldots+E_{ikn-1}$ because $E_{ik1}\le E_{ikn-1}$

Consequently, the intersection of the negative intervals is empty.

Finally, we will prove that if t is in one of the intervals $I_{ik2kp,\ p\in 2.kn}$, then it is in the interval I_{ik2k1}.

Let us suppose that $t\%T_i\notin I_{ik2k1}$, this means $t\%T_i\in]T_i-E_{ik1}..T_i[\cup\{0\}$.

If $t\%T_i=0$, then t is not element of any interval

In the case $t\%T_i \in]T_i-E_{ik1}..T_i[$, we will prove that T_i-E_{ik1} is greater than any other interval $I_{ik2kj,j=2..kn}$. It is sufficient for this proof, since the intervals are increasing, to prove that $T_i-E_{ik1} \geq E_{ik1}+E_{ik2}+\ldots+E_{ikn-1}$. So, we have to prove that $T_i \geq 2E_{ik1}+E_{ik2}+\ldots+E_{ikn-1}$; since by definition $T_i=E_{ik1}+E_{ik2}+\ldots+E_{ikn}$, therefore we have to prove that $E_{ik1}+E_{ik2}+\ldots+E_{ikn-1}+E_{ikn} \geq 2E_{ik1}+E_{ik2}+\ldots+E_{ikn-1}$, this is true because $E_{ikn} \geq E_{ik1}$.

Let us generalize to a task k_p of the serial transaction:

$k_j=k_1$	Difference $+C_{ik1}$ for $t\%T_i \in]0..T_i+\Phi_{ik_1}-\Phi_{ik_p}]$ i.e. for $t\%T_i \in]0..T_i-(E_{ik_1}+E_{ik_2}+\ldots+E_{ik_{p-1}})]$ since $T_i=\Sigma E_{ij}$ $t\%T_i \in]0..E_{ik_p}+E_{ik_{p+1}}+\ldots+E_{ik_n})]$ let us note I_{ikpk1} this interval
$k_j=k_2$	Difference $+C_{ik2}$ for $t\%T_i \in]\Phi_{ik_2}-\Phi_{ik_1}..T_i+\Phi_{ik_2}-\Phi_{ik_p}]$ i.e. for $t\%T_i \in]E_{ik_1}..T_i-(E_{ik_2}+\ldots+E_{ik_{p-1}})]$ since $T_i=\Sigma E_{ij}$ $t\%T_i \in]E_{ik_1}..E_{ik_1}+E_{ik_p}+E_{ik_{p+1}}+\ldots+E_{ik_j}$ let us note I_{ikpk2} this interval
$k_j=k_p$	Difference of $-C_{ikp}$ for $t\%T_i \in]0..\Phi_{ik_j}-\Phi_{ik_1}]$ i.e. for $t\%T_i \in]0..E_{ik_1}+\ldots+E_{ik_{p-1}}]$, let us note I_{ikpkp} this interval
$k_j=k_p+1$	Difference of $-C_{ikp+1}$ for $t\%T_i \in]\Phi_{ik_p+1}-\Phi_{ik_p}..\Phi_{ik_p+1}-\Phi_{ik_1}]$ i.e. for $t\%T_i \in]E_{ik_p}..E_{ik_1}+\ldots+E_{ik_p}]$, let us note $I_{ikpkp+1}$ this interval
$k_j=k_n$	Difference of $-C_{ikn}$ for $t\%T_i \in]\Phi_{ik_n}-\Phi_{ik_p}..\Phi_{ik_n}-\Phi_{ik_1}]$ i.e. for $t\%T_i \in]E_{ik_p}+E_{ik_{p+1}}+.$ $..+E_{ik_{n-1}}..E_{ik_1}+E_{ik_2}+E_{ik_3}+\ldots+E_{ik_{n-1}}]$, let un note I_{ikpkn} this interval

The proof uses the same way as before, except that for the general case, we show that there are always at least as many positive interval than negative intervals. Since the WCET cannot decrease, and since the positive intervals correspond to the first tasks of the transaction, the positive difference is always greater or equal than the negative difference.

- $t\%T_i \in]0..E_{ik1}]$: $t\%T_i \in I_{ikpk1}$ and $t\%T_i \in I_{ikpkp}$, and $\forall k_q>k_p$, $t\%T_i \notin I_{ikpkq}$ because the lower limit of these intervals is greater than $E_{ikp} \geq E_{ik1}$. So, there is at least one positive interval (giving a difference of C_{i1}) and at most one negative interval (giving a difference of C_{ikp}) and since $C_{i1} \geq C_{ikp}$, we obtain $W_{i1}(\tau_{ua}, t\%T_i \in]0..E_{ik1}]) - W_{ip}(\tau_{ua}, t\%T_i \in]0..E_{ik1}]) \geq 0$
- $t\%T_i \in]E_{ik1}..E_{ik1}+E_{ik2}]$: $t\%T_i \in I_{ikpk2}$ (positive intervals), $t\%T_i \in I_{ikpkp}$ (negative interval). It is possible that $t\%T_i \in I_{ikpkp+1}$ (negative interval), but in this case, $t\%Ti \in I_{ikpk1}$ (positive interval). On the contrary, $\forall k_q>k_{p+1}$, $t\%T_i \notin I_{ikpkq}$ because $E_{ikp}+E_{ikp+1} \geq E_{ik1}+E_{ik2}$. Since the execution times are nonincreasing, we have $W_{i1}(\tau_{ua}, t\%T_i \in]E_{ik1}..E_{ik1}+E_{ik2}]) - W_{ip}(\tau_{ua}, t\%T_i \in]E_{ik1}..E_{ik1}+E_{ik2}]) \geq 0$
- the same reasonning can be lead on the other possible intervals for $t\%T_i$ for every interval of length Eik_j.

□

5- Validation of the case study

Theroem 1 implies that in order to analyse an intermediate priority task, it is sufficient to test its response time when it's released at the same time as the first task of the serial transaction to obtain its tight worst-case response time with a classic response time analysis. Note that this theorem cannot be applied to a lower priority task, because the condition "decreasing WCET" is not satisfied in this case.

Let S be a set of transactions.

Let τ_{ua} be a task of S under analysis with execution time equal to C_{ua}.

Let us note $hp(\tau_{ua})$ the set of serial transactions in S such as τ_{ua} is a lower priority task. Let us note $it(\tau_{ua})$ the set of indices of serial transactions in S such as τ_{ua} is an intermediate priority task .

By applying Theorem 1, the interference applied by the serial transactions whose indices belong to $it(\tau_{ua})$ in a time interval of length t does not need any specific study related to transactions. It is given (tight upper bound) by:

$$\sum_{j \in it(\tau_{ua})} \left(\left\lfloor \frac{t}{T_j} \right\rfloor \cdot L_j + \min\left(\left\lceil \frac{t\%T_j}{p_j} \right\rceil, L_j \right) \right) \cdot C_j$$

In this formula, $\left\lfloor \frac{t}{T_j} \right\rfloor$ represents the number of periods T_j completed in the time interval of length t; and $\min\left(\left\lceil \frac{t\%T_j}{p_j} \right\rceil, L_j \right)$ represents the number of acquisition tasks activated in the remaining time($t\%T_j$).

We still need to use the technique defined in [TN04b] in order to study the interference of the serial transactions whose indices belong to $hp(\tau_{ua})$, leading to a pessimistic upper bound, but allowing us to validate the case study (see Table 3). This application is valid

because in the table 3, we can see that for all the tasks, the worst-case response time is lower than the deadline.

Tasks	Period	deadline	Priority	Worst-case response time
1	200000	200000	1	56156
2	20000	10000	7	6532
3	50000	30000	5	15532
4	20000	10000	6	6572
5	250000	140000	2	56096
6	60000	60000	4	54636
7	250000	160	11	124
8	20000	720	10	468
9	100000	80	12	12
10	250000	5000	9	3408
11	20000	7500	8	5620
12	100000	70000	3	55416

Table 3: Worst-case response time calculated with serial transaction method

6– Conclusion

In this article, we have presented a new task model: the serial transaction. A serial transaction Γ_i is compound with L_i short but urgent acquisition tasks activated each time a serial packet is received, and a less urgent but longer treatment task activated when a whole frame is received.

The number of acquisition tasks can be important (more than 120 in a real case study) and makes the exact calculation of response time intractable. Moreover, overestimating the worst-case response time of the urgent acquisition tasks wouldn't allow the validation of a task system.

After simplifying the way to evaluate the interference of a transaction and finding the critical instant candidate, we have shown that for tasks of intermediate priority, the critical instant always coincides with the release of the first task of the transaction (Theorem 1) . This new result allows us to calculate an exact worst-case response time for intermediate priority tasks (usually most tasks of a system), while we still use the method proposed in [TN04b] for the tasks of lower priority than a whole serial transaction. Our future work is generalizing the theorem 1 to a larger case of transactions called monotonic transactions. Moreover, an extension of this theorem taking jitters into account is investigated.

References

[Aud91] N.C. Audsley, Optimal priority assignment and feasibility of static priority tasks with arbitrary start times, Tech. Report YCS-164, University of York, nov. 1991.

[Der74] M.L. Dertouzos, Control robotics : the procedural control of physical processors, Proc. of IFIP Congress, 1974, pp. 807-813.

[DM89] M.L. Dertouzos, A.K. Mok, Multiprocessor on-line scheduling of hard real-time tasks, IEEE Transactions on Software Engineering 15(12), Déc. 1989, 1497-1506.

[GPS1] TIM-LC, TIM-LF, TIM-LP System Integration Manual, http://www.u-blox.com

[Gro99] E. Grolleau, Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de pétri en environnement monoprocesseur et multiprocesseur, thèse, ENSMA - Université de Poitiers, nov. 1999.

[IMU1] Crista Inertial Measurement Unit (IMU) Interface / Operation Document, May 2004, http://www.cloudcaptech.com.

[Lab74] J. Labetoulle, Un algorithme optimal pour la gestion des processus en temps réel, Revue Française d'Automatique, Informatique et Recherche Opérationnelle (Fév.1974), 11-17.

[LL73] C.L. Liu and J.W. Layland, Scheduling algorithms for mutltiprogramming in real-time environnement, Journal of the ACM 20(1) (1973), 46-61.

[LW82] J. Leung and J. Whitehead, On the complexity of fixed-priority scheduling of periodic, real-time tasks, Performance Evaluation (Netherland) 2(4) (1982), p.237-250.

[MPC1] MPC555/MPC556 User's Manual October 2000, http://e-www.motorola.com

[MS03] J. Mäki-Turja and M. Sjödin, Improved Analysis for Real-Time Tasks With Offsets –Advanced Model. Technical Report MRTC no. 101, Mälardalen Real-Time Research Centre(MRTC), May 2003

[Osek1] OSEK/VDX operating system specification 2.2.2 July 2002, http://www.osek-vdx.org.

[Osek2] OSEK/VDX System Generation OIL : Osek Implementation Langauge version 2.5, Juillet 2004, http://www.osek-vdx.org.

[OSM1] OSEKturbo OS/MPC5xx v2.2.1 Technical Reference, Juin 2003, http://www.metrowerks.com.

[OSM2] OSEKturbo OS/MPC5xx User's Manual, Juin 2003, http://www.metrowerks.com.

[OSM3] OSEKturbo performance information, http://www.metrowerks.com

[PG98] J.Palencia Gutierrez and M.Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In Proc. 19th IEEE Real-Time System Symposium (RTSS), December 1998

[Tin94] K. Tindell, Addind Time-Offsets to Schedulability Analysis, Technical Report YCS 221, Dept of Computer Science, University of York, England, January 1994

[TN04a] J.Mäki-Turja and M.Nolin. Faster Response Time Analysis of Tasks with Offsets. In Proc. 10th IEEE Real-Time Technology and Applications Symposium (RTAS), May 2004

[TN04b] J.Mäki-Turja and M.Nolin. Tighter Response Time Analysis of Tasks with Offsets. In Proc. 10th International conference on Real-Time Computing and Applications (RTCSA'04), August 2004

[XP92] J. Xu and D.L. Parnas, Pre-run-time scheduling of processes with exclusionrelations on nested or overlapping critical sections, Phoenix Conference on Computers and Communications (Phoenix, USA), Apr. 1992,pp. 6471-6479.

The Real-Time MatPLC

Mário de Sousa
Faculdade de Engenharia da Univ. do Porto
Dept. Eng. Electrotecnica e de Computadores
R. Dr. Roberto Frias, 4200-465 Porto
msousa@fe.up.pt

Adriano Carvalho
Faculdade de Engenharia da Univ. do Porto
Dept. Eng. Electrotecnica e de Computadores
R. Dr. Roberto Frias, 4200-465 Porto
asc@fe.up.pt

Abstract

The MatPLC is an open-source industrial control application, consisting of a core, generic modules, and tools for creating custom modules. Since many control and monitoring systems require strict time determinism, hard real-time capabilities were added to the MatPLC. The paper includes an outline of the MatPLC's architecture, and details the design changes required to add the hard real-time capabilities to the MatPLC framework. The execution times of the real-time version of the MatPLC were measured and analysed.

1. Introduction

The technology used in PLCs (Programmable Logical Controllers) have evolved with the times, to the point that many modern top of the range PLCs are actually full fledged computers in disguise, executing modern operating systems. More importantly, and in order to take advantage of economies of scale, many PLC vendors have started to adopt hardware similar to PCs (Personal Computers).

The MatPLC project was started with the intention of eliminating the lock the vendors have on the end-users, by taking advantage of open standards as well as open source operating systems running on the de facto standard PC platform. It has currently been successfully tested on a DIN rail mounted PC platform marketed by SixnetIO. Most industrial communication networks are supported (Modbus, Devicenet, Profibus, ASI, etc.), either directly or through the use of a network interface card manufactured by Hilscher. Software standards are also being taken into account. Currently the project includes a compiler for the IL (Instruction List) and ST (Structured Text) programming languages defined in IEC 61131-3.

Many industrial control applications require deterministic behaviour not only from the control algorithm, but also in the time domain. In order to be able to support these applications, it becomes necessary that the MatPLC's framework and the implementation itself be augmented in order to provide real-time guarantees. This paper discusses and details the design changes to the MatPLC framework, and the implementation effort of porting the MatPLC to a real-time platform. The small standard demo was run on the new real-time capable MatPLC in order to evaluate and validate this new version.

2. MatPLC Overview

The MatPLC [1-3] is an open-source control application, composed of autonomous but cooperating modules that execute in separate user-space processes, and access a common shared state stored in shared memory. Each of these modules is free to decide whether or not to execute in a standard PLC scan loop, which means that the industrial control application may be developed using either asynchronous programming common to most PLCs, or synchronous programming common to most other computer architectures. Both methods may even be used simultaneously, as long as they are not mixed within the same MatPLC module. The application builder may choose to let the modules execute autonomously, or to synchronize their activities.

Access to all shared resources is made through the MatPLC library routines which offer PLC-like semantics for the modules that wish to use them, such as inputs that only change at the beginning of the logic and outputs that are only written at the end of the logic. The library is divided into several sections (Fig. 1):

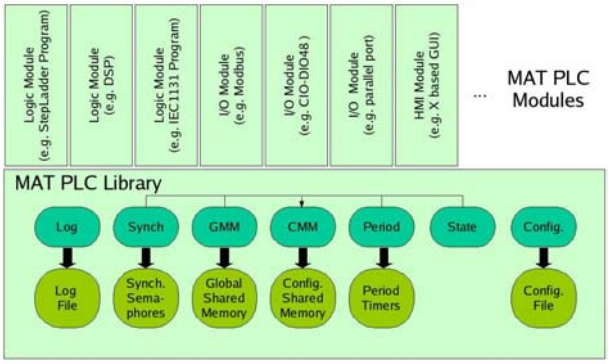


Figure 1. MatPLC Architecture, showing a selection of modules in the top part and the sections of the library in the bottom.

- configuration memory manager (cmm) - manages the shared memory area that stores core configuration data, guaranteeing that all modules share the same configuration;
- global memory manager (gmm) - manages the shared memory area used to store the state of the plc points;
- synch section - handles the synchronization between modules;
- period section - enforces scan loop timings;
- state section - handles module execution state;
- configuration section - parses the configuration files;
- log section - allows every module to produce log messages in a consistent manner. These are timestamped and written to a logging file.

A more detailed explanation of the internal mechanisms used within the more complex sections follows.

2.1.The Configuration Memory Manager

The cmm is used by the other sections to store the current configuration of the PLC. For example, the synch section stores how the modules should synchronise their execution amongst themselves, the gmm section stores the location of named PLC points, while the state section stores the identification of the synchronisation point being used to control the PLC run/halt state. The use of a shared memory location instead of a configuration file (that may be changed between the launch of two modules) guarantees that all modules use the same configuration. Every access to the configuration memory area is made through the cmm library section.

Several MatPLCs can run simultaneously on the same system. Each is distinguished by the configuration memory area it uses, which is identified by a unique number. When a module is launched, the identity of the MatPLC to which it should attach is specified as a command line parameter.

2.2.The Global Memory Map

Since the MatPLC architecture hinges on the simultaneous execution of several modules, every access to the global memory map by a specific module must be made to be atomic with respect to the other modules. Several modes are available for enforcing this constraint (Fig. 2), none of which is optimal for all possible scenarios in which the MatPLC is expected to be used. The application builder may choose any of the access modes for each executing module.

For the default 'local' mode, a local copy of the memory map is created for the module (Fig. 2 - Module B). When a module accesses a plc point, it is actually accessing its local memory map. Local and global memory maps are synchronized by calling the plc_update() function, which is protected by a semaphore, providing atomic updates with respect to other modules.

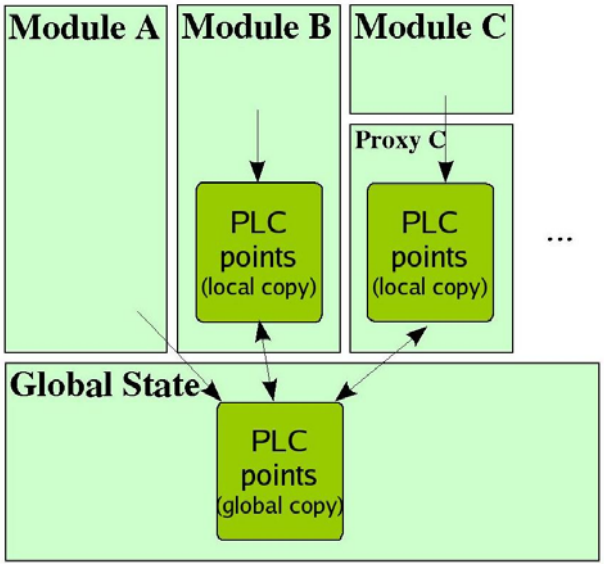


Figure 2. Synchronization of gmm memory maps (local, isolate and shared)

The second mode, `isolate' (Fig. 2 - Module C), completely isolates the module from the shared memory used by the PLC, and is mostly used by untrusted modules (e.g. still in a debugging stage). It uses sockets to forward the plc_update() function call from the module to the proxy, introducing significant overhead.

The third and last mode, `shared' (Fig. 2 - Module A), assumes that simultaneous access to the shared map will never occur, and therefore gives the module direct access to the global memory map with no synchronization enforced. This may be guaranteed if all modules are running a scan loop with each loop executing in turn (possible due to the synch section, described below), or if the modules access disjoint portions of the global map.

2.3.The Synch Section

The synch section allows the application builder to specify the sequence of execution of the running

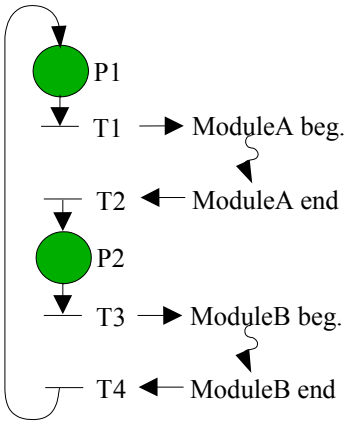


Figure 3. The synchronization model with an example Petri Net.

modules. This is achieved by specifying a Petri net, taking into account that particular synchronization points in each module (usually beginning and end of scan, but optionally others) are associated with the firing of a transition. When such a synchronisation point is reached during the module's execution, the module blocks until the transition fires (assuming it cannot fire immediately). Note that a transition will not fire unless a module is waiting on it; in this the semantics differ from those of a standard Petri net.

The synch section leverages the standard SysV semaphores to simulate the synchronization Petri net, using a single SysV semaphore set, with a semaphore for each place. Each transition is implemented by simultaneously waiting on all the appropriate semaphores, a functionality of SysV semaphores not supported by POSIX semaphores.

2.4.The Period Section

This section enforces maximum scan rates for each module. It uses POSIX timers to set an alarm that goes off at every multiple of the desired scan period, at which time an alarm counter is incremented. When a module is ready to start a new scan it decrements the alarm counter and continues with the scan. If there are no outstanding alarms, the scan is delayed until the next alarm goes off.

2.5.The State Section

This section of the library handles RUN/STOP modes both for the MatPLC as a whole and for each module individually. A module will only execute a scan if both the whole MatPLC and the module itself are in RUN mode.

In order for these modes to work correctly with the synch library, they are implemented by adding hidden places to the synch petri net. These places are connected with arcs to the begin of scan synch transitions in such a way that the transition will only fire if both the PLC and the module are in RUN mode, with the tokens being replaced after firing. A module is thus able to atomically verify the conditions required by both the state and the synch sections.

2.6.MatPLC Modules

Currently the MatPLC comes with several modules, which may be grouped into I/O (Input/Output) modules, logic modules, and human interface modules.

The I/O modules interface with physical devices through either local or remote digital and analog I/O. Modules are provided that allow access to local I/O cards based on the Intel 8255, a PC's parallel port, and all cards supported by the comedi [4] library. Support for all main fieldbus networks (Devicenet, ASI, Profibus, etc.) is achieved through the use of network cards made by hilscher [5], besides the three modbus variants that are also supported directly by a specific MatPLC module.

Logic modules include a DSP (Digital signal Processing) module that implements PID loops as well as digital filters, and a compiler that generates modules from control programs written in the standard text based PLC programming languages (IEC 61131-3 IL and ST [6]).

Human interface includes graphical interfaces based on the gtk and the tcl/tk libraries, as well as a text based interface module based on the curses library.

3.Real-Time MatPLC

In order to support RT (Real-Time) applications, the MatPLC framework has been augmented to become time-deterministic. This is achieved mainly by adding support for the concept of RT modules through the use of execution priorities.

In fact, and considering that a real-time control application is commonly composed of both hard (e.g. control) and non-hard real-time (e.g. graphical interface) components, it makes sense to re-use the existing modularity of the MatPLC to support the modularity of RT applications. The components of a RT application are therefore expected to be mapped onto MatPLC modules, which means that the revised MatPLC framework therefore must support modules with RT characteristics. Additionally, the MatPLC's core library routines had to be revised so that non-RT modules do not cause undue interference and blocking on the RT modules.

The concept of RT modules was therefore added to the MatPLC framework. These modules are largely identical to traditional MatPLC modules, but augmented to provide deterministic behaviour. This is achieved by allowing the user to specify a fixed execution priority for each RT module, as well as by eliminating potential blocking due to memory page swapping by the operating system's memory manager. All sections of the MatPLC's core were also analysed in order to remove any potential interference and blocking between the RT and non-RT modules.

Since the MatPLC runs over an operating system, the above functionality not only requires support from the operating system, but also assumes that the operating system itself is time deterministic. The MatPLC was at first coded on a non real-time version of Linux, therefore creating a RT version of the MatPLC required that it be first ported to a RT operating system. In order to maintain the most portability, it was decided to code the RT version of the MatPLC to the RT POSIX standard [7]. As no version of RT Linux was at the time completely RT POSIX compatible, QNX was chosen as the first target operating system.

3.1. CMM and GMM Sections

The cmm and gmm sections each use shared memory during run mode, and a single semaphore each for controlling access to those memory sections. However,

unlike the access to the cmm memory area which is only made during initialization, access to the gmm memory area is made during run-time by modules which may be executing under differing priorities. This introduces the possibility that unbounded priority inversion may occur. To work around this a scheduling protocol that bounds priority inversion (e.g. PCP – Priority Ceiling Protocol [8]) has to be used.

The original MatPLC implementaion used SysV sempahores to synchronise the access to the gmm memory area when using the 'local' access mode. Unfortunately SysV sempahores do not support priority inheritance protocols, and neither do POSIX semaphores. The only available option is to use POSIX mutexes, which do support PCP and similar scheduling protocols, but however are not mandated by the POSIX standard to work between threads residing on seperate processes. For the MatPLC, with each module running under a separate process, mutex synchronisation between processes was a requirement, otherwise a big revamping of the MatPLC framework would have to be attempted. Fortunately QNX allows mutexes to be used between processes, so the revamping of the MatPLC framework was not required.

Considering that mutexes had to be used for synchronising RT MatPLC modules, and that the mutexes with inter-process capability may not be available under all POSIX compliant operating systems, to have a truly portable MatPLC it became necessary to re-implement the gmm synchronisation to use either POSIX mutexes, as well as POSIX semaphores. Since the SysV semaphores were already being used, and two new options were also required, it was decided to encapsulate the synchronisation mechanism in a plc_mutex abstraction, which is mapped onto one of the three available options at compile time. The RT verson of the MatPLC, currently running on QNX, uses the POSIX mutex version to synchronise the access to the gmm shared memory, and configured to use the priority inheritace protocol.

Both the POSIX semaphore and the POSIX mutex variants require that the semapore/mutex be placed in memory that may be accessed simultaneously by all processes that synchronise to that semaphore/mutex. A cmm memory block was used to store the semaphore/mutex since the memory managed by the cmm is shared between all running MatPLC modules.

In the gmm 'shared' access mode no explicit synchronisation between the processes accessing the gmm global memory map is attempted by the gmm section. The processes are expected to either access disjoint areas of the memory map, or to never execute concurrently (which may be enforced by the synch section). This means that the gmm section does not make use of any synchronisation mechanism when in 'shared' access mode, and therefore did not require any changes to become time deterministic and RT ready when in this mode.

Another issue stems from the sockets used by the `isolate' mode of the gmm section, which introduce additional overhead which is difficult to impossible to evaluate, and therefore cannot be used in a RT deterministic setting. For this reason, support for the 'isolate' option under a RT MatPLC has for the moment been deferred to a later date, but most likely never.

Although it is not strictly necessary to support priority inheritance in the synchronisation mechanism used to control access to the cmm (only accessed at start-up of each module), the cmm itself now also uses the encapsulated plc_mutex that was created specifically for the gmm, making it safe to access the cmm after start-up if it ever becomes necessary. This potential future access would however not be entirely innocuous due to the additional bounded blocking that it would introduce to the RT MatPLC modules.

3.2.Synch Section

Before implementing RT behaviour on the MatPLC, the synch section had been originally implemented using SysV semaphores, making extensive use of their richer semantics; simultaneous and atomic waiting and posting on the same semaphore, or on different semaphores in the same semaphore set.

Since no RTOS currently supports SysV semaphores, this section had to undergo significant changes. As with the gmm section, it was decided to maintain the SysV implementation since it has higher execution speed and is therefore preferable when SysV semaphores are available and no RT requirements are necessary. Once again the SysV semaphore version was encapsulated inside a plc_synchsem abstraction (implemented as a library) that provides the same semantics as SysV semaphores.

Two new versions of the plc_synchsem were implemented, one using POSIX semaphores and the other using POSIX mutexes. The version to be used is decided at compile time depending on the synchronisation mechanisms supported by the operating system being used, as well as the RT requirements. Special effort was made to eliminate unbounded priority inversion and blocking in the two POSIX variants, especially on the POSIX semaphore version since only the POSIX mutex version provides automatic bounded blocking through the use of priority inheritance protocols. This means that the POSIX semaphore variant may also be used in a RT setting, although it may be a little slower.

For both POSIX variants a plc_synchsem consists of a data structure with the following shared data elements:

1. an array with the current value of each emulated semaphore in the semaphore set,
2. a list of processes currently blocked trying to synchronise to the semaphore set,

3. and a semaphore/mutex to control the access to the above data structures.

   Additionally, each process that will synchronise to a plc_synchsem also has a private semaphore (condition variable on the POSIX mutex variant) on which it will block, waiting to be release by another process.

The synchronisation algorithm for synchronising with a plc_synchsem follows the following steps:

1. lock the shared semaphore/mutex of the plc_synchsem;
2. verify if the conditions the calling process specified for synchronising are met; if false go to to 3, else go to 4.
3. add the process to the decreasing priority ordered list of currently blocked processes, release the semaphore/mutex locked in step 1, and then block on the private semaphore (or he condition variable for the POSIX mutex variant). When this process becomes unblocked (through the actions of another process executing step X), simply return.
4. make the required changes to the value of each semaphore in the plc_synchsem set.
5. Run sequentially through all processes (by decreasing priority) that are currently blocked and check whether the new semaphore values allows the process to become unblocked. As soon as the highest priority process that may become unblocked is found, then remove that process from the blocked processes list and (a: POSIX semaphore variant) add it to a list of processes to be unblocked later or (b: POSIX mutex variant) signal the private condition variable on which the process is blocked, and go to step 4.
6. The current semaphore values do not permit the unblocking of any further processes, so (a: POSIX semaphore variant) run through the list of processes to be unblocked and unblock them in decreasing priority order by signalling the private semaphore on which each process is blocked, or (b: POSIX mutex variant) do nothing.
7. release the semaphore/mutex locked in step 1.

For the POSIX semaphore version the above algorithm releases the processes in decreasing priority order so as to eliminate the possibility of priority inversion and unbounded blocking. If this were not done, the release of a mid-priority process may create the conditions to release a higher priority process. If the mid-priority process is released by a low priority process (i.e. the process that is synchronising with the plc_synchsem), then the mid-priority process will pre-empt the low priority process and therefore delay the unblocking of the high priority process. This occurrence would lead to unbounded blocking of the higher priority process, which is not desirable in RT systems.

For the POSIX mutex version it is no longer necessary to release the processes in decreasing priority order since the signalling of the private condition variable does not immediately release the blocked process. All blocked processes whose private condition variables were earlier signalled are only released simultaneously and atomically at the same time the global shared mutex is released in step 7, therefore eliminating the possibility of priority inversion occurring.

3.3.The New RT Section

A new section was added to the MatPLC core library to add support for the RT specific configuration parameters – process priority and memory management.

The new RT section basically sets, at module start-up, the priority of the process running each module to the priority requested by the user in the MatPLC's configuration file, and sets the scheduling algorithm to the POSIX fixed priority SCHED_FIFO. If no explicit priority is specified by the user, then the default scheduling algorithm is left unchanged, as is the initial priority.

Besides the priority, the RT section also configures the way the memory used by each module is managed by the underlying operating system. If at least one module is configured to run under RT priority, then that module, as well as all the others, are configured by this section to run with their memory locked to RAM, and with swapping to disk disabled. Note that it is not sufficient to lock the RT modules' memory to RAM, as these modules may experience bounded blocking from the remaining modules through the mutex used by the gmm. This means that even non-RT modules may at some intervals execute under an inherited RT priority, so in order to avoid undue delays, all modules must have their memory locked to RAM.

3.4.Log Section

The log section currently writes all logs to a user configurable file during run time. Since the most probable is to have the file residing on disk, file access times are not deterministic. Therefore this section also needs changing in order to support RT guarantees. Although no change has yet been attempted, it is expected that in the future RT version of this section all logs will be sent to a RT FIFO/message queue, where they may be later removed by a non real-time logging process. This process may then send these logs to a file, to a terminal or even to the UNIX system log.

For the moment RT modules may not produce any logs during run time so as not to introduce unbounded delays. Another option is to configure the logging file on a memory mapped file system where file access times are more deterministic.

3.5.Modules

Since the bulk of the code in the MatPLC is in the modules, one would expect that they would require the bulk of the porting effort. This, however, does not appear

to be the case. The majority of modules fall into two classes: those which are never going to be real-time (file loggers, graphical user interfaces, etc.) and those which require no change at all or very little (e.g. DSP module, modules generated by the IEC 61131 ST/IL compiler). This stems from the fact that the RT modules only execute asynchronous logic, with all synchronization activities residing in the MatPLC library calls that were already discussed above. However, these modules do need to be linked to time deterministic versions of the C, thread and maths libraries.

The only exception to the above broad division are the I/O modules, which do require significant work. In order to operate in real-time, such modules may only make use of time deterministic hardware drivers, besides being themselves time deterministic.

This means that currently no networked I/O may be accessed with deterministic time, including the I/O on fieldbus networks since the the device driver for the hilscher cards is not available for the QNX operating system on which the RT MatPLC currently runs.

4.Test and Evaluation

4.1.Experimental Set-up

The resulting RT MatPLC implementation was tested on a personal computer with a 350 Mhz Pentium II and 320 Mbytes of RAM, running the QNX 6.2.0 operating system. The basic demonstration set-up was run, consisting of two modules: a text based human interface module, and a logic execution module written in C. The logic module simply switches on one out of four 'lights' in sequence, while the text mode interface displays the status of these four 'lights'. Both modules ran asynchronously and used the 'local' method of accessing the gmm memory map.

With the above set-up, a measurement was made of the time elapsed between the beginning of two consecutive scans of the logic module. This time differential was measured for 25000 scan cycles of the logic module, and saved to memory during the experiment so as not to disturb the measurement itself. Only at the end of all measurements were the results stored to a file on disk.

The time itself was measured using the clock counter present on all Intel compatible CPUs, and read using the *rdtsc* assembly instruction. This counter internal to the CPU counts the number of clock cycles since the CPU was switched on. On the 350 Mhz CPU this counter presents a resolution of 2,85 ns.

4.2.Results

The demo was executed twice: the first run with both modules executing with the default (non RT) execution priority and scheduling algorithm. The second run had the logic module running under a higher priority, and using a fixed priority FIFO scheduling algorithm. The results are presented in the following figures.

As expected, when running under non-RT priority the execution period has high variability and suffers from large jitter, whereas with a high priority the execution period becomes more periodic.

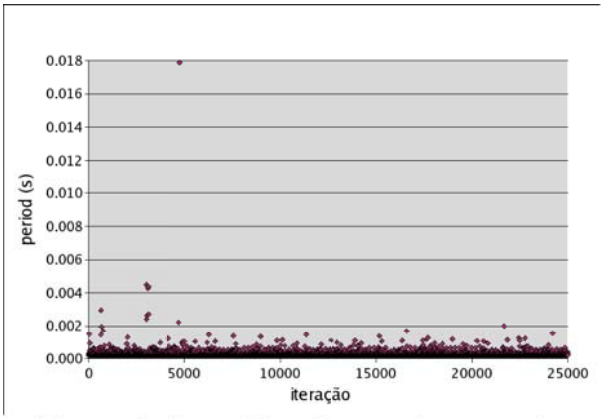


Figure 4. Execution times when running under non real-time priority

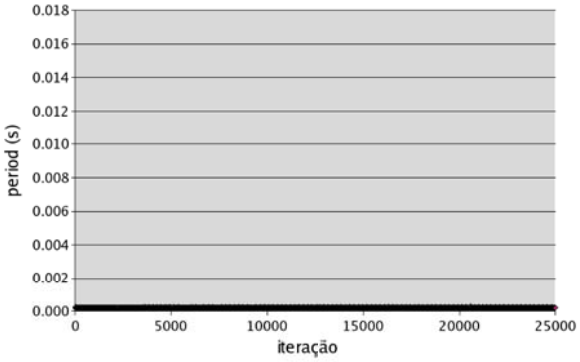


Figure 5. Execution times when running under high real-time priority.

5.Conclusions

In order to be able to support control applications with strict time constraints, the MatPLC's framework and the implementation itself were augmented in order to provide real-time guarantees.

The MatPLC's modularity meant that the framework did not require significant changes, merely requiring the concept of real-time modules. The code itself is also highly partitioned, with most synchronisation functions limited to the MatPLC's core library. In this library, the gmm, cmm and the synch sections required most attention in order to eliminate any possibility of unbounded priority inversion. A new RT section was also added to support the configuration of RT modules.

The small standard demo was run on the new real-time capable MatPLC in order to evaluate and validate this new version.

Code for the MatPLC can be obtained from the project's cvs server and on the website, http://mat.sf.net. The Real-Time version of the MatPLC has been merged with the main project, and may be obtained from the same location.

6.Acknowledgements

The authors would like to thank Curt Wuollet for taking the initiative of starting the MatPLC project. We also appreciate the contributions of Jiri Baum and Juan Carlos Orozco to the project.

References

[1] M. de Sousa, A. Carvalho, "MatPLC – the Truly Open Automation Controller", Proceedings of the 28th Annual Conference of IEEE Industrial Electronics, pp. 2278-2283, 2002

[2] M. de Sousa, A. Carvalho, "An IEC 61131-3 Compiler for the MatPLC", Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation, Lisbon-Portugal, 2003

[3] M. de Sousa, A. Carvalho, "Embedding the MatPLC", Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation, Catainia-Italy, 2005

[4] D. Chleef, "Comedi: Linux Control and Measurement Device Interface", http://www.comedi.org, available in December 2005

[5] http://www.hilscher.com, available in December 2005

[6] International Electrotechnical Commission, "IEC 61131-3, 2nd Ed. Programmable Controllers – Programming Languages" , International Electrotechnical Commission, Final Draft - 10th December 2001

[7] IEEE and The Open Group, "IEEE Std 1003.1, 2003 Edition", 2001-2003.

[8] L. Sha, R. Rajkumar, J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization", IEEE transactions on Computers 39 (9):1175-1185, 1990

Worst-case Execution Time

Code padding to Improve the WCET Calculability

Christine Rochange
Université Paul Sabatier
IRIT
31 062 Toulouse cedex 9
rochange@irit.fr

Pascal Sainrat
Université Paul Sabatier
IRIT
31 062 Toulouse cedex 9
sainrat@irit.fr

Abstract

The Worst-Case Execution Time of tasks with strict deadlines must be predictable: it must be possible to estimate this time both safely and tightly at an acceptable computing cost. Static WCET analysis is facilitated if parts of code can be analyzed more or less independently of one another. This is why it is desirable to prevent timing interferences between blocks. In this paper, we show how it is possible to transform the code to prevent timing effects between distant basic blocks on an execution path. Our approach consists in padding the code to space out basic blocks. Performance results show that the code size is sensibly increased but that the cost in terms of WCET degradation is moderate.

1. Introduction

Being able to estimate the Worst-Case Execution Time (WCET) of tasks is absolutely necessary for hard real-time systems. Measurement is generally inadequate because it cannot be guaranteed that all the possible execution paths have been tested. This is why academic research has focused on static WCET analysis.

The WCET estimated by static methods should obviously be safe since missing deadlines can have dramatic consequences in some critical systems. However, it should also be as tight as possible: WCET overestimation can have undesirable effects like the impossibility to schedule the tasks. It might also lead to oversized hardware.

Static methods compute an upper bound of the real WCET by combining information about the possible execution paths (produced by a preliminary analysis of the code) and the execution times of the basic blocks. These times can be determined by a cycle-level simulator of the target processor.

However, critical applications follow the general evolution towards more and more computing requirements. This is why advanced processor architectures tend to be used in critical systems [16]. Unfortunately, in a high-performance processor, a basic block does generally not execute the same way in the application code as it would do if it was executed alone. This is due to interferences (data dependencies, precedence constraints or resource conflicts) with other blocks on the execution path. To get the worst-case execution time of a block, these possible interferences should be taken into account. This is often a very complex task, as it will be explained in Section 2.

To keep the WCET analysis simple, we have recently proposed to modify the processor architecture to eliminate any possible timing effect between basic blocks [13]. The idea was to space out successive basic blocks in the pipeline in such a way that they cannot interfere. The proposed scheme obviously degrades the performance (in the order of 42% for an 4-way superscalar out-of-order processor) but the loss could be acceptable in the name of timing safety. However, the main problem is that such a processor does not exist for the moment. This is the reason why we suggest that the distance between blocks could be enforced by the compiler instead of the hardware.

Our approach consists in padding the code by inserting neutral filler-instructions, *i.e.* instructions that will not be executed but only fetched and decoded before being removed from the pipeline (like a true NOP). The lengths of the padding blocks are computed so that they eliminate all the possible interferences between basic blocks.

Note that this work focuses on timing interferences related to the use of the pipeline and of the internal processor resources. We do not address here the question of modeling caches, branch predictors, etc. This is why we will consider these components as perfect (*i.e.* with a very predictable behaviour) in the evaluation part.

The paper is organized as follows. Section 2 gives some background information on static WCET analysis and on the possible timing interferences between blocks in high-performance processors. It also overviews related work. We introduce our approach in Section 3. Performance results are analyzed in Section 4 and concluding remarks are given in Section 5.

2. Background

2.1. Static WCET estimation

Static analysis techniques add the execution times of basic blocks on the possible execution paths extracted either from the syntax tree [8] or from the control flow graph [6]. For example, the *Implicit Path Enumeration Technique* handles the search of the WCET as an optimization problem where:

- the objective function is the program execution time expressed as the sum of the basic block execution times weighted by their respective numbers of execution. As we will explain it in Section 2.3, it should also include the possible timing interferences between basic blocks.
- the constraints are the relations between the unit execution times. Some of them can be extracted from the control flow graph, others come from a preliminary flow analysis and express loop bounds, infeasible paths, etc.

Evaluating the WCET of the program comes to determining the numbers of execution of the basic blocks that maximize the objective function while meeting the constraints.

2.2. Timing interferences

As mentioned above, the expression of the program execution time should include inter-block timing interferences.

For very simple processors, such interferences are limited to adjacent blocks which overlap in the pipeline: the execution time of a two-block sequence is shorter than the sum of their respective execution times. In that case, all of the timing effects can be captured by measuring the execution times of blocks alone and of sequences of two blocks.

However, more advanced architectures make interferences between distant blocks possible, as it was shown by Engblom [3]. He has found that a block can interfere with a distant one, and this kind of interference is referred to as a *long timing effect* (LTE).

The execution time of a path can be computed as:

$$T = \sum_{i \in \mathcal{B}} t_i + \sum_{0 \le j < \ldots < k \le n} \delta_{j..k}$$

where $\mathcal{B}$ is the set of blocks (which are numbered from 0 to n), t_i is the execution time of block i and $\delta_{j..k}$ is the timing effect associated to the sequence of blocks $B_j \ldots B_k$. This is illustrated in Figure 1.

Sources of long timing effects include block alignment (*i.e.* the relation between the number of instructions in the block and the width of the pipeline) [12], long latency instructions, data dependencies, out-of-order execution, limited-capacity queues, etc.

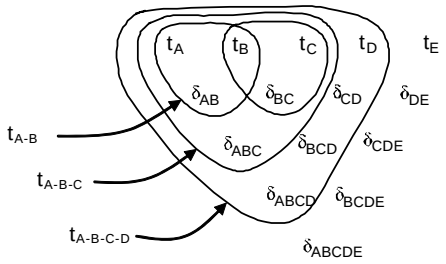


Figure 1. Engblom's timing model

Engblom has shown that long timing effects would span over unlimited block sequences: at the very worst, the first block of the program can affect the execution of the last block. Moreover, a long timing effect value ($\delta_{i\ldots j}$) can be negative as well as null or positive. A negative value should be taken into account to get a tight WCET estimation, but a positive value *must* be accounted for to compute a *safe* estimation.

2.3. Including timing interferences in WCET analysis

The original IPET method was developed considering very simple processor architectures where only adjacent basic blocks could interfere by overlapping in the pipeline. The corresponding gain was seen as the (negative) execution time of the edge linking the two blocks. Then, the edge execution time (weighted by the number of executions of the edge) was taken into account in the expression of the program execution time. For example, the WCET model for the control flow graph given in Figure 2 would have been:

$$\max T = x_A t_A + x_B t_B + x_C t_C + x_D t_D + x_E t_E + x_{AB}\delta_{AB} + x_{BC}\delta_{BC} + x_{CD}\delta_{CD} + x_{BE}\delta_{BE} + x_{ED}\delta_{ED}$$

$$1 = x_A = x_{AB} = x_B \qquad x_B = x_{BC} + x_{BE}$$
$$x_{BC} = x_C = x_{CD} \qquad x_{BE} = x_E = x_{ED}$$
$$x_D = x_{CD} + x_{ED} = 1$$

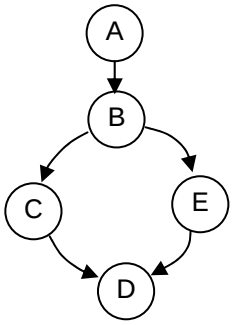


Figure 2. Example Control Flow Graph

Now, advanced processor architectures are often used for real-time systems and long timing

interferences should also be taken into account. We have found two different approaches in the literature.

The first one, described in [4], extends the original IPET model to include the long timing effects. For the example given in Figure 2, the model comes to:

$$\max T = x_A t_A + x_B t_B + x_C t_C + x_D t_D + x_E t_E + x_{AB}\delta_{AB} + x_{BC}\delta_{BC} + x_{CD}\delta_{CD} + x_{BE}\delta_{BE} + x_{ED}\delta_{ED} + x_{ABC}\delta_{ABC} + x_{BCD}\delta_{BCD} + x_{ABE}\delta_{ABE} + x_{BED}\delta_{BED} + x_{ABCD}\delta_{ABCD} + x_{ABED}\delta_{ABED}$$

$$1 = x_A = x_{AB} = x_B \qquad x_B = x_{BC} + x_{BE}$$
$$x_{BC} = x_C = x_{CD} \qquad x_{BE} = x_E = x_{ED}$$
$$x_D = x_{CD} + x_{ED} = 1$$
$$x_{ABC} \leq x_{AB} \qquad x_{ABC} \leq x_{BC} \qquad x_{ABC} \bullet x_{AB} - x_{BE}$$
$$x_{ABE} \leq x_{AB} \qquad x_{ABE} \leq x_{BE} \qquad x_{ABE} \bullet x_{AB} - x_{BC}$$
$$x_{BCD} = x_{BC} \qquad x_{BED} = x_{BE}$$
$$x_{ABCD} = x_{ABC} \qquad x_{ABED} = x_{ABE}$$

This solution weighs the expression of the objective function down and adds several constraints for each possible sequence in the execution path. Since LTEs can be as long as complete execution paths, the number of sequences to consider is potentially very high. Then the optimization problem might be very difficult to solve. Moreover, a value must be assigned to the LTE associated to each sequence of blocks: it must be computed from the execution times of all the sub-sequences. At the end, evaluating the LTE values comes to measuring every possible sequence of blocks, which is very time consuming in the general case.

Another approach consists in including the possible timing interferences in the execution times of blocks. When the target architecture can generate long timing effects, the execution time of a basic block should be evaluated by considering all the possible prefix paths and by keeping the highest value. While simulating numerous prefix paths could be unfeasible, the use of the abstract interpretation theory can make things more tractable [15]. The IPET model is then transformed as follows, where τ_A is the execution time of block A including the possible impact of other basic blocks:

$$\max T = x_A\tau_A + x_B\tau_B + x_C\tau_C + x_D\tau_D + x_E\tau_E$$

$$1 = x_A = x_B = x_C + x_E$$
$$x_D = x_C + x_E = 1$$

The algorithm for obtaining the adjusted unit execution times by abstract interpretation is not much detailed in papers, but it seems that it necessitates high computing power [14].

Moreover, including the effects of any possible prefix path in the execution time of a block leads to WCET overestimations since: (a) the flow analysis can find out that some prefix paths are infeasible, and (b) some prefix paths might not belong to the longest path and then should not be accounted for in the WCET. In the preceding example, τ_D includes the impact of block B on block D within the sequence BED (while δ_{BCD} might be null). If the flow analysis determines that block E is never executed and if δ_{BED} is positive, the WCET will be overestimated. Similarly, if the execution time of block C is far longer than that of block E, the longest path is along the path BCD and the impact of B on D in sequence BED should be ignored.

To sum up, the evaluation of unit execution times is costly in time for both approaches. The first solution also makes the IPET model more complex while the second one introduces some pessimism. These are the reasons why we are investigating solutions to limit timing interferences.

2.4. Related work

Li et al. [7] define a model based on dependence graphs to evaluate the execution time of a basic block in an out-of-order processor. However, they do not model superscalar execution and their experiments consider a very small core. Whether their model would scale to more realistic processors still has to be further investigated.

Heckmann et al. [5] use abstract interpretation to estimate the impact of previously executed blocks on the execution time of each basic block. This approach has been implemented in the aiT tool by the AbsInt company. While their method is an interesting alternative to exhaustive measurement (which is generally not affordable), each unit execution time includes the effects of all the possible prefix paths, which might result in WCET overestimations as shown in Section 2.3. Moreover, it seems that some pessimistic assumptions have sometimes to be taken to reduce the number of states. They might also lead to WCET overestimation.

In a recent work [13] we defined a processor pipeline where non-adjacent blocks cannot have timing interferences thanks to a fetch gating mechanism that enforces some distance between basic blocks in the pipeline. While this architecture makes the WCET easily computable by adding the execution times of the basic blocks among the possible execution paths, this solution does not solve today's problems since such an architecture does not exist yet.

As far as other parts of the processor are concerned (cache memories, branch predictor), guidelines to make their behaviour more predictable have also been proposed as an alternative to build too much complex models [2][11].

3. Code padding

3.1. General principle

The basic idea of the scheme proposed in this paper is close to the one that was behind our previous work [13]. To avoid long timing effects, basic blocks should not enter the pipeline one after the other: a certain distance should be enforced between them in such a way that the execution of a block cannot be disturbed by a previous block still in the pipeline. We suggest here that this distance could be enforced by the way of code padding, using neutral filler-instructions like NOPs. A *filler*-instruction is not executed and is removed from the pipeline after decoding. It does not require any other hardware resource than a slot in the fetch and decode stages. Some examples of filler-instructions in real processors will be given in Section 3.2.

The lengths of the code padding blocks have to be calculated by analysing the instructions belonging to basic blocks that might be executed consecutively and by determining their respective resource requirements. This analysis can be done by the compiler, and an algorithm is proposed in Section 3.3.

3.2. Neutral filler-instructions

To implement code padding, we need some instructions that use the fetch and decode stages to space out basic blocks, but are not executed (they should not consume computing resources) and not processed to the completion stage (otherwise, they might impact the execution time of the basic blocks). In this section, our purpose is to show that most instruction sets feature instructions that meet these constraints.

Most architectures have a NOP instruction that does not produce any result. In modern pipelines, NOP instructions are quashed from the pipeline after decoding in order to save the occupation of the functional units and the pipeline bandwidth.

Some processors have other instructions that do not go to the end of the pipeline. For example, on the PowerPC 750, fall-through branch instructions are removed from the instruction stream at dispatch. Then, an unconditional branch targetting the next instruction can be considered as a neutral instruction and used as a filler.

3.3. Code padding

The role of code padding is to avoid any possible interaction between distant blocks on an execution path. In the case where no long timing effect can occur, only the interferences between successive blocks are to be accounted for. Then the execution time of a sequence of *n* blocks can be computed as:

$$T = \sum_{i \in B} t_i + \sum_{0 \le j < n} \left(\sum_{s \in \mathcal{S}_j} \delta_{j,s} \right)$$

where $\mathcal{S}_j$ is the set of possible successors of block B_j.

A sufficient condition for this formula to be correct is that every possible sequence of two blocks executes exactly as if it was not preceded by other blocks in the pipeline. In this case, the LTE term — that normally stands for the distortion of the execution trace of the sequence by previous instructions — is null. This can be illustrated by the following example.

Let us consider three blocks A, B and C processing through a 3-stage pipeline with two non pipelined functional units, FU1 and FU2, that have a 3-cycle latency. Blocks A and C use FU1 while B uses FU2. The execution patterns are shown in Figure 3. The execution times of the blocks and the timing effects can be computed from these tables:

$t_A = t_B = t_C = 5$

$t_{AB} = 6 \quad \Rightarrow \quad \delta_{AB} = t_{AB} - t_A - t_B = 6 - 5 - 5 = -4$

$t_{BC} = 6 \quad \Rightarrow \quad \delta_{BC} = t_{BC} - t_B - t_C = 6 - 5 - 5 = -4$

$t_{ABC} = 8 \quad \Rightarrow \quad \delta_{ABC} = t_{ABC} - t_A - t_B - t_C - \delta_{AB} - \delta_{BC}$

$= 8 - 5 \times 3 - (-4) \times 2 = +1$

	1	2	3	4	5
FETCH	A				
FU1		A	A	A	
FU2					
COMPLETE					A

	1	2	3	4	5
FETCH	B				
FU1					
FU2		B	B	B	
COMPLETE					B

	1	2	3	4	5
FETCH	C				
FU1		C	C	C	
FU2					
COMPLETE					C

	1	2	3	4	5	6
FETCH	A	B				
FU1		A	A	A		
FU2			B	B	B	
COMPLETE					A	B

	1	2	3	4	5	6
FETCH	B	C				
FU1			C	C	C	
FU2		B	B	B		
COMPLETE					B	C

	1	2	3	4	5	6	7	8
FETCH	A	B	C					
FU1		A	A	A	C	C	C	
FU2			B	B	B			
COMPLETE					A	B		C

Figure 3. Execution of a 3-block sequence (example)

The *positive* LTE δ_{ABC} expresses that the execution pattern of the sequence B-C is distorted when it is preceded by block A, due to A and C conflicting for the use of FU1. This resource is free before the end of the fetch of B when it is executed alone, but it remains busy until two cycles after the end of the fetch of B when it is preceded by A.

Our purpose is to prevent this distortion and to make the sequence execute as shown in Figure 4. The approach consists in filling the black cell with a neutral instruction. Now, $\delta_{AB} = -3$ and $\delta_{ABC} = 0$.

	1	2	3	4	5	6	7	8
FETCH	A		B	C				
FU1		A	A	A	C	C	C	
FU2				B	B	B		
COMPLETE					A		B	C

Figure 4. Safe execution of a 3-block sequence (example cont'd)

Filler-instructions are added before a basic block to absorb any resource conflict that might occur with a previous block. As we will see in Section 4, the fetching of some basic blocks has to be delayed by several cycles if we want to prevent long timing effects. This means that these blocks should be preceded by a large number of neutral instructions, since a one-cycle delay is enforced by as many filler-instructions as the pipeline width.

To keep the code size acceptable, it is possible to group all the required filler-instructions into a common *padding block* that has multiple entry points and is terminated by a return branch (`blr`). Then every sequence of filler-instructions required to delay the fetch of a basic block can be implemented as a linked-branch (`bl`) to the appropriate entry point of the padding block. This is illustrated in Figure 5 where a 2-way pipeline is assumed. Note that the linked branch and the return branch each enforce a one-cycle delay.

4. Algorithm for code padding

To compute the padding lengths, the compiler first needs to collect timing information about the execution of sequences of basic blocks in the pipeline. Such information can be profiled by a cycle-level simulator of the processor that simulates blocks and up-to-*n*-block sequences (the simulation time is generally acceptable if *n* is small). Cycle-level simulation is required because precise dynamic information is needed to generate safe results. The simulation can be done within the compiler (provided it has an exact knowledge of the hardware) or by calling external software. Figure 6 gives an algorithm that analyses the resource needs of blocks and sequences: for each block B and for each resource R, it computes the time at which R is needed after B starts to be fetched (`n[R,B]`) and the time at which R is released by B after B has been completely fetched (`r[R,B]`). Release times are also derived for sequences: `r[R,S]` stands for the time at which resource R is available after sequence S has been entirely fetched.

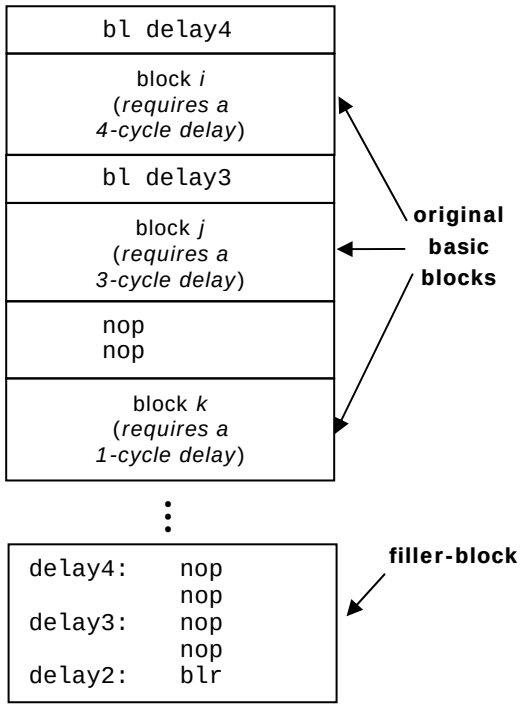


Figure 5. Padded basic blocks

```
foreach block B do {
   ff[B] ← first fetch cycle of B;
   lf[B] ← (last fetch cycle of B) + 1;
   foreach resource R do {
      n[R] ← cycle at which R is needed;
      r[R] ← cycle at which R is released;
               // 0 if R not used by B
      n[R,B] ← n[R] - ff[B];
      r[R,B] ← r[R] - lf[B];
   }
   d[B] ← 0;
}
foreach sequence B1-…-Bx (x < n) do {
   lf[Bx] ← (last fetch cycle of Bx) + 1;
   foreach resource R do {
      r[R] ← cycle at which R is released;
               // 0 if R not used by any Bi
      r[R,B1-…-Bx] ← r[R] - lf[Bx];
   }
}
```

Figure 6. Algorithm to analyse the resource requirements of blocks and sequences

4.1. *Depth-1* approach

As stated before, a long timing effect δ_{ABC} is not null if block A has an influence on how sequence B-C executes. On the contrary, δ_{ABC} is null if C executes after A-B exactly as after B. A *sufficient* — but not necessary — condition for this is that every resource (register, pipeline stage, functional unit, etc.) is released after A-B exactly at the same time as after B.

This assertion leads to the algorithm given in Figure 7. It analyses each two-block sequence to find out whether the first block has an impact on the availability of resources after the sequence. If so, the algorithm calls the `StrictDelay()` function that computes the distance *d* to put between the two blocks so that every resource is available after the sequence as soon as after the second block executed alone. Note that this distance is not always equal to the difference between `r[R,A-B]` and `r[R,B]`: it can be smaller but also larger due to timing anomalies, a phenomenon identified by Lundqvist [10]. For the moment, the `StrictDelay()` function computes the right distance by successive trials (the distance is upper-bounded by the size of the instruction window (fetch queue plus reoder buffer). A more clever algorithm based on execution graphs [7] is under development.

In the rest of this paper, this first algorithm will be referred to as the *depth-1* strategy.

```
foreach sequence A-B do {
   foreach resource R do {
      if r[R,A-B] > r[R,B] then {
         d ← StrictDelay(R,A-B);
         if d > d[B] then
            d[B] ← d;
      }
   }
}
```

Figure 7. *Depth-1* algorithm for computing the padding lengths

4.2. *Depth-n* strategy

The algorithm proposed in the previous section guarantees that every resource is available after sequence A-B exactly at the same time as after block B executed alone. This caution can be considered as excessive since the blocks executed after A-B might not require the resources delayed by A.

A more aggressive approach consists in examining the requirements of the possible successors of sequence A-B to determine whether a delay on the availability of a given resource induced by block A is likely to generate a long timing effect or not.

In the *depth-n* algorithm, the effective requirements of each basic block in every *n*-block sequence (B_0-B_1-…-B_{n-1}) are analyzed. Two kinds of situations necessitate that a distance is put between B_0 and B_1. The first case is when B_i uses a resource that is (a) not ready at the time B_i needs it, and (b) available later after B_0-…-B_{i-1} than after B_1-…-B_{i-1}. The second case is when the resource is ready on time for any block within the sequence but is released later after B_0-…-B_{n-1} than after B_1-…-B_{n-1}.

The analysis of the possible conflicts can be further refined for resources that can handle several instructions in parallel: they do not necessarily have to be completely free for blocks that use them but they should provide enough free slots to fulfil the needs.

In the case where a distance is to be enforced, the padding length is computed by the `MinimumDelay()` function (which implements the same approach as the `StrictDelay()` function). Figure 8 details the *depth-4* algorithm that analyses 5-block sequences and considers the exact requirements of the three last blocks to determine whether the second one has to be delayed after the first one.

```
foreach sequence A-B-C-D-E do {
   foreach resource R do {
      if n[R,C] > 0
         && r[R,A-B] > n[R,C]
         && r[R,A-B] > r[R,B] then {
         d ← MinimumDelay(R,A-B-C);
         if d > d[B] then
            d[B] ← d;
      }
      elsif n[R,D] > 0
            && r[R,A-B-C] > n[R,D]
            && r[R,A-B-C] > r[R,B-C] then {
         d ← MinimumDelay(R,A-B-C-D);
         if d > d[B] then
            d[B] ← d;
      }
      elsif n[R,E] > 0
            && r[R,A-B-C-D] > n[R,E]
            && r[R,A-B-C-D]> r[R,B-C-D] then {
         d ← MinimumDelay (R,A-B-C-D-E);
         if d > d[B] then
            d[B] ← d;
      }
      elsif
         r[R,A-B-C-D-E] > r[R,B-C-D-E] then {
         d ← StrictDelay(R,A-B-C-D-E);
         if d > d[B] then
            d[B] ← d;
      }
   }
}
```

Figure 8. *Depth-4* algorithm for computing the padding lengths

5. Performance results and discussion

5.1. Evaluation methodology

We have developed in SystemC a cycle-level simulator that models a generic processor architecture with parameterized features. The configuration we used for our tests is shown in Figure 9. The cache and the branch predictor are considered as perfect since modeling them is outside the scope of this work. The simulator is able to execute PowerPC code.

Pipeline width	**2-way**	**4-way**
fetch queue size	16	32
instruction cache	perfect (100% hit rate)	
branch predictor	perfect	
re-order buffer size	16	64
# of functional units (*latency*)		
integer add (*1 cycle*)	2	4
integer mul/div (*6 cycles*)	1	1
floating-point add (*3 cycles*)	1	1
fp mul (*6 cycles*)	1	1
fp div (*15 cycles*)	1	1
load/store (*2 cycles*)	2	2
data cache	perfect (100% hit rate)	

Figure 9. Simulated processor architecture

The results presented below were measured for several benchmarks commonly used in research on WCET analysis and presented in Figure 10 They implement standard algorithms: matrix arithmetic, signal processing, sorts.

`matmul`	matrix multiplication
`ludcmp`	LU decomposition
`jfdctint`	JPEG integer implementation of the forward Discrete Cosine Transform
`bsort`	bubble sort
`heapsort`	heap sort
`insertsort`	insert sort

Figure 10. Benchmarks

Figure 11 shows our framework for code padding. The object code is produced with the standard `gcc` compiler, targeted for the PowerPC instruction set. We have developed a utility that extracts the Control Flow Graph from the object code. The list of the basic blocks is used to drive the processor simulator which produces the execution trace of each block and of each possible sequence of up-to-5 blocks. The main tool of the chain is the *Interference Analysis* script that computes the padding lengths to eliminate any possible resource conflict between distant basic blocks. This script gets timing information from the simulator to compute the required delays down to the last cycle. Finally, the *Code Padding* script inserts the filler-instructions in the original assembly code.

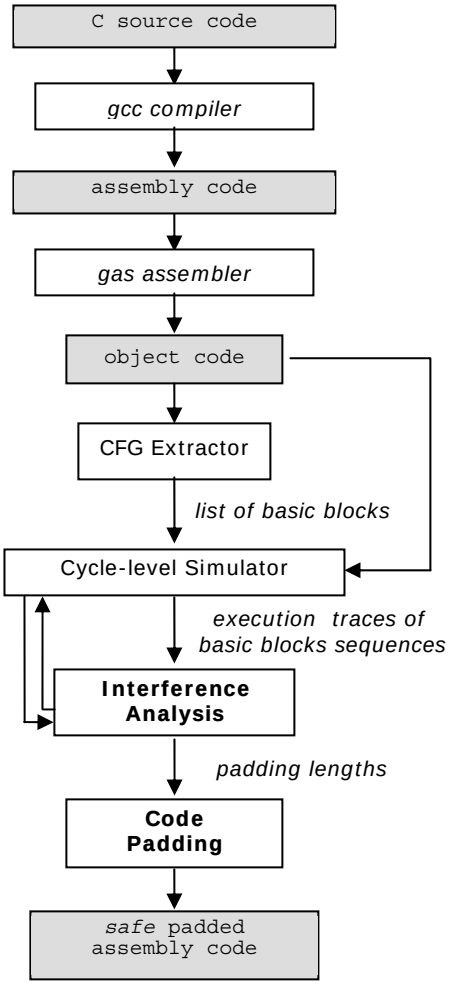


Figure 11. Code transformation framework

5.2. Impact of code padding on code size

Figure 12 shows the increase of the code size (number of static instructions) due to the filler-instructions added by the *depth-1* algorithm. The cost is undeniably sensible, especially for a 4-way target pipeline.

As shown in Figure 13, the increase is smaller when the analysis is done more in depth, *i.e.* when it takes into account the real requirements of the basic blocks. With the *depth-4* strategy, the mean increase is 18.28% for a 2-way pipeline, and 57.01% for a 4-way pipeline. We acknowledge that the increase is still noticeable but as we will discuss it in Section 9, we argue that it is the price of predictability.

	2-way	4-way
matmul	35.24%	76.19%
ludcmp	16.51%	28.20%
jfdctint	11.37%	126.97%
bsort	31.25%	76.25%
heapsort	25.00%	51.47%
insertsort	23.81%	59.52%
MEAN	***23.86%***	***69.77%***

Figure 12. Code size increase for the *depth-1* strategy

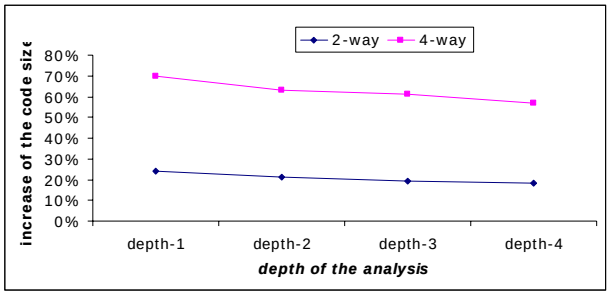


Figure 13. Code size increase as a function of the analysis depth

The cost in code size is higher for larger pipelines because (a) a single-cycle distance is enforced by as many NOPs as the pipeline width, and (b) a larger pipeline augments the overlapping of blocks and then augments the risks of resource conflicts. Results per benchmark are given in Figure 15.

Figure 14 gives further insight in how the increase is broken down into the length of the common padding block, the number of calls to this block and the number of NOPs added to implement the 1-cycle delays. The most severe increases in code size are due to the length of the padding block. For example, `jfdctint` requires a 649-instruction-long padding block while the original code has 519 instructions. This padding length is due to a 225-instruction-long basic block that seriously delays the availability of some resources.

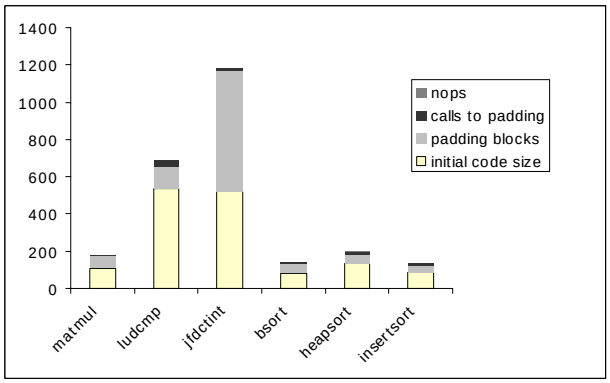
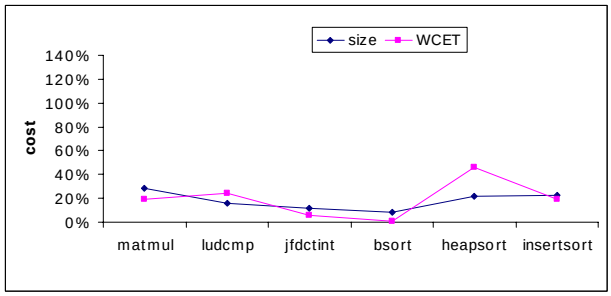
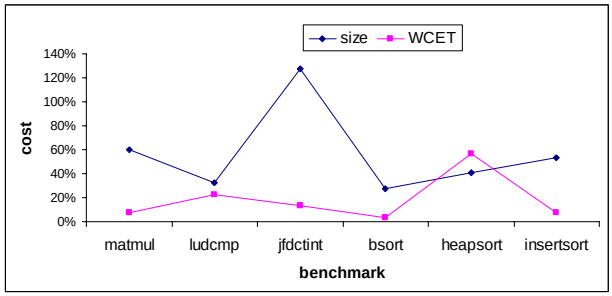


Figure 14. Breaking down of the code size increase (4-way pipeline, depth-4 policy)



(a) 2-way pipeline



(a) 4-way pipeline

Figure 15. Code size and WCET increase with the *depth-4* analysis

5.3. Impact on the real WCET

As said before, long timing effects can occur for long sequences of blocks (*i.e.* the execution of a basic block can have an impact on the execution of a very distant block). Measuring them involves analysing the execution traces of all the possible block sequences of any length which is very costly both in computing time and in memory requirements. This cost is generally unaffordable. However, we have analyzed up-to-6-block sequences and, for each of the benchmarks, we have observed some positive long timing effects (some of them spanning over 6-block sequences). This justifies the need for a solution to make the execution time predictable.

We have evaluated the real WCET of each benchmark code, without and with code padding. We used the symbolic execution method [9] that simulates every possible path. To make it possible, we have limited the size of the data so as to keep the number of possible paths reasonable. Figure 16 shows the results obtained with different analysis depths.

As expected, code padding, that enforces some delay between the execution of successive basic blocks and then limits the instruction parallelism, is responsible for an increase of the execution time. Note that the plotted time is the *real* WCET, not the estimated one (since we cannot make WCET estimations by static analysis when the target architecture generates long timing effects). Augmenting the depth of the analysis helps greatly in

limiting the cost in performance which comes to about 19% on average for the *depth-4* algorithm. This cost can be considered as moderate if we keep in mind that the WCET of the padded code can be estimated quickly, easily, tightly and, above all, safely.

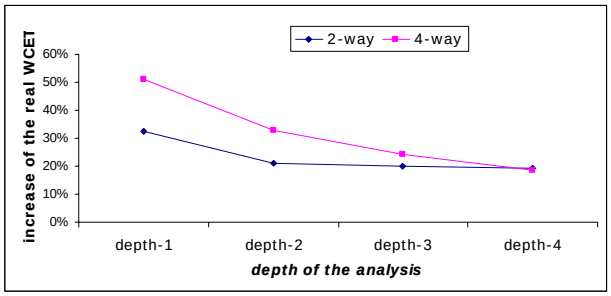


Figure 16. WCET increase as a function of the analysis depth

5.4. Discussion

When having to evaluate the WCET for a program that is to be run on a high-performance architecture, two strategies might be considered. The first one consists in using a method that takes into account any possible long timing effect (of any length and of any value). As far as we know, the only method doing that is the one implemented in the `aiT` tool by the AbsInt company. Its drawbacks include high computation times, complexity of the task of modeling the processor architecture (in the case where a new processor is targeted) and the use of pessimistic assumptions that might produce inaccurate WCET estimates.

The second possible strategy, which is the one we incline towards, aims to make the hardware/ software pair predictable. In [13], we proposed some modifications to the processor architecture to eliminate the possible interferences between distant blocks along the execution path. These modifications included two components: the first one prescheduled the instructions as they enter the reorder buffer; the second one acted as a gate that delays the fetch of a new basic block until it cannot be impacted by another block under execution in the pipeline. This scheme increased the mean execution time by 21% (2-way) to 42% (4-way).

The approach proposed here clearly has a lower cost in terms of execution time: it is smaller by one third for a 2-way processor (19.4% against 21%) and by more than one half for a 4-way processor (18.5% against 42%). This is because we compute the distance required between successive basic blocks off-line. Then we know exactly which instructions belong to the blocks and we exploit profiling information to identify the data and resource dependencies that result in timing interferences. On the contrary, the runtime mechanism proposed in [9] does not know anything about a block that is to be fetched. Then, it has to make pessimistic assumptions and it enforces unnecessarily long distances between the basic blocks.

Moreover, our solution does not need any particular hardware and only requires that a free instruction is available in the instruction set. As mentioned in Section 3.2, such an instruction exists in most processors. Then the code padding method can be used immediately (*i.e.* without waiting that a processor manufacturer decides to design a processor compatible with safe WCET evaluation). The required effort is moderate since the code transformation is done at the assembly level.

Code padding has a cost both in code size and in execution time. However, if we want to keep the evaluation of the WCET simple, the only alternative is to use simpler processors (scalar, with in-order instruction scheduling, etc.) that were proved to be LTE-free. However, they might not meet the performance requirements.

6. Conclusion

This paper deals with timing interferences that make the evaluation of the WCET of a task complicated, pessimistic and possibly unsafe. This problem has already been addressed in a previous work where a processor was designed to prevent timing interferences between basic blocks while keeping most part of the performance. However, the proposed solution has not yet been implemented in a real-life processor. Our purpose is to show how the prevention of timing interferences can be done by transforming the source code, which does not require any specific hardware.

Our approach consists in profiling the execution of basic blocks and of *n*-blocks sequences extracted from the Control Flow Graph of the application. This can be done using a cycle-level simulator and is much faster than simulating all the possible execution paths. Execution profiles are then analyzed to detect data dependencies and resource conflicts that could generate interferences between distant blocks. Filler-instructions, which are discarded from the pipeline as soon as they have been decoded, are inserted in the source code to enforce a distance between blocks so that the interferences are eliminated.

Performance analysis show that, even if the number of added padding instructions is significant, the impact on the worst-case execution time is moderate (a mean slowdown of 19% has been measured).

The increase in the code size and in the real WCET is sensible but this is the cost to pay for predictability, and thus for safety. The WCET of padded codes can be evaluated accurately using simple state-of-the-art methods.

References

[1] A. Anantaraman, K. Seth, K. Patil, E. Rotenberg, F. Mueller, "Virtual Simple Architecture (VISA): Exceeding the Complexity Limit in Safe Real-time Systems", *30th Int. Symp. on Computer Architecture*, may 2003.

[2] F. Bodin, I. Puaut, "A WCET-oriented static branch prediction scheme for real-time systems", *17th Euromicro Conf. on Real-Time Systems*, july 2005.

[3] J. Engblom, *Processor pipelines and static worst-case execution time analysis*, PhD thesis, Uppsala University, april 2002.

[4] A. Ermedahl, *A Modular Tool Architecture for Worst-Case Execution Time Analysis*, PhD thesis, Uppsala University, june 2003.

[5] R. Heckmann, M. Langenbach, S. Thesing, R. Wilhelm, "The influence of processor architecture on the design and the results of WCET tools", *Proceedings of the IEEE*, vol. 91, n°7, july 2003.

[6] Y.-T. Li, S. Malik, "Performance Analysis of Embedded Software using Implicit Path Enumeration", *ACM SIGPLAN Notices*, vol. 30, n°11, 1995.

[7] X. Li, A. Roychoudhury, T. Mitra, "Modeling Out-Of-Order Processors for Software Timing Analysis", *IEEE Real-Time Systems Symposium*, december 2004.

[8] S.-S. Lim, S. Min, M. Lee, C. Park, H. Shin, C. S. Kim, "An Accurate Instruction Cache Analysis Technique for Real-Time Systems", *Workshop on Architectures for Real-Time Applications*, 1994.

[9] T. Lundqvist, P. Stenström, "An Integrated Path and Timing Analysis Method based on Cycle-Level Symbolic Execution", *Real-Time Systems*, 17(2), 1999.

[10] T. Lundqvist, P. Stenström, "Timing Anomalies in Dynamically Scheduled Processors," *IEEE Real-Time System Symposium (RTSS'99)*, december 1999.

[11] I. Puaut, D. Decotigny, "Low-Complexity Algorithms for Static Cache Locking in Multitasking Hard Real-Time Systems", *23rd Int. Real-Time Systems Symp.*, december 2002.

[12] C. Rochange, P. Sainrat, "Towards Designing WCET-predictable Processors", *3rd Workshop on Worst-Case Execution Time Analysis*, june 2003.

[13] C. Rochange, P. Sainrat, "A Time-Predictable Execution Mode for Superscalar Pipelines with Instruction Prescheduling", *ACM International Conference on Computing Frontiers*, may 2005.

[14] J. Souyris, E. Le Pavec, G. Himbert, V. Jegu, G. Borios, R. Heckmann, "Computing the Worst-Case Execution Time of an Avionics Program by Abstract Interpretation", *5th Workshop on WCET Analysis*, july 2005.

[15] H. Theiling, C. Ferdinand", Combining Abstract Interpretation and ILP for Microarchitecture Modelling and Program Path Analysis", *IEEE Real-Time Systems Symposium*, december 1998.

[16] R. Wilhelm, J. Engblom, S. Thesing, D. Whalley, "Industrial Requirements for WCET Tools", *3rd Workshop on WCET Analysis*, june 2003.

A Distributed and Verifiable Loop Bounding Algorithm for WCET Computation on Constrained Real-Time Embedded Systems

Nadia Bel Hadj Aissa, Gilles Grimaud, David Simplot-Ryl
IRCICA/LIFL, Univ. Lille 1, UMR CNRS 8022
INRIA Futurs, POPS research group*
{aissa, grimaud, simplot}@lifl.fr

Abstract

Most of classical WCET techniques rely on the fact that all the system is known during conception phase, i.e. hardware and software parts. In the context of mobile code for small devices like smartcards or RFID tags, these assumptions cannot be true because of heterogeneous hardware and unknown software environment. On the other hand, these small devices have not enough computation power to compute themselves the WCET of loaded applications. In this paper, we propose a distributed method which allows to generate a portable WCET pre-computation, including automatic loop detection, which is given with the mobile code. This pre-computation – which is automatic and do not use annotations – is verified by the mobile host by using a lightweight proof which is embedded in the mobile in PCC manner. We present experimental results by applying our method on the kernel of a smartcard dedicated operating system that proves the validity of the proposed method.

1. Introduction

The advent of pervasive mobile devices (***e.g.*** smart phones, smart cards, sensors, RFID tags...) emphasized the necessity for hardware manufacturers to increase the number of produced units at a constant cost rather than increase their performance. In fact, increasing clock speeds is not the answer for battery-operated devices where lowering memory footprint, power consumption, and cost is a main issue. It follows that, for these technologies, the constraints on memory size and computing power are durably established facts.

On the other hand, the recent interest for the mobile code paradigm challenges the traditional infrastructure models and implies the ability for the constrained devices to load code from outside dynamically. Mobile code can be represented by machine code, allowing maximum execution speed on the target machine but thereby sacrificing platform independence. Alternatively, the code can be written in a portable bytecode fashion (*"Write Once Run Anywhere"*), thus offering platform independence. The code will be, then, interpreted by a virtual machine or compiled by a JIT compiler to obtain native code. Generally, the bytecode is produced on traditional data-processing supports (***i.e.*** code producer) before being deployed on the constrained devices (***i.e.*** code consumer). These new features bring considerable flexibility to the device in order to satisfy customers evolving expectations and needs and make it possible to handle the stringent hardware constraints that characterizes mobile device technology.

However, the choice of mobile code paradigm raises major security issues. The code consumer may download an untrusted program created by a code producer, who is possibly badly disposed. This mistrust relationship between the code supplier and the code recipient leads to the inception of proof-based security approaches, for instance. In these approaches [11, 12], a proof created at compile time by the code supplier, is packaged with untrusted code. By a straightforward inspection of the code and the certificate [1], the code consumer can verify the validity of the proof and thus the compliance with a safety policy.

Indeed, a code consumer should be free to reject code that may threaten his system and does not adhere to a particular safety policy. In general, a safety policy needs to address the concerns of confidentiality, integrity and availability. The former issues were thoroughly studied. The availability criterion involving resource-related issues (***e.g.*** ensuring that the program will not compute for more than a given amount of time, or that it will not take up an amount of computing power or memory above a certain threshold) is often neglected.

In this paper, we address a scenario where the code consumer needs to ensure the execution of a dynamically loaded application within strict timing requirements. Predicting the timing behavior of a mobile code (***e.g.*** the worst case execution time, WCET) allows allocating correctly the computational resources between the different

*This work is partially supported by grants from the CPER Nord-Pas-de-Calais/FEDER TAC COM'DOM, the European IST-FP6 INSPIRED project.

competing tasks in the underlying operating system and makes it possible to prevent availability attacks. Indeed, by misleading the WCET computation algorithm, malicious programmers could intentionally minimize the processor time consumption of their programs to launch a denial-of-service attack. It also improves the deployment scheme as the code recipient would rather know in advance if the downloaded application will definitely run within the amount of CPU available on the system and meet its deadline.

The existing timing analysis methods are executed atomically in one execution site where hardware architecture, runtime environment, and compilation or interpretation strategy are known in advance. In a mobile code execution scenario, the producer cannot unilaterally determine the CPU needs of a program because it depends closely on the number of CPU cycles consumed by the target processor. In addition, the memory and CPU consumptions of existing algorithms quickly increase with the complexity of the program. This means that it is not realistic to shift the burden of computing the WCET of dynamically loaded application on the consumer (***i.e.*** constrained device).

Thus, we proposed in [2] to split the WCET computation process in two phases. The first step is executed when the source code is compiled on the producer. Then, the computation process had to be finalized when the code is deployed on the host system. The challenges are to distribute the computation efficiently to do the heaviest operations on the producer and to endow the consumer with the ability to check the safety of the information inferred by untrusted parties.

As far as worst case execution time computation is concerned, evaluating loop bounds represents a critical issue and is the major source of timing unpredictability. We propose in this paper a novel scheme for safely computing loop bounds on constrained devices by using a PCC based approach. On the producer side, the intermediate code is statically analyzed to extract loop bounds that will be used for timing analysis. We chose to determine loop bounds by proceeding in a manner that is similar to standard type derivation. An inference engine scans the instructions and tracks the values taken by the program variables. According to the transition rules, a variable can be a constant, an open range of values, or a loop index. This process is repeated iteratively until a fix-point is reached. Then, the state types are stored for the entry point to each basic block which constitutes the proof. The WCET proof is sent to the consumer with the intermediate code. The target system needs a single pass to check the consistency of the loop bounds inferred by the producer with the code. An on-the-fly compiler produces the native code corresponding to the underlying platform. The WCET computation process is finalized and a global WCET value is calculated by the target system.

The rest of the paper proceeds as follows. First, we present some working hypothesis that should define preliminary assumptions for our work. We describe our loop bounds detection tool, followed by some examples that illustrate how the types are derived statically on the code producer. The linear verification process is then explained through the same examples. Some experimental results of our work on the kernel of *Camille* [5] operating system are presented. Finally, we consider some future work.

2. Preliminaries

Bounding the number of loop iterations is a well-founded research area. In the general case, it is impossible for an automatic analysis to determine whether a given loop will execute a definite number of times. This is one obvious consequence of the undecidability of the *Halting Problem*. Some work has already been achieved to predict loop bounds automatically by Healy et al. [9]. Their approach, based on classic control-flow analysis, uses block dominance and loop frontiers and can give quite tight predictions of the number of iterations in loops with integer indexes. Gustafsson [8] used abstract interpretation [4] to automatically determine loop bounds and false paths.

Note that our work is basically different. Indeed, one of our goals shall be to detect loops statically on the producer side. However, the main difference with the approaches cited previously is that we must provide a way to the consumer to verify the loop bounds inferred by untrusted code suppliers. The verification overhead should be less important than the effort due to detect loop bounds in the first place. Therefore, the code recipient needs only to trust its own loop bounds checker. It also should induce, if the method is to be effective, a linear verification effort much simpler than the tools required to analyze statically the code to extract loop bounds.

Since not all the loops have a predictable behavior, suitable for our analysis, we present, in the following, some preliminary assumptions on the recognizable loop patterns. Then, we introduce the precedence relation defined on the basic blocks involved in these patterns. This relation will be used by our typing algorithm presented in the following section.

2.1. Recognizable loop patterns

The loops that are candidates for our static analysis follow the execution pattern[1] explained below :

1. *Initialization step:* contains loop initialization code that should execute unconditionally. The loop counter variable is explicitly set up to a starting value that represents the lower bound of the loop execution count.

2. *Counter update step:* contains the instructions that are intended to perform loop control variable updates (***e.g.*** incrementation).

[1] An execution pattern defines the sequence of steps taken during the visit of the loop.

3. *Condition evaluation step:* contains the evaluation of the loop termination condition. The loop counter variable is compared to the upper bound of the loop execution count. If the termination condition is fulfilled, the loop is stopped, else go to the loop header and repeat.

Note : We consider only the case where the lower bound, step and upper bound values are expressed as constants and can be calculated at analysis time. The loop condition can also be evaluated at the beginning of the repeating section of code. In this case, if the loop condition is true an iteration is carried out else the loop is terminated.

We are aware that not all the loops fit in this simple pattern. However, we chose to focus on this kind of loop constructs based on the observation that many loop constructs iterate a fixed number of times and do some simple work every iteration and have a single target of the loop exit. In [13], an inspection of industrial code leads to the following conclusions : 94% of the analysed loops had a single target of the loop exit and can be syntactically bound, only a few loops actually depend on outer loops and should be simple to bound. In addition, The evaluation of the RTEMS operating system source code in [3] shows that the program constructs were quite simple. No nested loops, unstructured code or recursion were found.

2.2. Precedence Relation

To detect loop bounds, we need to identify each step and transition in the execution pattern. Figure 1 sketches the control flow graphs of a pre-test and a post-test loop corresponding to the patterns described previously.

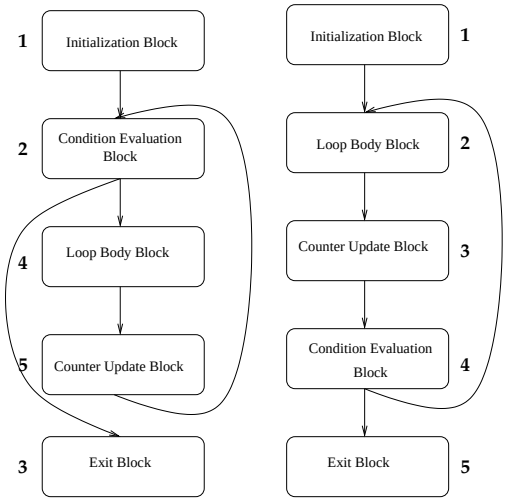


Figure 1. Pre-test and Post-test loops

The control flow of a given program is usually depicted as a directed graph $G(V, E)$, where V is the set of basic blocks of code and an edge $(x, y) \in E$ represents a possible flow of control from x to y. We define a precedence relation $\prec$ between basic blocks denoting their execution order in the loop traversal. Let x and y be two vertices of the control flow graph. We have $x \prec y$ if and only if x always happens before y in all execution paths. For the sake of simplicity and for implementation reasons, we use an label function $order$ which gives a number to each node such that: $\forall x, y \in V \quad x \prec y \Rightarrow order[x] < order[y]$. This labeling function can be assimilated to a depth-first topological order of a graph where the nodes are ordered sequentially. Indeed, a depth-first traversal of a graph visits all the nodes marking them as they are visited. The number of the last descendant of each node is also saved, thus, enabling an efficient test of precedence in the Depth First Traversal Tree.

In Figure 1, the ordering of the different basic blocks indicates that in the case of a pre-test loop the identified steps will execute consecutively as follows : Initialization Step (**1**) – Condition evaluation step (**2**) – Counter update step (**5**). On the other hand, for the post-test loop, the loop execution pattern will be formed in sequence by : Initialization Step (**1**) – Counter update step (**3**) – Condition evaluation step (**4**).

3. Loop Bounds Detection on the Code Producer

The set of rules that we propose in this work aims at guaranteeing that the loop bounds inferred statically on the code producer will be respected at run-time on the code recipient. Static analysis of compiled-programs provides information about expected program behavior in order to minimize dynamic checks and so runtime overhead. Mainly, it is considered necessary in our context to notify, as soon as possible at the moment of installation of the downloaded program, any possible run-time error. Traditionally, static analysis is associated with Abstract Interpretation [4]. Recently, however, much interest has been shown in the potential of type inference as a means of performing static analysis as well as ensuring program correctness on compiled code when the application is deployed on the target. Our approach is casted as a type inference problem where types are used to express loop bounds. In the following, we describe the formalization of our type system.

3.1. The Instruction Set

The instruction set, used in the remainder of this paper, is a finite set of elementary arithmetic operations and basic control-flow operations as illustrated in Figure 2. A program P is a sequence of instructions where each instruction is referred to with an instruction counter, *pp*. When *P* is a program, we write *Dom(P)* for the domain of *P* (its set of program counters); $P[pp]$ is defined only for $pp \in Dom(P)$.

Note : We denote by *Var* the set of the program variables, *Cst*, the set of the constants, and *Labels*, the set of the instructions targeted by a branch operation. *v* ranges over *Var*, *c* ranges over *Cst* and L_i ranges over *Labels*. cmp ranges over $\{<, \leq, >, \geq\}$. The instruction 4 in Fig-

Instructions	I ::=	v $\leftarrow$ c	(1)
		v $\leftarrow$ v + c	(2)
		b $\leftarrow$ v cmp c	(3)
		vd $\leftarrow$ vs op tabArgs	(4)
		Jump L_i	(5)
		Jumpif v, L_i	(6)
		Return v	(7)
Program	P ::=	PI | I	

Figure 2. Instruction Set

ure 2 represents any operation or method invocation on *vs* with the list of arguments *tabArgs*. The *Return* instruction is used to terminate the instruction flow and to restore the context of the caller.

3.2. A type system for loop bounds detection

A variable in a given program can have different types as illustrated in Figure 3. If a variable has an irrelevant state for loop bounds detection, its type is set to $\top$. A variable can also be a constant value, an open range of integer values (where α is the lower bound of the range and σ is the incrementation step), a conditional (where var is compared to a constant value ψ using an operator cmp) or a loop iterator (where α is the lower bound of the range, ψ the upper bound and σ the incrementation step). We consider a distinction between a possible iterator and a confirmed one. The flags ? and ! indicate a possible and a confirmed iterator, respectively. A possible iterator is obtained when a variable is consecutively initialized, incremented and compared to an upper bound by a single execution path. A confirmed iterator is obtained when the variable is consecutively initialized, incremented and compared to an upper bound by all the execution paths.

The types are tagged with an order number e except the $\top$ type. This ordering number makes it possible to figure out, at which step of the loop execution pattern, the type has been created. Thus, it becomes possible to be ensured of the precedence relationship between the specific basic blocks identified in Figure 1. For this purpose, we define a function χ that attributes an ordering number to each instruction. When a new type T is created at an instruction i, the function χ determines the ordering number of the basic block to which the instruction belongs ($\chi(i) = e$). Then, the newly created type is stamped with the resulting value (T_e).

Types	T ::=	$\top$	Irrelevant
		$\{\alpha\}_e$	Constant value
		$[\alpha \stackrel{\sigma}{:} [_e$	Open range
		$(var, cmp, \psi)_e$	Conditional
		$[\alpha \stackrel{\sigma}{:} \psi]^?_e$	Possible iterator
		$[\alpha \stackrel{\sigma}{:} \psi]^!_e$	Confirmed iterator

Figure 3. Type Syntax

3.3. Operational semantics

We model a state of an execution as a tuple $\langle pp, T\rangle$, where pp is the program counter, T is the current state of program variables, represented by a total map from the set of variables to the set of values. The execution of each instruction except "return", changes the state of execution of the program P from state $\langle pp, T\rangle$ to state $\langle pp', T'\rangle$. The operational semantics, illustrated in Figure 4, describe how the evaluation of instructions affects the program state and informally behave as follows:

Rule 1: The evaluated instruction is an assignment to a constant value. The type of v is changed on $\{\alpha\}_e$ with e the ordering number of the basic block that includes the assignment instruction. The next instruction to be evaluated corresponds to the valid program point $pp + 1$.

Rule 2: The instruction at the current program point is the result of the evaluation of a comparison of a variable v and a constant ψ. The type of the variable b is set to $(var, cmp, \psi)_e$ with e the ordering number of the basic block that includes the comparison instruction. The next instruction to be evaluated corresponds to the valid program point $pp + 1$.

Rule 3: The evaluated instruction corresponds to the incrementation of a variable v by a constant step σ. If the variable v has been already initialized (***i.e.*** its current type is $\{\alpha\}_e$), its type is changed to $[\alpha \stackrel{\sigma}{:} [_{e'}$ with e' the ordering number of the basic block that includes the incrementation instruction. The next instruction to be evaluated corresponds to the valid program point $pp + 1$.

Rule 4: If the variable v has been already initialized, incremented and compared to an upper bound (***i.e.*** its current type is $[\alpha \stackrel{\sigma}{:} \psi]^?_e$), the re-evaluation of the incrementation instruction ensures that the variable v is a confirmed iterator by this path and its type can be changed to $[\alpha \stackrel{\sigma}{:} \psi]^!_{e'}$. e' denotes the ordering number of the basic block that includes the incrementation instruction. The next instruction to be evaluated corresponds to the valid program point $pp + 1$.

Rule 5: In the case of a post-test loop pattern, the variable b contains the evaluation of the comparison between a variable v of the program and a constant value with the comparison operator ranging over $\{<, \leq\}$. The variable v is an open interval $[\alpha \stackrel{\sigma}{:} [_{e'}$. The evaluation of the conditional branch instruction induces changes at two points of the program. The first corresponds to the fulfillment of the branching condition and can be reached by the given $LabelId$. If a variable v was initialized, incremented and compared to an upper bound, then its type can be changed to $[\alpha \stackrel{\sigma}{:} \psi]^?_{e''}$ with e'' the ordering number of the basic block labeled by $LabelId$. The second program point corresponds to the failure of the condition when the program

implicitly branches to $pp + 1$. It propagates irrelevant information for loop bounding and the types of all the variables is set to $\top$ at this program point.

Rule 5': In the case of a pre-test loop pattern, the variable b contains the evaluation of the comparison between a variable v of the program and a constant value with the comparison operator ranging over $\{>, \geq\}$. The variable v is an open interval $[\alpha \stackrel{\sigma}{..}[_{e'}$. The evaluation of the conditional branch instruction induces changes at two points of the program. The first corresponds to the fulfillment of the branching condition and can be reached by the given $LabelId$. The variable v exceeds the upper bound of the loop, the types of all the variables will be set to $\top$ at the basic block labeled by $LabelId$. The second program point $(pp + 1)$ corresponds to the failure of the condition and indicates that the loop iterates one more time. Thus, the type of v can be changed to $[\alpha \stackrel{\sigma}{..} \psi]^{?}_{e''}$ with e'' the ordering number of the basic block labeled by $pp + 1$.

Rule 6: This rule corresponds to the evaluation of any method invocation different from those listed previously. It produces an irrelevant value for our analysis, thus, changing the type of the destination variable to $\top$.

3.4. Unification Rules

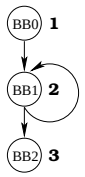
Some instructions may have multiple preceding paths of execution and the types constructed on these paths have to be merged. This can only occur at the targets of jumps corresponding to the entries of basic blocks. Therefore, a set of unification rules must be written and applied when an unconditional or a conditional branch instruction is evaluated. In classic typing systems, a hierarchy of types, represented by a partially ordered set of classes, is often used for the unification operation. In our type system, the precedence relation defined between the types raises new issues. The unification rules must handle the precedence relation that exists between the different types. If we consider v a variable of our program, the current type of $T[v]_a$, and $T[v]_b$ the new type that is created after the evaluation of an unconditional or a conditional branch instruction. When a is inferior to b, it means that the type tagged by a has been created in a step that precedes the type computed in b. The Table 1 shows the different unification rules depending on the types that have to be merged. For lack of space, we do not put the unification rules that involves the conditional type. If a conditional type $(var, cmp, \psi)_a$ is unified with $(var, cmp, \lambda)_b$, the resulting type is $(var, cmp, \min(\psi, \lambda))_b$. The unification of a conditional and any other type gives an irrelevant information and the resulting type is therefore set to $\top$.

3.5. Examples

To illustrate the type derivation procedure explained previously, consider the code samples in Figure 5, 6 and 7. Each sample program consists of a sequence of instructions that belongs to the instructions set, defined in 3.1. A control flow graph annotated with the precedence order of each basic block is joined to each fragment of code.

Sample Program 1 Consider the sample program in Figure 5 that corresponds to a post-test loop pattern. The program P uses two variables i and tmp to iterate 10 times. As explained in 2.1, the loop pattern should consist of the consecutive execution of an initialization step, a counter update step, and a condition evaluation step. This loop pattern will allow us to detect an iterator.

BB	pp	Label	Instruction
0	1		i ← 0
1	2	L1	i ← i + 1
	3		tmp ← i ≤ 9
	4		jumpif tmp L1
2	5		return i



S	pp	T[i,tmp]	pp'	T'[i,tmp]
0	∅	$\emptyset$, $\emptyset$	1	$\top$, $\top$
1	1	$\top$, $\top$	2	$\{0\}_1$, $\top$
2	2	$\{0\}_1$, $\top$	3	$\left[0 \stackrel{1}{..}\right[_2$, $\top$
3	3	$\left[0 \stackrel{1}{..}\right[_2$, $\top$	4	$\left[0 \stackrel{1}{..}\right[_2$, $(i, \leq, 9)_2$
4	4	$\left[0 \stackrel{1}{..}\right[_2$, $(i, \leq, 9)_2$	2	$\left[0 \stackrel{1}{..} 9\right]^{?}_2$, $\top$
			5	$\top$, $\top$
5	2	$\left[0 \stackrel{1}{..} 9\right]^{?}_2$, $\top$	3	$\left[0 \stackrel{1}{..} 9\right]^{!}_2$, $(i, \leq, 9)_2$
6	3	$\left[0 \stackrel{1}{..} 9\right]^{!}_2$, $(i, \leq, 9)_2$	4	$\left[0 \stackrel{1}{..} 9\right]^{!}_2$, $(i, \leq, 9)_2$
7	4	$\left[0 \stackrel{1}{..} 9\right]^{!}_2$, $(i, \leq, 9)_2$	2	$\left[0 \stackrel{1}{..} 9\right]^{!}_2$, $\top$
			5	$\top$, $\top$
8	2	$\left[0 \stackrel{1}{..} 9\right]^{!}_2$, $\top$	3	$\left[0 \stackrel{1}{..} 9\right]^{!}_2$, $(i, \leq, 9)_2$
9	5	$\top$, $\top$	∅	$\emptyset$, $\emptyset$

Figure 5. Listing of Sample Program 1, CFG and Execution trace of type inference procedure

Figure 5 shows the execution trace of the type derivation algorithm. From left to right, the columns indicate the analysis step, the current program counter, the current types of the local variables, the program counter of the next instruction to be evaluated and finally the types of the local variables after the evaluation of the current instruction.

Execution starts by initializing both the types of variables i and tmp to $\top$. At step 1, the first instruction assigns a constant value to the variable i. The type of i is then tagged with the precedence order of the corresponding basic block and becomes $\{0\}_1$. The type of tmp remains unchanged. These new types are transmitted to the next instruction to be evaluated which is indicated by the program counter 2. At step 2, the incrementation of i changes its type from $\{0\}_1$ to $\left[0 \stackrel{1}{..}\right[_2$. At step 3, the instruction 3 is evaluated. The variable tmp contains the result of the comparison between i and a constant value. Its type is set to a conditional on the variable i.

At step 4, the evaluated instruction is a conditional branch. According to our typing rules, the types of the

$$\frac{P[pp] = v \leftarrow \alpha \quad \alpha \in Cst \quad (pp+1) \in Dom(P)}{\langle pp, T\rangle \mapsto \langle pp+1, T'\rangle \text{ where } T'[v] = \{\alpha\}_e}$$
– Rule 1 –

$$\frac{P[pp] = b \leftarrow v \; cmp \; \sigma \quad cmp \in \{<, \leq, >, \geq\}, \sigma \in Cst \quad (pp+1) \in Dom(P)}{\langle pp, T\rangle \mapsto \langle pp+1, T'\rangle \text{ where } T'[b] = (var, cmp, \psi)_e}$$
– Rule 2 –

$$\frac{P[pp] = v \leftarrow v + \sigma \quad \sigma \in Cst \quad T[v] = \{\alpha\}_e \quad (pp+1) \in Dom(P) \quad \chi(pp) = e'; e' \geq e}{\langle pp, T\rangle \mapsto \langle pp+1, T'\rangle \text{ where } T'[v] = [\alpha \mathbin{\dot{..}} [_{e'}}$$
– Rule 3 –

$$\frac{P[pp] = v \leftarrow v + \sigma \quad \sigma \in Cst \quad T[v] = [\alpha \mathbin{\dot{..}} \psi]^?_e \quad (pp+1) \in Dom(P) \quad \chi(pp) = e'; e' \geq e}{\langle pp, T\rangle \mapsto \langle pp+1, T'\rangle \text{ where } T'[v] = [\alpha \mathbin{\dot{..}} \psi]^!_{e'}}$$
– Rule 4 –

$$\frac{P[pp] = Jumpif\ b, LabelId \quad T[b] = (var, cmp, \psi)_e, \text{cmp} \in \{<, \leq\} \quad T[v] = [\alpha \mathbin{\dot{..}} [_{e'} \quad (pp+1), LabelId \in Dom(P) \quad \chi(LabelId) = e''; e'' \geq e' \geq e}{\langle pp, T\rangle \mapsto \langle pp+1, T'\rangle \text{ where } \forall\, v \in Var, T'[v] = \top \quad \langle LabelId, T'\rangle \text{ where } T'[v] = [\alpha \mathbin{\dot{..}} \psi]^?_{e''}}$$
– Rule 5 –

$$\frac{P[pp] = Jumpif\ b, LabelId \quad T[b] = (var, cmp, \psi)_e, \text{cmp} \in \{>, \geq\} \quad T[v] = [\alpha \mathbin{\dot{..}} [_{e'} \quad (pp+1), LabelId \in Dom(P) \quad \chi(pp+1) = e''; e'' \geq e' \geq e}{\langle pp, T\rangle \mapsto \langle LabelId, T'\rangle \text{ where } \forall\, v \in Var, T'[v] = \top \quad \langle pp+1, T'\rangle \text{ where } T'[v] = [\alpha \mathbin{\dot{..}} \psi]^?_{e''}}$$
– Rule 5' –

$$\frac{P[pp] = vd \leftarrow vs \; op \; tabVar}{\langle pp, T\rangle \mapsto \langle pp+1, T'\rangle \text{ where } T'[vd] = \top}$$
– Rule 6 –

Figure 4. Operational Semantics

a \ b	$\top$	$\{\alpha\}_b$	$[\alpha \mathbin{\dot{..}} [_b$	$[\alpha \mathbin{\dot{..}} \psi]^?_b$	$[\alpha \mathbin{\dot{..}} \psi]^!_b$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\{\beta\}_a$	$\top$	$\{\min(\alpha,\beta)\}_b$	$[\min(\alpha,\beta) \mathbin{\dot{..}} [_b$	$[\alpha \mathbin{\dot{..}} \psi]^?_b$	$[\min(\alpha,\beta) \mathbin{\dot{..}} \psi]^!_b$
$[\beta \mathbin{\dot{..}} [_a$	$\top$	$\{\alpha\}_b$	$[\min(\alpha,\beta) \overset{\min(\sigma,\phi)}{\dot{..}} [_b$	$[\min(\alpha,\beta) \overset{\min(\sigma,\phi)}{\dot{..}} \psi]^?_b$	$[\min(\alpha,\beta) \overset{\min(\sigma,\phi)}{\dot{..}} \psi]^!_b$
$[\beta \mathbin{\dot{..}} \gamma]^?_a$	$\top$	$\{\alpha\}_b$	$\top$	$[\min(\alpha,\beta) \overset{\min(\sigma,\phi)}{\dot{..}} \max(\psi,\gamma)]^?_b$	$[\min(\alpha,\beta) \overset{\min(\sigma,\phi)}{\dot{..}} \max(\psi,\gamma)]^?_b$
$[\beta \mathbin{\dot{..}} \gamma]^!_a$	$\top$	$\top$	$\top$	$\top$	$[\min(\alpha,\beta) \overset{\min(\sigma,\phi)}{\dot{..}} \max(\psi,\gamma)]^!_b$

Table 1. Unification Rules

local variables will be changed on two different program points. The first branch target corresponds to the fulfillment of the condition and sets the type of i to a possible iterator $[0 \mathbin{\dot{..}} 9]^?_2$. Indeed, the identified steps of the loop pattern (initialization, update, condition evaluation) has been consecutively executed and we can infer that the variable i is an iterator at least by one execution path. The second branch target corresponds to the failure of the condition and sets the types of all the variables to $\top$.

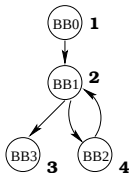
At step 5, the type information determined at the previous step has to be merged with the type stored at the current program point. Thus, a unification had to be made between the following types: $(\{0\}_1, \top)$ and $([0 \mathbin{\dot{..}} 9]^?_2, (i, \leq, 9)_2)$. According to our unification rules, the type of i on label $L1$ must be set to $[0 \mathbin{\dot{..}} 9]^?_2$ as its precedence order 2 is greater than the precedence order of $\{0\}_1$. This denotes that the initialization step occurred before the incrementation one in the loop execution pattern. At step 6, the instruction 3 is evaluated. i is incremented and we can infer that the variable i is an iterator by all the execution paths. The type of i is set to a confirmed iterator $[0 \mathbin{\dot{..}} 9]^!_2$.

At steps 7 and 8, this type is propagated with respect of the unification rules. The analysis stops at the step 8 when the types of the variables i and tmp at the program point 3 still unchanged.

At step 9, the evaluated instruction is a return instruction that stops the program flow and does not propagate any types. This is the last analysis step. The inference engine reached a fix-point and identified an iterator where the lower bound is 0, the incrementation step is 1 and the upper bound is 9.

Sample program 2 Consider the sample program in Figure 6 that corresponds to a pre-test loop pattern. The program P uses two variables i and tmp to iterate 10 times. As explained in 2.1, this loop pattern should consist of the consecutive execution of an initialization step, a condition evaluation step and a counter update step. This loop pattern will allow us to detect an iterator.

BB	pp	Label	Instruction
0	1		i $\leftarrow$ 0
1	2	L1	tmp $\leftarrow$ i $>$ 9
	3		jumpif tmp L2
2	4		i $\leftarrow$ i $+$ 1
	5		jump L1
3	6	L2	return i



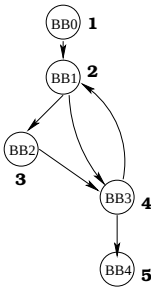
S	pp	T[i,tmp]	pp'	T'[i,tmp]
0	$\emptyset$	$\emptyset, \emptyset$	1	$\top, \top$
1	1	$\top, \top$	2	$\{0\}_1, \top$
2	2	$\{0\}_1, \top$	3	$\{0\}_1, (i, >, 9)_2$
3	3	$\{0\}_1, (i, >, 9)_2$	4	$\{0\}_1, (i, >, 9)_2$
			6	$\top, \top$
4	4	$\{0\}_1, (i, >, 9)_2$	5	$\left[0 \stackrel{1}{..} \right[_4, (i, >, 9)_2$
5	5	$\left[0 \stackrel{1}{..} \right[_4, (i, >, 9)_2$	2	$\left[0 \stackrel{1}{..} \right[_4, (i, >, 9)_2$
6	2	$\left[0 \stackrel{1}{..} \right[_4, \top$	3	$\left[0 \stackrel{1}{..} \right[_4, (i, >, 9)_2$
7	3	$\left[0 \stackrel{1}{..} \right[_4, (i, >, 9)_2$	4	$\left[0 \stackrel{1}{..} 9\right]^?_4, (i, >, 9)_2$
			6	$\top, \top$
8	4	$\left[0 \stackrel{1}{..} 9\right]^?_4, (i, >, 9)_2$	5	$\left[0 \stackrel{1}{..} 9\right]^!_4, (i, >, 9)_2$
9	5	$\left[0 \stackrel{1}{..} 9\right]^!_4, (i, >, 9)_2$	2	$\left[0 \stackrel{1}{..} 9\right]^!_4, (i, >, 9)_2$
10	2	$\left[0 \stackrel{1}{..} 9\right]^!_4, \top$	3	$\left[0 \stackrel{1}{..} 9\right]^!_4, (i, >, 9)_2$
11	3	$\left[0 \stackrel{1}{..} 9\right]^!_4, (i, >, 9)_2$	4	$\left[0 \stackrel{1}{..} 9\right]^!_4, (i, >, 9)_2$
			6	$\top, \top$
12	4	$\left[0 \stackrel{1}{..} 9\right]^!_4, (i, >, 9)_2$	5	$\left[0 \stackrel{1}{..} 9\right]^!_4, (i, >, 9)_2$
13	6	$\top, \top$	$\emptyset$	$\emptyset, \emptyset$

Figure 6. Listing of Sample Program 2, CFG and Execution trace of type inference procedure

After initializing all the variables of the program to $\top$, our tool begins processing instructions. At step 1, the evaluation of the assignment instruction changes the type of i to $\{0\}_1$ for the next step. The step 2 evaluates the instruction 2 which is a comparison between the variable i and a constant value. The type of tmp had to be changed to $(i, >, 9)_2$. At step 3, the evaluated instruction is a conditional branch. In this case, due to the unusable types of the variables, the conditional branch instruction only propagates the inferred types to the jump targets. At step 4, the first jump target is evaluated and corresponds to the implicit branch to the next instruction. As i is incremented in this step, its type had to be set to $\left[0 \stackrel{1}{..} \right[_4$. At step 5, the unconditional branch propagates the types to the label L1. At step 6, as the instruction has been already evaluated, a unification between the following types had to be made: $\{0\}_1$ and $\left[0 \stackrel{1}{..} \right[_4$. According to our unification rules, the type of i on label $L1$ must be set to $\left[0 \stackrel{1}{..} \right[_4$ as its ordering number 4 is greater than the ordering number of $\{0\}_1$. This denotes that the initialization step occurred before in the loop execution pattern. As far as the variable tmp is concerned, its type is changed to $\top$ as it is indicated in Table 1. At step 7, the evaluated instruction is a conditional branch and all the steps required to create an iterator are at least validated by on execution path. So the type of i is changed to a possible iterator on a jump target and to top on the other. At step 8, i is incremented and its type can be changed to a confirmed iterator $\left[0 \stackrel{1}{..} 9\right]^!_4$. In the remaining steps, according to the unification rules, the confirmed iterator is propagated until we reach a fix-point in step 12.

Sample program 3 Consider the sample program in Figure 7. This example corresponds to a post-test loops. In Example1, an iterator was detected as the identified steps were executed consecutively as follows : Initialization step – Counter update step – Condition evaluation step. In this example, the code has been modified to force the program to execute the following pattern : Initialization step – Counter update step – Initialization step – Condition evaluation step. As an initialization step occurs after a counter update one, the detection of the iterator is compromised.

BB	pp	Label	Instruction
0	1		i $\leftarrow$ 0
1	2	L1	i $\leftarrow$ i $+$ 1
	3		tmp $\leftarrow$ i *op* 4
	4		jumpif tmp L2
2	5		i $\leftarrow$ 4
3	6	L2	tmp $\leftarrow$ i $\leq$ 9
	7		jumpif tmp L1
4	8		return i



S	pp	T[i,tmp]	pp'	T'[i,tmp]
0	$\emptyset$	$\emptyset, \emptyset$	1	$\top, \top$
1	1	$\top, \top$	2	$\{0\}_1, \top$
2	2	$\{0\}_1, \top$	3	$\left[0 \stackrel{1}{..} \right[_2, \top$
3	3	$\left[0 \stackrel{1}{..} \right[_2, \top$	4	$\left[0 \stackrel{1}{..} \right[_2, \top$
4	4	$\left[0 \stackrel{1}{..} \right[_2, \top$	5	$\left[0 \stackrel{1}{..} \right[_2, \top$
			6	$\left[0 \stackrel{1}{..} \right[_2, \top$
5	5	$\left[0 \stackrel{1}{..} \right[_2, \top$	6	$\{4\}_3, \top$
6	6	$\{4\}_3, \top$	7	$\{4\}_3, (i, \leq, 9)_4$
7	7	$\{4\}_3, (i, \leq, 9)_4$	2	$\{4\}_3, (i, \leq, 9)_4$
			8	$\{4\}_3, (i, \leq, 9)_4$
8	2	$\{4\}_3, \top$	3	$\{4\}_3, \top$
9	3	$\{4\}_3, \top$	3	$\{4\}_3, \top$
10	4	$\{4\}_3, \top$	5	$\{4\}_3, \top$
			6	$\{4\}_3, \top$
11	5	$\{4\}_3, \top$	6	$\{4\}_3, \top$
12	8	$\{4\}_3, (i, \leq, 9)_4$	3	$\emptyset, \emptyset$

Figure 7. Listing of Sample Program 3, CFG and Execution trace of type inference procedure

Execution starts by setting the types of all the variables to $\top$. Through the steps $1, 2, 3$ the variable i is first initialized then incremented. Its type is changed to $\left[0 \stackrel{1}{..} \right[_2$.

At step 4, the evaluated instruction is a conditional branch that propagates the type information to instruction 5 and 6. At step 5, the instruction 5 assigns a constant value to i. Thus, the type of i is changed to $\{4\}_3$. At this program point, the loop pattern consists of the following steps : initialization – incrementation – initialization. At step 6, a unification between an open interval stamped by the precedence order 2 ($\left[0 \stackrel{1}{..} \right[_2$) is replaced by a constant value stamped by the precedence order 3 ($\{4\}_3$) as indicated by the unification rules. tmp is changed to a conditional on the variable i. At step 7, the branch instruction transmits the types to the corresponding jump targets. At step 8, i is a constant stamped with the precedence order 3. i is incremented at a basic block stamped with a precedence order 2. The type of i still unchanged as suggested by the rule 3 in Figure 3.3. The next steps propagate this type until a fix-point is reached and an iterator is never detected.

3.6. Proof generation

Before generating the proof that will be packaged with the code sent to the consumer, an ultimate step will consists of rejecting any program when we can determine offline that its behavior is unsafe. An unsafe behavior is detected off-line when the loop bounds detection tool identifies a backedge that is not described by a bounded iterator. In this case, the timing consumption of the mobile code cannot be bounded and verified in the code recipient.

Finding backedges relies also on the precedence relation explained in section 2. Indeed, the depth first traversal of the control flow graph assigns two timing stamps for each basic block. One time stamp is related to the time when a basic block b is first discovered and is noted d[b]. The second accounts for the time when the algorithm finishes examining the basic block list of successors and is noted f[v]. An edge $(b \rightarrow b')$ is a backedge if $d[b] < d[b']$ and $f[b] > f[b']$. Note that the discovery time stamp was used for tagging the different types computed by the inference engine.

Determining these time stamps on the consumer represents an important overhead and implies heavy computations. Before sending the code to the consumer, we apply a transformation in order to guarantee the respect of the precedence order of the basic blocks. This transformation guarantees that the loop bounds will be verified on the consumer by a straight forward inspection of code. However, reordering the code probably affects the runtime behavior of a method or increases the code size and the number of local variables slightly as mentioned in [10].

Furthermore, to minimize the workload affected to the consumer in order to compute the WCET of the mobile code, we use a parser to flatten the control flow graph of the program into a tree. This eases the computation of the WCET by the recipient part of the system, since searching the most costly path is less resource-demanding in a tree than in a cyclic graph. Conditional statements are represented by separate branches in the tree. Loops are replaced by a tag on the node representing the execution count of the block. In the case of nested loops, the inner loop is tagged by the product of its execution count and the outer loop one, as illustrated in [2].

Once the tagged-tree is built, it is sent to the target system within the binary containing the code and the established proof of type correctness. The embedded compiler is responsible for searching the most costly branch in the tagged-tree. The proof consists of an array containing the types of the program variables at each jumping target and is sketched in 4.1. A verification step is mandatory as far as the tagged-tree is concerned but is beyond the scope of this paper.

4. Verification on the consumer side

A safe mobile code that can be executed by a consumer must satisfy the following requirements:

1. A basic block has always a precedence order smaller than those of its successors.

2. The loop bounds should be in the range of values inferred by the static analysis done on the producer.

4.1. Loop bounds verification process

Our loop bounds detection tool performed the iterative type checking analysis at the code producer's end. The proof that will be packaged with the code sent to the consumer consists of annotations added to the intermediate code. Indeed, at the beginning of each basic block, the inferred types of the local variables at this program point are added as an annotation. We now describe how these annotations are verified at the code consumer end:

- At the beginning of a basic block, set the derived types of each variable of the program to the annotated types.

- Make one linear pass through the statements of the block, applying the inference rules explained before to the derived types.

- At the end of a basic block, check that the derived types are coherent with the annotated types of all its successors. If the annotations do not verify this property, we reject them.

4.2. Examples

In the following, we illustrate the verification process explained in 4.1.

Sample program 1 The annotated type table joined to the code received by the consumer is represented below:

Label	i	tmp
L1	$\left[0 \stackrel{1}{..} 9\right]_2^1$	$\top$

Table 2 shows the steps of the verification process of the code sample 1. At step 1, according to our operational semantics explained in 3.3, the derived types of the program variables i and tmp corresponds to $\{0\}_1, \top$. At step 2, the evaluated instruction is a jump target. We must check that the current state indicated by $(\{0\}_1, \top)$ is coherent with the annotations done on the producer ($\left[0 \doteq 9\right]^!_2, \top$). Informally, we must verify that the initialization step occurred before the creation of the iterator type. As the precedence order of the current state is smaller than the annotated type, the conformity of the proof is verified and current state is set to ($\left[0 \doteq 9\right]^!_2, \top$). For the following instructions, the derived types are computed with regard to the typing rules that were used by the loop detection tool.

Step	pp	Instruction	$T_a[i, tmp]$	$T_d[i, tmp]$
1	1	i $\leftarrow$ 0	$\emptyset$	$\{0\}_1, \top$
2	2	i $\leftarrow$ i + 1	$\left[0 \doteq 9\right]^!_2, \top$	$\left[0 \doteq 9\right]^!_2, \top$
3	3	tmp $\leftarrow$ i $\leq$ 9	$\emptyset$	$\left[0 \doteq 9\right]^!_2, \top$
4	4	jumpif tmp L1	$\emptyset$	$\left[0 \doteq 9\right]^!_2, (i, >, 9)_2$
5	5	return i	$\emptyset$	$\left[0 \doteq 9\right]^!_2, (i, >, 9)_2$

Table 2. Verification process on the consumer of the sample program 1

Sample program 2 The annotated type table joined to the code received by the consumer is represented below:

Label	i	tmp
L1	$\left[0 \doteq 9\right]^!_4$	$\top$
L2	$\top$	$\top$

Table 3 shows the steps of the verification process of the code sample 2. At step 1, according to our operational semantics explained in 3.3, the derived types of the program variables i and tmp corresponds to $\{0\}_1, \top$. At step 2, the evaluated instruction is a comparison operation and corresponds to a jump target. We must check that the current state indicated by $(\{0\}_1, \top)$ is coherent with the annotations done on the producer ($\left[0 \doteq 9\right]^!_4, \top$). We must verify that the initialization step occurred before the creation of the iterator type. As the precedence order of the current state is smaller than the annotated type, the conformity of the proof is verified and current state is set to ($\left[0 \doteq 9\right]^!_4, \top$). For the following instructions, the derived types are computed with regard to the typing rules that were used by the loop detection tool.

Sample program 3 For the sample program 3, our loop bounds detection tool never detects an iterator. As this code fragment does not correspond to a recognizable loop pattern, our tool rejects the program. In the following, we show that our verification tool rejects the code if a malicious programmer tries to mislead it by sending a false proof.

Step	pp	Instruction	$T_a[i, tmp]$	$T_d[i, tmp]$
1	1	i $\leftarrow$ 0	$\emptyset$	$\{0\}_1, \top$
2	2	tmp $\leftarrow$ i $>$ 9	$\left[0 \doteq 9\right]^!_4, \top$	$\left[0 \doteq 9\right]^!_4, \top$
3	3	jumpif tmp L2	$\emptyset$	$\left[0 \doteq 9\right]^!_4, \top$
4	4	i $\leftarrow$ i + 1	$\emptyset$	$\left[0 \doteq 9\right]^!_4, \top$
5	5	jump L1	$\emptyset$	$\left[0 \doteq 9\right]^!_4, \top$
6	6	return i	$\emptyset$	$\top, \top$

Table 3. Verification process on the consumer of the sample program 2

The annotated type table joined to the code received by the consumer is represented below:

Label	i	tmp
L1	$\left[0 \doteq 9\right]^!_2$	$\top$
L2	$\top$	$\top$

Table 4 shows the steps of the verification process of the code sample 3. At step 1, according to our operational semantics explained in 3.3, the derived types of the program variables i and tmp corresponds to $\{0\}_1, \top$. At step 2, the evaluated instruction corresponds to a jump target. We must check that the current state indicated by $(\{0\}_1, \top)$ is coherent with the annotations done on the producer ($\left[0 \doteq 9\right]^!_2, \top$). We must verify that the initialization step occurred before the creation of the iterator type. As the precedence order of the current state is smaller than the annotated type, the conformity of the proof is verified and current state is set to ($\left[0 \doteq 9\right]^!_2, \top$). At step 3, the proof continuity is ensured. At step 4, a constant value 4 is assigned to i. This initialization occurs in a basic block that occurs after the creation of an iterator. The verifier rejects the code as the proof does not correspond to the mobile code received by the consumer.

Step	pp	Instruction	$T_a[i, tmp]$	$T_d[i, tmp]$
1	1	i $\leftarrow$ 0	$\emptyset$	$\{0\}_1, \top$
2	2	i $\leftarrow$ i + 1	$\left[0 \doteq 9\right]^!_2, \top$	$\left[0 \doteq 9\right]^!_2, \top$
3	3	jumpif tmp L2	$\emptyset$	$\left[0 \doteq 9\right]^!_2, \top$
4	4	i $\leftarrow$ 4	$\emptyset$	Proof Inconsistence
5	5	tmp $\leftarrow$ i $\leq$ 9	$\emptyset$	Proof Inconsistence
6	6	jumpif tmp L1	$\emptyset$	Proof Inconsistence
6	6	return i	$\emptyset$	$\top, \top$

Table 4. Verification process on the consumer of the sample program 3

5. Experimental Results

We experimented our loop bounds detection tool within the *Façade* framework [7]. This framework is based on a typed intermediate language designed for resource-limited devices and mainly smart cards. Thanks to this intermediate language, it was possible to devise an extensible operating system called *Camille* [5]. In *Camille*, applications and operating system extensions are programmed

using a high level language as C or Java for instance, then translated into an Façade Intermediate language by a code converter or a dedicated compiler, before they are loaded in the embedded system. *Camille* already supports the Proof Carrying Code model. Indeed, extensions are validated when loaded in the operating system by a verifier, which ensures their type-correctness.

Our algorithm was applied to the *Camille* kernel as it corresponds to our requirements. Indeed, *Camille* itself is written using a type-safe subset of the C language and can be translated in *Façade* using a customized version of `GCC`. Table 5 describes the kernel source code in terms of footprint, number of lines. More information about the source can be found in [6].

	Kernel C Files	**Kernel Façade Files**
Size (kB)	195	148.5
Number of lines	5962	6606

Table 5. Description of Camille Kernel code

Our off-line tool, first, builds the control flow graph and determines the precedence order on the 53 components constituting the kernel. The analysis is done on the intermediate code and detects 614 basic blocks. Then, the tool proceeds by an iterative type checking analysis. The instruction set used in 3.1 can be considered as a subset of the The Façade language. It uses the CardInt class with the methods *AsIs* for assignment, *+I* for addition, and *opI* for arithmetic comparison.

In the kernel written in C language, we accounted 60 *For* loops and 13 *While* loops. The *Façade* code corresponding to the translation of the kernel files was inspected by our loop bounds detection tool concerning the recognizable loop patterns. It contains 120 methods covering essentially arithmetical operations and memory management. The tool allows to bound 70% of the kernel methods. The remaining methods contains infinite loops or depend closely on the system inputs (***e.g.*** I/O Stream). Our type analysis took fairly 5 iterations on the set of classes to reach a fixed-point.

6. Conclusion and Future Work

We presented in this paper the scheme we propose to safely compute WCET in a resource-constrained operating system. By distributing the computation between the producer side running on a powerful workstation and the consumer side specific to the hardware included in the mobile device, we are able to circumvent the very strict memory and CPU limitation of the device. We guarantee the safety of our scheme by establishing a proof of loop bounds at compilation time on the producer and by verifying its correctness at installation time on the consumer.

In this paper, we focused on determining the upper bound on the number of iteration in loops in order to control CPU usage in the worst case. The proposed approach can also benefit from an integration with live memory analysis which aims at determining an upper bound on the amount of the memory actually referenced by a program. It can also be used to control the system resource usage by accounting the number allocation and deallocation in loop constructs.

References

[1] E. Albert, G. Puebla, and M. Hermenegildo. An abstract interpretation-based approach to mobile code safety. *Electronic Notes in Theoretical Computer Science*, 132(1):113–129, 2005.

[2] N. Bel Hadj Aissa, C. Rippert, and G. Grimaud. Distributing the WCET Computation for Embedded Operating Systems. In *Proc. of the 25th IEEE International Real-Time Systems Symposium, Work In Progress Session*, Lisbon, Portugal, December 2004.

[3] A. Colin and I. Puaut. Worst-case execution time analysis of the RTEMS real-time operating system. In *Proc. of the 13th Euromicro Conference on Real-Time Systems*, pages 191–198, Delft, The Netherlands, june 2001.

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th Symposium on Principles of Programming Languages (POPL)*, pages 238–252, 1977.

[5] D. Deville, A. Galland, G. Grimaud, and S. Jean. Smart Card Operating Systems: Past, Present and Future. In *Proc. of the 5th NORDU/USENIX Conference*, February 2003.

[6] D. Deville, Y. Hodique, and S.-R. I. Safe collaboration in extensible operating systems: A study on real time extensions. *Special Issue on System and Networking for Smart Objects – International Journal of Computers and Applications (IJCA)*, January 2005.

[7] G. Grimaud, J.-L. Lanet, and J.-J. Vandewalle. FaÇade: A typed intermediate language dedicated to smart cards. In *Software Engineering — ESEC/FSE*, number 1687, pages 476–493. Springer-Verlag, 1999.

[8] J. Gustafsson. *Analyzing Execution-Time of Object-Oriented Programs Using Abstract Interpretation*. PhD thesis, Department of Computer Systems, Uppsala University, may 2000.

[9] C. Healy, M. Sjodin, V. Rustagi, D. Whalley, and R. van Engelen. Supporting timing analysis by automatic bounding of loop iterations. *Real-Time Systems*, 18(2/3):129–156, may 2000.

[10] X. Leroy. Bytecode verification on java smart cards. *Software, Practice and Experience*, 32(4):319–340, 2002.

[11] G. Necula. Proof-carrying code. In *Proc. of the 24th ACM Symposium on Principles of Programming Languages*, January 1997.

[12] E. Rose and K. H. Rose. Lightweight bytecode verification. In *Proc. of the OOPSLA'98 Workshop on Formal Underpinnings of Java*, Vancouver, BC, Canada, November 1998.

[13] C. Sandberg. Inspection of industrial code for syntactical loop analysis. In *Proc. of the 4th International Workshop on Worst-Case Execution Time Analysis, in association with the 16th ECRTS conference*, Catania, Italy, June 2004.

Dynamic Instruction Cache Locking in Hard Real-Time Systems

Alexis Arnaud Isabelle Puaut
IRISA, Campus universitaire de Beaulieu,
35042 Rennes Cédex, France
Email: {aarnaud/puaut}@irisa.fr

Abstract

Cache memories have been widely used in order to bridge the gap between high speed processors and relatively slower main memories, and thus to improve the overall performance of systems. However in the context of hard real-time systems, they are a source of predictability problems. A lot of progress has been achieved to model caches to statically determine safe and precise bounds on the worst-case execution times (WCETs) estimates of tasks on architectures with caches. Nonetheless cache-aware WCET analysis techniques may not always be applicable or may be too pessimistic, because some memory accesses are unknown statically. Another reason may come from a poorly documented or non-deterministic cache line replacement policy. An alternative approach is to lock cache lines so as to make memory access times entirely predictable.

In this paper, we consider an instruction cache and a task. We propose a an algorithm which partitions the task into a set of regions. Each region owns statically a locked cache contents determined offline.

A set of tasks is used to experimentally analyze the effects of the algorithm on the worst-case cache miss rate (WCCMR). A sharp improvement is observed, as compared with a system without any cache. Furthermore it is observed that the results obtained on WCCMRs compare to the results obtained from static analysis of a cache whose policy is to replace least recently used (LRU) cache lines. Contrary to cache analysis techniques, our algorithm depends neither on the scheduling policy, nor on the cache line replacement policy. As a further property, it works at the machine language level, and thus does not require any source code.

Keywords : hard real-time systems, cache memories, worst-case execution time

1 Introduction

1.1 Cache memories and real-time issues

Caches are small buffer memories with low latency which are inserted between the CPU and the main memory. They benefit from the spatial and temporal locality often found in instruction and/or data streams in order to store, at any time, memory references which are likely to be addressed in a near future. They operate transparently. Therefore no change is required in the memory addressing scheme. They bring an improvement of the overall performance of computer systems. However two phenomena make it hard to know statically memory access in the worst case:

- Intra-task interferences which occur when a task overrides its own cache lines, mainly because of the relatively small size of the cache as compared with the task's memory demands.
- In preemptive multitasking systems, preemptions cause inter-task interferences. Namely when the execution is switched from a task A towards a task B, some cache blocks used by A may be evicted by B.

In the industry, there is a growing demand of hard real-time systems with improved performance and cheaper hardware. Thus the challenge here is to accomodate the performance goal of cache memories with predictability requirements of hard real-time systems.

1.2 Cache memories in hard real-time systems

There are at the present time two categories of approaches for safely incorporating cache memories in hard real-time systems. In the first one, *cache analysis*, caches operate without any restriction. Static analysis techniques (cache-aware WCET analysis [9, 7] and schedulability analysis

[6]) predict their worst-case impact on the system schedulability. They assume that the cache line replacement policy is known.

The second category of approaches consists in using caches in a restricted or customized manner in order to adapt them to the needs of hard real-time systems and schedulability analysis.

Cache-partitioning techniques assign portions of a cache to some specified tasks in order the guarantee that for each task its most recently used code or data will remain in the cache while the processor executes another task. The partitioning can be made at the hardware [5] or software level [8]. Since the dynamic behavior of the cache is isolated within each partition, inter-task interferences are eliminated. The counterpart is that the per-task available amount of memory is reduced, hence decreased performance. Furthermore static cache analysis is still required to tackle intra-task interferences.

An alternative is to use *cache locking techniques*. Locking a cache line consists in loading some contents in a cache and inhibiting the cache line replacement policy. If all the cache lines are locked, we say that the state of the cache is a *locked cache state*. Predictability is strictly ensured if contents is chosen offline. This feature is available on several commercial processors (among others: Motorola ColdFire MCF5249, Motorola PowerPC 603e, ARM 940T).

Given a task, its code is subdivided into one or more zones. Each such zone has a locked cache state. Consequently, executions of the task are subdivided into temporal windows, in each of which the cache is locked. When there is more than one zone, the locking scheme is said to be *dynamic*, whereas for only one zone, it is *static* [11, 2].

If the locking method is *global*, at every instant, each task owns a portion of the cache space. No cache reload is needed when a task is preempted. In the case of a *local* locking method (see for example [10, 3]), each task owns the entire cache. To ensure this, the cache is reloaded each time a preemption occurs.

1.3 Paper contents and contributions

This paper explores the use of *local dynamic locking* of instruction caches in hard real-time systems. Dynamic cache locking is attractive from several points of views. First of all, it improves the worst and average-case performance of tasks, as compared with the case where the same tasks do not use any cache at all.

When using dynamic instruction cache locking techniques, the interactions between the dynamic properties of caches and other architectural components such as pipelines or branch predictors are less complex, making easier the analysis of these components in validation tools of hard real-time systems. Dynamic instruction cache locking can also be used when no cache analysis method can apply accurately, due for instance to non-deterministic or poorly documented cache replacement strategies.

It may be also suitable for designing mixed systems providing both tasks with hard real-time constraints and tasks with soft real-time constraints which may use unrestricted caches.

In this paper, we propose algorithms for finding a partition of the machine code of a given task into regions, and to determine a locked state of the instruction cache for each such region. It is performed in a non-blind manner by using memory access patterns obtained by profiling the task. The goal is to improve the worst-case performance as compared with a system with no cache, in such a way that this performance be comparable with results obtained from static analysis of the same cache whose replacement policy is the least recently used (LRU).

1.4 Paper organization

The remainder of the paper is organized as follows. Section 2 gives an overview of the proposed local dynamic instruction cache locking strategy. Then we detail the experimental setup and performance measurements used for validating our approach in Section 3. In Section 4, we give an overview of other studies related to our work. Finally we conclude in Section 5 with a summary of the paper contributions.

2 A dynamic instruction cache locking technique

In this section we describe our method which supports dynamic instruction cache locking. After introducing the assumptions and notations (§2.1) and giving a first glance at the method (§2.2), the central objects of this work, namely regions, are studied in paragraph 2.3. Then we detail how to associate a locked cache state to a region of a program in order to improve the worst-case performance of this program (§2.4). Finally, an algorithm for partitioning a program into such regions is described in the paragraph 2.5.

2.1 Assumptions

2.1.1 Architecture and program model

In our model, we consider a CPU provided with a one-level set-associative instruction cache.

We will consider a progamm presented in binary form. Each subroutine owns a unique return point. Indirect jumps are excluded. Moreover the program is assumed to execute within a finite amount of time.

Throughout this paper, for any program, we will associate to each of its subroutines a control flow graph (CFG). A control flow graph is an abstract representation of a subroutine. Each node in the graph represents a basic block, i.e. a sequential piece of code with a unique entry point and a unique exit point. Directed edges are used to represent jumps in the control flow.

2.1.2 Reloading and locking operation

Reloading and locking the cache may be done by inserting instructions calling a special subroutine. However, in this work, this operation is assumed to be done without modifying the program. We use debug registers which raise an exception at specified values of the program counter. An exception handler does the job of reloading and locking the cache. The benefit is that the program's memory map is left unchanged.

2.2 Overview

We propose to apply a local dynamic cache locking strategy which aims to improve the WCET of a program as compared with the case of a system with no cache. The main issue is to avoid performing an exhaustive search of all the possible subdivisions of the program and of all possible cache contents for each subdivision, as this would result in a combinatorial explosion. The proposed method consists in the following two steps :

1. **Profiling.**
   We determine, from executions of the program with various entry data sets, a collection of execution paths along with their execution frequencies. These paths must verify the following two conditions: (i) as many basic blocks as possible are reached; (ii) no path can be deduced from other paths with set operations, so that the number of paths is minimal. From this profiling information, we compute for each basic block an execution frequency.

2. **Program partitioning**
   A greedy algorithm is applied on the set of basic blocks. At the initial state, the program is presented as the set of basic blocks of its control flow graph. Each such basic block is a *region*. At each step of the algorithm, regions are aggregated into new regions. Each region has a locked cache state. Two basic operations, merging and inlining, allow to create new regions from existing ones. The goal of the algorithm is to determine a set of regions minimizing the WCET estimation of the program.

2.3 Regions

The notion of region is central in this work. Given a subroutine whose CFG is known, a region R is a connected part of this CFG. Namely, between any couple B_1, B_2 of basic blocks of R, there exists at least one non-directed path between them. R may be of one of two types :

- R is a *simple region* if it has a locked cache state which is known statically. This state is computed with an algorithm described in section 2.4. The addresses through which other regions of the program may enter R are *cache reload points*. When one of them is reached, the cache is reloaded with the locked cache state of R.

- Suppose R spans all the basic blocks of its subroutine. If there is a significant benefit from avoiding cache reloads when entering and exiting from this subroutine, R may be *inlined*. In this case, R *inherits* the cache state of any region in which the subroutine was called.

2.4 Computation of a locked cache state for a simple region

Consider a simple region R in a program. We provide it with a locked cache state. Namely, for each cache line, we select from this region the memory line such that: (i) it can be loaded in that cache line; (ii) its execution frequency is the highest; (iii) the gain obtained from having this memory line in the cache is more important than the cost of loading and locking it in the cache.

The last condition is true if the execution frequency of this memory line exceeds a constant proportional to the average number of times a cache reload occurs when entering R.

This locked cache state is chosen so as to minimize, among all possible choices, an heuristic which is the approximate time spent, during any execution of the program, in the basic blocks of R plus the average time spent reloading the locked cache state of R. The proof of this property vaguely follows the lines of the main proof presented in [11], so we will not detail it here.

2.5 The Region Merging and Inlining algorithm

In this Section, first we define two basic operations on regions, namely merging and inlining (§2.5.1). Then, in order to improve the WCET of a program, an algorithm (§2.5.2) partitions the program into regions using these two operations.

2.5.1 Basic operations on regions

Merging Let R_1 and R_2 be two simple regions that are connected. Merging these two regions into a new simple region R means that:

- R aggregates the blocks of R_1 and R_2
- The locked cache state of R is computed by the algorithm presented in the paragraph 2.4.

We will use the notation $R = R_1 \oplus R_2$ to express the fact that the region R is obtained as the result of merging R_1 with R_2.

Inlining Suppose a subroutine contains only a simple region R. There may be a potential benefit by avoiding cache reloads when calling and exiting this subroutine. The general idea for the inlining operation is to allow the this subroutine to inherit the locked cache state from the subroutine which has just called it.

We now define a *calling region* CR of R the following way (cf. figure 1):

- CR is a simple region.
- There exists at least one chain $(f_0, \ldots, f_{m-1})$ of subroutine calls leading from CR to R: (i) the call towards f_0 lies in CR; (ii) if $m \geq 2$, each $f_0, \ldots f_{m-2}$ represent an inlined region; (iii) f_{m-1} calls towards the subroutine representing R.

Now let CR_i $(1 \leq i \leq n)$ be the calling regions of R. Then *inlining* R means that, for each CR_i:

- For each memory line of R, its frequency is assumed to be scaled up by the proportion, among all the calling regions, of calls from CR_i towards R.
- Its locked cache state is computed (§2.4) offline from the knowledge of the memory lines of both CR_i and R.

From now on, the locked cache state of R is inherited from the locked cache state of the last accessed calling region during runtime (cf. figure 1).

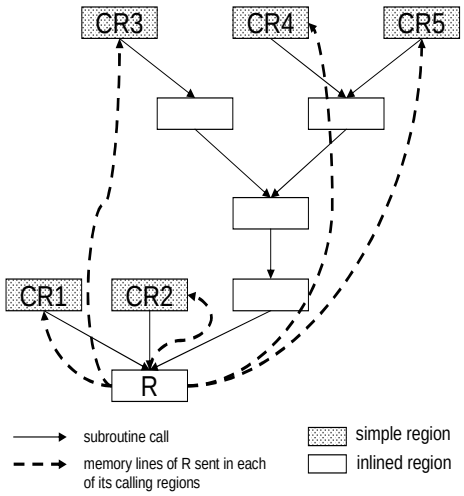


Figure 1. A region R and its calling regions CR_i. Inlining operation on R.

2.5.2 Description of the algorithm

In this subsection, we give a description of an algorithm which determines a partition of a program P into regions in order to to minimize the WCET of P. We propose a sub-optimal strategy, the RMI (Region Merging and Inlining) greedy algorithm. RMI takes as an input the partition of P in basic blocks, which are initial regions. At each iteration, a pair of regions is chosen and merged once for all, thus giving a new partition choice. RMI keeps also track of the current best partition. Inlining operations are involved when updating the best partition. When completed, RMI returns the best found partition of the program.

Quality of an operation. Let Ω_{pre} be a partition of P, and Ω_{post} the partition resulting from an operation (merging or inlining) on Ω_{pre}. The quality criterion of this operation is based on the difference, noted δ, between the WCETs of P with the locked cache states from respectively Ω_{post} and Ω_{pre}. The best operation gives the lowest value of δ, noted δ_{min}. It represents on the WCET of P its best improvement if $\delta_{\text{min}} < 0$, and its least deterioration otherwise.

In order to choose among some possible operations on Ω_{pre} the best one, the EvalOp algorithm (cf. algorithm 1) must be called each time such an operation was attempted on Ω_{pre}. Given an operation, the WCET of the resulting partition Ω_{post} and its quality criterion δ are computed (ℓ. 1-2). Then EvalOp updates the information on the partition resulting from the best operation on Ω_{pre} if needed (ℓ. 3).

Algorithm 1 EvalOp algorithm

Require: P: program; Ω_{pre}: partition of P; Ω_{post}: partition after an operation; δ_{min}: best quality criterion

Ensure: Ω_{min}: partition resulting from the best operation; δ_{min}

1: WCET(Ω_{post}) $\leftarrow$ WCET of P with Ω_{post};
2: $\delta \leftarrow$ WCET(Ω_{post}) - WCET(Ω_{pre});
3: **if** $\delta \leq \delta_{\text{min}}$ **then** $\Omega_{\text{min}} \leftarrow \Omega_{\text{post}}$; $\delta_{\text{min}} \leftarrow \delta$;

Description of the RMI algorithm. First note that a partition Ω of P into regions gives rise to a search space. Namely this search space contains all the partitions that can be deduced from Ω by operations (merging and inlining) on its regions.

Algorithm 2 RMI algorithm

Require: P: program, Ω_{init}: initial partition of P, S_{max}: max size of a set of locked cache states

Ensure: Ω_{best}: best found partition

1: $\Omega_{\text{cur}} \leftarrow \Omega_{\text{init}}$; $\Omega_{\text{best}} \leftarrow \emptyset$;
2: WCET(Ω_{cur}) $\leftarrow$ WCET of P with Ω_{cur};
3: WCET(Ω_{best}) $\leftarrow$ WCET of P (no cache);
4: **while** a subroutine has more than 1 simple region in Ω_{cur} **do**
5: $\quad\Omega_{\text{cur}} \leftarrow$ TryMerge(P, Ω_{cur});
6: $\quad$**if** WCET(Ω_{cur}) $\leq$ WCET(Ω_{best}) and Size(Ω_{cur}) $\leq S_{\text{max}}$ **then** $\Omega_{\text{best}} \leftarrow \Omega_{\text{cur}}$;
7: $\quad\Omega_{\text{inlined}} \leftarrow \Omega_{\text{cur}}$;
8: $\quad$**while** there are inlineable regions in Ω_{inlined} **do**
9: $\quad\quad\Omega_{\text{inlined}} \leftarrow$ TryInline(P, Ω_{inlined});
10: $\quad\quad$**if** WCET(Ω_{inlined}) $\leq$ WCET(Ω_{best}) and Size(Ω_{cur}) $\leq S_{\text{max}}$ **then** $\Omega_{\text{best}} \leftarrow \Omega_{\text{inlined}}$;
11: $\quad$**end while**
12: **end while**

The RMI algorithm (cf. algorithm 2) starts from the initial solution search space corresponding to the basic blocks of P stored in the current partition choice Ω_{cur} (ℓ. 1). At each iteration, RMI searches for the best merging between a pair of regions (ℓ. 5) by calling the TryMerge algorithm (cf. algorithm 3), thus updating Ω_{cur}, and equivalently reducing the solution search space. It then updates the information on the best partition (ℓ. 6-11). The whole process is iterated until no merging operation is possible in the solution search space, which means that, in Ω_{cur}, only one simple region remains in each subroutine (ℓ. 5). When choosing the best partition Ω_{best} of P, RMI first compares Ω_{best} against Ω_{cur} (ℓ. 6), and updates it if needed. Then, starting from Ω_{cur}, a greedy algorithm is used to choose a sequence of inlining operations (ℓ. 8-11) by calling the TryInline algorithm (cf. algorithm 4). At each step, the current choice is stored in Ω_{inlined}. After a choice was made, Ω_{best} is updated if needed.

Description of the TryMerge algorithm. Given a partition Ω of the program P, for each pair of mergeable regions, the TryMerge algorithm tries to merge them and builds a test partition Ω_{test} (ℓ. 3). If the EvalOp algorithm decides that Ω_{test} results from the current best merging operation, it is stored in Ω_{min} (ℓ. 4). After completion, Ω is updated with the partition stored in Ω_{min} representing the best merging operation (ℓ. 6).

Algorithm 3 TryMerge algorithm

Require: P: program, Ω: partition of P

Ensure: Ω

1: $\delta_{\text{min}} \leftarrow \delta_{\text{max}}$;
2: **for** each connected pair of simple regions $(R_1, R_2) \in \Omega$ **do**
3: $\quad\Omega_{\text{test}} \leftarrow (\Omega \backslash \{R_1, R_2\}) \cup \{R_1 \oplus R_2\}$;
4: $\quad(\Omega_{\text{min}}, \delta_{\text{min}}) \leftarrow$ EvalOp(P, Ω, Ω_{test}, δ_{min});
5: **end for**
6: $\Omega \leftarrow \Omega_{\text{min}}$;

Description of the TryInline algorithm. Given a partition Ω of P, for each subroutine which contains only one simple region R, the TryInline algorithm builds a test partition Ω_{test} in which R is inlined (ℓ. 3-7). As for TryMerge, the EvalOp algorithm is used to choose the current best inlining operation (ℓ. 8) whose corresponding partition is stored in Ω_{min}. After completion, Ω contains the partition corresponding to the best inlining operation (ℓ. 10).

Algorithm 4 TryInline algorithm

Require: P: program, Ω: partition of P

Ensure: Ω

1: $\delta_{\text{min}} \leftarrow \delta_{\text{max}}$;
2: **for** each inlineable region $R \in \Omega$ **do**
3: $\quad\mathcal{CR}$: set of calling regions of R in Ω;
4: $\quad R' \leftarrow R$; $\mathcal{CR}' \leftarrow \mathcal{CR}$;
5: $\quad\Omega_{\text{test}} \leftarrow \Omega \backslash \{R, \mathcal{CR}\}$;
6: $\quad$Inline R' in $\mathcal{CR}'$;
7: $\quad\Omega_{\text{test}} \leftarrow \Omega_{\text{test}} \cup \{R', \mathcal{CR}'\}$;
8: $\quad(\Omega_{\text{min}}, \delta_{\text{min}}) \leftarrow$ EvalOp(P, Ω, Ω_{test}, δ_{min});
9: **end for**
10: $\Omega \leftarrow \Omega_{\text{min}}$;

As regards the worst-case complexity of the RMI algorithm in terms of the basic operations involved, merging and inlining, it is quadratic in the

number of basic blocks of the considered program. This property is shown in the annex A.

3 Experimental results

This section deals with an experimentation designed to validate the approach adopted in this work. In the paragraph §3.1, the experimental protocol and the assumptions are detailed. Then in the following paragraph (§3.2), we evaluate the impact of our method on the worst-case performance.

3.1 Experimental setup

Hardware and timing model. As the worst-case performance with regards to an instruction cache is our only concern, we assume an executive support from a 32 bit MIPS R3000 processor at instruction level only. In our model, this processor provides only one architectural component, namely an instruction cache. Its cache line replacement policy is the LRU policy. Moreover we suppose that this cache can be totally locked.

The size of the cache ranges from 512 bytes to 4 kilobytes, and its associativity is equal to 1 (thus it is direct-mapped). The application performance with respect to the cache is our only concern in this study. Therefore the timing model for the processor is very simple. The worst case performance of a task under a given configuration of the cache is measured in worst case cache miss rate (WCCMR).

When the cache is dynamically locked, a special routine of the underlying operating system is assumed to manage the reloading and the locking of the cache. *As the performance of this routine is highly critical, it is assumed to be stored into a scratch-pad memory* [12]. As we focus on cache misses, only operations loading memory lines into the cache are taken into account. Thus, given a locked cache state S, the worst number of cache misses of this routine is assumed to be equal to $|S|$, i.e. the number of cache lines in S.

Generation of execution traces. In order to profile programs, a MIPS R3000 processor emulator at instruction level is used to generate execution traces.

Estimation of worst case miss rates. The WCCMRs of programs, presented in binary form, are computed with the Heptane [1] static WCET analysis tool [4]. Within the context of this work, it uses a technique based on abstract syntaxic trees. In such a tree, the leaves are basic blocks, while the other nodes are sequences, if-then-else control structures, or loop structures. The WCET and the WCCMR are computed bottom-up by formulae which establish for each node a partial WCET (resp. WCCMR) depending on its children nodes. The WCET (resp. WCCMR) of the root node is then the WCET (resp. WCCMR) of the analyzed program.

Heptane includes hardware modeling capabilities to estimate safely but precisely the numbers of hits and misses in the worst case on architectures with instruction caches, pipelines and simple branch predictors. In the present study, Heptane's pipeline and branch prediction modeling modules were switched off since our focus is on instruction caches only. In addition of a cache analysis module, Heptane was incorporated a module that takes into account the presence of a dynamically locked instruction cache. This new module uses a file describing the set of cache states and cache reload points of the program to be analyzed. It classifies instructions into two categories : *miss* and *hit*. An instruction is classified a a *hit* if it is locked in the instruction cache, and is classified as a *miss* otherwise.

Experimentation process. Given a program and a parametrization of the instruction cache, the experiment proceeds in two steps (see figure 2). First, the set of cache states and cache reload points is computed by the RMI algorithm. For this purpose, execution traces are generated with Nachos.
The second step is the performance evaluation itself. The WCCMR is computed with Heptane. Two cases are considered: (i) a system with a dynamic instruction cache (i.e. operating in its normal behavior); (ii) a system with a dynamically locked instruction cache.

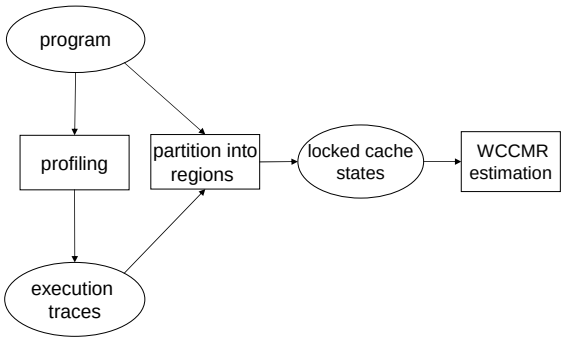


Figure 2. Experimental protocol

The experiments were conducted on three benchmark tasks, whose features are summarized

[1] Heptane is an open-source software available at http://www.irisa.fr/aces/software/software.html

in figure 3. The third column gives, for each task, the code size in bytes.

Name	Description	Size
minver	matrix inversion	4584
matmult	matrix product	1328
jfdctint	integer DCT transformation	3424

Figure 3. Characteristics of tasks

3.2 Performance of dynamic instruction cache locking

In this paragraph, we interpret the results obtained from the experimentation. First, we examine the worst-case performance improvement obtained with our approach (§3.2.1). Then we study some properties of the RMI algorithm itself (§3.2.2).

3.2.1 Worst-case performance

We compare the worst-case performance of the tasks in two situations: (i) the cache is dynamically locked; (ii) the cache is dynamic wih a LRU policy. The figures 4, 5, and 6 describe the results of the experiments. In the locked case, the WCCMR comprises the cache misses due to the task itself, and the cache misses arising from cache reloading operations.

Impact of the cache size. As seen on figures 4, 5, and 6, in both locked and LRU cases, the worst-case performance is far better than without any cache (in this situation, the WCCMR would be equal to 100%).

In the dynamic case, the WCCMR sharply decreases when increasing the cache size, as the cache conflict probability decreases.

In the locked case, when increasing the cache size, we observe a general tendency towards the decrease of the part of the WCCMR which represents the reload overhead, . But for a notable exception in the case of the task jfdctint with a 1 KB cache, a similar tendency applies as regards the part of the WCCMR from the task itself.

Now we compare the worst-case performance between the locked cache and the dynamic case. In this aim, we compute, for each task and each cache size, a ratio between the total WCCMR in the locked case and the WCCMR in the dynamic case. With the exception of two results (jfdctint with a 1 KB cache, and minver with a 4 KB cache), the average ratio is equal to 1 for jfdctint, 1.44 for minver and 0.83 for matmult. Thus the results are in the same order of magnitude in the locked and dynamic situations.

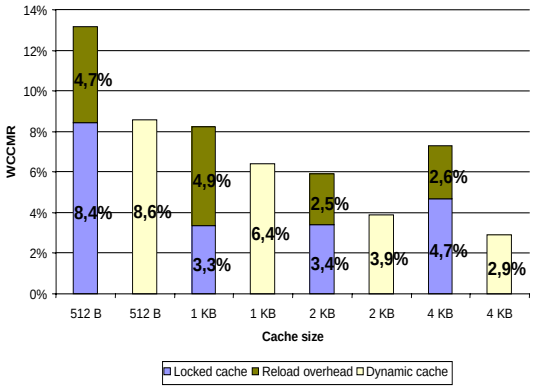


Figure 4. WCCMR results for minver.

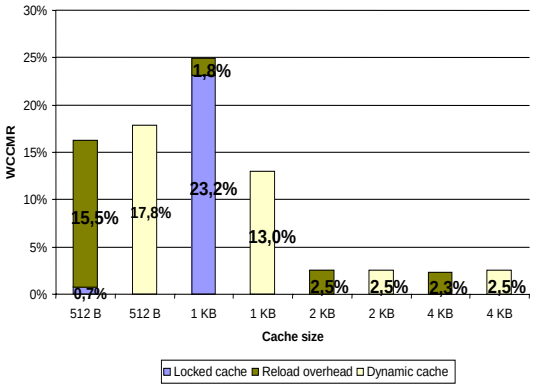


Figure 5. WCCMR results for jdctint.

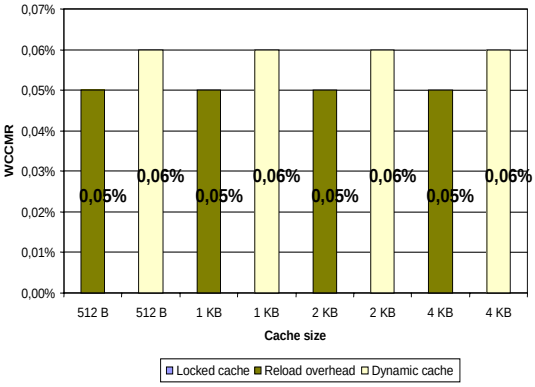


Figure 6. WCCMR results for matmult.

Impact of the associativity degree. On the figure 7, we consider the task minver and a 1 KB cache. The impact of the associativity degree is illustrated both in the locked and LRU cases.

We observe that the worst-case performance of the locked cache scales well when increasing the associativity degree. This can be explained by the fact that, for a given cache size, a cache contents computed by the Lock-MP algorithm for a specified associativity degree remains valid for other associativity degrees. This confers to the RMI algorithm a low sensitivity to the variations of this parameter.

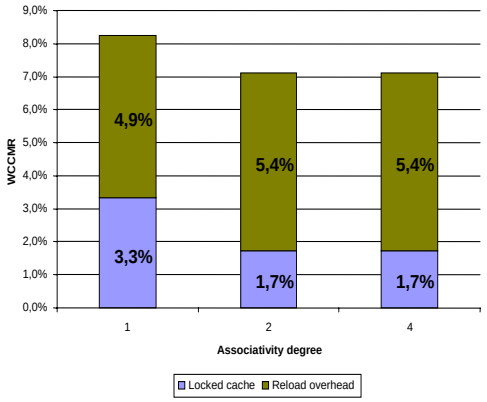


Figure 7. Compared impact of the associativity degree of a 1 KB cache for the task minver.

3.2.2 Properties of the RMI algorithm

Performance. As noticed before, the worst-case complexity of the RMI is quadratic in the number of basic operations, merging or inlining, on the initial set of basic blocks of a task. The figure 8 indicates, among others, for each task and for each cache parametrization, the number IB of initial basic blocks and the time T it took to compute a set of locked cache states. The study of the quantity T/IB^2 shows that the computation time T in seconds follows the approximate law $T = 0.5IB^2$.

Locked cache states. As regards the number of locked cache states determined by the RMI algorithm, the figure 8 shows that this number decreases when the size of the cache increases. This is is essentially due to the fact that the cache contents selection algorithm Lock-MP accepts a more important number of useful memory lines in a less important number of locked cache states.

A notable fact is that, even when the size of a task is inferior or approximately equal to the size of the cache, the RMI algorithm may determine more than one locked cache state. It can be seen in the figure 8 in the case of the task minver for a 4 KB cache, and in the case of the task matmult for most of the cache sizes. A reason for rejection of valuable memory lines by the Lock-MP algorithm is the existence of conflicts due to placement constraints in a set-associative cache. The RMI algorithm may address this issue by creating more locked cache states when there is a benefit from considering those rejected memory lines.

Task	Cache size	Nb of basic blocks	Computation time	Nb of cache states
minver	512 B	135	3h 6min 49s	10
	1 KB	135	2h 30min 52s	6
	2 KB	135	2h 35min 13s	4
	4 KB	135	2h 42min 12s	3
jfdctint	512 B	23	5min 22s	12
	1 KB	21	4min 32s	3
	2 KB	19	3min 12s	3
	4 KB	19	3min 13s	1
matmult	512 B	23	2min 48s	2
	1 KB	23	2min 48s	2
	2 KB	22	2min 50s	2
	4 KB	23	3min	2

Figure 8. Results characteristics

4 Related work

Studies have been performed for static instruction cache locking in multitasking hard real-time systems. In [2], a global approach is proposed. The cache state minimizing the cache-aware response time (CRTA) [1] of each task is chosen. It is achieved with a genetic algorithm. The fitness function is a weighted mean of the response time of each task. The same authors explored a local approach in [3] with the same algorithm for cache contents selection.

In [11], two greedy algorithms have been designed for a global locking scheme. Both have a pseudo-polynomial complexity. From task periods and access statistics of instruction blocks along the worst-case execution path of each task, each algorithm selects a cache state so as to minimize a well chosen cache-aware metric, and thus to improve the task set schedulability. A local variant is proposed in [10].

As explained in [10], static cache locking lacks some scalability. If the ratio between the size of the task set and the size of the cache memory is very high, only a very small fraction of the task set will benefit from the cache. Our work is applied on a per-task basis, and thus is a local approach. It is designed to overcome the scalability problem by al-

lowing the locked state of the cache to be reloaded at some addresses of a program.

The work [13] is a combination of dynamic data cache locking and static cache analysis. Given a task, at compile time, an algorithm computes the regions in the code where one cannot accurately determine all possible cache contents required for analyzing the state of the data cache, because of memory references which cannot be statically known. Such regions are enclosed with a pair of statements so that the cache is locked in them. A locality analysis based on the study of reuse vectors selects the data to be loaded in the cache. In order to address the multitasking issues, it is assumed that the data cache is partitioned among the tasks of the system. Also the knowledge of the cache replacement policy is required.

As compared with this work, our approach proposes a scheme in which the instruction cache is *always* locked. Thus our method does not depend on the cache line replacement policy, and may be used in cases when static cache analysis fails. Moreover our work does not depend on any partition of the cache. Therefore it does not require additionnal partitioning techniques, and it can be easily applied in situations in which the number of tasks of the system may vary.

Finally, scratch-pad memories [12] are an alternative to instruction or data caches. These are on-chip static memories with low latencies. As a consequence they may reconcile performance and predictability. They generally provide lower capacities than caches and consume far less power. Because of the addressing scheme, the code of tasks must be explicitly modified in order to benefit from scratch-pad memories. Thus, as compared with our scheme, this approach requires more compiler support. We believe that the addressing transparency provided by instruction caches is a key advantage, because it alleviates the need for code transformations.

5 Conclusion

The key benefit of instruction cache locking is to make the memory access times entirely predictable and to be a technique that eliminates intra-task conflicts. It can be applied in situations where static cache analysis cannot be used (e.g. when the cache has a non deterministic or undocumented cache line replacement policy). Moreover, it may make easier the analysis of other architectural components. In this work, we have proposed a local dynamic cache locking strategy and an algorithm for determining a finite number of cache configurations for a given task. Its additional features are independence from any scheduling policy (it is a per-task strategy), unnecessity to access the source code of programs, scalability with regards to cache associativity. With regard to performance evaluation against a system without any instruction cache, a sharp improvement is observed on the miss rates in the worst case. Moreover for many cache parametrizations, the worst-case performance is in the same order of magnitude as results from static LRU cache analysis.

As a further work, it would be interesting to explore the transposition of the RMI algorithm (i.e. we keep the basic merging and inlining operations) from a greedy algorithm towards a genetic algorithm. The main reason is that a genetic algorithm exhibits a better exploration of a solution space and thus might find sets of locked cache states which would lead to better improvements on worst-case performances. Another direction would be to adapt this work in other situations. It could be easily achieved for multi-level instruction caches. Finally, the adaptation to data caches should be investigated.

A Worst-case complexity of the RMI algorithm

As regards the worst-case complexity of the RMI algorithm, we now show that it is quadratic in terms of involved operations (merging and inlining). First we detail a worst-case scenario. Suppose our program comprises N_S subroutines $F_0 \dots F_N$, each with N_R basic blocks assimilated to simple regions. In each F_k, the N_R regions are consecutive. We consider the following calling hierarchy: for each k, F_k calls the subroutines $F_{k+1} \dots F_{N_S-1}$. For the sake of simplicity, we assume here that the main subroutine may be inlined. Starting from the value $k = 0$, RMI repeats the following steps until only one simple region remains in each subroutine: (i) choose a pair of regions of F_k, then merge them; (ii) in the remaining subroutines $F_{k+1}, \dots, F_{N_S-1}$, no pair of regions is chosen for merging; (iii) if only one simple region remains in F_k, then increment k; (iv) try to inline each of the subroutines $F_0, \dots, F_{k-1}$, but never choose one.

Given a value of k, each of the subroutines $F_0, \dots, F_{k-1}$ contain only one region simple. At a given stage, assume the subroutine F_k has $N_R - i + 1$ regions. Then $N_R - i$ mergings are tried before making a choice. As each of the remaining $N_S - k - 1$ subroutines $F_{k+1}, \dots, F_{N_S-1}$ has N_R simple regions, overall $(N_S - k - 1)(N_R - 1)$

mergings are tried without any choice being made. As regards the subroutines $F_0, \ldots, F_{k-1}$, k inlining operations are tried without any success. Thus, at a given stage, $(N_R - i) + (N_S - k - 1)(N_R - 1)$ operations are done. As the number of regions of F_k can vary from 2 to N_R for mergings, i ranges from 1 to $N_R - 1$. Now summing over the N_S subroutines, we obtain the following number of operations: $\sum_{k=0}^{N_S-1} \sum_{i=1}^{N_R-1} [(N_R - i) + (N_S - k - 1)(N_R - 1) + k]$. The computation of this sum yields $\frac{1}{2} N_S^2 N_R (N_R - 1)$ operations. Thus this value is in $O((N_S N_R)^2)$. As $N_S N_R$ is the number of basic blocks of the program, the worst-case complexity of the RMI algorithm in number of operations is quadratic with the number of basic blocks.

References

[1] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *Proceedings of the 1996 Real-Time technology and Applications Symposium (RTAS '96)*, pages 204–212. IEEE Computer Society, June 1996.

[2] A. Marti Campoy, A. P. Ivars, and J. V. Busquets Mataix. Static use of locking caches in multitask preemptive real-time systems. In *Proceedings of the IEEE/IEE Real-Time Embedded Systems Workshop (RTES'01)*, 2001.

[3] A. Marti Campoy, A. P. Ivars, F. Rodriguez, and J. V. Busquets Mataix. Static use of locking caches vs. dynamic use of locking caches for real-time systems. In *IEEE Canadian Conference on Electrical and Computer Engineering*, Montreal, Canada, May 2003.

[4] A. Colin and I. Puaut. A modular retargetable framework for tree-based wcet analysis. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, pages 37–44, The Netherlands, June 2001.

[5] D. B. Kirk. Smart (strategic memory allocation for real-time) cache design. In *Proceedings of the 10th IEEE Real-Time Systems Symposium (RTSS '89)*, pages 229–237, Santa Monica, CA, USA, December 1989.

[6] C. Lee, J. Hahn, Y. Seo, S. Min, R. Ha, S. Hong, C. Park, M. Lee, and C. S. Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Transactions on Computers*, 47(6):700–713, June 1998.

[7] T. Lundqvist and P. Stenström. An integrated path and timing analysis method based on cycle-level symbolic execution. *Real-Time Systems*, 17(2-3):183–207, 1999.

[8] F. Mueller. Compiler support for software-based cache partitioning. In *LCTES '95: Proceedings of the ACM SIGPLAN 1995 workshop on Languages, compilers, & tools for real-time systems*, pages 125–133, New York, NY, USA, 1995. ACM Press.

[9] F. Mueller. Timing analysis for instruction caches. *Real-time systems*, 18(2):217–247, May 2000.

[10] I. Puaut, A. Arnaud, and D. Decotigny. Analyse de performance de méthodes de verrouillage statique de caches dans les systèmes temps-réel strict. In *Proc. of the 12th International Conference on Real-Time Systems (RTS'04)*, March 2004.

[11] I. Puaut and D. Decotigny. Low-complexity algorithms for static cache locking in multitasking hard real-time systems. In *Proceedings of the 23rd IEEE International Real-Time Systems Symposium (RTSS '02)*, Austin, TX, USA, December 2002.

[12] S. Steinke, N. Grunwald, L. Wehmeyer, R. Banakar, M. Balakrishnan, and P. Marwedel. Reducing energy consumption by dynamic copying of instructions onto onchip memory. In *ISSS '02: Proceedings of the 15th international symposium on System Synthesis*, pages 213–218, New York, NY, USA, 2002. ACM Press.

[13] X.Vera, B. Lisper, and J.Xue. Data caches in multitasking hard real-time systems. In *24th IEEE International Real-Time Systems Symposium (RTSS '03)*, Cancun, Mexico, 2003. IEEE Computer Society Press.

Uniprocessor Scheduling II

Polynomial Time Approximate Schedulability Tests for Fixed-Priority Real-Time Tasks: some numerical experimentations

Pascal Richard
Laboratoire d'Informatique Scientifique et Industrielle
ENSMA
1, avenue Clément Ader
Téléport 2 - BP 40109
86961 Futuroscope Cedex (France)
pascal.richard@univ-poitiers.fr

Abstract

Efficient schedulability tests are required for analyzing large task systems or for designing on-line admission controllers. We next focus on periodic fixed-priority tasks. For fixed-priority tasks with constrained deadlines (i.e., deadlines are less than or equal to periods), no exact polynomial time feasibility test is known. We propose several polynomial time algorithms with performance guarantees (with an ijnput accuracy parameter) and compare them with known exact feasibility tests (running in pseudo-polynomial time) and a fully polynomial time approximation scheme (FPTAS). Our main objective is to define the capabilities of such algorithms according to the system workload and an accuracy parameter defining the quality of results to compute.

1 Introduction

Large real-time systems are emerging in many applications, including industrial automation, defense and telecommunication. For these systems, the exact workload cannot be predicted and there are significant runtime uncertainties due to the controlled environment or system resources states. In many case, best effort strategies are required to admit or reject works. After the admission control, all admitted tasks must meet their timing requirements. Many admission controllers are dedicated to improve some Quality of Service (QoS) metrics or a benefit function. They are usually based on two on-line algorithms: an admission controller that checks if a new task can be accepted without any consequence on already admitted tasks and a scheduler that chooses the next task to run among uncompleted admitted tasks. In many systems, tasks are assumed to be periodic, but their first release time is not predictable (i.e., tasks are released over time) and can be killed due to system mode changes. Due to such a dynamic arrival of works, these real-time systems must cope with temporary overloaded conditions (using an admission controller to regulate admitted workload while ensuring that task deadlines will be met). Polynomial time schedulability tests are necessary to define efficient admission controllers.

Checking the feasibility of a task system is usually a hard computational problem, that cannot be solved in polynomial time in the number of tasks. Exact feasibility tests are known for periodic fixed-priority tasks [10, 11] and run in pseudo-polynomial time. Furthermore, their execution times can vary from one execution to another according to the task parameters [13, 6]. Nevertheless, there are two ways for defining efficient schedulability tests that consists on:

- improving initial values of an exact feasibility test as in [13] or [6]. But, the worst-case computational complexity of such tests is still pseudo-polynomial,
- defining an approximate schedulability test running in polynomial time as in [7, 1, 9].

Next, we focus on the second promising way. Approximation algorithms perform a compromise between computational effort to decide the feasibility of task systems and the quality of taken decisions. If the approximate algorithm concludes that a task system is feasible, then it will be true at run-time for all possible behaviors of these tasks. But, if the answer is negative, then we cannot conclude that the task system will be infeasible at run-time.

The paper is organized as follows: Section 2 presents the task model considered in the remaining of the paper. Section 3 presents known exact feasibility tests and approximate feasibility tests for periodic fixed-priority tasks. Section 4 presents some computational complexity results and new polynomial time feasibility tests for fixed-priority real-time tasks. Section 5 presents experimental results based on numerical simulations.

2 The task model

We consider uniprocessor real-time systems running periodic tasks. A periodic task τ_i defines a set of jobs. Every periodic task is known and implemented in the software architecture. Thus, job parameters are always known before starting the system. Every task τ_i is defined by three parameters and denoted $\tau_i(C_i, D_i, T_i)$. C_i is the worst-case execution requirement of τ_i, D_i its the relative deadline (the time window between its release and its completion), and T_i its period between two successive releases. We assume that deadlines are constrained: $D_i \leq T_i, 1 \leq i \leq n$, where n is the number of tasks in the system.

Every job generated by a periodic task is scheduled using a fixed-priority. At any time, the highest priority job is run among available ones. According to such a basic dispatching policy, the optimal priority assignment can be performed off-line using the Deadline Monotonic [2] priority ordering. We assume that task priorities are known before starting the system (i.e. priority assignment is done off-line) and tasks are indexed using the priority ordering, thus τ_1 is the highest priority task.

3 Review of feasibility tests for preemptive fixed-priority task systems

Three main approaches are used to define schedulability tests: analyzing the system utilization factor (i.e., $\sum_{i=1}^{n} C_i/T_i$), analyzing the processor demand or analyzing worst-case response times of tasks. For fixed-priority tasks, tests are known for checking a sufficient schedulability condition of tasks having deadlines equal to periods such as [12]. A necessary and sufficient schedulability condition can be computed in pseudo-polynomial time for systems having constrained-deadlines using a processor demand analysis or by computing worst-case response times of tasks. But, no polynomial time algorithm nor NP-hardness result are currently known for the feasibility problem related to the studied task model. Next, we only present results and schedulability tests that will be used in the remainder of the paper.

3.1 Exact algorithms

For a given task τ_i, the scenario leading to its worst-case response time R_i is achieved when task τ_i is released at a *critical instant* (i.e., simultaneously with all higher priority tasks) [12]. The processor demand analysis is based on the total execution time required by a task τ_i and can be expressed as of function of time. In a periodic synchronous task system, the total execution time requested by task τ_i is (request bound function):

$$rbf_i(t) = \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{1}$$

The *cumulative request bound function* allows to compute the worst-case response time of task τ_i:

$$W_i(t) = C_i + \sum_{j=1}^{i-1} rbf_i(t) \tag{2}$$

For task τ_i, its exact worst-case response time R_i^* is the minimal solution to the equation:

$$W_i(R_i^*) = R_i^* \tag{3}$$

Joseph and Pandya [10] proposed a recursive algorithm to solve the previous equation. But an iterative algorithm can be defined using successive approximation of response times in order to reach the smallest fixed-point of Equation 3 (this lead to simple recursion formula). The feasibility test consists on: first, computing worst-case response times of all tasks, and second, checking that $R_i^* \leq D_i, 1 \leq i \leq n$. The corresponding algorithm is pseudo-polynomial and the number of iterations before reaching the smallest fixed-point widely varies from one task system to another and is highly dependent on task parameters [13, 6].

Lehoczky *et al.* [11] provided a processor demand analysis for checking task feasibility that will lead in practice to a different feasibility test. Their main result is stated hereafter:

Theorem 1 *[11] In a synchronous task system, task τ_i is feasible if, and only if, there exists a time $t \in (0, D_i]$ such that $W_i(t) \leq t$.*

Such a result defines an alternative way to check feasibility of a task system, without explicitly computing worst-case response times. The cumulative request bound function (defined in Equation 2) only changes for a finite set of values (i.e., when tasks are released). Thus, the number of time instants to check the feasibility of task τ_i in Theorem 1 is defined by the following testing set (for constrained-deadline task systems):

$$S_i = \{bT_j | j = 1 \ldots i, b = 1 \ldots \lfloor D_i/T_j \rfloor\} \cup \{D_i\} \tag{4}$$

Thus, checking task τ_i feasibility requires to verify if:

$$\min_{t \in S_i} \left(\frac{W_i(t)}{t} \right) \leq 1 \tag{5}$$

As a consequence if one instant $t \in S_i$ satisfies $W_i(t) \leq t$ then τ_i is feasible and no more time instant has to be checked to decide the feasibility of τ_i. According to Theorem 1, a practical implementation of such a test usually requires to check only a subset of S_i. But, the computational complexity of this algorithm depends on the ratio: D_i/T_j. As a consequence, the algorithm runs in pseudo-polynomial time. In [5], an improvement of this test is presented, but this algorithm is still running in pseudo-polynomial time.

In practice, the algorithms proposed by [10] and [11] can lead to a quite different number of iterations. But, their pseudo-polynomial complexities are not acceptable to define an on-line admission controller and furthermore, the numbers of iterations are too dependent on task parameters.

3.2 Approximation algorithms

An approximation algorithm is a polynomial time algorithm that is used to solve efficiently NP-hard (optimization) problems. There exist several ways to define a solution in polynomial time with performance guarantees in comparison with an exact algorithm (always computing the optimal value of an optimized function). Let A be an approximation algorithm and OPT be an exact algorithm. For any instance, vales returned by A or OPT for a given instance I are respectively denoted $A(I)$ and $OPT(I)$. The (relative) performance guarantee of the algorithm A is defined by a *ratio* $A(I)/OPT(I)$ while considering any possible instance I of a given optimization problem. The competitive ratio of A is thus defined by: $r_A = \inf_{anyI} A(I)/OPT(I)$, where I is a instance of the considered problem. Thus, the ratio defines the worst-case performance guarantee while considering all possible instances of the optimization problem. An approximation algorithm is a polynomial time algorithm having a ratio bounded by a constant. Note that an algorithm A is optimal (i.e., always leads to the optimal value of the optimized objective function) if, and only if, $r_A = 1$.

A approximation scheme is a parametric approximation algorithm (thus running in polynomial time) that takes an input problem instance and an error bound $0 < \epsilon < 1$. The error bound defines an *accuracy* input parameter. The ratio of an approximation scheme must be defined as follows: $r_A \leq 1 + \epsilon$. A Polynomial-Time Approximation Scheme (PTAS) is an algorithm that runs in polynomial time in the length of the input. A fully polynomial time algorithm (FPTAS) is a PTAS that satisfies an additional condition: it is also polynomial in $1/\epsilon$. That is the best result that can be achieved to solve an NP-hard problem. Only few optimization problems admit FPTAS.

Since few years, approximation algorithms gain a great interest in the real-time research community. To the best of our knowledge, no approximation algorithm has been proposed to calculate approximate response times of tasks with performance guarantees (we shall provide such a result in the next section). Nevertheless, checking feasibility is not an optimization problem, but only a *decision problem*. As a consequence, approximation algorithm principles cannot be exploited without revisiting their definition. In fact, several frameworks have been proposed to reuse approximation algorithm concepts and thus defining several approaches to perform *approximate schedulability analysis*:

- Chakraborty *et al.* [7] proposed approximation scheme that always provide the good answer if the task system is not schedulable, but can give a wrong answer in the other case with a bounded error ϵ (i.e., it returns not schedulable whereas the task system is feasible).
- Based on the results obtained by [1] for EDF, Fisher and Baruah [9] proposed another definition: if a task system is stated as infeasible then it is really not feasible on a slower processor (with speed $1 - \epsilon$).

Even if these two frameworks are different, the performance guarantee of an approximation algorithm is obtained by bounding the error on the exact value of the function $rbf(t)$ and its approximate version. We only present the function proposed in [9] that is directly linked (and will be reused) to the problem we cope with in this paper.

The function $rbf_i(t)$ is a non-decreasing step function. The number of steps is not bounded by any polynomial function in the size of task parameters. One way to define an polynomial-time approximation scheme is to consider a limited number k of steps (polynomially bounded in the number of tasks in the system) and then to use a linear function to define an upper bound of $rbf_i(t)$. The number of steps that will be considered while computing the approximate request bound function is defined as follow:

$$k = \lceil 1/\epsilon \rceil - 1 \tag{6}$$

Then, the approximate demand bound function $\overline{rbf}_i(t)$ is defined by considering the first k steps of $rbf_i(t)$:

$$\begin{aligned} \overline{rbf}_i(t) &= rbf_i(t) && \text{if} \quad t \leq (k-1)T_i \\ &= C_i + t\frac{C_i}{T_i} && \text{otherwise} \end{aligned}$$

Then, the *approximate cumulative request bound function* is defined by:

$$\overline{W}_i(t) = C_i + \sum_{j=1}^{i-1} \overline{rbf}_j(t) \tag{7}$$

To complete the test, Fisher and Baruah use exactly the same principle than those proposed by Lehoczky et al. [11] but defining a testing set, but having a polynomial number of entries according to the input task system size and the accuracy parameter (ϵ):

$$\overline{S}_i = \{bT_j | j = 1 \ldots i-1, b = 1 \ldots k\} \cup \{D_i\} \tag{8}$$

where k is defined in Equation 6. A basic implementation of this approximate schedulability test leads to an $O(n^2/\epsilon)$ algorithm [9]. Clearly, if ϵ is closed to 0, then the number of iterations performed by the algorithm is quite huge and should not be acceptable into an on-line admission controller (even if it is a polynomial time algorithm from a theoretical point of view). Thus, numerical experimentations are necessarily required according

to the application in order to define a *good* value for ϵ for the considered task systems. Note that such an approximation scheme has been extended to task systems with arbitrary deadlines in [8] (i.e., periods and deadlines are not related).

4 New Algorithms

We first present some computation complexity results, and then, we propose three new polynomial time algorithms for checking the feasibility of fixed-priority tasks with constrained-deadlines.

4.1 Computational complexity of feasibility problems

The computation complexity theory classifies decision problems according to their internal complexity. Checking feasibility of a task system is obviously a decision problem. Nevertheless, no computational complexity result is known for the feasibility problem related to the studied task model. This decision problem is not known $\mathcal{NP}$-hard, nor belonging to $\mathcal{NP}$. Before defining approximation algorithms, we first state a computational complexity result for fixed-priority tasks, then we recall that verifying that tasks scheduled under EDF (Earliest Deadline First) leads to a very different class of problems in the computational complexity theory, unless $\mathcal{P}$ equals $\mathcal{NP}$.

Theorem 2 *Checking deadlines for synchronous fixed-priority tasks having constrained-deadlines is a decision problem belonging to $\mathcal{NP}$.*

Proof: In order to show the problem to be in $\mathcal{NP}$, we have to prove that a task set can be decided feasible using a polynomial time non-deterministic algorithm. If the non-deterministic part of such an algorithm "guesses" a scheduling point t in the testing set defined in Equation 4 for checking the feasibility of a task τ_i, then a necessary and sufficient condition according to Theorem 1 is: $W_i(t) \leq t$. Such a test is done in polynomial time since the Equation 2 is computable in linear time. Repeating this principle for every task leads to a polynomial time test (using a non deterministic algorithm). Thus, the considered feasibility problem belongs to $\mathcal{NP}$. □

Note that the same feasibility problem will be in co-$\mathcal{NP}$ if we consider an EDF scheduler (thus, one can checked in polynomial time for a given date t that a task system is infeasible, but checking that a task system is feasible requires more than a polynomial amount of time [3]).

Theorem 3 *[3] Checking deadlines for synchronous tasks having constrained deadlines, to be scheduled under EDF, is a decision problem belonging to co-$\mathcal{NP}$.*

Proof: In order to show the problem to be in $\mathcal{NP}$, we have to prove that a task set can be decided infeasible using a polynomial time non-deterministic algorithm. If one "guesses" a time instant t, then for checking that task τ_i is not schedulable, it is necessary and sufficient to check there exist a time instant t such that : $dbf(t) > t$ (see [3] for the definition of the demand bound function $dbf(t)$). This is done in polynomial time since $dbf(t)$ is computable in linear time. Thus, a non-deterministic algorithm can check the infeasibility of a task system in polynomial time. Thus, the considered feasibility problem belongs to co-$\mathcal{NP}$. □

Algorithm 1: Linear Time Approximation Algorithm

Data : $n, (\tau_1, \ldots, \tau_n)$
$\tilde{R}_1 = C_1$;
$r = s = 0$;
for *i=2..n* **do**
  $r = r + C_{i-1}/T_{i-1}$;
  $s = s + C_{i-1}$;
  $\tilde{R}_i = (s + C_i)/(1 - r)$;
end
return $(\tilde{R}_1, \ldots, \tilde{R}_n)$;

Next, we present several polynomial time algorithms to check the feasibility of fixed-priority tasks with constrained deadlines.

4.2 A linear time approximation

Consider the workload function stated in Equation 2 and let R_i^* be the exact worst-case response time of τ_i. In order compute an approximate worst-case response time (i.e., an upper bound), one can relax the integral values of t/T_j while computing the interference of any higher priority task τ_j. That is to say:

$$R_i^* \leq C_i + \sum_{j=1}^{i-1} \left(1 + \frac{R_i^*}{T_j}\right) C_j$$

For obtaining a lower bound of the worst-case response time of τ_i:

$$R_i^* \geq C_i + \sum_{j=1}^{i-1} \frac{R_i^*}{T_j} C_j$$

Using the two previous inequations, we obtain:

$$\frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} \leq R_i^* \leq \frac{\sum_{j=1}^{i} C_j}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} \qquad (9)$$

We use such an upper bound to approximate the worst-case response time of τ_i. Then, these upper bounds of tasks can be used to define a linear time feasibility algorithm (i.e., running in $O(n)$, where n is the number of tasks) that computes response time upper bounds $\tilde{R}_i$ as presented in Algorithm 1.

We first establish a negative result concerning the performance guarantees of Algorithm 1 while considering any possible task systems with constrained-deadline. Then, we shall show that under a simple assumption that Algorithm 1 has a bounded performance guarantee.

Theorem 4 *Let R_i^* be the exact worst-case response of τ_i and $\tilde{R}_i$ be the upper bound computed by Algorithm 1, then the ratio $\overline{R}_i/R_i^*$ is not bounded (i.e., Algorithm 1 has no performance guarantee).*

Proof : Consider the following task system with two tasks: $\tau_1(1-\epsilon,1,1)$ and $\tau_2(K\epsilon,K,K)$, where ϵ satisfies $0<\epsilon<1$ and K is an arbitrary integer number such that $K>1$. Note that periods are proportional, thus a necessary and sufficient condition for the task system to be schedulable under the Rate Monotonic scheduling rule is $C_1/T_1+C_2/T_2\leq 1$. The utilization factor is:

$$U=\frac{C_1}{T_1}+\frac{C_2}{T_2}=(1-\epsilon)+\frac{K\epsilon}{K}=1$$

Thus, the task system is schedulable under Rate Monotonic and the exact worst-case response times and those obtained by Algorithm 1 are:

$$\begin{aligned} R_1^* &= \overline{R}_1=1-\epsilon \\ R_2^* &= K \qquad\qquad \overline{R}_2=\frac{1+(K-1)\epsilon}{\epsilon} \end{aligned}$$

Thus, the worst-case performance guarantee of Algorithm 1 is obtained while considering τ_2:

$$\lim_{\epsilon\to 0}\frac{\overline{R}_2}{R_2^*}=\lim_{\epsilon\to 0}\frac{1}{K\epsilon}+\frac{(K-1)}{K}=\lim_{\epsilon\to 0}\frac{1}{\epsilon}=\infty$$

□

A similar result can be achieved for the performance guarantee of the lower bound $\tilde{R}_i$ that is defined by: $\max_{i=1\ldots n}\frac{R_i^*}{\tilde{R}_i}$.

Theorem 5 *Let R_i^* be the exact worst-case response of τ_i and $\tilde{R}_i$ be the lower bound computed in Equation 9, then the ratio $R_i^*/\tilde{R}_i$ is not bounded (i.e., the lower bound has no performance guarantee).*

Proof : Consider the following task system with two tasks: $\tau_1(K,2K,2K)$ and $\tau_2(\epsilon,2K,2K)$, where ϵ satisfies: $0<\epsilon<1$ and K is an arbitrary number such that $K>1$. Note that periods are equal. Thus, it is quite easy to see that the Rate Monotonic scheduling algorithm leads to a feasible schedule.

The exact worst-case response time for task τ_2 is $R_2^*=K+\epsilon$. And the lower bound defined by Equation 9 is:

$$\tilde{R}_2=\frac{\epsilon}{1-\frac{K}{2K}}=2\epsilon$$

As a consequence, we verify:

$$\lim_{\epsilon\to 0}\frac{R_2^*}{\tilde{R}_2}=\lim_{\epsilon\to 0}\frac{K+\epsilon}{2\epsilon}=\infty$$

As a consequence, such a lower bound has no performance guarantee. □

We now prove that if task parameters satisfy a simple condition, then Algorithm 1 is an approximation algorithm.

Theorem 6 *If we assume that there exists a constant K such that $K>\sum_i(C_i)/\min_i C_i$ for any task system, then Algorithm 1 has a performance guarantee not greater than K.*

Proof: Starting from equation 9:

$$\frac{C_i}{1-\sum_{j=1}^{i-1}\frac{C_j}{T_j}}\leq R_i^*\leq\frac{\sum_{j=1}^{i}C_j}{1-\sum_{j=1}^{i-1}\frac{C_j}{T_j}}$$

Thus,

$$\frac{\overline{R}_i}{R_i^*}\leq\frac{\sum_{j=1}^{i}C_j}{C_i}\leq K$$

Thus, under the assumption, Algorithm 1 is a K-approximation. □

Thus, one can hope that Algorithm 1 is quite interesting for evaluating task systems having small tasks with similar worst-case execution times. For such systems, Algorithm 1 provides an efficient $O(n)$-time approximation algorithm for computing worst-case response times of tasks. But, when there are high variations on task lengths, then the algorithm cannot be efficient, since the constant K can be a huge number.

Using a similar argument, we define an assumption such that the lower bound defined in Equation 9 has a performance guarantee in comparison with exact worst-case response times of tasks.

Theorem 7 *If we assume that there exists a constant K such that $K>\sum_i(C_i)/\min_i C_i$ for any task system, the lower bound of the worst-case response time defined in Equation 9 has a performance guarantee not greater than K.*

Proof: As in the previous proof, starting from equation 9 we directly obtain:

$$\frac{R_i^*}{\tilde{R}_i}\leq\frac{\sum_{j=1}^{i}C_j}{C_i}\leq K$$

□

We investigate next sections, two new approximation algorithms requiring more computational efforts (i.e., that are not running in linear time).

4.3 A deterministic approximation algorithm

The algorithm proposed by Joseph and Pandya [10] is based on computing the smallest fixed-point of Equation 3. The algorithm runs in pseudo-polynomial time since the number of iterations is not known to be bounded by a polynomial number in the task system size.

A simple way to achieve a bounded number of iterations is to stop computations at most after k iterations. If the smallest fixed-point is reached before k iterations then the algorithm returns the exact worst-case response times.

Algorithm 2: Deterministic Approximation Algorithm

Data : $n, (\tau_1, \ldots, \tau_n), \epsilon$
$(\overline{R}_1, \ldots, \overline{R}_n)$=Algorithm1$(n, (\tau_1, \ldots, \tau_n)$;
$k = \lceil \frac{1}{\epsilon} \rceil - 1$;
$r = s = 0$;
for *i=2..n* **do**
  $l = 0$;
  $t = C_i$;
  while *(*$t < W_i(t)$ **and** $l < k$ **and** $t \leq D_i$*)* **do**
    $l = l + 1$;
    $t = W_i(t)$;
  end
  if $t = W_i(t)$ **then**
    $\overline{R}_i = t$;
  end
end
return $(\overline{R}_1, \ldots, \overline{R}_n)$;

Algorithm 3: Randomized Approximation Scheme

Data : $n, (\tau_1, \ldots, \tau_n), \epsilon$
$k = \lceil \frac{1}{\epsilon} \rceil - 1$;
$Feasible$=True;
$i = 1$;
while $i \leq n$ **and** $Feasible$ **do**
  f_i=False;
  $j = 1$;
  while $j \leq k$ **and not** f_i **do**
    Choose randomly a time $t \in S_i$;
    if $\overline{W}_i(t) \leq t$ **then**
      f_i=True;
    end
    $j = j + 1$;
  end
  $i = i + 1$;
  $Feasible = f_i$;
end
return *Feasible* ;

Otherwise, it returns the upper bound presented in the previous section (i.e., using Algorithm 1):

$$\overline{R}_i = \frac{\sum_{j=1}^{i} C_j}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}}$$

The number k is a parameter that must be based on an *accuracy* constant: $\epsilon, 0 < \epsilon < 1$. As Fisher and Baruah, we define it as follows[1]:

$$k = \left\lceil \frac{1}{\epsilon} \right\rceil - 1$$

Algorithm 2 presents the pseudo-code of the deterministic approximation algorithm. In order to improve the algorithm efficiency, we first run Algorithm 1 that defines initial values of approximate worst-case response times. The algorithm runs in $O(\frac{n^2}{\epsilon})$ since the workload $W_i(t)$ is computed in $O(n)$.

As a direct consequence of the result presented in Theorem 4, we can establish that Algorithm 2 is an approximation algorithm under the following condition: there is a constant K such that $K > \sum_i (C_i) / \min_i C_i$ for any task system.

Theorem 8 *If we assume that there exists a constant K such that $K > \sum_i (C_i) / \min_i C_i$ for any task system, then Algorithm 2 has a performance guarantee not greater than K.*

4.4 A randomized approximation scheme

The last proposed algorithm is based on the Lehoczky, Sha and Ding's feasibility test. This algorithm checks the processor demand using a testing set S_i for any task τ_i. The size of such a set is not known to have a polynomial number of items. The feasibility test enumerates the testing set and stops when a time t that verifies $W_i(t)/t \leq 1$. The worst-case behavior of such a test is achieved when all items in the testing set have been checked. The number of iterations for analyzing task τ_i is at most $\sum_{j=1}^{i} \left\lfloor \frac{D_i}{T_j} \right\rfloor$. From the implementation point of view, the order in which items in S_i's are enumerated is not important.

A simple way to define an approximation scheme based on the Lehoczky, Sha and Ding's exact feasibility test is to limit the size S_i while checking the feasibility of task τ_i. Once again, we fix such a number using an accuracy constant $\epsilon, 0 < \epsilon < 1$ as follows: $k = \lceil \frac{1}{\epsilon} \rceil - 1$.

In order to ensure the algorithm to be an approximation scheme we also have to use the approximate workload $\overline{W}_i(t)$ (i.e., Equation 7) rather than the exact workload $W_i(t)$ (i.e., Equation 2). If such a function is not used, we cannot ensure that the algorithm has competitive ratio bounded by a constant (i.e., to ensure that is an approximation algorithm). As a consequence, Algorithm 3 is a simple randomized version of the algorithm proposed in [9].

We define a randomized approximation scheme by enumerating randomly at most k items in each S_i with the same probability (i.e., a uniform law). While considering such items if no of them leads to a positive answer, then we state the task system to be infeasible. The corresponding algorithm has a computational complexity of $O(\frac{n^2}{\epsilon})$. If ϵ tends to 0, then the randomized approximation scheme has the same behavior than Lehoczky, Sha and Ding's exact feasibility test.

[1]We same the same definition of k in order to allow comparisons of algorithms in the Section dedicated to numerical experimentations.

Algorithm Names	Authors
LSD89	Lehoczky, Sha and Ding, 1989
JP86	Joseph and Pandya, 1986
FB05	Fisher and Baruah, 2005
UB	Section 4.2
DET	Section 4.3
RAND	Section 4.4

Table 1. Algorithm name abbreviations used in the paper

5 Numerical results

We first describe the simulation environment and then numerical results.

5.1 Experimentation environment

We compared all presented methods (see Table 1 for the complete list). Task systems are randomly generated in order to achieved a given processor workload. The maximum worst-case execution time is fixed to 100 units of time and deadlines are constrained for all tasks (i.e., $D_i \leq T_i, 1 \leq i \leq n$). The simulator parameters are:

- the processor workload are 0.5 and 0.9,
- the number of tasks are between 2 and 50 tasks in every task systems,
- considered epsilon values are from 0.01 to 0.46 with a step 0.05 (Note that if $\epsilon \geq 0.5$, then $k = \lceil \frac{1}{\epsilon} \rceil - 1$ is always equal to 1).

For every value of these parameters, 25 task systems have been randomly generated and all methods have been run and compared. In the following, algorithms will be denoted as indicated in Table 1.

We only focus on two output parameters:

- the number of validated task systems,
- the number of iterations performed by the algorithms, which indicate the number of times that the workload function is computed during the test (i.e., Equation 2).

We are aware that simulation environment can have biasing effects on results [4], nevertheless every simulation results is always valid only within the confine of the stochastic model defined in the simulator. We note that results presented in the next section are valid for our simulation environment, and only for it.

5.2 Simulation Results

Figures 1 and 2 present numerical results for task systems having a processor utilization equal to 0.5. Figure 1 gives the number of validated task systems (i.e., the output status of the test is *feasible*). The algorithm

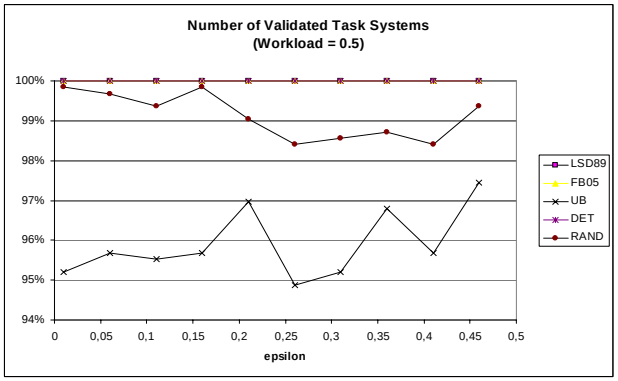


Figure 1. Number of validated task systems: all methods achieved good performances (Workload 0.5)

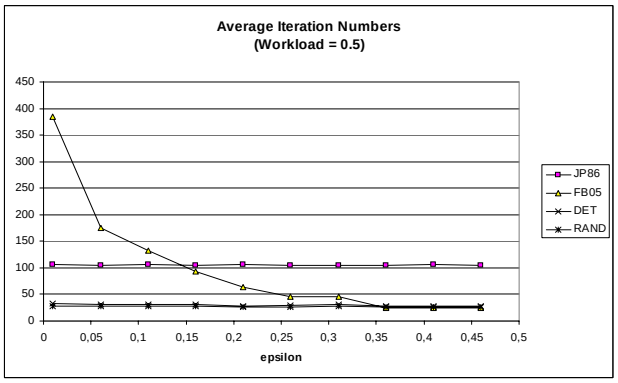


Figure 2. Iteration numbers according to epsilon values (Workload 0.5)

LSD89 is used as a reference. As we can see, all methods achieved good performances. More precisely, Figure 1 shows that LSD89, FB05 and DET have equivalent results. The randomized algorithm has lower performances in comparison with DET. The linear time approximation algorithm (based on the upper bound presented in Section 4.2) leads to acceptable results since in more than 94 percent it achieves a positive result (i.e., the same result than an exact feasibility test).

The average iteration number of JP86 remains constant for every epsilon value, because epsilon is not an input parameter for that algorithm. One can note that algorithm FB05 requires more iterations than JP86 for task sets when small epsilon values are considered. But, when epsilon is up to 0.25, then FB05 needs the same average iteration numbers than the other approximation algorithms and furthermore achieves better results.

Figures 3 and 4 present the same kink of results for a processor workload equal to 0.9. Clearly from Figure 3,

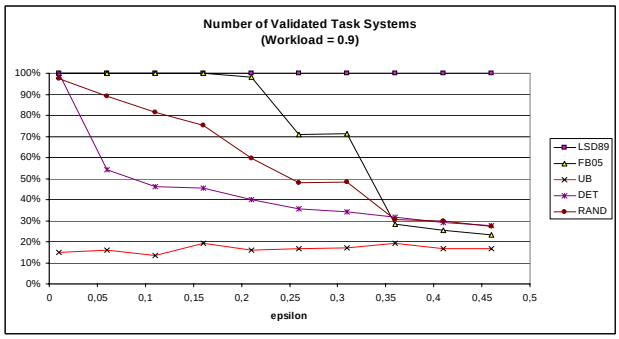


Figure 3. Number of validated task systems: all methods achieved good performances (Workload 0.9)

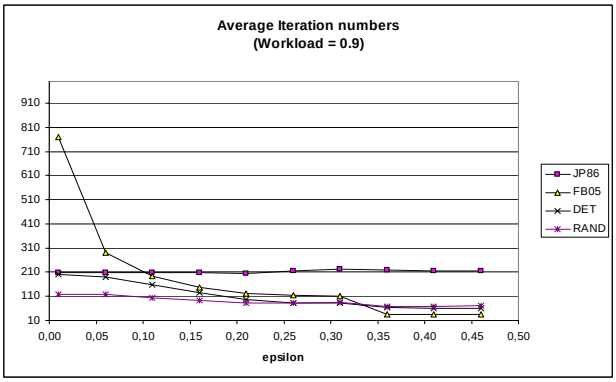


Figure 4. Iteration numbers according to epsilon values (Workload 0.9)

FB05 is more competitive in comparison to other approximation algorithms. But, when epsilon is up to 0.25 then its performance against an exact feasibility test decreases drastically to 30 percent of positive results. One can note that the linear approximation algorithm is not competitive enough when the processor utilization is high.

According to Figure 4, the average iteration numbers have slopes in comparison with numerical results achieved for a processor utilization equal to 0.5. Once again, FB05 becomes interesting for values around 0.25 since it requires the same average number of iterations and achieves better results.

As a conclusion, we must say that FB05 is better than the proposed polynomial time approximation algorithms for epsilon values near to 0.25. For optimizing the processor utilization, we conclude that FB05 is the better algorithm among those proposed here, but the accuracy parameter ϵ must be carefully chosen in order to control the quality of results. When the processor utilization is not high, then admission control can be efficiently done using the linear time approximation algorithm (denoted UB), that has been presented in Section 4.2.

6 Conclusion

Efficient feasibility tests are required for implementing an admission controller for large real-time systems. We focused on feasibility tests with a polynomial time complexity for defining efficient admission controllers. We presented computational complexity results and compared several approximate feasibility tests. We shown the checking the feasibility of tasks with constrained-deadlines belongs to $\mathcal{NP}$ when tasks have fixed-priorities, whereas the same problem with EDF belongs to co-$\mathcal{NP}$. We proposed three simple approximate algorithms and compared them with exact feasibility tests [10, 11] and one existing polynomial time approximation scheme [9].

Numerical results shown that if the processor utilization is not high, then admission control can be efficiently done in linear time. When the processor utilization increases, then we can use the Fisher and Baruah's fully polynomial time approximation scheme. According to our results, it could also interesting to evaluate exact feasibility tests since in many situations they can be as powerful than polynomial time approximation schemes even if their worst-case computational complexities lead to pseudo-polynomial time algorithms. But, there is still a small gap between polynomial time admission control and exact tests based pseudo-polynomial time algorithms.

The fully polynomial-time approximation scheme proposed in [9] is to decide if a given task system is feasible on a unit speed processor. But, it is not the case then the test ensures that the task system is infeasible upon a slower processor (the slowdown is related to the accuracy parameter). Thus, we want to use such techniques in order to define an efficient scheduling algorithm for tasks to be run upon a variable speed processor for power aware computer systems.

We must also conclude that the existence of approximation algorithms (or better approximation schemes) for computing worst-case response times of tasks is still an important open issue. Most of known papers do not cope with any performance guarantee in comparison with exact values of worst-case response time. Thus, we think that for most real-world systems validated with such schedulability tests lead to oversizing the real-time system features.

References

[1] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. *proc. Euromicro Int. Conf. on Real-Time Systems (ECRTS'04)*, pages 187–195, 2004.

[2] N. Audsley, A. Burns, M. Richardson, and A. Wellings. Hard real-time scheduling: the deadline monotonic approach. *proc. 8th IEEE Workshop on Real-Time Operating Systems and Software, Atlanta*, pages 127–132, 1991.

[3] S. K. Baruah, R. R. Howell, and L. E. Rosier. Algorithms and complexity concerning the preemptive scheduing of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

[4] E. Bini and G. Buttazzo. Biasing effects in schedulability measures. *Euromicro Int. Conf. on Real-Time Systems (ECRTS'04)*, 2004.

[5] E. Bini and G. Buttazzo. Schedulability analysis of periodic fixed-priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, November 2004.

[6] R. Bril, W. Verhaege, and E. Pol. Initial values for on-line response time calculations. *proc. Int Euromicro Conf. on Real-Time Systems (ECRTS'03), Porto*, 2003.

[7] S. Chakraborty, S. Kunzli, and L. Thiele. Approximate schedulability analysis. *proc. 23rd Int. Symposium on Real-Time Systems (RTSS'02)*, 2002.

[8] N. Fisher and S. Baruah. A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. *proc. Euromicro Int. Conf. on Real-Time Systems (ECRTS'05)*, pages 117–126, July 2005.

[9] N. Fisher and S. Baruah. A polynomial-time approximation scheme for feasibility analysis in static priority systems with bounded relative deadlines. *proc. Real-Time and Embedded Systems (RTS'05), Paris*, 2005.

[10] M. Joseph and P. Pandya. Finding response times in a real-time systems. *The Computer Journal*, 29(5):390–395, 1986.

[11] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. *proc. Real-Time System Symposium (RTSS'98)*, pages 166–171, 1989.

[12] J. C. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[13] M. Sjodin and H. Hansson. Improved response time analysis calculations. *proc. IEEE Int Symposium on Real-Time Systems (RTSS'98)*, 1998.

Feasibility Conditions with Kernel Overheads for Periodic Tasks with Fixed Priority Scheduling on an Event Driven OSEK System

Franck Bimbard
Cedric/CNAM
292 Rue St Martin FR-75141
75007 Paris Cedex 03
bimbard@ece.fr

Laurent George
ECE, LACSC
53, rue de Grenelle
75007 Paris
lgeorge@ieee.org

Abstract

In this paper we show how to extend classical real-time feasibility conditions for preemptive fixed priority scheduling of periodic tasks to consider kernel overheads. The kernel considered in this paper is the event driven OSEK kernel. We identify the sources of overhead that influence the response time of the tasks. In such a system the overheads are due to the context switching and the mechanisms used to activates/terminates and reschedules tasks and to the granularity of the periodic timer used to implement the periodic task model. We show how to take into account those overheads in the classical feasibility conditions. We compare the theoretical modified feasibility conditions with kernel overhead to the results obtained on a real implementation. We show that the kernel overheads cannot be neglected and that the theoretical results are valid and can be used for a real-time dimensioning of an OSEK system.

1 Introduction

Fixed priority scheduling in real-rime systems has been extensively studied in the last thirty years. The task model considered in this paper is the periodic model. The problem is to schedule a periodic task set $\tau = \{\tau_1, ..., \tau_n\}$ with a preemptive fixed priority scheduling. A periodic task τ_i is defined by:

- C_i : The worst case execution time (WCET).
- T_i : The period of the task.
- D_i : The deadline constraint (a task released at time t must be executed by $t + D_i$).
- P_i : The fixed priority (priority 0 is the lowest priority).

The starting point for preemptive fixed priority scheduling is in [7] that proposed a simple polynomial time sufficient feasibility condition for the Rate Monotonic (RM)algorithm. The Feasibility Conditions (FC) have then been extended by [4] in the case where $\forall i, D_i \leq T_i$ and by [8], [3] for tasks with no obvious relation between D_i and T_i valid for all the tasks. The feasibility conditions are based on the worst case response time computation for any periodic task. The scheduling model used in the FC is the event driven model. The FC are pseudo-polynomial but do not consider kernel overheads. The preemption cost is considered either null or is included as an extra duration in the WCET of the tasks, leading to imprecise FC.

In the time driven model, [8] showed how to take into account the cost of the scheduler. The scheduler behaves as periodic tasks with a preemption cost that can be taken into account in the feasibility conditions.
Yet, in the event driven model, the solution to increase the durations of the tasks to take into account kernel overheads can be very pessimistic [5] as it always considers a worst case maximum number of preemptions for a task.

In this paper, we consider an event driven implementation of OSEK. OSEK standard has been initiated in 1993 by several german companies like BMW, Bosch, Daimler-Benz, Opel, and Siemens. The objectives were to save money, with a standard OS and to increase the software compatibility between manufacturers by using standard interfaces for all processors and network protocols. The OSEK operating system offers the necessary functionality to support event driven control. Yet, the current approach used for system dimensioning leads to overestimate the overhead of the operating system, without a precise analysis

of the operating system leading to a pessimistic dimensioning. e.g. developers generally limit the CPU of the tasks to allocate the rest of the CPU to the operating system without a good characterization of the OS. In this paper, we propose to characterize the overheads of an OSEK kernel to propose a deterministic system dimensioning.

We study the sources of kernel overheads for the fixed priority scheduling of preemptive periodic tasks in the case where $\forall i, D_i \leq T_i$. We show how to integrate the overheads of the kernel in the classical theoretical feasibility conditions and show that this extension is valid for a real-time dimensioning. In section 2, we recall the principles of an OSEK kernel. We then describe in section 3 the environment used and the sources of kernel overhead. We identify different sources of overhead i.e. the time granularity chosen for the periodic timer used for the periodic model implementation may introduce a variation in the actual period chosen OSEK. We then focus on the task activation/termination and on the context switch overheads. In section 4, we show how to integrate the identified kernel overheads in classical theoretical real-time analysis. In section 5, we propose to compare the theoretical worst case response times to kernel overhead with the experimental results obtained with a real OSEK implementation showing that our analysis is relevant for system dimensioning. Finally, we conclude.

2 OSEK characteristics

In subsection 2.1, task management is exposed. The scheduling policy is detailed in subsection 2.2. Then, the alarm mechanism, used to implement the periodic task model is described in subsection 2.3.

2.1 Task management

Two different task concepts are provided by the OSEK operating system: basic tasks, and extended tasks. Extended tasks are distinguished from basic tasks by being allowed to wait for events for communications between tasks and resources management. The OSEK operating system is responsible for saving and restoring task context in conjunction with task state transitions whenever necessary. We are interested in this paper in the overheads due to the switching task mechanism and to the alarms treatment used to implement the periodic task model. We therefore focus on basic tasks which have three possible states:

- **Running :** In the running state, the CPU is assigned to the running task, so that its instructions can be executed. Only one task can be in this state at any time, while all the other states can be adopted simultaneously by several tasks.
- **Ready :** All functional prerequisites for a transition into the running state exist, and the task only waits for election of the processor.
- **Suspended:** In the suspended state the task is passive and can be activated.

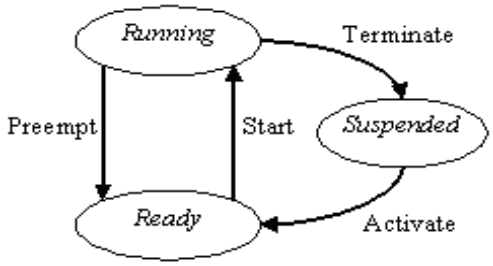


Figure 1. Basic task state model

We now describe the transitions between the states exposed in figure 1:

Transition	Former state	New state	Description
Activate	Suspended	Ready	A new task is set into the ready state by the service ActivateTask.
Start	Ready	Running	A ready task selected by the scheduler is executed.
Preempt	Running	Ready	The scheduler decides to start another task. The running task is put in the ready state.
Terminate	Running	Suspended	The running task completes and self-suspends by the service TerminateTask.

Table 1. States and status transitions for basic tasks

In the OSEK operating system, a task can terminate by calling the service TerminateTask. Ending the task without a call to TerminateTask is strictly forbidden and causes undefined behavior. Task activation is performed using the operating system service ActivateTask. After activation the task is ready to execute from the first statement. The following figure illustrates the interactions between two tasks suspended at time 0 and the evolution of their states with time. Task τ_1 is set to the running state and is later preempted by a task τ_2, of higher priority.

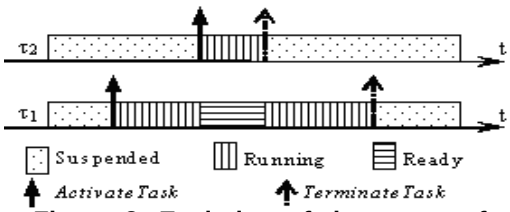


Figure 2. Evolution of the states of two basic tasks

2.2 Scheduling policy

In the OSEK operating system, there are three different scheduling policies: full preemptive, non preemptive, and mixed preemptive. In the latter case, a system is composed of both preemptive and non-preemptive tasks. In this paper, we consider Full preemptive scheduling as it maximizes the kernel overheads. Full preemptive scheduling means that a task which is presently running may be put into the ready state, as soon as a higher priority task has got ready. The preempted task context is saved so that it can be resumed at the location where it was preempted.

2.3 Alarm mechanism

The alarm mechanism allows implementing the periodic task model. Each alarm has two parameters: the time where it starts for the first time, and its period. Each time an alarm occurs, it activates its associated task. This mechanism uses the OSEK time base to count the time which is different from the CPU time (clock cycle). This OSEK time base is also called "Tick Time".
The OSEK time base has a time granularity of period T_{tick}, multiple of the clock cycle. The use of a timer permits to the processor to create a periodical interruption. The CPU load due to this interruption is discussed in subsection 3.2.

3 Kernel overheads

Because several OSEK versions exist we have to describe our development environment. Our OSEK operating system is based on the OSEK-OS-specification version 2.2 [2] and is provided by Vector Corp. Our target device is a dsPIC30F6014 which is provided by Microchip Corp. and excited by a quartz at 7,3728 MHz. The integrated Phase Lock Loop multiplies this frequency by 16. According to the structure of dsPIC, the internal cycle time is equal to: $T_{cycle} = \frac{4}{16 \times 7372800} = 33,91ns$.
We propose different measurements to validate our tests and experimentations and measure the overheads due to OSEK. In subsection 3.1, we explain our measurement methods. Then, we describe the OSEK's overheads in subsection 3.2. After which the overheads measurements are shown in subsection 3.3.

3.1 Measurements methods

We have done two kinds of measurements. We first determine the influence of the Tick Time on both the worst case response times and the actual values of the periods chosen by the kernel. Then we study the influence of the kernel on the worst case response times of the tasks.
The worst case response times of the tasks depends on the WCET of the tasks. To reduce the uncertainty of the WCET determination, each task is only composed of a simple empty loop which corresponds to "$for(i = 0; i < EndLoop; i++)$;". We use a standard simulation tool MPLab, which is also provided by Microchip Corp., to determine the WCET, depending on the value of the "$EndLoop$" constant. Thus, no uncertainty is introduced in the execution times of the tasks (we only want to measure the kernel overheads, not the WCET uncertainty).
We have integrated, in our OSEK's source code, a software which automatically measures the worst case response times of several activations for each task in the worst case scenario corresponding to lemma 2. This software uses a 16-bit timer for its measurements which are stored in RAM. Once all measurements are made, these results are transmitted via a serial port at 115200 bauds. Thus, the transmission does not influence the obtained results.

3.2 OSEK's overheads

Like any operating system, OSEK needs to generate its own time base, called the Tick Time, having a period T_{tick}. As described in subsection 2.3, this Tick Time is used to manage the alarms in charge of implementing the periodic task model. Under certain conditions, this management can add a CPU load which cannot be neglected. In paragraph 3.2.1, an experiment is done to show how this Tick Time influences the execution time of a single task. This experiment also illustrates the impact of this Tick Time actual value, chosen by OSEK, of the period of a task. In addition, because the switching task mechanism creates another CPU load, it can also affect the tasks when it is too frequent. In Paragraph 3.2.2, an experiment is done to show how this mechanism can also influence the duration of a task.

3.2.1 Tick Time

To illustrate the Tick Time influence, we consider the following example where a single task is run by the system. This task has a period equal to $100ms$ and a duration equal to $50ms$. We present its response time (r) for a given Tick Time period T_{tick}:

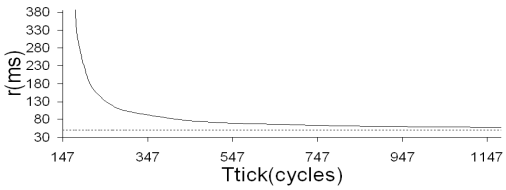


Figure 3. Comparison between the WCET (dotted curve) to the measured response time (continuous curve) of the task

The overhead of the OSEK operating system increases when T_{tick} decreases. Consequently, the task response time increases in the same way. As we can see on figure 3, the response time is multiplied by at least 2 or more when the Tick Time is below 295 cycles. The response time is strongly increased when the Tick Time is equal to 147 cycles.

We now, examine the maximum absolute error obtained on the considered periodic task for a given value of T_{tick}. As explained in subsection 2.3, the periodic model is based on an alarm mechanism which depends on the Tick Time. That is why, the period is more precise when Tick Time is multiple of it. The period of the task is rounded to the nearest multiple of T_{tick}. The actual period, denoted T_i^* for a task τ_i chosen by OSEK kernel is as follows:

$$T_i^* = \left(1 + \lfloor \frac{T_i - T_{tick}/2}{T_{tick}} \rfloor\right) T_{tick}$$

We can notice that its error never exceeds the duration $T_{tick}/2$. Our OSEK implementation uses a 16-bit timer to generate the Tick Time. Hence T_{tick} cannot exceed $2222\mu s$ $(65535 \times 33,91ns)$. Finally, in this interval, the Tick Time should be the greatest value multiple of the greatest common divisor between all periods of the tasks to cancel the imprecision $T_{tick}/2$.

3.2.2 Switching task mechanism

We now consider a preemptable task of WCET $50ms$. This task is interrupted by a higher priority task which is empty. The Tick Time is constant and equal to 2949 cycles. Consequently, the Tick Time constantly increases the response time of $3,5ms$. Thus, when the period of the higher priority task decreases, we observe the deviation which is only due to the switching task mechanism on figure 4.

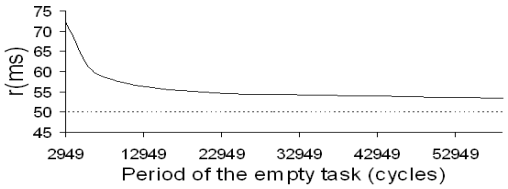


Figure 4. WCET (dotted curve) vs measured response time (continuous curve) of the task

We can see that the difference between theoretical and real durations increases when the period of the higher priority task decreases. In other words, switching task mechanism is non-negligible. We show how to take it into account in section 4.

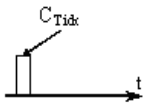
3.3 Overheads measurements

Now that the overheads are identified, the objective is to measure them in order to integrate them into the feasibility conditions. In subsection 3.3.1, we begin with an illustration of the measured durations. Then, the durations of the overheads are given in subsection 3.3.2.

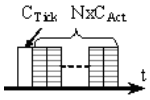
3.3.1 Illustration of the measured events

In this subsection, we determine the durations of the overheads previously exposed. Each time a Tick Time occurs, there are three possible scenarios:

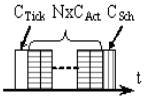
The Tick Time only manages the alarms but no task must be activated



The Tick Time manages the alarms and N tasks must be activated



The Tick Time manages the alarms, N tasks must be activated and one of them must be set to the running state (potentially stopping the execution of a running task)



The two last scenarios illustrate the cases where alarms occur and activate their associated task.

3.3.2 Measurements

The following table gives the notations used for each overheads:

Symbol	Description
C_{tick}	The execution time of the alarms management that occurs every T_{tick}.
C_{act}	The execution time required to activate a task. The state of an activated task is set to ready.
C_{sched}	The execution time required to schedule the task (if any) that has just been activated and that has the highest priority among all the tasks in the ready state.
C_{term}	The execution time to terminate the task and reschedule.

Table 2. Notations of the measurements

The following table gives the results of our measurements:

Symbol	WCET (cycles)
C_{tick}	179
C_{act}	393
C_{sched}	166
C_{term}	300

Table 3. Execution times of kernel overheads

4 Real-Time analysis with kernel overheads

We consider in this paper, that for any task τ_i, $D_i \leq T_i$. We now recall classical results in the uniprocessor context for real-time scheduling.

- A task is said to be non-concrete if its first release time is not known in advance. In this paper, we only consider non concrete first request times, as the activation request times are supposed to be unpredictable.

For any task τ_i,

- $hp(i)$ denotes the set of tasks having a higher or equal priority than τ_i except τ_i.
- $lp(i)$ denotes the set of tasks having a strictly lower priority than τ_i.
- Time is assumed to be discrete (task arrivals occur and task executions begin and terminate at clock cycles; the parameters used are expressed as multiples of the clock cycles); in [1] it is shown that there is no loss of generality with respect to feasibility results through restricting the schedules to being discrete, once the task parameters are assumed to be integers (multiples of the clock cycles) i.e. a discrete schedule exists, if and only if a continuous schedule exists.
- An idle time of level τ_i is defined as a time t, such that there are no tasks in $hp(i) \cup \tau_i$ released before time t pending at time t. An interval of successive idle times of level τ_i is called an idle period of level τ_i.
- A level τ_i busy period is defined as a time interval $[a, b)$, such that there is no idle time of level τ_i in $[a, b)$ and such that both a and b are idle times of level τ_i.
- The worst case busy period of level τ_i is the first busy period resulting from the scenario where all tasks τ_j in τ are first requested at time 0, and are periodic from 0.

Notice that this definition of the worst case level τ_i busy period is slightly different from the one proposed by [6] where only tasks in $hp(i) \cup \tau_i$ where considered. With kernel overhead, we show in theorem 1 that a task in $lp(i)$ can also have an influence on the worst case response time of a task τ_i.

- FP denotes any arbitrary Fixed Priority scheduling with Highest Priority First used on-line.
- $U_{no} = \sum_{i=1}^{n} \frac{C_i}{T_i}$ is the processor utilization factor, i.e., the fraction of processor time spent in the execution of the task set [7] without kernel overhead. An obvious necessary condition for the feasibility of any task set is $U_{no} \leq 1$ (this is assumed in the sequel).

Lemma 1 *[4] The worst-case response time r_i of a non-concrete periodic task τ_i (with $D_i \leq T_i, \forall i \in [1, n]$) scheduled FP is found in the worst case busy period of level τ_i and r_i is the solution of the following equation:*
$r_i = C_i + \sum_{j \in hp(i)} \lceil \frac{r_i}{T_j} \rceil$

Proof: The worst case busy period of level τ_i proposed in this paper is the same as the one proposed in [4] when we consider that all the execution times of the overheads are null. The equation of r_i is the one proposed in [4]. ■

Lemma 2 *The worst case response time of a periodic task with kernel overheads is found in the worst case busy period of level τ_i.*

Proof: For a task τ_i, the kernel overheads are maximized when the number of activations of tasks in $hp(i)$ and $lp(i)$ are maximized. Leading to the same worst case scenario for tasks in $hp(i)$ as in [4]. Notice that for tasks in $lp(i)$, we only have to consider the overheads of the tasks activations (achieved by the ActivateTask) whose number is maximized when tasks in $lp(i)$ are released as described in the worst case busy period of level τ_i. ■

Theorem 1 *The worst case response time r_i of a periodic task τ_i with the OSEK kernel overheads is the solution of the following equation:* $r_i = C_{act} + C_i + C_{term} + \sum_{j \in hp(i)} \lceil \frac{r_i}{T_j^*} \rceil (C_{act} + C_j + C_{term}) + \sum_{j \in lp(i)} \lceil \frac{r_i}{T_j^*} \rceil C_{act} + \max(\sum_{\tau_j \in hp(i)} \lceil \frac{r_i}{T_j^*} \rceil, 1) C_{sched} + \lceil \frac{r_i}{T_{tick}} \rceil C_{tick}$.

Proof: We consider a task τ_i released in its worst case busy period of level τ_i. The worst case response time of τ_i is composed of three terms:

- The first term is equal to: $C_{act} + C_i + C_{term} + \sum_{j \in hp(i)} \lceil \frac{r_i}{T_j^*} \rceil (C_{act} + C_j + C_{term})$. For any request of a task in $hp(i) \cup \tau_i$, the kernel must activate, run and terminate the task. The equation of lemma1 is updated accordingly.

- The second term is equal to: $\sum_{j \in lp(i)} \lceil \frac{r_i}{T_j^*} \rceil C_{act}$. For any task in $lp(i)$, the scheduler must at least activate the tasks according to their request times and put it in the ready state. The second part corresponds to the maximum duration required to activate the tasks in $lp(i)$.

- The third term is equal to: $\max(\sum_{\tau_j \in hp(i)} \lceil \frac{r_i}{T_j^*} \rceil, 1) C_{sched} + \lceil \frac{r_i}{T_{tick}} \rceil C_{tick}$. The scheduler is called to save/restore the context of task τ_i every time a task with a priority higher to τ_i is run. If $hp(i)$ is empty then the scheduler is called once for τ_i. The maximum number of scheduler calls is bounded by: $\max(\sum_{\tau_j \in hp(i)} \lceil \frac{r_i}{T_j^*} \rceil, 1) C_{sched}$. We must also take into account the alarm overhead. By assumption, all the alarms are managed by a periodic timer of period T_{tick} of duration C_{tick}. Leading to an overhead equals to: $\lceil \frac{r_i}{T_{tick}} \rceil C_{tick}$.

■

We now propose a sufficient feasibility condition for the dimensioning of our OSEK system.

Theorem 2 *A sufficient feasibility condition for the scheduling of periodic tasks scheduled with preemptive FP with the OSEK kernel overheads is (where T_α is the period of task with the maximum priority):*

$$\forall \tau_i \in \tau, r_i \leq D_i \quad (1)$$

$$U_{no} + \sum_{\tau_j \in \tau} \frac{C_{act} + C_{term} + C_{sched}}{T_j^*} \leq 1 \quad (2)$$

Proof: Equation (1) is straightforward. Equation (2) is clearly necessary as the kernel overheads add a duration $C_{act} + C_{term}$ to every tasks and the scheduler may be called for each task activation. ■

5 Experimentation

In this section, we experiment the previous theoretical results on a given task set. This task set is composed of five preemptive periodic tasks described in table 4.

Task	C_i (cycles)	D_i (cycles)	T_i (cycles)	P_i
τ_5	15920	55720	87560	4
τ_4	55720	318400	796000	3
τ_3	71640	636800	1273600	2
τ_2	398000	2547200	2308400	1
τ_1	796000	5094400	4855600	0

Table 4. Task Set

Note that the T_{tick} parameter has been chosen to be the small in order to increase the CPU load due to the Tick Time. This CPU load is, in worst case, equal to $C_{tick}/T_{tick} = 0,22$. Task τ_1 has the lowest priority and will be often preempted by higher-priority tasks which should largely increase its execution time.

We now compare for any task τ_i the theoretical response time without overhead r_i^0, the theoretical response time with kernel overheads r_i^1 and the measured response time r_i^2 in a real OSEK system. We now determine two significant ratios in table 6 for each task to characterize the performance of our theoretical worst case response time with kernel overheads.
The first column of results provides the percentage of deviation between the theoretical response time with kernel overheads and the real response time

Task	r_i^0 (cycles)	r_i^1 (cycles)	r_i^2 (cycles)
τ_5	15920	23721	20997
τ_4	71640	117642	112309
τ_3	159200	232303	223926
τ_2	652720	1014316	961556
τ_1	1838760	3611823	3461052

Table 5. Comparison between the different response times

Task	$(1 - \frac{r_i^2}{r_i^1}) \times 100$	$(1 - \frac{r_i^0}{r_i^1}) \times 100$
τ_5	11,48%	32,89%
τ_4	4,53%	39,10%
τ_3	3,61%	31,47%
τ_2	5,20%	35,65%
τ_1	4,17%	49,09%

Table 6. Performance theoretical worst case response time with kernel overheads

obtained with our OSEK system. The deviation is higher for the task τ_5 as it has the smallest execution time. The overheads are accordingly more important. The deviations obtained for the tasks τ_4, and τ_3 decrease as their execution times increase. The tasks τ_2, and τ_1 are more influenced by the context switching mechanism in OSEK.
In all cases, the deviations are small and enable to use the theoretical approach for a real-time dimensioning.

The second column of results shows the deviation between the theoretical approach with and without kernel overheads showing that the deviation ranges from 31,47% to 49,09%. Hence, the kernel overheads cannot be neglected and influences significantly the worst case response times of the tasks.

6 Conclusion

In this paper we have studied the impact of kernel overheads in the theoretical feasibility conditions of preemptive fixed priority scheduling of periodic tasks. We have considered an event driven OSEK system proposed by Vector Corp. We have identified the sources of kernel overheads and have shown how to integrate them in the worst case response times of the tasks, used by the feasibility conditions. We have shown in our experiments that the overestimation of the theoretical worst case response times does not exceed 11,48% and that the feasibility condition without kernel overhead are not valid.

References

[1] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *Real-Time Systems*, Vol. 2, pp. 301-324, 1990.

[2] V. Corp. Osek/vdx operating system v. 2.2.3 specification.

[3] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive scheduling real-time uniprocessor scheduling. *INRIA Research Report*, No. 2966, September 1996.

[4] M. Joseph and P. Pandya. Finding response times in a real-time system. *BCS Comp. Jour.*, 29(5), pp. 390-395,, 1986.

[5] D. Katcher, H. Arakawa, and J. Strosnider. Engineering and analysis of fixed-priority schedulers. *IEEE Trans. on Soft. Eng.*, 19, pp920-934, 1993.

[6] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Proceedings 11th IEEE Real-Time Systems Symposium*, pp 201-209, Dec. Lake Buena Vista, FL, USA, 1990.

[7] L. C. Liu and W. Layland. Scheduling algorithms for multi-programming in a hard real time environment. *Journal of ACM*, Vol. 20, No 1, pp. 46-61, January 1973.

[8] K. Tindell, A. Burns, and A. J. Wellings. An extendible Approach For Analysing Fixed Priority Hard Real-Time Tasks. *Real-Time Systems 6(2)v*, 1994.

Index