Computer Performance Evaluation. Modelling Techniques and Tools.
Lecture Notes in Computer Science, Springer Verlag, April 2002, to appear.

A Tool for Controlling Response Time in Real-Time Systems

Pascal Richard

Laboratory of Applied Computer Science
ENSMA - Téléport 2 - BP 40109
86961 Futuroscope Cedex, France
`pascal.richard@ensma.fr`

Abstract. In hard real-time systems, classical scheduling policies only cope with satisfaction of deadline constraints. In this paper, to every periodic task is associated a weight that models the importance of the task in terms of worst-case response time. These parameters are set off-line by the designers of the real-time software in order to control the quality of the on-line schedule. According to these weights, a set of feasible fixed-priorities are computed so that the mean weighted response time of the tasks is minimized. We propose a branch and bound algorithm to solve this problem. An example is completely detailed and numerical results on randomly generated problems are lastly presented to show the efficiency of the developed tool.

1 Introduction

In hard real-time systems, tasks periodically read data from sensors and respond to the controlled system through actuators. A response to an external event must occur within a fixed interval of time. So, every task occurrence (a job) is subjected to a deadline. These timing constraints are usually inherited from the environment of the controlled system. If one deadline is not met, some dramatic events can occur [1]. Usually, schedulers implemented in commercial real-time operating systems have no knowledge about tasks arriving in the future. From the validation view-point, tasks are a priori known, but the running time of each job is unknown until the job finishes - only worst-case execution times are known.

In some real-time applications, tasks are not only subjected to complete by their deadlines, but more control of the on-line schedule is required. Quality of Service (QoS) is interpreted in many ways (e.g. jitters, response times, etc.). In real-time environments, the objective is then to optimize a performance measure while respecting the release times and the deadlines of the tasks. There are two approaches in order to deal with such optimization problems using schedulers of real-time kernels. On the one hand, we can modify the temporal parameters of the tasks in order to improve the behaviour of the classical priority assignment such as Deadline Monotonic (DM) and Earliest Deadline First (EDF). On the other hand, if we focus on fixed-priority schedulers, we can directly compute a set of priorities so that the associated on-line schedule leads to good performances. Such tools are needed at the design step in order to build up a schedule with the expected performances.

In non real-time environments, response time (or flow time) is a widely-used performance metric to optimize continuous job arrivals. When each task has to be responsive in a single processor problem, then the optimized performance measure is the maximum response time of the tasks. For that purpose, the First-In-First-Out rule (FIFO) is an exact algorithm [2]. If the whole system has to be responsive, then the optimized metric is the average response time. It is well-known that the Shortest Remaining Processing Time rule (SRPT) leads to an optimal schedule. Recently, it has been shown that Shortest Processing Time rule is asymptotically optimal for the same problem [3]. But, when deadlines are added to the problem, then it becomes $\mathcal{NP}$-hard [4].

Weighted versions of these problems have also been studied. Their interests are to control the maximum response times of some tasks while keeping the whole system responsive. In particular, the stretch (also called the slowdown) is the weighted sum of response times when weights are the inverse of the processing times. This metric has been used to study the performances of operating systems and parallel systems [5]. The computational complexity of minimizing the average stretch is unknown. But, if arbitrary weights are allowed, then minimizing the weighted total response time is $\mathcal{NP}$-hard in the strong sense [6].

Hereafter, we study periodic tasks that will be on-line scheduled according to fixed priorities. Each task τ_i is defined by four deterministic parameters (C_i, D_i, T_i, w_i), where C_i is the worst-case execution time, D_i is the deadline relative to the arrival of the task in the system, T_i is the period between two successive arrivals and w_i is a real number that models the importance of the task in terms of worst-case response time. The tasks are simultaneously released at time 0 and we also assume that $D_i \leq T_i$, for every task τ_i, $1 \leq i \leq n$. A task τ_i is feasible if its worst-case response time R_i is less than or equal to its relative deadline. Among all feasible priority assignment to the tasks, we aim to find one that minimizes the weighted sum of the worst-case response times. So, the objective function is $\min(\sum_j w_j R_j)$. According to the previous discussion, it is easy to prove that this scheduling problem is $\mathcal{NP}$-hard in the strong sense.

The parameters w_i are set by the designers of the real-time software for defining the importance of task τ_i in terms of worst-case response time. If there are no particular constraints on the response time of a task (but it must be completed by its deadline), then its weight is set to 0. In this way, this task will have no influence on the objective function. We then compute the fixed priorities π_i so that the objective function is minimized and all deadlines are met. Deadline Monotonic policy usually leads to bad performances for minimizing the weighted sum of worst-case response times. The main idea to increase the quality of the schedule is to schedule a task having a small weight as high a priority as possible, even if its deadline is not the lowest one among pending tasks.

The paper is organized as follows. In Section 2 we shall present our branch and bound algorithm. Section 3 presents a detailed example and numerical results on randomly generated problems. Lastly, we conclude and give further perspectives of research.

2 Branch and Bound Algorithm

Branch and Bound algorithms are very used to solve combinatorial problems. When they are applied to scheduling problems, partial schedules are stored in vertices of a search tree. Leaves of the search tree define feasible solutions of the problem. Since the number of feasible solutions is exponential due to the computational complexity of the problem, the only way to find a solution in an acceptable amount of time is to detect as soon as possible vertices that will not lead to an improvement of the best known solution. At each step, the explored vertex is separated, that is to say its children are defined by assigning a priority to one task in the previous partial solution. These vertices are pruned (deleted) if the lower bound of the criterion is greater than or equal to the best known solution or if the constraints of the problem are not satisfied. The implementation of a branch and bound method is based on a set of non-explored vertices, called the Active Set.

2.1 Initial Solution

Before beginning the branch and bound procedure, an initial feasible solution (i.e., a feasible set of priorities) is computed in order to allow comparison of criteria with the first vertices generated in the first level of the search tree. This solution allows the computation of an upper bound of the objective function. Since $D_i \leq T_i$ and tasks are synchronously released at time 0, then:

- Deadline Monotonic policy leads to a feasible schedule if one exists,
- The worst-case response times of the tasks can be computed exactly [7].

Assigning priorities by using the Deadline Monotonic algorithm is performed in polynomial time, but we do not know if there exists a polynomial time algorithm for verifying that every deadline is met in the corresponding schedule (i.e. the complexity status of feasibility of synchronously released and fixed-priority tasks is unknown). But the worst-case response time of a task can be computed in pseudo-polynomial time (i.e., in polynomial time assuming that the input is unary encoded) by using the method presented in [7]. Thus, as far as we know, checking feasibility of given priorities can be performed in pseudo-polynomial time.

We assume hereafter that 1 is the highest priority level. Let $[i]$ denote the index of the task at priority level i. The worst-case response time of a task τ_i depends on the workload of the higher priority tasks. This workload at time t is $W(t)$:

$$W(t) = \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_{[j]}} \right\rceil C_{[j]} \tag{1}$$

The worst-case response time of τ_i is then obtained by solving the recurrent equation $t = C_i + W(t)$, that is to say:

$$\begin{cases} R_i^{(0)} = C_i \\ R_i^{(k)} = W(R_i^{(k-1)}) + C_i \end{cases} \tag{2}$$

$$R_i = R_i^{(k)} = R_i^{(k-1)} \tag{3}$$

According to the Deadline Monotonic priority assignment, we can compute the worst-case response times of the tasks by using (2), and then compute the value of the objective function. The task ordering is performed in polynomial time, as well as computing the objective function. Thus, the computational complexity of this heuristic is due to the feasibility problem of the tasks.

We shall now propose a better heuristic, that always leads to a better solution in comparison to the Deadline Monotonic schedule. We assign the priority in reverse order. Such backward priority ordering is based on an interesting property of fixed-priority schedule.

Lemma 1 *[8] Let the tasks assigned lowest priority levels $i, i+1, \ldots, n$ be feasible under that priority ordering, then there exists a feasible schedule that assigns the same tasks to levels $i, \ldots, n$.*

Our heuristic assigns priorities to the tasks from the lowest priority to the highest one, and at each step it chooses the task that increases as little as possible the objective function among all tasks that are feasible for the current priority level. As a direct consequence of Lemma 1, the heuristic always finds a feasible priority assignment if one exists. In Figure 1, we give the pseudo-code of the heuristic.

2.2 Vertex Branching Rule

A vertex is simply defined by the index of a task. Levels of the vertices in the tree are interpreted as priority levels of tasks. The first level in the tree defines the tasks assigned to the lowest priority level, and in the same way, the leaves define tasks assigned to the highest priority level. The number of vertices in a search tree is clearly exponential in the size of the problem. Since the number of vertices at level l is the number of permutations of k elements among n, then the number of vertices at level k is: $\frac{n!}{(n-k)!}$. Thus, the total number of vertices in the search tree is: $\sum_{i=0}^{n-1} \prod_{k=0}^{i} (n-k)$.

The vertex branching rule selects the next vertex to be explored. This strategy has an important impact on the largest size of the active set. The most commonly used strategy to solve scheduling problems is the depth-first search one. The main interest of this method is: the size of the active set is polynomially bounded in the size of the problem (it is not the case for other strategies, especially when a breadth-first search is used). In that way, the maximum number

```
Heuristic: Backward Priority Rule
   Let X be the set of tasks and n be the number of tasks
   Let z=0 be the objective function
   While X is not empty
      Let X' the tasks feasible at priority level n
       among tasks belonging to X
      Let a be the task with the lowest weighted
       worst-case response time r in X'
      z=z+a.w*r
      Assign priority level n to a
      n=n-1
      X=X\{a}
   End Loop
End
```

Fig. 1. Heuristic pseudo-code

of vertices stored simultaneously in the active set is $n(n+1)/2$, generated while reaching the first leaf of the search tree.

When a leaf is reached, the value of its objective function is compared with the best known solution. If it is lower, then the current leaf is used to update the upper bound of the criterion. The search continues while the active set is not empty (i.e., the tree has been completely explored). When the tree has been completely explored, then the best known upper bound, improved during the search, is an optimal solution.

2.3 Lower Bound

For the explored vertex, the value of the criterion associated to the tasks assigned to priority levels can be exactly computed, since their worst-case response times are already known. But for the others, we need a lower bound in order to have an estimation of the quality of the vertex. Note that a task τ_i with a lower priority than another task τ_j cannot increase the response time of τ_j in all feasible schedules.

One way to compute a lower bound of the weighted sum of the worst-case response times is to apply the modified Smith's rule [9]. Tasks without priority are scheduled in non-decreasing order of the ratio C_i/w_i, and we relax three constraints of our problem:

- preemption is allowed
- tasks are periodic.
- tasks are subjected to deadline constraints.

Tasks are then sequenced without preemption using this rule and the relaxed constraints. Their completion times are then computed. Subsequently, the weighted sum of the worst-case response times is computed. We shall prove that this method leads to a lower bound of the objective function for the explored vertex. Figure 2 presents the pseudo-code of the lower bound computation. $t.w$ and $t.C$ denote respectively the weight and the worst-case execution time of a task t ; $v.S$ is the set of tasks with assigned priorities in the vertex v ; $v.R[t]$ is the worst-case response time of t for vertex v ; T is the set of tasks of the system.

```
Algorithm: Lower Bound
Input:  Vertex v
Output: Integer lb
Var : Integer z, Set of Tasks U
Begin
    lb=0
    For every task t in v.S Do
        lb=lb+t.w*v.R[t]
    End For
    U=T\v.S
    z=0
    For every task t in U in non-decreasing order of t.C/t.w Do
        z=z+t.C
        lb=lb+z*t.w
    End For
    Return lb
End
```

Fig. 2. Lower Bound pseudo-code

Theorem 1 *A lower bound of the optimal weighted sum of the worst-case response times is computed by the Lower Bound algorithm.*

Proof: The first loop in the Lower Bound procedure calculates the value of the criterion for the tasks having priorities. Since these tasks are assigned to lower priority levels, then their worst-case response times can exactly be computed using the system of equations (2) and (3). For the other tasks (having no assigned priority levels), their worst-case response times are dependent of the assignment of the higher priority levels. We shall show for these tasks that a lower bound of the criterion is obtained by applying the modified Smith's rule [9]. Since we relax the assumption of the periodic arrival of tasks, higher priority tasks occurs strictly one time. Thus, the worst-case response times cannot increase. Subsequently, consider two priority assignments of the tasks: σ and σ'. Assume that the priority assignment in the schedule σ respect the modified Smith's rule, and not in σ' due only to the interchange of τ_i and τ_j. The values of the criterion in σ_1, σ_2 are the same in σ than in σ'. The only variation in the respective objective functions z and z' can only be due to the interchange of τ_i and τ_j:

$$z' - z = (C_j w_j + (C_i + C_j)w_i)) - (C_i w_i + (C_i + C_j)w_j)) \tag{4}$$

$$z' - z = C_j w_i - C_i w_j \tag{5}$$

Since the tasks respect the modified Smith's rule, then we verify that the criterion cannot be improved in σ' since $C_i/w_i \leq C_j/w_j$: $z' - z \geq 0$. It follows that a lower bound of the criterion has been established.□

2.4 Vertex Elimination Rules

Elimination rules are of prime interest in branch and bound methods because they can drastically prune the search tree. We next detail three conditions to prune a vertex, that are:

- immediate selection,
- a deadline is not met,
- the lower bound is greater than the best known solution (upper bound).

The immediate selection rule detects children that will not lead to a feasible solution, before generating them. The following result leads to such a rule.

Theorem 2 *For all pair τ_i, τ_j of tasks, if:*

$$\left\lceil \frac{T_i}{T_j} \right\rceil \times C_j + C_i > D_i \tag{6}$$

then the priority of τ_i must be higher than the priority of τ_j.

According to this result, we can define a set of constraints that must be satisfied during the branching step. For instance, if we detect that τ_i must have a higher priority than τ_j according to Theorem 2, then a task τ_i will not be generated in the first level of the search tree. Furthermore, τ_i can be assigned to a priority level if, and only if, τ_j has already been assigned one in a predecessor vertex. The constraints are obviously computed and checked in polynomial times.

We now detail the test of deadline constraints for the current vertex. The vertex branching rule generates children of the vertex currently explored while respecting the immediate selection rule. If the vertex of level k is separated, then all its children are assigned to the priority level $n - k$ (remember that priorities are assigned in reverse order). The first step is to verify that the tasks assigned to this priority level meet their deadlines ($\tau_{[n-k+1]}$). Since the vertex is newly generated, then we always know that the tasks assigned to the priority levels $n..n - k + 1$ are feasible. These tasks do not interfere with the execution of $\tau_{[n-k+1]}$. We can check the feasibility of $\tau_{[n-k+1]}$ by solving the equations (2) and (3). Furthermore, its response time depends only on the tasks without assigned priority levels.

Theorem 3 *To prune a vertex, it is unecessary to check the schedulability of the tasks without assigned priority levels.*

Proof: Let S be a set of feasible tasks according to the Deadline Monotonic algorithm, then every $S' \subset S$ is also schedulable according to Deadline Monotonic. Since the initial solution of the branch and bound procedure is obtained by using the Deadline Monotonic priority assignment, then there exists a feasible priority assignment for the tasks without priority in the current vertex. Lemma 1 allows to conclude. $\Box$

The second way to prune a vertex is performed when the best solution is lower than the lower bound of the current vertex. Generation of the children can only increase lower bounds. So, when a leaf should be reached, the value of the objective function will not be improved, and it will be greater than or equal to the best known upper bound.

One can remark that checking the feasibility of a child vertex is performed in pseudo-polynomial time. This complexity is only due to the computation of the worst-case response time of the task at the current priority level in the explored vertex (i.e., when equations (2,3) are solved).

3 Experimentation

We shall present a detailed example and numerical results on randomly generated instances of the scheduling problem.

3.1 A detailed example

We next consider the task set presented in Figure 3. The utilization factor of the processor (i.e., $\sum_i C_i/T_i$) is $U = 0.792$, and applying the heuristic presented in Section 2.1 leads to a feasible schedule, and the value of the objective function for that schedule is $ub = 254$. The priority assignment and the associated response-times according to the heuristic are also given in Table 3, respectively in columns $H(\pi_i)$ and $H(R_i)$. Figure 4 presents the search tree. The immediate selection rule defines two constraints: τ_4 must have a higher priority than τ_1 and τ_2. The complete search tree has 326 vertices. The optimal value is 174, so the improvement of the initial solution is up to 46 percent. The implicit enumeration performed by the branch and bound generates 15 vertices, so less than one percent of the search tree has been explored to find the optimal priority assignment.

3.2 Numerical results

This method has also been experimented with randomly generated problems. For a given number of tasks, we randomly generate 25 instances. Problems have between 6 and 28 tasks, and the utilization factor of the processor is 0.5 for all instances. Uniform law is used by the random generator and the characteristics of the generation are $C_i \in [1..10]$, $D_i = T_i$ are computed in order to have a utilization factor close to 0.5, and $w_i \in [0..20]$. Numerical experimentation has

Tasks	C_i	D_i	T_i	w_i	$H(\pi_i)$	$H(R_i)$
τ_0	5	15	30	2	1	5
τ_1	7	50	50	1	4	45
τ_2	8	50	100	3	5	21
τ_3	3	20	25	5	3	12
τ_4	2	7	7	4	2	7

Fig. 3. Task set of the detailed example

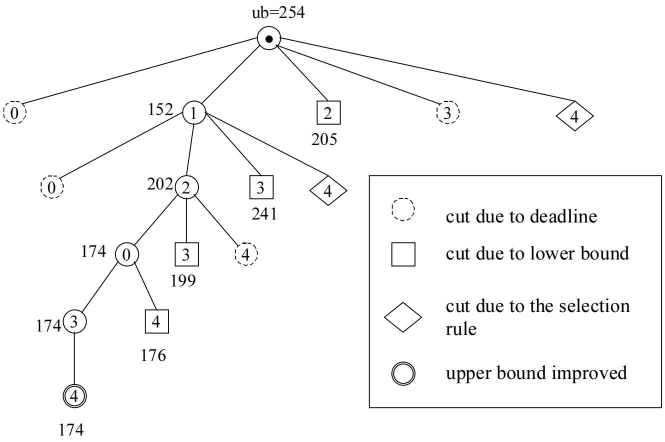

Fig. 4. Search tree of tasks of the detailed example

been performed on a Pentium III/450 MHz personal computer. A time limit has been fixed to one hour for every instance.

Figure 5 gives the average resolution time according to the problem size. The best solution indicates at what time the optimal solution has been reached. The elapsed time is the total duration to solve a given problem. The difference between the elapsed time and the time at which the optimal solution has been found, is the delay to prove that the best known solution is an optimal one. As shown in Figure 5, the optimal solution is quickly reached, but proving optimality takes a long time when the size of the problem increases. Figure 5 also presents the number of instances solved versus of unsolved instances for each size of a problem. We can see that no instance with 28 tasks has been solved within the time limit of one hour for each instance.

4 Conlusion

We have presented a method for computing fixed priorities of periodically released tasks so that the weighted sum of their worst-case response times is minimized. These priorities are then used by an on-line scheduler while respecting

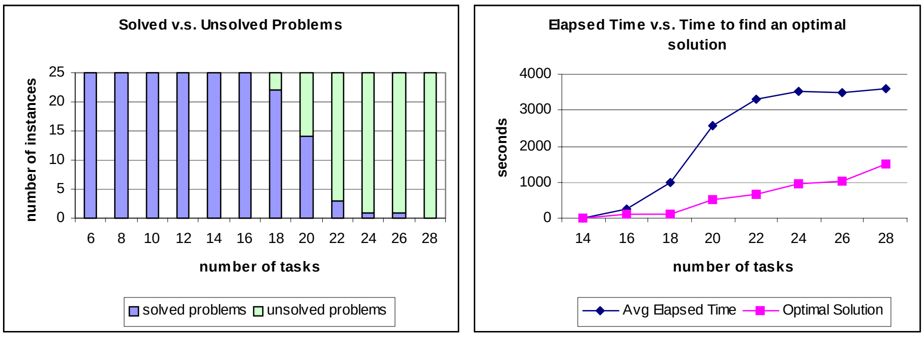

Fig. 5. Solved instances and computation times - Time limit: 1 hour.

the hard deadline constraints. Numerical results have been presented to show the efficiency of the method. Perspectives of this work are to extend the existent tool to other optimization criteria and distributed real-time systems. For that purpose, the priority assignment method presented in [10] will be extended in order to meet QoS requirements.

References

1. Liu, J.: Real-Time Systems. Prentice Hall (2000)
2. Bender, M., Chakrabarti, S., Muthukrishnan, S.: Flow and stretch metrics for scheduling continuous job streams. proc. Symp. on Discrete Algorithms (1998)
3. Kaminsky, P., Simchi-Levi, D.: Asymptotic analysis of an on-line algorithm for single machine completion time problem with release dates. Operations Research Letters **29** (2001) 141–148
4. Du, J., Leung, J.: Minimizing mean flow time with release time and deadline constraints. in: proc. of Real-Time System Symposium (1988) 24–32
5. Muthukrishnan, S., Rajaraman, R., Shaheen, A., Gehrke, J.: Online scheduling to minimize average stretch. proc. IEEE Symp. on Foundations of Computer Science (1999) 433–443
6. Chejuri, C., Khanna, S., Zhu, A.: Algorithms for minimizing weighted flow time. proc. ACM Symp. on Theory of Computing (2001)
7. Joseph, M., Pandya, P.: Finding response-time in a real-time system. BCS Computer Journal **29** (1986) 390–395
8. Audsley, N.: Optimal priority assignment and feasibility of static priority assignment with arbitrary start times. YCS 164, University of York (1991)
9. Smith, W.: Various optimizers for single-stage production. Naval Research Logistic Quaterly **3** (1956) 481–486
10. Richard, M., Richard, P., Cottet, F.: Task and message priority assignment in automotive systems. 4th Int. Conf. on Fieldbus Systems and Their Applications (2001)