tel-00554265, version 1 - 10 Jan 2011

THESE

pour l'obtention du Grade de

DOCTEUR DE L'ECOLE NATIONALE SUPERIEURE DE MECANIQUE ET D'AEROTECHNIQUE
(Diplôme National – Arrêté du 7 août 2006)

Ecole Doctorale : Sciences et Ingénierie pour l'Information
Secteur de Recherche : Informatique et Applications

Présenté par :

Thi Huyen Chau NGUYEN

****

Approximation des temps de réponse des tâches sporadiques à priorité fixe dans les systèmes monoprocesseurs

****

Directeur de Thèse : Pascal RICHARD

****

Soutenue le 25 Novembre 2010
devant la Commission d'Examen

****

JURY

Maryline CHETTO	Professeur, Université de Nantes	Rapporteur
Laurent GEORGE	Maître de Conférences, HDR, Paris 12	Rapporteur
Joël GOOSSENS	Maître de Conférences, Université Libre de Bruxelles	Examinateur
Emmanuel GROLLEAU	Maître de Conférences, HDR, ENSMA	Examinateur
Isabelle PUAUT	Professeur, Université de Rennes 1	Examinateur
Pascal RICHARD	Professeur, Université de Poitiers	Examinateur

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Remerciements

Je tiens à remercier le Professeur Guy PIERRA, ancien directeur du LISI, ainsi que le Professeur Yamine AIT-AMEUR, actuel directeur du LISI, de m'avoir accueillie aussi chaleureusement et avoir mis à ma disposition tous les moyens techniques et scientifiques nécessaires à l'exercice de mon travail.

Je remercie également Pascal RICHARD, mon directeur de thèse, sans qui tous mes travaux n'auraient pas pu être réalisés. Je le remercie pour son soutien, ses conseils lors de nos conversations, ainsi que pour ses qualités humaines et son écoute dans les moments difficiles.

Je voudrais adresser aussi mes remerciements à Emmanuel GROLLEAU qui m'a orientée vers l'équipe temps réel et qui m'a aidée de façon enthousiaste depuis les premiers jours où je suis arrivée en France, pendant mon stage de mastère et ma thèse au LISI.

Merci aux Professeurs Maryline CHETTO et Laurent GEORGE d'avoir accepté la lourde charge de rapporteurs, ainsi qu'au Professeur Isabelle PUAUT pour l'honneur qu'elle m'a fait en acceptant d'être le Président de mon jury.

Je voudrais également remercier mes parents, ma sœur et mon frère du soutien inconditionnel qu'ils m'ont apporté. Je n'oublie bien évidemment pas mes amis François, Nabil, Idir, Chedlia, Yassine, qui ont été, bien malgré eux, des rochers sur lesquels je pouvais m'agripper dans les moments difficiles.

Il y a encore tant de personnes que je souhaite remercier, en particulier les membres du LISI : Loé, Christian, Valéry, Frédéric, Michaël, Claudine, Chimène, Sybille, Sadouanouan, Stéphane, Patrick. A toutes ces personnes, je souhaite leur dire merci pour tes les merveilleux moments que j'ai passés en leur compagnie.

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Table des matières

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Chapitre 1

Introduction

1. Définition générale

1.1. Systèmes temps réels

Les systèmes temps réels sont des systèmes informatiques qui doivent réagir sous des contraintes temporelles strictes aux événements provenant de l'environnement contrôlé. En conséquence, la correction de ces systèmes dépend non seulement de la correction des calculs mais aussi des dates auxquelles les résultats sont produits. Une réaction survenant trop tard peut être inutile voire même dangereuse. Voici quelques exemples d'applications qui demandent de réagir en temps réel :

- contrôle des processus de production complexes,
- applications automobiles,
- systèmes de contrôle des vols,
- systèmes de télécommunication,
- automatisation industrielle,
- robotique,
- systèmes militaires,
- etc.

1.2. Ordonnancement temps réel

Les systèmes informatiques temps réels se décomposent en un exécutif temps réel (noyau, exécutif ou système d'exploitation temps réel) et un programme applicatif. Pour concevoir, développer et faire évoluer le système, un programme temps réel est d'un point de vue logiciel considéré comme un système multitâche. Le travail du système informatique consiste par conséquent à gérer l'exécution et la concurrence de l'ensemble des tâches en optimisant l'occupation du processeur et en veillant à respecter les contraintes temporelles des tâches. L'ordonnancement des tâches est effectué par l'ordonnanceur de l'exécutif qui à chaque instant alloue le processeur à une tâche selon une politique fondée sur les priorités des tâches. Tous ces éléments sont regroupés sous la notion d'ordonnancement temps réel. Si une contrainte temporelle n'est pas respectée, une défaillance du système ou une faute temporelle est révélée. Dans un système temps réel dur, chaque tâche est soumise à une échéance temporelle stricte.

Le non respect d'une contrainte temporelle n'est pas admissible à l'exécution. Ainsi pour ces systèmes, la vérification du respect des échéances est effectuée dès la conception du système.

Dans la suite nous ne considérons que des tâches à priorité fixe devant s'exécuter sur une plateforme monoprocesseur. Les priorités sont calculées durant la conception du système et ne changent pas durant l'exécution de l'application.

1.3. Tâche temps réel

Une tâche temps réel est l'entité de base des programmes temps réels. C'est un calcul qui est exécuté par le CPU de façon séquentielle et non-réentrante. Les tâches temps réels permettent ainsi de contrôler le procédé externe via un ensemble de capteurs et d'actionneurs intégrés au procédé. Elles utilisent les primitives de l'exécutif temps réel sur lequel elles sont exécutées pour communiquer entre elles, faire des acquisitions de données (depuis des capteurs du système contrôlé) ou encore actionner des commandes sur le système contrôlé. Mais pour respecter les caractéristiques du procédé (par exemple, désactiver à temps le contrôleur de vitesse d'une voiture dont on actionne les freins), les tâches sont soumises à des contraintes temporelles.

Les tâches réalisent classiquement une acquisition depuis des capteurs couplés au procédé contrôlé, un traitement, puis un envoi de données vers des actionneurs. Ces traitements sont généralement périodiques. Les contraintes temporelles sont définies par une échéance, relative au réveil de la tâche. L'échéance relative (ou délai critique) spécifie le délai maximum autorisé entre le réveil d'une tâche et sa terminaison. Chaque instance de tâche doit donc s'exécuter dans une fenêtre temporelle de longueur égale au délai critique. Cette fenêtre temporelle associée à une instance de tâche débute ainsi par sa date de réveil et se termine par une date d'échéance (dite absolue).

2. Notations liées aux tâches et à l'ordonnancement

2.1. Données sur les tâches

Nous allons tout d'abord expliciter et définir l'ensemble des notations utilisées pour spécifier les caractéristiques et le comportement de nos tâches :

- τ : l'ensemble de n tâches ;
- τ_i : la tâche numérotée i ;
- C_i : sa pire durée d'exécution ou WCET (worst-case execution time) ;
- $r_i = r_{i,1}$: sa première date d'activation lorsqu'elle est connue. Dans ce cas on parle de tâche concrète. Parfois réglable au gré du concepteur (offset free). Lorsqu'elle est inconnue on parle de tâche non concrète. Dans le cas des tâches concrètes, on parle de tâches synchrones au démarrage (ou simultanées) si tous les r_i sont nuls, asynchrones au démarrage (ou non simultanées) sinon ;
- T_i : sa période d'activation ; à partir de sa première date d'activation, créant une instance (a job) $\tau_{i,1}$ à exécuter, une tâche est activée toutes les T_i unités de temps, donnant à chaque fois naissance à une instance $\tau_{i,j}$. Les instances sont non réentrantes les unes par rapport aux autres (exécution FIFO des instances qui sont actives simultanément) ;
- D_i : son délai critique (relative deadline) qui définit la contrainte temporelle de la tâche.
- J_i : Une instance $\tau_{i,k}$ d'une tâche τ_i à gigue de démarrage (gigue sur activation ou jitter ou release jitter) est activée entre les dates $r_{i,k}$ et $r_{i,k} + J_i$. Une telle tâche concrète avec gigue de

tel-00554265, version 1 - 10 Jan 2011

démarrage est définie par $< r_i, C_i, D_i, J_i, T_i >$, et une non concrète par $< C_i, D_i, J_i, T_i >$. La gigue permet de représenter les délais introduits par l'exécutif pour démarrer une instance et/ou les communications (entrantes) de la tâche.

Une instance $\tau_{i,j}$ de la tâche τ_i doit être exécutée avant son échéance absolue (deadline) $d_{i,j} = r_{i,j} + D_i$. Les instances $\tau_{i,1}$ et $\tau_{i,2}$ respectent leur échéance alors que $\tau_{i,3}$ viole son échéance. Les caractéristiques temporelles des tâches, ainsi que les dates de réveil, les échéances absolues sont représentées figure 1.1 avec les symboles suivants :

- $\uparrow$: symbole d'une date de réveil d'une instance de tâche ;
- $\downarrow$: symbole d'une date d'échéance (absolue) d'une instance de tâche ;

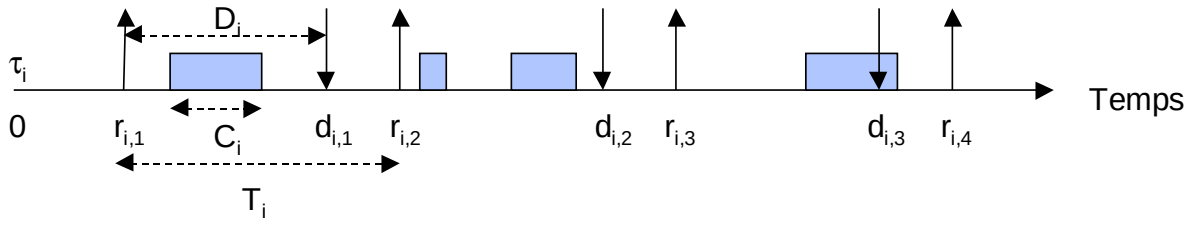

FIGURE 1.1 – *Notations liées aux tâches et à l'ordonnancement.*

2.2. Les caractéristiques supplémentaires liées à l'ordonnancement

Plusieurs indicateurs permettent de caractériser les tâches et leurs comportements :

- $u_i = \frac{C_i}{P_i}$: Facteur d'utilisation par une tâche τ_i (fraction de temps durant laquelle la tâche occupe le processeur) ;
- $U = \sum_{1 \leq i \leq n} u_i$: Facteur d'utilisation d'une ensemble de tâches ;
- $u_i^c = \frac{C_i}{\min(T_i, D_i)}$: Facteur de charge du processeur par une tâche τ_i ;
- $U^c = \sum_{1 \leq i \leq n} u_i^c$: Facteur de charge d'un ensemble de tâches ;
- $R_i^{(*)}(k)$: le temps de réponse de la k^{ieme} requête d'une tâche τ_i, c'est la différence de temps entre l'instant où la tâche termine son exécution et l'instant où elle a été activée ;
- $R_i^{(*)}$ ou R_i : le pire temps de réponse d'une tâche τ_i est le maximum des temps de réponse de ses requêtes $R_i{}^{(*)} = Max_{k \geq 1} R_i{}^{(*)}(k)$.

Nous considérons dans la suite, que la tâche la plus prioritaire est celle dont le numéro est le plus faible.

2.3. Classification des modèles de tâches

Une taxonomie des problèmes d'ordonnancement repose sur la définition précise des modèles de tâches et des plateformes d'exécution. Les définitions des algorithmes d'ordonnancement et des tests associés sont fortement liés aux caractéristiques du modèle de tâche considéré. Nous donnons ci-après les principaux éléments permettant de caractériser un modèle de tâche :

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Types de périodicité :
- Tâches périodiques : les activations successives sont séparées par une période constante ;
- Tâches sporadiques : Tâches réveillées par un événement externe caractérisé avec un délai minimal entre 2 événements successifs ;
- Tâches apériodiques : Tâches réveillées par un événement externe. Le délai entre 2 événements successifs, ainsi que la fréquence ne sont pas connus.

Types d'échéance :
- Échéance sur requête : $D_i = T_i$;
- Échéance : $D_i \leq T_i$;
- Échéance arbitraire : D_i et T_i sont indépendants l'un de l'autre.

Types de réveil :
- Tâches concrètes : toutes les dates de réveil sont connues
  - Tâches à départ simultané $\forall i, r_i = 0$;
  - Tâches à départ différé $\exists i / r_i \neq 0$;
  - Tâches à départ libre ("offset free") : les dates r_i peuvent être librement choisies par le concepteur.
- Tâches non concrètes : les dates de réveil sont inconnues. L'ordonnancement doit être validé quelles que soient les dates de réveil des tâches. Nous pouvons citer par exemple les modèles de tâches suivant :
  - Modèle RMA (Rate Monotonic Analysis) ;
  - Transactions et Modèle Multiframe Généralisé.

Autres caractéristiques importantes des tâches :
- Tâches indépendantes : toutes les tâches peuvent être préemptées par toute tâche n'importe quand ;
- Tâches non préemptibles : une tâche ne peut être préemptée ;
- Tâches partageant des ressources : l'accès aux ressources critiques s'effectue à l'aide d'un protocole de gestion de ressource afin d'assurer l'exclusion mutuelle des accès, éviter famine et interblocage ;
- Tâches soumises à contraintes de précédence : certaines tâches doivent attendre des messages ou synchronisations provenant d'autres tâches pour pouvoir débuter leur exécution.
- Tâches à suspension : une tâche peut se suspendre durant une opération d'entrée sortie et reprendre son exécution lorsque celle-si est terminée.

3. Objectifs de l'étude

Dans beaucoup de systèmes temps réels, des tâches spécifiques doivent se terminer avant leurs échéances impératives. Dans la suite nous considérons les systèmes de tâches à priorité fixe, sporadiques s'exécutant sur un unique processeur avec la préemption autorisée.

Deux grandes catégories d'algorithmes ont été proposées pour vérifier des conditions nécessaires et suffisantes de faisabilité des tâches : analyse du temps de réponse (Response Time Analysis ou RTA) [4,18] et l'analyse de la demande processeur (Processor Demand Analysis ou PDA) [20]. Ces deux

approches sont connues pour avoir une complexité au mieux pseudo-polynomiale selon le modèle de tâches considéré. Le problème du calcul des pires temps de réponse est connu NP-Difficile au sens faible ([13] pour le modèle de tâche de Liu et Layland [23]. Notons toutefois que pour ce modèle de tâche, l'existence d'un algorithme polynomial pour déterminer la faisabilité d'un système de tâches est un problème ouvert.

Les concepteurs de systèmes temps réel doivent définir, optimiser et analyser de sensibilité des paramètres du système et ces étapes nécessitent en pratique l'exécution d'un très grand nombre de tests d'ordonnançabilité, parfois à travers un environnement logiciel intéractif. La *complexité* de ces algorithmes les rend peu appropriés si l'analyse d'ordonnançabilité doit être exécutée un grand nombre de fois. Dans ces scénarios, un algoritme pseudo-polynomial de test d'ordonnançabilité peut être trop lent. A la place, un algorithme plus rapide mais fournissant un résultat approché d'ordonnançabilité, plutôt qu'exact, peut être acceptable. C'est notamment le cas :

1. durant un processus intéractif de conception d'un système ou de son prototypage rapide. Le concepteur du système effectue un très grand nombre d'appels à l'algorithme de tests d'ordonnançabilité.
2. pour les systèmes admettant dynamiquement de nouvelles tâches : dans un système en exécution, l'admission d'une nouvelle tâche nécessite d'exécuter un test d'ordonnaçabilité avant que cette tâche puisse débuter son exécution. Des contraintes temporelles serrées peuvent exclure l'utilisation des tests exacts d'ordonnançabilité dans ces contextes.
3. durant l'analyse d'un système distribué de grande taille lorsque par exemple une analyse holistique ([36]) est utilisée. Les tests d'ordonnançabilité sont alors exécutés un très grand nombre de fois.

Un autre aspect à considérer est que dans certains cas, comme par exemple pour une utilisation hors-ligne dans un logiciel de conception, un test booléen peut s'avérer fournir des informations insuffisantes, et un test fondé sur le calcul des temps de réponse des tâches est typiquement requis. Dans le contexte des techniques d'approximation, il n'est pas seulement nécessaire de vérifier la faisabilité du système de façon approchée, mais il est nécessaire en plus de déterminer les pires temps de réponse approchés des tâches. Pour ces utilisations, des méthodes de calcul de bornes supérieures des pires temps de réponse des tâches ont été définies (par exemple, [6, 24, 33]).

En 2007, fondé sur les résultats préliminaires de Richard et al. [31], Fisher et al. [16] ont montré qu'il est possible de définir un test approché qui détermine des approximations des pires temps de réponse des tâches, à l'aide d'algorithmes paramétriques (schémas d'approximation). La faible erreur encourue et la complexité polynomiale de ces méthodes les rendent adaptées pour les utiliser pratiquement pour les applications mentionnées plus haut.

Dans ce contexte, nous nous sommes principalement intéressés aux conditions suffisante d'ordonnançabilité et à la qualité des tests approchés à travers les techniques d'approximations polynomiales des problèmes NP-Difficile : approximation linéaire et schéma d'approximation pour déterminer des valeurs approchées des pires temps de réponse des tâches à priorité fixe. Dans la suite, nous allons utiliser des algorithmes d'approximation pour concevoir des tests efficaces d'ordonnançabilité (c'est-à-dire s'exécutant en temps polynomial) mais en introduisant une petite erreur dans le processus de décision de l'ordonnançabilité des tâches. Cette erreur est contrôlé à l'aide d'un paramètre de précisions ϵ (accuracy parameter). En d'autres termes, ils réalisent un compromis entre l'effort de calcul pour décider de l'ordonnançabilité d'un système de tâches et la qualité des décisions prises. Ce type

tel-00554265, version 1 - 10 Jan 2011

d'approche a été développé pour l'algorithme d'ordonnancement EDF [1, 2, 11] et pour les systèmes de tâches à priorité fixe [14, 16, 31, 32]. Certains de ces algorithmes ([**?**, **?**]) permettent simultanément de vérifier l'ordonnançabilité d'une tâche et de calculer une borne supérieure de son pire temps de réponse de la façon suivante :

- Si le test répond "faisable", alors la tâche analysée est garantie d'être faisable sur un processeur de capacité 1 (c-à-d, de vitesse unitaire) et une borne supérieure de son pire temps de réponse peut être calculée à l'aide de la méthode de déduction présenté dans [31].
- Si le test retourne "non faisable", la tâche est garantie de ne pas être ordonnançable sur un processeur plus lent, dont la capacité est $(1 - \epsilon)$. Notons, qu'aucune conclusion ne peut être établie si un processeur de capacité unitaire est considéré, et aucune borne du temps de réponse ne peut être déduite.

Ces bornes supérieures des pires temps de réponse (s'il en est) peuvent introduire une perte de précision dans le processus de décision. Il est souhaitable que cette perte de précision soit quantifiée avec précision afin de mesurer le compromis entre la précision des calculs et les ressources supplémentaires qui seront nécessaires si un tel test approché est utilisé.

4. Organisation du mémoire et contributions

Le plan du mémoire est le suivant : dans le chapitre Chapitre 2, nous rappelons les principaux résultats connus pour l'ordonnancement des tâches à priorité fixe. Les trois chapitres suivants présentent nos contributions :

(a) Dans le Chapitre 3, nous proposons une borne supérieure de calcul des temps de réponse des tâches fondée sur une approximation linéaire de la demande processeur. La borne supérieure du temps de réponse d'une tâche est alors calculée en temps linéaire. A l'aide du ratio d'approximation classique, nous montrons que cette borne n'a pas de garantie de performance vis-à-vis du temps de réponse exact de la tâche analysée. Nous montrons que c'est aussi le cas pour les bornes connues dans la littérature. Nous montrons que la nouvelle borne domine celles connues et nous établissons une garantie de performance à l'aide de la technique d'augmentation de ressource. Précisément, la borne supérieure calculée est une borne inférieure du pire temps de réponse exact si le système est implémenté sur un processeur de capacité $1/2$. En d'autres termes, un facteur d'augmentation de 2 de la capacité du processeur est la borne supérieure du prix à payer pour utiliser cette nouvelle borne de pire temps de réponse. Des résultats d'évaluations numériques seront présentés afin d'obtenir une garantie de performance en moyenne vis-à-vis du calcul exact des pires temps de réponse des tâches.

(b) Dans le Chapitre 4, nous présentons des algorithmes paramétriques permettant d'améliorer les tests approchés de faisabilité d'un ensemble de tâches à échéance contrainte. Nous proposons ensuite plusieurs extensions des ces schémas d'approximation pour déterminer la faisabilité d'un système de tâches. Ensuite, ces schémas d'approximation sont étendus par une nouvelle "méthode déduite" pour calculer des bornes supérieures des temps de réponse des tâches. Puis, nous conduisons une analyse pire cas afin d'évaluer la qualité des bornes supérieures des temps de réponse à l'aide de la technique d'augmentation de ressource utilisée classiquement dans la théorie de l'approximation polynomiale des problèmes NP-Difficile. Enfin, nous présentons les résultats numériques pour comparer nos algorithmes à ceux existants sur la base de configuration de tâches

tel-00554265, version 1 - 10 Jan 2011

générées aléatoirement. De ces tests nous dérivons la garantie de performance moyenne de nos bornes supérieures des pires temps de réponse calculées.

(c) Dans le chapitre Chapitre 4, nous revisitons l'algorithme de Fisher et Baruah présenté dans [14] qui analyse l'ordonnançabilité des systèmes de tâches à priorités fixes et à échéance arbitraire. Nous prouverons à l'aide de contre-exemples que ce test n'est pas correct dans le cas général, mais demeure valide lorsque l'assignation des priorités respecte l'ordonnancement Rate Monotonic. Nous modifions ensuite ce test de faisabilité des tâches afin de corriger les problèmes rencontrés par l'algorithme dans le cas général tout en conservant les principes originels de la méthode proposée par Fisher et Baruah.

Les contributions des travaux présentés dans ce mémoire ont été effectuées en collaboration avec des personnalités exterieures à notre équipe de recherche : Sanjoy K. Baruah (University of North Carolina at Chapel Hill), Nathan Fisher (Wayne State University at Detroit) et Enrico Bini (Scuola Superiore Sant'Anna, Pisa).

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Chapitre 2

Ordonnancement à priorité fixe

1. Introduction

Dans ce chapitre nous présentons les principaux résultats sur l'ordonnancement des tâches à priorité fixe, ainsi que les techniques d'analyse d'ordonnançabilité, qui constituent la base de nos travaux.

1.1. Modèle de tâche

Nous considérons les systèmes de tâches sporadiques à priorité fixe, avec les hypothèses suivantes :
- Toutes les tâches dans le système de tâches τ sont indépendantes, c'est-à-dire, il n'y a aucunes relations de précédence et aucunes contraintes de ressource.
- Aucune tâche ne peut se suspendre d'elle même, par exemple sur des opérations d'entrée-sortie.
- Toutes les tâches sont entièrement préemptives.
- Les délais introduits par les changements de contexte du noyau sont supposés négligeables.
- les niveaux de priorités des tâches sont représentés par des nombres entiers de 1 à n. Le niveau 1 correspond à la tâche la plus prioritaire.

1.2. Algorithme d'ordonnancement et principe d'ordonnancement

Un algorithme d'ordonnancement consiste en un ensemble de règles qui, à tout moment, déterminent l'ordre dans lequel les tâches sont exécutées. Un algorithme est déterministe lorsqu'il prend les mêmes décisions lorsqu'il rencontre une même situation plusieurs fois. Un algorithme d'ordonnancement est dit en-ligne lorsque les décisions d'ordonnancement sont prises durant l'exécution des tâches et chaque fois qu'une tâche se réveille ou qu'une tâche en exécution se termine. Selon cette politique d'ordonnancement en-ligne, à chaque instance, l'ordonnanceur utilise la priorité de chaque tâche. La tâche ayant la plus haute priorité va être exécutée. En ordonnancement à priorité fixe, plusieurs stratégies d'affectation des priorités existent. Les plus connues sont : Rate Monotonic qui affecte les priorités aux tâches selon les périodes croissantes (la tâche la plus prioritaire est celle avec la plus petite période) et Deadline Monotonic qui affecte les proirités selon le même principe mais en se basant sur les échéances relatives des tâches.

Example 1 Soit un système de tâches non concrètes τ = {τ_1=<1,4>, τ_2=<9,12>} donné sous la forme $<C_i, T_i>$. Une affectation des priorités selon RM donnera Prio(1)=1, Prio(2)=2. ■

Analyse : Supposons qu'au début, il n'y ait que la tâche τ_2 qui demande le processeur. A l'instant 3, supposons que la tâche τ_1 arrive. Puisqu'elle est plus prioritaire, la tâche $\tau 2$ est préemptée et $\tau 1$ débute son exécution. Lorsqu'elle se termine, la tâche τ_2 devient la plus prioritaire et s'exécute à nouveau.

Un algorithme ordonnancement fiablement un système de tâches si toutes les échéances sont respectées. Un algorithme d'ordonnancement est optimal s'il est capable de produire un ordonnancement respectant toutes les échéances s'il en existe un. Cette propriété d'optimalité des algorithmes d'ordonnancement est souvent étudiée au sein de classe d'algorithmes, comme par exemple la classe des algorithmes d'ordonnancement à priorité fixe.

1.3. Résultats classiques

1.3.a. Condition nécessaire triviale

Une condition nécessaire (CN) triviale est que $U \leq 1$ pour qu'un système de tâches soit ordonnançable dans un système monoprocesseur.

1.3.b. Rate Monotonic ([23])

Contexte : Tâches indépendantes, périodiques, à échéance sur requête : une tâche est définie par $<C_i, T_i>$ avec implicitement $D_i = T_i$.

Algorithme : Affecter une priorité en fonction de la période : plus la période est petite, plus la priorité est élevée.

Résultats :

- Rate Monotonic est optimal dans la classe des algorithmes à priorité fixe pour les systèmes à échéance sur requête et à départ simultané. Cela signifie que si un algorithme à priorités fixes ordonnance fiablement un système de tâches à échéance sur requête τ, alors RM ordonnance fiablement τ. De façon équivalente, tout système à échéance sur requête ordonnançable par un algorithme à priorité fixe est ordonnançable par RM.
- **Théorème 1** *Pour des tâches indépendantes, le pire temps de réponse pour une tâche de priorité i advient lorsque toutes les tâches de priorité supérieure à i sont activées simultanément avec la tâche étudiée.*
  - Remarque : Si un algorithme ordonnance fiablement un système de tâches simultanées τ={$<C_i$, $T_i>$} $i = 1..n$, alors il ordonnance fiablement tout système de tâches τ={$<r_i, C_i, T_i>$} $i = 1..n$.

1.3.c. Deadline Monotonic ([22])

Contexte : Tâches indépendantes, périodiques, à échéance contrainte : une tâche est définie par $<C_i, D_i, T_i>$.

tel-00554265, version 1 - 10 Jan 2011

Algorithme : Affecter une priorité en fonction de l'échéance relative : plus l'échéance relative est petite, plus la priorité est élevée.

Résultats :
- Deadline Monotonic est optimal dans la classe des algorithmes à priorité fixe pour les systèmes à échéance contrainte et à départ simultané. Cela signifie que si un algorithme à priorités fixes ordonnance fiablement un système de tâches à échéance contrainte τ, alors DM ordonnance fiablement τ. De façon équivalente, tout système à échéance contrainte ordonnançable par un algorithme à priorité fixe est ordonnançable par DM.
- Notons bien que dans le cas des systèmes à échéance sur requête, un cas spécial des systèmes à échéance contrainte, DM et RM sont équivalents, c-à-d, tous les deux algorithmes affectent des priorités aux tâches d'une manière identique.

1.3.d. Théorème de l'instant critique ([23] généralisé par [3])

Dans un ordonnancement donné, tout dépassement d'échéance intervient nécessairement dans un intervalle de temps durant lequel le processeur exécute une tâche. Un tel intervalle de temps est appelé *période d'activité* du processeur. Pour les ordonnancements de tâches à priorité fixe, une *période d'activité de niveau i* est définie par un intervalle de temps durant lequel sont exécutées uniquement des tâches de niveaux de priorité supérieur ou égal à i.

Contexte : Algorithmes à priorité fixe.

Résultats :

Théorème 2
- *Le pire temps de réponse d'une tâche intervient lors de la plus longue période d'activité de son niveau de priorité.*
- *La plus longue période d'activité débute à l'instant critique lorsque toutes les tâches sont simultanées, c'est-à-dire à l'instant où elles sont toutes réveillées simultanément.*

Conséquence : Pour une simulation de tâches à départ simultanées, on pourrait se contenter de construire la première période d'activité du processeur en considérant toutes les tâches et vérifier au sein de celle-ci que toutes les échéances sont respectées. Cette période d'activité utilisé pour valider le système est appelé l'intervalle d'étude dans la littérature.

1.3.e. Intervalle d'étude : cyclicité des ordonnancements

Contexte :
- Dans [21], Leung et Merrill ont montré que l'intervalle d'étude est borné pour analyser la faisabilité des tâches indépendantes sur un processeur.
- Dans [12], Geniet et Grolleau ont généralisé ce résultat à (presque) tout algorithme d'ordonnancement déterministe avec tâches dépendantes ou indépendantes sur un processeur.

tel-00554265, version 1 - 10 Jan 2011

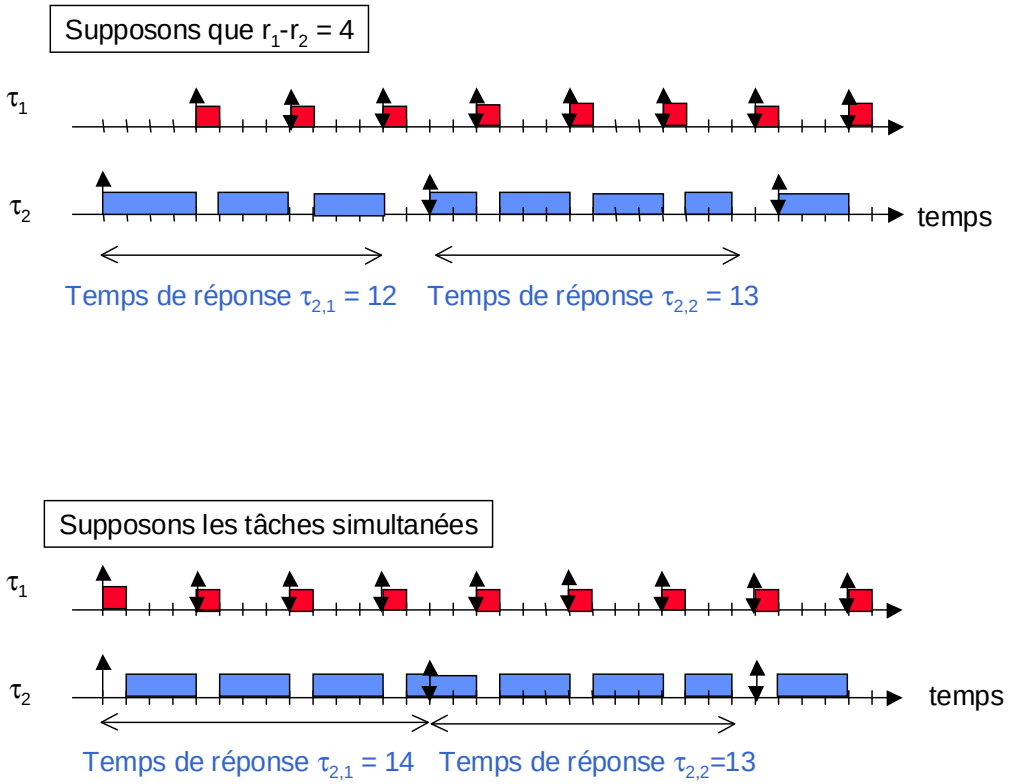

FIGURE 2.1 – *Théorème de l'instant critique.*

Résultats :

- Pour un algorithme d'ordonnancement déterministe A ordonnançant des tâches concrètes à départ simultané, alors s'il existe un ordonnancement valide sur l'intervalle d'étude [0..PPCM(T_i)[, alors A ordonnance fiablement le système de tâches, car l'ordonnancement complet est obtenu en répétant indéfiniment la séquence σ_A.
- Dans le même contexte, mais pour un système de tâches à départ différé, s'il existe une séquence valide σ_A crée par A sur l'intervalle d'étude [0..max(r_i)+2PPCM(T_i)[, alors le système est alors fiablement ordonnancé par A.

Conséquences

- Une simulation doit être menée sur une durée PPCM(T_i) pour des tâches simultanées et sur une durée max(r_i)+2PPCM(T_i) pour des tâches différées.
- Longeur de l'intervalle d'étude :
  - Non polynomial, une borne supérieure de PPCM(1,2,3,4..,m) est 3^m.
  - La simulation est donc une technique fortement exponentielle.
    - En général, proscrite des techniques de validation, mais utilisée pour comprendre un fonctionnement ou en ordonnancement hors-ligne.

Example 2 Soit un système de tâches non concrètes τ={τ_1=<1,4>, τ_2=<10,14>} donné sous la

tel-00554265, version 1 - 10 Jan 2011

forme $<C_i, T_i>$. Une affectation de priorités RM donnerait Prio(1)=1, Prio(2)=2 ■

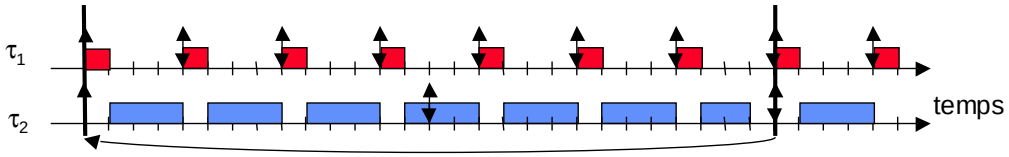

FIGURE 2.2 – *Cyclicité des ordonnancements.*

Remarque 1 *Le système se retrouve dans le même état à l'instant 28 qu'à l'instant 0.*

2. Tests d'ordonnançabilité

2.1. Définition

Un test de faisabilité (ou ordonnançabilité) vérifie le respect des échéances avant de démarrer l'application (en phase de conception).

2.2. Classification des tests d'ordonnançabilité

2.2.a. Test exact/test approché

Tests exacts : fournissent des conditions nécessaires et suffisantes de la garantie du respect des échéances des tâches.

Tests approchés : ne sont que des conditions suffisantes d'ordonnançabilité. Ils sont soit des approximations des tests exacts mais avec une complexité algorithmique moindre, soit ils ne permettent pas de conclure pour toutes les configurations de tâches. Dans ce dernier cas, lorsque la réponse des tests d'ordonnançabilité retourne Oui, la configuration de tâches est ordonnançable ou dans le cas inverse, le test ne peut pas conclure.

2.2.b. Techniques

L'analyse du facteur d'utilisation est un test d'ordonnançabilité qui revient à calculer le facteur d'utilisation d'une configuration de tâches puis à vérifier qu'il n'excède pas la valeur seuil propre à l'algorithme d'ordonnancement utilisé.

L'analyse du temps de réponse est un test d'ordonnançabilité en deux étapes :

1. Tout d'abord, le temps de réponse R_i (ou une borne supérieure) de chaque tâche est calculé.
2. Puis le respect des échéances est testé en vérifiant : $R_i \leq D_i$, $1 \leq i \leq n$ (complexité algorithmique en $\mathcal{O}(n)$).

L'analyse de la demande processeur revient à tester pour tout intervalle de temps $[t_1, t_2]$ que la durée maximum cumulée (ou une borne supérieure) des exécutions des requêtes qui ont leur réveil et leur échéance dans l'intervalle est inférieure à $t_2 - t_1$ (c'est-à-dire, n'excède pas la longueur de l'intervalle).

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

2.3. Concepts et notions de base liés aux tests d'ordonnançabilité

2.3.a. Période d'activité

Définition 1 *Une période d'activité du processeur est un intervalle de temps]a, b[de l'ordonnancement tel que le processeur a exécuté toutes les requêtes arrivées avant la date a et a terminé à la date b toutes les requêtes arrivées à partir de la date a.*

Définition 2 *Une période d'activité du processeur de niveau i est un intervalle de temps où le processeur n'exécute que des tâches ayant une priorité supérieure ou égale à i (level-i busy period).*

2.3.b. Caractérisation des scénarios pires cas

Définition 3 *Un scénario est l'ensemble des dates de réveil des tâches permettant de caractériser le début d'une période d'activité du processeur où se produira une faute temporelle (s'il en existe).*

Il y a deux cas :

1. Le scénario se produit nécessairement dans la vie du système : le test d'ordonnançabilité résultant sera alors exact et définit une condition nécessaire et suffisante pour que le système de tâches soit ordonnançable.
2. Le scénario ne se produit pas forcément dans la vie du système : le test sera alors approché puisque la demande processeur aura été surestimée. Ceci introduit donc du pessimisme dans le test d'ordonnançabilité, c'est-à-dire qu'une borne supérieure de la demande processeur est calculée. Le test sera uniquement une condition suffisante d'ordonnançabilité ; si le test renvoie vrai alors le système est ordonnançable, sinon on ne peut pas conclure.

2.3.c. Fonctions d'évaluation de la demande processeur

Nous présentons dans la suite deux fonctions fondamentale pour analyser un système de tâches. Nous supposons ci-après que les tâches sont à départ simultané ($r_i = 0$ pour tout i).

Request Bound Function :
C'est la demande processeur d'une tâche τ_i dans l'intervalle de temps $[0, t[$, notée $\text{RBF}(\tau_i, t)$ (Request Bound Function) est définie par la durée maximum cumulée de travail processeur associée aux réveils de τ_i sur $[0, t[$.

$$\text{RBF}(\tau_i, t) \stackrel{\text{def}}{=} \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{2.1}$$

Demand Bound Function :
La demande processeur des tâches devant se terminer avant la date t (c'est-à-dire dont l'échéance est avant ou à la date t), notée $\text{DBF}(t_1, t_2)$ (Demand Bound Function) est définie par la durée cumulée des requêtes dont la date d'activation et l'échéance sont dans l'intervalle de temps$[0, t]$:

$$\text{DBF}(0, t) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) \times C_i \tag{2.2}$$

Workload Function :
$W(t)$: fonction de travail du processeur sur l'intervalle $[0, t[$ cumule l'activité de toutes les tâches :

$$W(t) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \text{RBF}(\tau_i, t) \tag{2.3}$$

$W_i(t)$: La durée cumulée des tâches de priorité supérieure ou égale à i et réveillées dans l'intervalle $[0, t[$.

$$W_i(t) \stackrel{\text{def}}{=} \sum_{j=1}^{i} \text{RBF}(\tau_j, t) \tag{2.4}$$

2.3.d. Ensemble d'ordonnancement, point critique et temps de réponse

Les fonctions précédentes sont des fonctions en escalier dont la valeur évolue d'un C_i à chaque réveil de $\tau_i, 1 \leq i \leq n$. ENtre deux réveil ces fonctions sont stationnaires. Ainsi, pour exploiter de telles fonctions, il est suffisant de ne considérer que les dates de réveils des tâches, c'est-à-dire un point sur chaque palier de la fonction considérée. Ces instants remarquables sont appelés points d'ordonnancement.

Nous montrons dans la suite comment déterminer le pire temps de réponse de tâches à priorité fixe, échéances contraintes et à départ simultané. Nous définissons tout d'abord formellement les points d'ordonnancement, puis nous montrons comment déterminer le pire temps de réponse d'une tâche depuis la fonction 2.3.

Définition 4 ([20]) *L'ensemble des points d'ordonnancement (ou l'ensemble d'ordonnancement) S_i d'une tâche τ_i est l'ensemble d'instants où les tâches de priorité supérieure ou égale à τ_i sont activées :*

$$S_i \stackrel{\text{def}}{=} \{aT_j \mid j = 1 \dots i-1, a = 1 \dots \lfloor \frac{D_i}{T_j} \rfloor\} \cup \{D_i\} \tag{2.5}$$

Définition 5 ([31]) *Le point critique (ou point critique d'ordonnancement) de τ_i est le premier point d'ordonnancement de τ_i auquel la valeur de la Workload Function $W_i(t)$ est inférieure ou égale à la capacité du processeur :*

$$t^* \stackrel{\text{def}}{=} \min\{t \in S_i \mid W_i(t) \leq t\} \tag{2.6}$$

Définition 6 *Pour un système de tâches tel que $U \leq 1$, le pire temps de réponse d'une tâche τ_i peut être défini comme :*

$$R_i \stackrel{\text{def}}{=} (\min\{t > 0 \mid W_i(t) = t\}) \tag{2.7}$$

($W_i(t)$ dénote la demande processeur cumulée et sera définie formellement dans la prochaine section.)

Dans Figure 2.3 est une illustration de ces notions. Sur l'axe des abscisse, les points $t = \{2, 4, 6, 8, 10, 12, \dots\}$ sont les points d'ordonnancement de la tâche τ_i. Le point $t = 8$ est le point critique de τ_i. Enfin, le point sur la Workload function qui correspond à l'instant $t = 7$ est l'intersection entre la Workload Function $W_i(t)$ et la fonction de capacité du processeur, alors il définit le temps de réponse de τ_i.

tel-00554265, version 1 - 10 Jan 2011

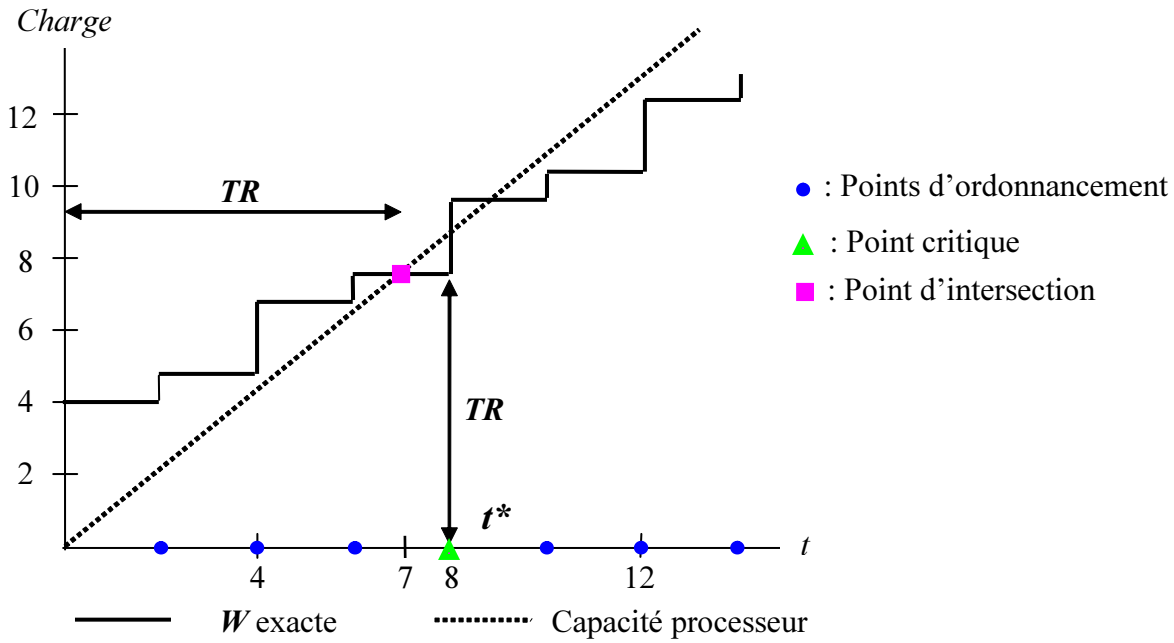

FIGURE 2.3 – Points d'ordonnancement, point critique et temps de réponse

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

2.3.e. Techniques d'approximation

1. **Algorithmes d'approximation**

   Ce sont les algorithmes utilisés pour résoudre de façon approchée des problèmes d'optimisation. La principale motivation pour concevoir un tel algorithme est de trouver une solution suffisamment proche de l'optimum dans un temps raisonnable.

2. **Borne d'erreur et ratio de garantie de performance**

   L'intérêt d'un algorithme d'approximation est de posséder une garantie de performance vis-à-vis de la valeur calculée par une méthode exacte (optimale). Précisément, pour toute instance I d'un problème d'optimisation combinatoire, soit $A(I)$ un algorithme approché calculant une borne supérieure du critère optimisé et $OPT(I)$ la valeur exacte du critère optimisé calculé par un algorithme optimal, alors la borne d'erreur $\epsilon\,(0 < \epsilon < 1)$ de l'algorithme A est définie par :

   $$\frac{A(I) - OPT(I)}{OPT(I)} \leq \epsilon \tag{2.8}$$

   Un algorithme approché A a une garantie de performance bornée par le ratio c si $r_A = A(I)/OPT(I) \leq c$ (pour un problème de minimisation) pour toute instance I du problème combinatoire étudié. Ce ratio définit donc les pires résultats que pourra atteindre l'algorithme approché A en considérant toutes les instances possibles d'un problème d'optimisation.

3. **Classification des algorithmes d'approximation**

   Trois classes d'algorithmes d'approximation sont généralement étudié dans la littérature :
   - Une approximation polynomiale est un algorithme d'approximation avec un ratio constant.
   - Un schéma d'approximation est un algorithme paramétrique, de paramètre ϵ, qui peut s'approcher aussi près que possible de la valeur optimale de la fonction optimisée. Le ratio d'un schéma d'approximation polynomiale (PTAS - Polynomial Time Approximation Scheme) s'écrit sous la forme : $r_A \leq 1 + \epsilon$.
   - Un schéma d'approximation est complet (FPTAS - Fully polynomial Time Approximation Scheme) s'il est un PTAS et que l'algorithme est en plus polynomial en fonction du paramètre $1/\epsilon$. Un FPTAS est le meilleur résultat d'approximation pouvant être obtenu pour résoudre un problème NP-Difficile.

4. **Application des algorithmes d'approximation aux problèmes de l'ordonnancement temps réel**

   Un test d'ordonnançabilité fondé sur l'analyse de la demande processeur n'est pas un problème d'optimisation, mais un problème de décision (c'est-à-dire, qui retourne une valeur binaire). Toutefois, les techniques d'approximation vont pouvoir être utilisées, moyennant une adaptation de la définition de la garantie de performance. Nous présentons à la fin de la section suivante le schéma d'approximation polynomiale de [15] utilisant le paramètre ϵ avec la sémantique suivante :
   - Si le test répond ordonnançable alors le système est ordonnançable quel que soit son comportement à l'exécution.
   - Si le test répond non ordonnançable, alors il est non-ordonnançable avec certitude sur un processeur plus lent (avec la vitesse $1 - \epsilon$). Mais sur un processeur de vitesse unitaire, aucune décision ne peut être prise.

tel-00554265, version 1 - 10 Jan 2011

3. Principaux tests d'ordonnançabilité

3.1. Analyse du facteur d'utilisation

3.1.a. Condition suffisante de Liu & Layland 73

Contexte :
Soit τ une configuration de n tâches périodiques indépendantes à échéance sur requête et à départ simultané.

Test :
Dans [23], le système τ est ordonnancé fiablement par RM si :

$$U \leq n(2^{1/n} - 1) \tag{2.9}$$

Analyse :

$$\lim_{n\to\infty} \left(n \times \left(2^{1/n} - 1 \right) \right) = \ln 2 \approx 0.69$$

- Il en résulte que tout système de tâches indépendantes à échéance sur requête de charge inférieure à 69% est ordonnancé fiablement par RM.
- La complexité du test est linéaire.

Ce test a été amélioré dans [9], où le système τ est prouvé fiablement ordonnancé par RM si :

$$U \leq \prod_{i=1}^{n} (1 + u_i) \tag{2.10}$$

Toutefois, même cette borne hyperbolique domine la borne de Liu et Layland, la limite de facteur d'utilisation reste la même dans le pire cas (c'est-à-dire, $ln2$).

Lorsque les périodes des tâches sont des multiples 2 à 2, alors une condition nécessaire et suffisante peut-être établie à l'aide du facteur d'utilisation du système de tâche : $U \leq 1$.

3.2. Analyse du temps de réponse

3.2.a. Cas des tâches à échéance contrainte (test de [18])

Contexte :
Tâches indépendantes à échéance.

Résultats :
- Soit $hp(i)$ l'ensemble des indices de tâches de priorité supérieure (ou égale) à la priorité de τ_i, en excluant i.
  - Par exemple, soient $\tau = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ de priorité décroissante, $hp(3) = \{1, 2\}$
- Le temps de réponse de τ_i est donné par la charge à traiter plus l'interférence des tâches de priorité supérieure, en démarrant à l'instant critique, car le pire temps de réponse de τ_i intervient dans la plus longue période d'activité de niveau de priorité prio(i), et celle-ci démarre (y compris pour les tâches à échéances arbitraires) à l'instant critique.

Test d'ordonnançabilité :
Le pire temps de réponse de τ_i est le plus petit point fixe de la suite :

$$R_i^{(0)} = C_i \tag{2.11a}$$

$$R_i^{(m+1)} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{(m)}}{T_j} \right\rceil C_j \tag{2.11b}$$

Si l'on note $R_i^{(*)}$ la valeur du point fixe, la condition nécessaire et suffisante pour qu'un système soit ordonnançable par une affectation de priorité (RM ou autre) est que :

$$\forall i \in [1..n], R_i^{(*)} \leq D_i \tag{2.12}$$

Analyse :
- La suite converge si et seulement si $U \leq 1$.
- Ce test est de complexité pseudo-polynomiale.

Une preuve détaillée sur ces deux derniers points est présentée dans [35] (Annexe A, p. 27).

3.2.b. Cas des tâches à échéance arbitraire (test de [19])

Contexte :
Tâches indépendantes à échéance arbitraire.

Résultats :
La formule de calcul de [18] ne peut pas s'appliquer telle qu'elle est définie, car elle ne prend pas en compte la possibilité pour une instance $\tau_{i,j}$ de la tâche τ_i d'interférer avec les instances suivantes de τ_i. Cependant le résultat suivant reste valide :
- Le temps de réponse de τ_i est donné par la charge à traiter plus l'interférence des tâches de priorité supérieure, en démarrant à l'instant critique, car le pire temps de réponse de τ_i intervient dans la plus longue période d'activité de niveau de priorité prio(i), et celle-ci démarre (y compris pour les tâches à échéances arbitraires) à l'instant critique.
- Il faut donc aller jusqu'à la fin de la période d'activité initiée par l'instant critique et regarder dans cette période d'activité quelle est l'instance de la tâche τ_i avec le plus long temps de réponse.

Test d'ordonnançabilité :
C'est l'idée exploitée par [19]. L'idée est qu'il faut tester toutes les échéances des instances situées dans la première période d'activité. On évalue ainsi l'activité cumulée des tâches de priorités supérieures ou égales à τ_i :

$$W_i^{(0)}(k) = kC_i \tag{2.13a}$$

$$W_i^{(m+1)}(k) = kC_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i^{(m)}(k)}{T_j} \right\rceil C_j \tag{2.13b}$$

tel-00554265, version 1 - 10 Jan 2011

$R_i^{(*)}(k) = W_i(k) - (k-1)T_i$ donne le temps de réponse de $\tau_{i,k}$ (i.e. k^{ieme} instance de τ_i).

On applique cette formule de la façon suivante : on commence à $k = 1$ (on a alors le même fonctionnement que [18]).

- Si $R_i^{(*)}(1) > D_i$, il y a violation d'échéance (on peut tout de même continuer si l'on souhaite obtenir le pire temps de réponse).
- Si $R_i^{(*)}(1) > T_i$, la deuxième instance fait partie de la première période d'activité, on termine donc pour $k = 1$, mais on sait qu'il faudra tester aussi $k = 2$. De même si $R_i^{(*)}(2) > 2T_i$, il faudra tester $k = 3$, etc. tant que $R_i^{(*)}(k) > kT_i$ on teste $k + 1$. Le processus converge si et seulement si $U \leq 1$. Chaque date de fin d'instance $\tau_{i,k}$ est comparée à $d_{i,k}$.

Si nous définissons $k_i \stackrel{\text{def}}{=} \min\left\{k \mid R_i^{(*)}(k) \leq kT_i\right\}$ et $R_i^{(*)} \stackrel{\text{def}}{=} \max_{k \leq k_i} R_i^{(*)}(k)$ alors $R_i^{(*)}$ est le pire temps de réponse de la tâche τ_i. Donc, la condition nécessaire et suffisante pour qu'un système soit ordonnançable par une affectation de priorité (RM ou autre) est que :

$$\forall i \in [1..n], R_i^{(*)} \leq D_i \tag{2.14}$$

Analyse :

Ce test est de complexité pseudo-polynomiale.

3.3. Analyse de la demande processeur

3.3.a. Test de [Leh 90] dans [19]

Contexte :

Soit τ une configuration de n tâches périodiques indépendantes à échéance arbitraire. Une tâche est définie par $<r_i, C_i, D_i, T_i>$.

Résultats :

Théorème 3 *Le pire temps de réponse pour une instance de τ_i se produit pendant une période d'activité du niveau i lancée par un instant critique $r_1 = \ldots = r_i = 0$.*

Selon ce théorème, pour déterminer l'ordonnançabilité du système, nous ne considérons que le pire scénario où se trouve l'instant critique.

Définition 7

$$W_i(k,x) \stackrel{\text{def}}{=} \min_{t \leq x} \left(\left(\sum_{j=1}^{i-1} C_j \left\lceil \frac{t}{T_j} \right\rceil + kC_i \right) / t \right) \tag{2.15}$$

La quantité de $\sum_{i-1} C_j \left\lceil \frac{t}{T_j} \right\rceil + kC_i$ donne les demandes processeur cumulatives totales faites par toutes les instances de $\tau_1, ., \tau_{i-1}$ et les premières k instances de τ_i pendant $[0, t]$. Le k^{ieme} instance de τ_i respectera son échéance si et seulement si cette quantité est inférieure ou égale à t pour au moins un certain $t \leq d_i$, parce qu'à un tel instant le processeur aura accompli tout kC_i et tout le travail exigé des tâches plus prioritaires. En fait, la plus petite valeur de t pour laquelle $\sum_{j=1}^{i-1} C_j \left\lceil \frac{t}{T_j} \right\rceil + kC_i = t$ est

tel-00554265, version 1 - 10 Jan 2011

l'instant où cette instance est accomplie. En outre, la période d'activité de niveau i qui a commencé à temps 0 finira avec la terminaison de la k^{ieme} instance de τ_i s'il n' y a plus de traitement à faire au niveau de priorité i ou supérieur.

Définition 8

$$N_i \stackrel{\text{def}}{=} \min\{k \mid W_i(k, kT_i) \leq 1\} \tag{2.16}$$

N_i est le nombre d'instances de la tâche τ_i faisant partie de la période d'activité de niveau i.

Test d'ordonnançabilité de la tâche τ_i : la condition nécessaire et suffisante pour qu'une tâche τ_i soit ordonnançable par une affectation de priorité (RM ou autre) est que :

$$\max_{k \leq N_i} W_i(k, (k-1)T_i + D_i) \leq 1 \tag{2.17}$$

Test d'ordonnançabilité du système de tâches : la condition nécessaire et suffisante pour qu'un système τ soit ordonnançable par une affectation de priorité (RM ou autre) est que :

$$\max_{1 \leq i \leq n} \max_{k \leq N_i} W_i(k, (k-1)T_i + D_i) \leq 1 \tag{2.18}$$

Analyse :

Ce test est de complexité pseudo-polynomiale (l'analyse présentée dans [35] tient dans le cas des échéances aribtraires). Il existe quelques methodes pour réduire les intervalles à considérer dans Eq. (2.17) dans [25] et dans [7], mais ces deux methodes nécessite au pire d'examiner $\mathcal{O}(2^n)$ points d'ordonnancement.

3.4. Test approché de Fisher et Baruah ([15])

Techniques d'approximation ont été récemment utilisées pour définir des tests de faisabilité approchés. Ces tests sont exécutés en temps polynomial en fonction de la taille d'ensemble de tâches et d'un paramètre de précision"$1/\epsilon$" (c-à-d, ils sont FPTASs).

3.4.a. Contexte

Tâches indépendantes à échéance contrainte.

3.4.b. Schéma d'approximation

La fonction $\text{RBF}(\tau_i, t)$ est une fonction en escalier non - décroissante. Le nombre de paliers dans cette fonction n'est pas borné polynomialement dans la taille du système à ordonnancer. Dans [15] est défini un schéma d'approximation polynomiale par ne considérer qu'un nombre borné k de paliers. Au-delà, une fonction linéaire (continue) sera utilisée pour définir une borne supérieure de $\text{RBF}(\tau_i, t)$. Le nombre de paliers va être défini à l'aide du paramètre d'erreur ϵ :

$$K = \left\lceil \frac{1}{\epsilon} \right\rceil - 1 \tag{2.19}$$

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

3.4.c. Approximation de la demande processeur

Définition 9 *La Request Bound Function approchée de la tâche τ_i, qui sera notée $\delta(\tau_i, t)$ est définie par :*

$$\delta(\tau_i, t) \stackrel{\text{def}}{=} \begin{cases} \text{RBF}(\tau_i, t) & si\ t \leq (k-1)T_i, \\ (t + T_i)u_i & sinon \end{cases} \tag{2.20}$$

Définition 10 *La Request Bound Function cumulative approchée (Workload Function approchée) de la tâche τ_i, qui sera notée $\widehat{W}_i(t)$ est définie par :*

$$\widehat{W}_i(t) \stackrel{\text{def}}{=} C_i + \sum_{j=1}^{i-1} \delta(\tau_j, t) \tag{2.21}$$

Théorème 4 *Dans un système de tâches périodiques (ou sporadiques), la tâche τ_i est ordonnançable par DM (Deadline Monotonic) si $\exists t \in$ (0,D_i] tel que $\widehat{W}_i(t) \leq t$.*

Théorème 5 *Si $\forall t \in$ (0,D_i], $\widehat{W}_i(t) > t$, tâche τ_i est non-ordonnançable par DM sur un processeur avec la vitesse $(1 - \epsilon)$.*

Définition 11 *Ensemble de points d'ordonnancement (Scheduling Set) est défini par :*

$$\widehat{S}_i \stackrel{\text{def}}{=} \{bT_j | j = 1 \ldots i-1, b = 1 \ldots k\} \cup \{D_i\} \tag{2.22}$$

3.4.d. Test d'ordonnaçabilité :

- Si pour toutes les tâches $\tau_i \in \tau$, il existe un certain $t \leq \widehat{S}_i$, tel que $\widehat{W}_i(t) \leq t$, alors τ est ordonnançable.
- Sinon, τ n'est garanti pour être non-ordonnançable que sur un processeur de capacité $(1 - \epsilon)$.

3.4.e. Analyse

Comme l'ensemble de points d'ordonnancement (Scheduling Set) possède un nombre polynomial d'entrées dans la taille du système de tâches à analyser et du paramètre de précision $1/\epsilon$, ce test d'ordonnançabilité est un schéma d'approximation complètement polynomiale (FPTAS).

4. Principales bornes de temps de réponse

4.1. Définition

Pour certains systèmes temps réels, tels que dans les systèmes de contrôle-commande, il est nécessaire de connaître le temps de réponse de tâche et non seulement la décision binaire sur l'ordonnançabilité des tâches fournie par les tests d'ordonnançabilité. Cette valeur définit une métrique importante

reliée au retard maximum d'une tâche (s'il ne respecte pas son échéance). Aucun algorithme polynomial n'est connu pour calculer R_i pour le modèle de tâches considéré. Calculer efficacement le pire temps de réponse sera résolu en utilisant les techniques d'approximation. Nous définissons maintenant formellement le pire temps de réponse approché (borne de temps de réponse) selon un paramètre de précision ϵ.

Définition 12 *Soit ϵ une constante et R_i le pire temps de réponse "exact" d'un tâche τ_i, alors une borne supérieure approchée de son temps de réponse $\widehat{R_i}$ satisfait :*

$$R_i \leq \widehat{R_i} \leq (1+\epsilon)R_i \tag{2.23}$$

Si l'algorithme d'approximation satisfait les conditions suivantes : $0 < \epsilon < 1$ et le temps d'exécution est polynomial en fonction de la taille d'entrée et de $1/\epsilon$, alors c'est un schéma d'approximation complet FPTAS).

Deux principales approches ont été conçues pour calculer des bornes supérieures de *wcrt* en temps polynomial, qui sont toutes les deux basées sur l'approximation linéaire de la fonction RBF (request bound function).

4.2. Bornes de temps de réponse de temps linéaire.

4.2.a. Borne de Sjödin et Hansson

Dans [34] [1] est présentée une manière simple de définir une borne inférieure du pire temps de réponse qui est continue en les paramètres du système. En utilisant le même principe de relaxation, une borne supérieure du pire temps de réponse d'une tâche τ_i peut être facilement définie :

$$R_i \leq \frac{\sum_{j=1}^{i} C_j}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} = ub_i^{SH} \tag{2.24}$$

4.2.b. Borne de Bini et Baruah

La borne ub_i^{SH} a été récemment améliorée par Bini et Baruah [6] et étendue aux tâches à échéances arbitraires : Bini et Baruah [6] ont montré que la pire charge de travail, qui est la durée cumulée des tâches que le processeur exécute une tâche τ_i dans n'importe quel intervalle de longueur t, peut être bornée par une fonction d'approximation linéaire (c-à-d, voir [6] pour plus de détails) :

$$\text{LA}^{BB}(\tau_i, t) \stackrel{\text{def}}{=} U_i\, t + C_i(1 - U_i) \tag{2.25}$$

En utilisant une telle fonction linéaire, [6] présente une borne supérieure du pire temps de réponse d'une tâche τ_i :

$$R_i \leq \frac{C_i + \sum_{j=1}^{i-1} C_j(1 - U_j)}{1 - \sum_{j=1}^{i-1} U_j} = sup_i^{BB} \tag{2.26}$$

1. Cette borne est été préalablement présentée dans [17].

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

4.2.c. Analyse

- Toutes les deux bornes sont de complexité linéaire.
- Mais nous avons montré dans [5, 10] que ces fameuses bornes supérieures ne possèdent pas de borne d'erreur constante (c-à-d, il existe des ensembles des tâches tels que les bornes supérieures des pires temps de réponse sont de c fois de plus de R_i où c est un nombre arbitraire). Ainsi, les algorithmes correspondants (en $\mathcal{O}(n)$) ne sont pas des algorithmes d'approximation pour calculer de bornes supérieures des pires temps de réponse. Mais, nous avons montré en utilisant une technique d'augmentation de ressource que ces bornes linéaires sont des bornes supérieures sur un processeur de vitesse unitaire et des bornes inférieures sur un processeur de vitesse deux fois plus lent. Ainsi, une augmentation de la capacité du processeur d'un facteur de 2 est une borne supérieure du prix à payer pour utiliser des bornes supérieures du temps de réponse efficacement calculable et qui sont des fonctions continues dans les paramètres du système.

4.3. Bornes de temps de réponse schéma d'approximation.

4.3.a. Borne de Fischer, Goossens, Nguyen et Richard

En utilisant l'approche de l'analyse approchée de faisabilité présentée par Fisher et Baruah (Section 3.4), nous avons montré comment estimer les pires temps de réponse approchés (bornes supérieures) dans [16, 26, 31].

Nous nous rappelons de la notion de l'instant critique d'une tâche donnée τ_i : c'est le premier point d'ordonnancement de τ_i auquel la valeur de la Workload Function $W_i(t)$ est inférieure ou égale à la capacité du processeur. Dans [31], nous avons montré que la valeur de la Workload Function $W_i(t)$ à ce point critique (si un tel point existe) correspond exactement au pire temps de réponse de τ_i.

Théorème 6 ([31]) *Le pire temps de réponse d'une tâche τ_i, tel que $W_i(t^*) \leq t^*$ (c.-à-d., la tâche est faisable), est exactement $R_i = W_i(t^*)$.*

Dans [16], nous avons proposé aussi une autre fonction pour approcher la fonction RBF sous l'hypothèse que tous les paramètres de tâches sont des nombres entiers.

$$\gamma(\tau_i, t) \stackrel{\text{def}}{=} \begin{cases} \left\lfloor \frac{t+T_i-1}{T_i} \right\rfloor C_i & \text{si } t \leq (k-1)T_i, \\ (t + T_i - 1)\frac{C_i}{T_i} & \text{sinon.} \end{cases} \tag{2.27}$$

A partir de cette $\gamma(\tau_i, t)$, nous avons défini un schéma d'approximation en suivant le principe de Section 3.4. Ensuite, se basant sur ce FPTAS et employant Théorème 6, nous avons proposé une "méthode déduite" pour déduire une borne supérieure de temps de réponse.

Définition 13 ([16]) *Considérons une tâche τ_i telle qu'il existe un instant t satisfaisant $\widehat{W}_i(t) \leq t$, alors une borne supérieure de son pire temps de réponse est définie par :*

$$\begin{aligned} \widehat{t^*} &\stackrel{\text{def}}{=} \min\left\{ t \in \widehat{S}_i \middle| \widehat{W}_i(t) \leq t \right\}, \\ \widehat{R_i} &\stackrel{\text{def}}{=} \widehat{W}_i(\widehat{t^*}) \end{aligned} \tag{2.28}$$

4.3.b. Analyse

- Notons bien que si l'algorithme d'ordonnançabilité ne donne pas une réponse positive, alors notre approche ne peut dériver aucune borne supérieure (mais, nous pouvons employer la borne définie in [6] par exemple).
- Comme le test de Fisher et Baruah (Section 3.4), l'algorithme présenté dans [**?**] pour calculer des bornes de temps de réponse est un FPTAS, c-à-d, la complexité est polynomiale dans la taille du système de tâches à analyser et dans le paramètre de précision $1/\epsilon$.
- Nous avons montré que ces bornes supérieures des pires temps de réponse n'ont pas de garantie de performance non plus, c-à-d, une borne supérieure du temps de réponse d'une tâche peut être très éloignée de la valeur exacte du temps de réponse.

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Chapitre 3

Continuité et ordonnançabilité

1. Introduction

Pendant un processus de conception interactive et de prototypage rapide de système, le concepteur fait typiquement un grand nombre d'appels à un algorithme d'analyse de l'ordonnançabilité, puisque les hypothèses de conception sont modifiées selon les résultats d'analyse d'ordonnançabilité (et d'autres techniques d'analyse comme le calcul des temps de réponse). Dans de tels scénarios, un algorithme pseudo-polynomial pour calculer l'ordonnançabilité de l'ensemble de tâches peut être inadmissiblement lent ; au lieu de cela, il peut être acceptable d'employer un algorithme plus rapide qui fournit un résultat approché plutôt qu'exact (Le calcul des temps de réponse exacts peut-être particulièrement coûteux quand les valeurs des paramètres sont des grands nombres).

Une des caractéristiques du calcul des pires temps de réponse est que les fonctions utilisées dans le test ne sont pas continues (à cause de la partie entière déterminant le nombre d'instances réveillées dans un intervalle de temps $[0, t)$ dans l'équation 5.1). Ce type de discontinuité est un obstacle majeur à la définition d'un processus de conception incrémental et interactif de systèmes temps réel. Idéalement, un tel processus de conception devrait explorer (à l'aide d'un logiciel interactif) l'espace des ordonnancements faisables en fonction des valeurs des paramètres des tâches. La conception serait de plus considérablement facilitée si des changements mineurs d'un des paramètres ne mènent qu'à des modifications mineures des propriétés de système et ne remettent pas en cause sa conception dans sa globalité.

Malheureusement, ces discontinuités sont inévitables, puisqu'elles résultent du réveil périodique des tâches, et pas uniquement de la technique de calcul des temps de réponse. La conséquence est qu'une modification mineure d'un des paramètres des tâches peut remettre en cause la conception du système complet avec les conséquences dramatiques quant aux coûts de développement du système.

Pour cette raison, nous croyons qu'en phase de conception d'un système temps réel, il y a un certain avantage à étudier *des bornes supérieures* du pire temps de réponse qui ne souffrent pas de telles discontinuités en fonction des paramètres de système. Si nous employons des bornes supérieures des temps de réponse ne reposant que sur des fonctions continues (plutôt que les des fonctions discontinues) alors l'espace des systèmes faisables devient alors continu. Par exemple, si nous étions à un point faisable dans l'espace d'état de conception, nous pourrions sans risque faire de petits changements sur

les paramètres de tâches sans conséquence majeure sur la conception complète du système.

Bien évidemment, une telle borne supérieure continue pour valider le système *n'est pas nécessairement de bonne qualité, voire serrée* . Clairement, en choisissant une borne supérieure continue, nous réalisons un compromis entre l'exactitude des calculs et les propriétés associées à la continuité, qui ont été mentionnées précédemment. Toutefois, il est souhaitable que cette perte d'exactitude soit *quantifiée* numériquement pour que le concepteur de système puisse déterminer si la perte d'exactitude vaut les profits que les bornes continues fournissent au processus de conception de système.

Pour récapituler les remarques faites ci-dessus : nous cherchons les bornes supérieures du pire temps de réponse qui soient *continues* en fonction des paramètres de système ; *efficacement calculables* ; et aient *une déviation quantifiable* par rapports aux pires temps de réponse exactes. Dans la suite de ce chapitre nous étudions deux bornes et analysons leurs propriétés. Le travail présenté ci-après est le fruit d'une collaboration avec S. Baruah et E. Bini [5, 10].

2. Borne de Sjödin et Hansson

Nous avons déjà présenté la borne de Sjödin et Hansonn ([17,34]) dans le chapitre précédent. Cette borne inférieure définit le pire temps de réponse d'une tâche τ_i qui est continue en les paramètres du système de tâches à échéance contrainte. Nous rappelons la formulation de cette borne supérieure du pire temps de réponse d'une tâche τ_i afin de faciliter la lecture et y faire référence dans la suite de ce chapitre :

$$R_i \leq \frac{\sum_{j=1}^{i} C_j}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} = ub_i^{SH} \tag{3.1}$$

Preuve : Nous rappelons de l'Eq. (2.4) que la fonction de travail $W_i(t)$ de la tâche τ_i est égale à $C_i + \sum_{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j$.

Dans [18] est indiqué que le pire temps de réponse R_i d'une tâche τ_i est le petit point fixe de la suite $W_i(t)$:

$$\begin{aligned} W_i(R_i) &= R_i \\ \Leftrightarrow R_i &= C_i + \sum_{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil \end{aligned}$$

Depuis nous avons $\left\lceil \frac{R_i}{T_j} \right\rceil < \frac{R_i}{T_j} + 1 \forall j$, nous obtenons :

tel-00554265, version 1 - 10 Jan 2011

$$\begin{aligned}
\left\lceil \frac{R_i}{T_j} \right\rceil C_j &\leq \left(\frac{R_i}{T_j} + 1 \right) C_j \ 1 \leq j \leq i-1 \\
\left\lceil \frac{R_i}{T_j} \right\rceil C_j &\leq \left(\frac{R_i}{T_j} \right) C_j + C_j \ 1 \leq j \leq i-1 \\
R_i = C_i + \sum_{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j &\leq \sum_i C_j + R_i \sum_{i-1} \frac{C_j}{T_j} \\
R_i \left(1 - \sum_{i-1} \frac{C_j}{T_j} \right) &\leq \sum_i C_j \\
R_i &\leq \frac{\sum_i C_j}{\left(1 - \sum_{i-1} \frac{C_j}{T_j} \right)} = ub_i^{SH}
\end{aligned}$$

■

3. Borne de Bini et Baruah

La borne ub_i^{SH} a été récemment améliorée par Bini et Baruah [6] tout en considérant les tâches à échéance arbitraire. Cette borne déjà présentée dans le chapitre précédent est rappelée ci-après :

$$R_i \leq \frac{C_i + \sum_{j=1}^{i-1} C_j (1 - \frac{C_j}{T_j})}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} = ub_i^{BB} \tag{3.2}$$

La preuve de l'exactitude de cette dernière borne du temps de réponse est un peu complexe puisqu'elle tient aussi pour les tâches à échéance arbitraire (voir [6] pour des détails). Nous fournissons ci-dessous une preuve plus simple dans le cas des tâches à échéance contrainte :

Lemme 1 *Une période d'activité de niveau i ne peut pas se terminer dans aucun intervalle $[kT_j; kT_j + C_j)$, $1 \leq j \leq i$, où k est un nombre entier arbitraire.*

Preuve : Nous prouvons le résultat par contradiction. Supposons qu'une période d'activité est finie à la date $t \in [kT_j; kT_j + C_j)$, alors l'instance correspondante de τ_j est réveillée et ne peut pas être terminée avant le temps $kT_j + C_j$, même en s'exécutant sans aucune préemption. Ceci contredit le fait que la période d'activité de niveau i est finie. ■

Lemme 2 *La demande processeur de la tâche τ_j, $1 \leq j \leq i$ dans une période d'activité de niveau i est dans le pire cas définie par :*

$$\text{RBF}(\tau_j, t) = \left\lceil \frac{t}{T_j} \right\rceil C_j = \left\lfloor \frac{t + T_j - C_j}{T_j} \right\rfloor C_j$$

où t est la durée de la période d'activité synchrone de niveau i.

tel-00554265, version 1 - 10 Jan 2011

Preuve : Nous considérons une tâche τ_j, $1 \leq j \leq i$. La demande processeur exacte de τ_j est définie par Request Bound Function $\text{RBF}(\tau_j, t) = \lceil \frac{t}{T_j} \rceil C_j$.

Nous prouvons que quand le temps t égale la longueur de la période d'activité de niveau i, alors $\text{RBF}(\tau_j, t) \leq \left\lfloor \frac{t+T_j-C_j}{T_j} \right\rfloor C_j$, $1 \leq j \leq i$.

Nous emploierons la règle de l'égalité indirecte : Soient a, b, k des nombres naturels, puis $a = b$ si, et seulement si, $\forall k, k \leq a \Leftrightarrow k \leq b$. Considérons un nombre entier arbitraire k tels que $k \leq \left\lceil \frac{t}{T_j} \right\rceil$. Puisque $\lceil x \rceil < x + 1$ pour tout nombre réel x, alors :

$$k \leq \left\lceil \frac{t}{T_j} \right\rceil < \frac{t}{T_j} + 1$$

Ainsi, $T_j(k-1) < t$. Selon le Lemme 5, t doit satisfaire $rT_j + C_j \leq t$, pour $r = k - 1$. Par conséquent :

$$\begin{aligned} T_j(k-1) + C_j &\leq t \\ k &\leq \frac{t + T_j + C_j}{T_j} \end{aligned}$$

Puisque k est un nombre entier :

$$k \leq \left\lfloor \frac{t + T_j - C_j}{T_j} \right\rfloor$$

En utilisant la règle de l'égalité indirecte et puisque k est arbitraire, alors le lemme suit. ■

Théorème 7 *Soit R_i le pire temps de réponse de τ_i :*

$$R_i \leq \frac{C_i + \sum_{j=1}^{i-1} C_j(1 - U_j)}{1 - \sum_{j=1}^{i-1} U_j}$$

Preuve : Nous rappelons de l'Eq. (2.11) que dans le cas des systèmes de tâches à échéance, le pire temps de réponse R_i est le plus petit point fixe de la suite $W_i(t)$, d'une façon équivalente, $R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$. Puisque le pire temps de réponse d'une tâche τ_i est la durée de la période d'activité de niveau i [20], alors en utilisant le Lemme 6 :

$$\begin{aligned} R_i &= C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{R_i + T_j - C_j}{T_j} \right\rfloor C_j \\ &\leq C_i + \sum_{j=1}^{i-1} \left(\frac{R_i + T_j - C_j}{T_j} \right) C_j \\ &\leq \frac{C_i + \sum_{j=1}^{i-1} C_j(1 - U_j)}{1 - \sum_{j=1}^{i-1} U_j \frac{C_j}{T_j}} \end{aligned}$$

■

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

4. Analyse

Les bornes exactes des pires temps de réponse ne sont pas nécessairement continues en fonction des paramètres de tâche ; en outre, la complexité informatique de la détermination de telles bornes est actuellement inconnue (et suspectée ne pas être polynomiale à moins que P = NP). Les bornes supérieures ub^{SH} et ub^{BB}, cependant, sont continues et sont efficacement calculables à temps linéaire en fonction du nombre de tâches. Ces fonctions sont continues, calculables en temps polynomial, mais quelle est la qualité de ces valeurs approchées des temps de réponse ? Dans la suite de ce paragraphe, essayons de répondre à cette question en quanitifiant l'écart entre la borne et la valeur exacte du temps réponse, en employant des techniques d'optimisation combinatoire.

4.1. Analyse de ratio classique

En optimisation combinatoire, la garantie de performance d'un algorithme approché est souvent analysée par son *ratio d'approximation*. Soit a la valeur obtenue par l'algorithme A qui résout un problème de minimisation, et *opt* la valeur exacte calculée par un algorithme optimal (c.-à-d., le minimum global de la fonction optimisée) ; l'algorithme A a un ratio d'approximation c, où $c \geq 1$, si et seulement si $opt \leq a \leq c \times opt$ pour toutes les entrées possibles de l'algorithme A (Si un tel c n'existe pas, alors on dit que l'algorithme A n'a aucun ratio d'approximation.).

Les prochains résultats montrent que les deux bornes supérieures ub_i^{SH} et ub_i^{BB} présentées dans les sections précédentes n'ont pas de garantie de performance.

4.1.a. La borne de Sjödin et Hansson

Dans [30], nous avons montré que cette borne supérieure bien connue n'a aucune garantie de performance (c.-à-d. il existe des ensembles de tâches tels que la borne supérieure ub_i^{SH} est c fois plus grande que R_i, où c est un nombre arbitrairement grand). Ainsi, l'algorithme correspondant en $\mathcal{O}(n)$ n'est pas un algorithme d'approximation pour calculer le pire temps de réponse.

Théorème 8 *[30] La borne supérieure ub_i^{SH} calculée par Eq. (3.1) n'a aucune garantie de performance.*

Preuve : Considérons le système de tâches suivant avec deux tâches : $\tau_1(1\epsilon, 1, 1)$ et $\tau_2(K\epsilon, K, K)$ données sous la forme $< C_i, D_i, T_i >$, où ϵ satisfait $0 < \epsilon < 1$ et K est un nombre entier arbitraire tel que $K > 1$. Notons que les périodes sont proportionnelles deux à deux, ainsi une condition nécessaire et suffisante pour que le système de tâches soit ordonnançable selon la règle d'ordonnancement Rate Monotonic est $C_1/T_1 + C_2/T_2 \leq 1$.

Le facteur d'utilisation est :

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = (1 - \epsilon) + \frac{K\epsilon}{K} = 1$$

Ainsi, le système de tâches est ordonnançable avec Rate Monotonic et le pire temps de réponse exact et la borne supérieure pour tâche τ_2 sont :

$$R_2 = K \qquad ub_2^{SH} = \frac{1 + (K-1)\epsilon}{\epsilon}$$

Ainsi, la pire garantie de performance est obtenue :

$$\lim_{\epsilon \to 0} \frac{ub_2^{SH}}{R_2} = \lim_{\epsilon \to 0} \frac{1}{K\epsilon} + \frac{(K-1)}{K} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} = \infty$$

■

4.1.b. La borne de Bini et Baruah

Théorème 9 *La borne supérieure ub_i^{BB} de [6] (Eq. (4.5) dessus) n'a pas de ratio d'approximation.*

Preuve : Nous prouvons ceci en démontrant un système et une tâche τ_i avec lesquels ub_i^{BB}/R_i tend à ∞. Toute tâche dans notre système aura $D_i = T_i$; par conséquent, nous représentons des paramètres d'une tâche τ_i par une paire ordonnée (C_i, T_i).

Considérons l'ensemble de tâches suivant :

$\tau_1 = (K, 2K + \epsilon)$, $\tau_2 = (K, 2K + \epsilon)$ et $\tau_3 = (\epsilon, 2K + \epsilon)$, où ϵ est un nombre positif arbitrairement petit et K est un nombre quelconque plus grand que ϵ.

Dans la Figure 3.1, nous dépeignons la séquence d'arrivée synchrone de cet ensemble tâches. Nous vérifions que l'utilisation $U_1 + U_2 + U_3$ est 1 et puisque les tâches ont toutes les périodes égales à $2K + \epsilon$, alors $R_3 = 2K + \epsilon$ en utilisant la politique d'ordonnancement du Rate Monotonic. La borne supérieure du pire temps de réponse est :

$$\begin{aligned} ub_3^{BB} &= \frac{\epsilon + 2K(1 - \frac{K}{2K+\epsilon})}{1 - \frac{2K}{2K+\epsilon}} \\ &= \frac{\epsilon(2K+\epsilon) + 2K(2K+\epsilon-K)}{2K+\epsilon-2K} \\ &= \frac{(2K+\epsilon)\epsilon + 2K(K+\epsilon)}{\epsilon} \\ &= 4K + \epsilon + \frac{2K^2}{\epsilon} \end{aligned}$$

Ainsi,

$$\lim_{\epsilon \to 0} ub_3^{BB} = \lim_{\epsilon \to 0} \left(4K + \epsilon + \frac{2K^2}{\epsilon} \right) = \infty$$

et en conséquence le ration d'approximation n'est pas borné et le théorème est prouvé. ■

4.2. Analyse d'augmentation de ressource

Les théorèmes ci-dessus indiquent que les bornes supérieures ub_i^{SH} et ub_i^{BB} du temps de réponse de [6, 34] n'offrent pas de garantie de performance, selon la mesure conventionnelle de ratio d'approximation. Dans les paragraphes suivants, nous appliquons la technique d'augmentation de ressource pour mesurer la déviation de ces bornes par rapport aux pires temps de réponse des tâches.

tel-00554265, version 1 - 10 Jan 2011

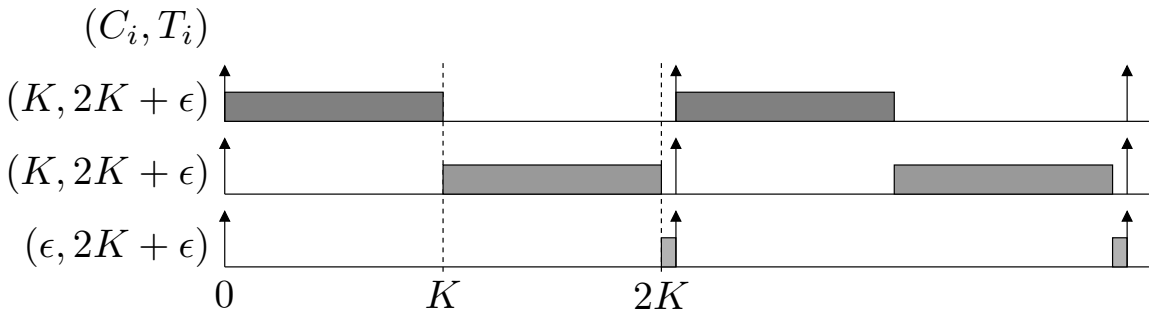

FIGURE 3.1 – *Exemple avec aucun ratio d'approximation de la borne de Bini et Baruah.*

4.2.a. Borne de Bini et Baruah

Nous introduisons la notation suivante :

$$\text{LA}^{BB}(\tau_i, t) \stackrel{\text{def}}{=} U_i\, t + C_i(1 - U_i) \tag{3.3}$$

Lemme 3

$$\forall t \geq C_i \qquad \text{LA}^{BB}(\tau_i, t) \leq 2\,\text{RBF}(\tau_i, t)\ .$$

Sketch de preuve: Nous rappelons de l'Eq. (5.1) que

$$\text{RBF}(\tau_i, t) = \left\lceil \frac{t}{T_i} \right\rceil C_i$$

Puisque $\text{RBF}(\tau_i, t)$ est une fonction en escalier, $\text{LA}^{BB}(\tau_i, t)$ et une fonction linéaire strictement croissante, il est clair que le ratio $\text{LA}^{BB}(\tau_i, t)$ / $\text{RBF}(\tau_i, t)$ est une fonction monotone croissante par morceaux $]kT_i, (k+1)T_i]$ $k \geq 0$, c.-à-d. $\text{LA}^{BB}(\tau_i, t)$ / $\text{RBF}(\tau_i, t)$ est strictement croissante sauf à l'instant t qui est multiple de la période T_i où $\text{LA}^{BB}(\tau_i, t)$ / $\text{RBF}(\tau_i, t)$ atteint alors un maximum local. Puisque les maxima locaux de $\text{LA}^{BB}(\tau_i, t)$ / $\text{RBF}(\tau_i, t)$ se trouvent dans les instants $t = (k+1)T_i$ $k \geq 0$, alors pour trouver le maximum global de cette fonction, nous ne devons considérer que les valeurs $kT_i, k \geq 1$.

Nous avons :

$$\begin{aligned} \frac{\text{LA}^{BB}(\tau_i, kT_i)}{\text{RBF}(\tau_i, kT_i)} &= \frac{kC_i + C_i(1 - U_i)}{kC_i} \\ &= 1 + \frac{1 - U_i}{k} \end{aligned}$$

Puisque $U_i \geq 0$ $\forall i \geq 1$ et $k \geq 1$, ce ratio est inférieur de 2. Alors, le lemme est prouvé. ■

Lemme 4 *Borne ub_i^{BB} est égale à la plus petite valeur de t satisfaisant l'équation suivante*[1] *:*

$$t = C_i + \sum_{j<i} \text{LA}^{BB}(\tau_j, t) \tag{3.4}$$

1. En fait, ub_i^{BB} s'avère justement être la *seule* valeur de t satisfaisant cette équation ; cependant, ce fait n'est pas utile à notre propos.

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Preuve : D'abord, nous prouvons que ub_i^{BB} est une solution à cette équation. Par définition de ub_i^{BB} de l'Eq. (4.5), nous avons :

$$
\begin{aligned}
ub_i^{BB} &= \frac{C_i + \sum_{j<i} C_j(1-U_j)}{1-\sum_{j<i} U_j} \\
ub_i^{BB}(1-\sum_{j<i} U_j) &= C_i + \sum_{j<i} C_j(1-U_j) \\
ub_i^{BB} &= C_i + \sum_{j<i} U_j\, ub_j^{BB} + \sum_{j<i} C_j(1-U_j) \\
ub_i^{BB} &= C_i + \sum_{j<i} \text{LA}(\tau_j, ub_i^{BB})
\end{aligned}
$$

donc impliquant que ub_i^{BB} est la solution de l'Eq. (3.4).

Pour voir que ub_i^{BB} est la *plus petite* valeur de t satisfaisant l'Eq. (3.4), observons que le taux d'accroissement à droite de l'équation (RHS) dans l'Eq. (3.4) par rapport à t est $(\sum_{j<i} U_j)$. Ce dernier terme est forcément < 1 et par conséquent, le partie droite de l'équation Eq. (3.4) ne se développe pas plus rapidement que sa partie gauche (LHS). De plus, RHS=LHS à $t = ub_i^{BB}$, ce doit être le cas pour RHS>LHS et pour tout $t < ub_i^{BB}$. Par conséquent le lemme est prouvé. ■

Nous pouvons employer le Lemme 4 pour prouver une borne sur l'augmentation de ressource pour ub_i^{BB}. Puisque ub_i^{BB} est la plus petite valeur de t à satisfaire l'Eq. (3.4) plus haut, ce doit être le cas pour tous $t < ub_i^{BB}$

$$
\begin{aligned}
t &< C_i + \sum_{j<i} \text{LA}^{BB}(\tau_j, t) \\
t &< C_i + 2\sum_{j<i} \left\lceil \frac{t}{T_j} \right\rceil C_j \quad \text{(utiliser Lemma 3)} \\
t &< 2\left(C_i + \sum_{j<i} \left\lceil \frac{t}{T_j} \right\rceil C_j \right) \\
\frac{1}{2}\, t &< C_i + \sum_{j<i} \left\lceil \frac{t}{T_j} \right\rceil C_j
\end{aligned}
$$

C'est-à-dire, le travail cumulée dans l'intervalle $[0, t)$ des instances ayant une priorité plus grande ou égale à τ_i avant la fin de la première instance de la tâche τ_i excède la capacité d'un processeur avec une capacité de calcul 1/2.

Par conséquent aucun $t < ub_i^{BB}$ ne peut correspondre au pire temps de réponse de τ_i si un processeur de capacité de calcul 1/2 est considéré. Le résultat suivant (Théorème 10) suit :

Théorème 10 *La borne ub_i^{BB} (Eq. (4.5)) est*

1. *Une borne supérieure du pire temps de réponse de τ_i ; et*

2. *Une borne inférieure du pire temps de réponse de* τ_i si le système est implémenté sur un processeur avec une vitesse $1/2$.

Nous montrons maintenant par un exemple simple que cette borne d'augmentation de ressource est serrée. Considérons l'ensemble de K tâches où $\tau_i = (K+1, 2K^2)$, $\forall i = 1, \ldots, K-1$, et $\tau_K = (1, 2K^2)$. Dans un processeur fonctionnant à la moitié de la vitesse, des durées d'exécution des tâches sont doublées et le facteur d'utilisation de système est $U = \frac{2(K-1)(K+1)+2}{2K^2} = 1$. Depuis les périodes sont toutes égales à $2K^2$, le temps de réponse de τ_K sur le processeur de capacité $1/2$ (dénoté par $\hat{R}_K$) est $2K^2$, parce que

$$\begin{aligned} 2\,C_K + \sum_{j=1}^{K-1} \left\lceil \frac{\hat{R}_K}{T_j} \right\rceil 2\,C_j &= 2 + (K-1) \left\lceil \frac{2K^2}{2K^2} \right\rceil 2(K+1) \\ &= 2(1 + (K-1)(K+1)) = 2K^2 = \hat{R}_K \end{aligned}$$

Maintenant calculons le ratio entre la borne supérieure de temps de réponse ub_K^{BB} et le temps de réponse exact $\hat{R}_K$ de τ_K sur le processeur de moitié-vitesse.

$$\begin{aligned} \lim_{K \to +\infty} \frac{ub_K^{BB}}{\hat{R}_K} &= \frac{1 + (K-1)(K+1)\left(1 - \frac{K+1}{2K^2}\right)}{2K^2\left(1 - (K-1)\frac{K+1}{2K^2}\right)} \\ &= \frac{2K^2 + (K^2-1)(2K^2-K-1)}{2K^2(K^2+1)} = 1 \end{aligned}$$

ce qui signifie que pendant que K se développe nous pouvons établir des ensembles de tâches dont le ratio d'augmentation de ressource est arbitrairement près de $1/2$.

Comment le concepteur de systèmes va-t-il interpréter le Théorème 10 ci-dessus ? D'abord, il est garanti que ub_i^{BB} est en effet une borne supérieure sur R_i ; par conséquent, c'est une évaluation sûre du pire temps de réponse exact. Et tandis que le Théorème 10 ne peut pas borner la quantité par laquelle ub_i^{BB} dépasse la valeur réelle de R_i (en effet, la Subsection 4.1 a prouvé qu'une telle borne n'existe pas), le concepteur est assuré que le pire temps de réponse n'aurait pas plus être meilleur que ub_i^{BB} si le système avait été implémenté sur un processeur deux fois plus lent.

Autrement dit, *une augmentation de la capacité du processeur d'un facteur deux est une borne supérieure du prix à payer pour utiliser une borne supérieure du temps de réponse calculable efficacement et qui est une fonction continue dans les paramètres du système.*

4.2.b. Borne de Sjödin et Hansson

Il est simple de voir que la borne supérieure ub_i^{SH} est dominée par la borne ub_i^{BB}. Donc le résultat du Théorème 10 s'applique aussi à la borne de Sjödin et Hansson.

5. Expérimentations

Dans la section précédente, nous avons établi la garantie pire cas de ub_i^{BB} pour deux modèles différents d'approximation : ratio d'approximation classique et le ratio d'augmentation de ressource.

tel-00554265, version 1 - 10 Jan 2011

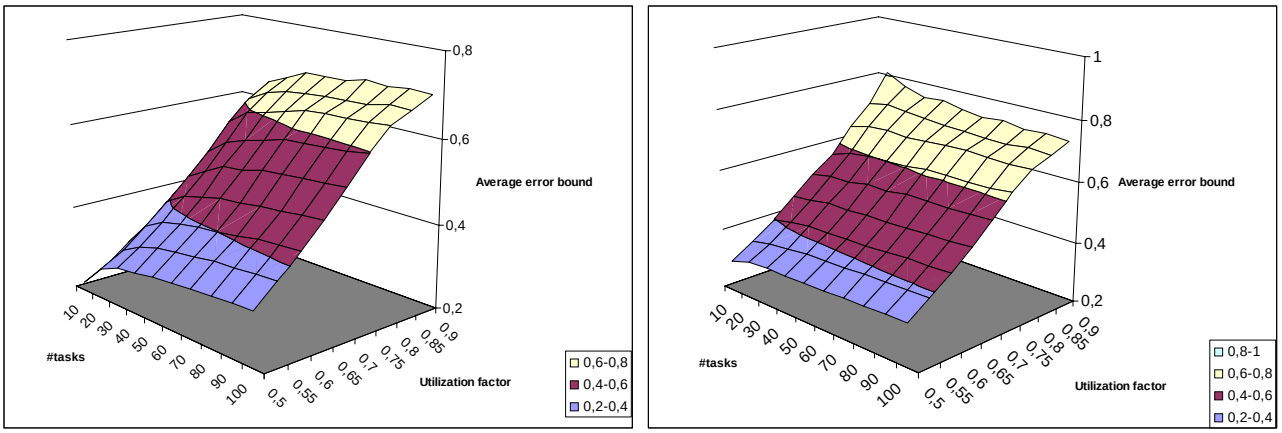

FIGURE 3.2 – *(a) SupB (Equation 4.5), (b) SupH (Equation 3.1).*

Dans cette section, nous décrivons des expérimentations numériques que nous avons exécutées afin de capturer la garantie de performance en moyenne.

5.1. Modèle stochastique

Nous avons aléatoirement généré des ensembles de tâches avec échéances relatives non reliées aux périodes. Nous avons produit d'abord les facteurs d'utilisation des tâches, puis leurs périodes et délais critiques ; et pour finir, leurs pires durées d'exécution. La distribution probabiliste considérée est la distribution uniforme. Les facteurs d'utilisation non biaisés étaient produits en employant l'algorithme UUniFast présenté dans [8]. Les périodes ont été aléatoirement générées dans l'intervalle [1,2500] et les délais critiques de l'intervalle [1,2600]. Les pires durées d'exécution sont calculées afin d'avoir des facteurs d'utilisation non biaisés (voir [8] pour des détails). Pour une taille de problème donnée, la simulation a été repliquée 400 fois.

5.2. Comparaison des bornes linéaires

Nous avons comparé d'abord la borne de l'Eq. (4.5) — referencée ici comme la borne SupB — et la borne de l'Eq. (3.1) — referencée comme la borne SupH — au pire temps de réponse exact, ce que nous avons calculé en temps pseudo-polynomial en utilisant l'algorithme exact. Nous avons contrôlé deux indicateurs : la déviation moyenne, calculée comme $(ub - opt)/opt$ (où ub est la valeur calculée par la borne linéaire et opt est le pire temps de réponse exact) et le nombre de systèmes de tâches qui ont été validés en utilisant les bornes linéaires.

Comme représenté dans la Figure 3.2, il s'avère que la borne d'erreur moyenne dépend principalement du facteur d'utilisation pour les deux bornes linéaires. La taille de l'ensemble de tâches a un impact faible sur la borne d'erreur moyenne. Dans la Figure 3.3, nous observons que la borne SupB (Eq. (4.5)) améliore approximativement de 5 % la borne SupH (Eq. (3.1)). La Figure 3.4 compare le nombre de systèmes de tâches faisables qui ont été validés par les bornes linéaires présentées.

tel-00554265, version 1 - 10 Jan 2011

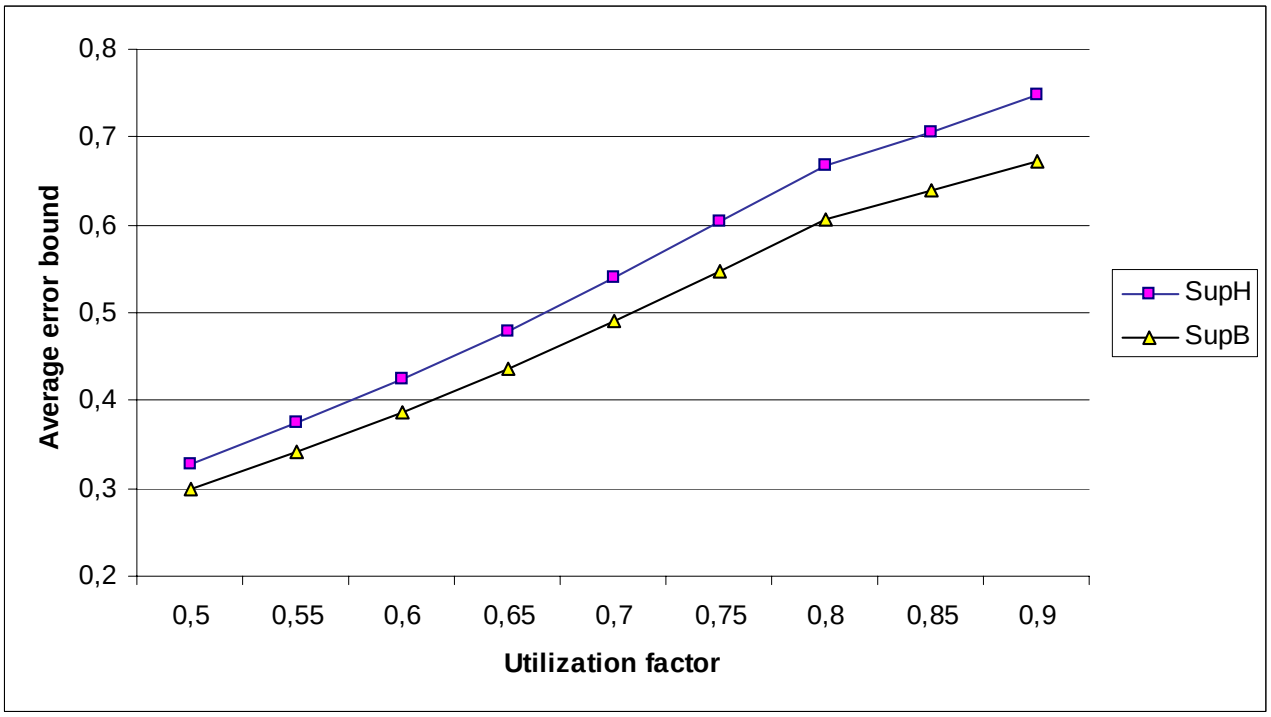

FIGURE 3.3 – *Comparaison de bornes linéaires SupB et SupH.*

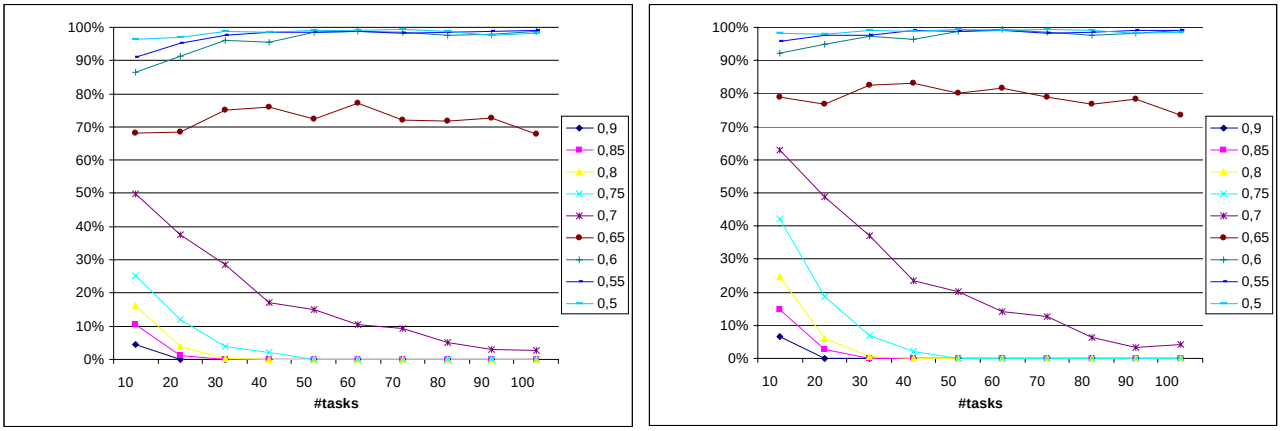

FIGURE 3.4 – *Ensembles faisables de tâches validés par les bornes linéaires : (a) SupB (Eq. (4.5)), (b) SupH (Eq. (3.1)).*

tel-00554265, version 1 - 10 Jan 2011

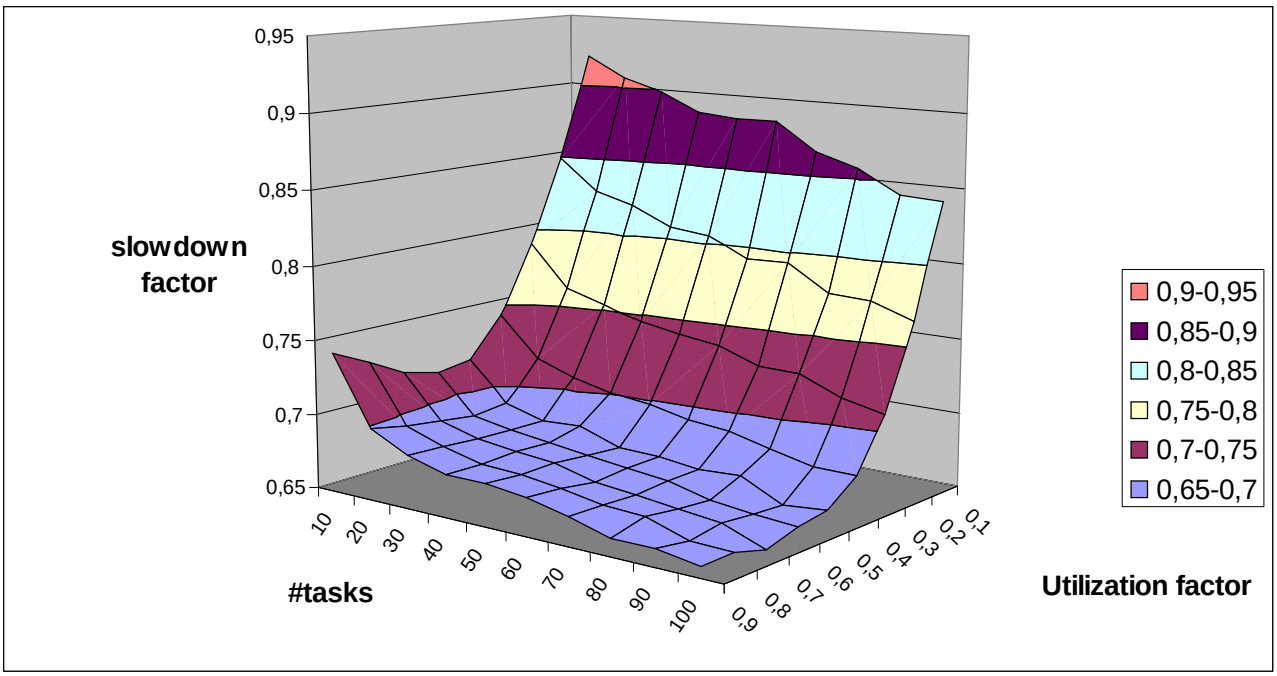

FIGURE 3.5 – *Facteur moyen de ralentissement s.*

5.3. Analyse d'augmentation de ressource

De façon analytique, nous avons déterminé le facteur de ralentissement s (slowdown factor) dans le pire cas de sorte que la borne calculée par SupB devienne le temps de réponse réel sur un processeur de vitesse s (c.f. ratio 1/2). Nous avons conduit une simulation afin de vérifier en pratique quelle est la valeur de ce ratio. Pour chaque ensemble de tâches, nous avons calculé (en utilisant une recherche dichotomique) le facteur exact de ralentissement s de sorte que SupB soit le temps de réponse réel sur un processeur de vitesse s. Nous avons surveillé le facteur de ralentissement moyen selon le nombre de tâches et le facteur d'utilisation. Des résultats de simulation sont présentés dans la Figure 3.5. Il peut être noté que le facteur de ralentissement est principalement lié au facteur d'utilisation plutôt qu'au nombre de tâches dans le système.

Comme prévu, les facteurs de ralentissement sont toujours entre $\frac{1}{2}$ et 1.

6. Conclusion

Nous avons étudié dans ce chapitre la borne supérieure du pire temps de réponse de Bini et Baruah [6], pour les tâches sporadique à échéance arbitraire. Nous avons proposé une preuve simple de cette borne dans le cas de tâches à échéance contrainte et analysé cette borne par déterminer les garanties de performance de celle-ci. Nous avons établi qu'aucun ratio d'approximation constant ne peut être défini dans [5] (l'inaproximabilité des temps de réponse pour les tâches à priorité fixe a été démontré en 2008 dans [13] sous l'hypothèse P $\neq$ NP) et montrer à l'aide de la technique d'augmentation de

tel-00554265, version 1 - 10 Jan 2011

ressource que la borne supérieure calculée est une borne inférieure du pire temps de réponse exact si le système est implémenté sur un processeur de capacité 1/2 [10].

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Chapitre 4

Tâches à échéances contraintes avec gigues d'activation

1. Introduction

Dans ce chapitre, nous continuons l'étude des systèmes de tâches sporadiques, à priorité fixe et échéances contraintes devant s'exécuter sur une plateforme mono-processeur. Dans ce contexte, nous rappelons que l'algorithme Deadline Monotonic (DM) est optimal pour fiablement ordonnancer un système de tâches. Dans le chapitre précédent, nous avons étudié des algorithmes de complexité linéaire pour déterminer les pires temps de réponse de tâches à priorité fixe. Dans la suite, nous allons utiliser des algorithmes d'approximation permettent de concevoir des tests efficaces d'ordonnançabilité (c'est-à-dire s'exécutant en temps polynomial) mais en introduisant une petite erreur dans le processus de décision de l'ordonnançabilité des tâches. Cette erreur est contrôlé à l'aide d'un paramètre de précisions ϵ (accuracy parameter). En d'autres termes, ils réalisent un compromis entre l'effort de calcul pour décider de l'ordonnançabilité d'un système de tâches et la qualité des décisions prises.

L'objectif de ce chapitre est de déterminer des bornes supérieures des pires temps de réponse pouvant être calculées de façon efficace et avec une erreur quantifiable vis-à-vis des temps de réponse exacts. Nous fournissons une nouvelle définition des fonctions de demande processeur (*Request Bound Function*) qui conduiront à un nouveau schéma d'approximation (FPTAS) pour analyser les systèmes de tâches à priorité fixe et échéances contraintes. Sur ce nouveau test approché d'analyse de la faisabilité d'un système de tâche, nous définissons deux nouvelles méthodes pour déduire des bornes supérieures des pires temps de réponse des tâches faisables. Notre FPTAS et nos nouvelles méthodes de déductions sont des améliorations des résultats présentés dans [16, 31, 32]. Puis, nous analysons les garanties de performances de nos schémas d'approximation en analysant le ratio classique d'approximation et en utilisant la technique d'augmentation de ressource. Enfin, nous présenterons des résultats d'expérimentations numériques pour comparer nos bornes à celles connues dans la littérature afin de déterminer la garantie de performances en moyenne. Le travail présenté dans la suite est le fruit d'une collaboration avec E. Bini [27, 28]

Le plan du chapitre est le suivant : le paragraphe 2 présente les résultats connus pour valider un

tel-00554265, version 1 - 10 Jan 2011

système de tâches à priorité fixe s'exécutant sur une plateforme monoprocesseur. Le paragraphe 3 présente une amélioration du tests approchés présenté dans [16] et deux nouvelles méthodes pour déduire du test des bornes supérieures des pires temps de réponse des tâches. Le paragraphe 4 présente des résultats sur les bornes d'erreurs des algorithmes d'approximation. Le paragraphe 5 présente les expérimentations numériques. Enfin, nous apportons des conclusions dans le paragraphe 6.

2. Définitions

2.1. Modèle de tâche

Nous rappelons les principales notations pour faciliter la lecture non linéaire du mémoire. Un système de tâches sporadiques τ_i, $1 \leq i \leq n$, est défini par les pires durée d'exécution (WCET) C_i, une échéance relative D_i et une période T_i qui mesure l'intervalle minimum entre l'arrivée de deux instances successives d'une tâche τ_i. Le facteur d'utilisation d'une tâche τ_i est la fraction de temps que τ_i demande le processeur : $U_i \stackrel{\text{def}}{=} C_i/T_i$. Le facteur d'utilisation d'un système de tâche : $U \stackrel{\text{def}}{=} \sum_{i=1}^{n} \frac{C_i}{T_i}$. Nous supposons que les échéances sont contraintes : $D_i \leq T_i$. Une telle hypothèse est réaliste pour de nombreuses applications réelles et conduit à des algorithmes de test d'ordonnançabilité plus simples [19]. La gigue sur activation J_i d'une tâche τ_i est le plus grand délai entre le réveil d'une tâche et son passage à l'état prêt par l'ordonnanceur de tâches. Nous supposons que $J_i < T_i$; nous supposons aussi que tous les paramètres des tâches sont des entiers (c-à-d., modèle de temps discret).

Nous supposons que toutes les tâches s'exécutent sur le même processeur, qu'elles sont indépendantes les unes des autres et ne suspendent pas elles-mêmes. Toutes les tâches ont des priorités fixes. A chaque instant la tâche dont la priorité est la plus grande parmi celles qui sont prêtes est choisie pour exécution. Sans perte de généralité, nous supposons que les tâches sont indexées dans l'ordre inverse de leur priorité : τ_1 est la tâche la plus prioritaire et τ_n est la moins prioritaire.

2.2. Résultats connus sur l'analyse des temps de réponse

2.2.a. Analyse exacte

Aucun algorithme polynomial en temps ne peut exister pour calculer les pires temps de réponse des tâches étudiées ([13]). Des algorithmes pseudo-polynomiaux sont connus et reposent sur la fonction de demande processeur d'une tâche τ_i à l'instant t (notée $\text{RBF}(\tau_i, t)$) et la demande processeur cumulée (notée $W_i(t)$) des tâches à la date t pour les tâches de priorité supérieure ou égale à i (voir [36] pour plus de détails) :

$$\text{RBF}(\tau_i, t) \stackrel{\text{def}}{=} \left\lceil \frac{t + J_i}{T_i} \right\rceil C_i \tag{4.1}$$

$$W_i(t) \stackrel{\text{def}}{=} C_i + \sum_{j=1}^{i-1} \text{RBF}(\tau_j, t) \tag{4.2}$$

Dans [20], une période d'activité de niveau i est définie comme l'intervalle de temps durant lequel seules des tâches de priorités supérieures ou égales à i s'exécutent.

Une approche courante pour vérifier l'ordonnançabilité des tâches à priorité fixe est de calculer le pire temps de réponse exacte R_i. Ce pire temps de réponse d'une tâche τ_i se définit formellement comme :

Définition 14 *Sous l'hypothèse que le processeur n'est pas surchargé (le facteur d'utilisation est strictement inférieur à 1), le pire temps de réponse d'une tâche τ_i peut être défini comme suit :*

$$R_i \stackrel{\text{def}}{=} \min\{t > 0 \mid W_i(t) = t\} + J_i$$

Des algorithmes exacts de calcul des pires temps de réponse sont connus dans la littérature. Ils utilisent des approximations successives en partant d'une borne inférieure R_i du temps de réponse, et calcule le pire temps de réponse (*wcrt*) comme le plus petit point fixe tel que $W_i(t) = t$.

Une autre approche de calcul des *wcrt* a été introduite dans in [31]. Nous présentons ci-après des détails sur cette méthode puisque les principes sous-jacents seront réutilisés dans la suite. Il a été démontré que le pire temps de réponse d'une tâche peut se calculer à l'aide d'une analyse de la demande processeur (voir [20] for details), pour chaque tâche respectant son échéance (et seulement pour celles-ci). Cette approche peut être étendue aux systèmes de tâches à échéances arbitraires comme nous le verrons dans le prochain chapitre. Elle peut aussi être facilement étendue aux tâches assujettis à des gigues sur activation comme le rappelle le résultat suivant :

$$S_i \stackrel{\text{def}}{=} \{aT_j - J_j \mid j = 1 \dots i-1, a = 1 \dots \lfloor \frac{D_i - J_i + J_j}{T_j} \rfloor\} \cup \{D_i - J_i\} \tag{4.3}$$

Dans [36], il a aussi été démontré que τ_i est ordonnançable si, et seulement si, $\exists t \in S_i, W_i(t) \leq t$. Il existe des méthodes réduisant le nombre de points d'ordonnancement à considérer dans l'analyse comme dans [25] et dans [7], mais ces deux méthodes engendrent une complexité exponentielle en $\mathcal{O}(2^n)$.

Fondé sur l'ensemble de points d'ordonnancement défini dans l'Eq. (4.3), il est défini dans [31] la notion de point *point d'ordonnancement critique* (sous l'hypothèse que la tâche τ_i est faisable) :

Définition 15 *Un point critique d'ordonnancement d'une tâche faisable τ_i est :*

$$t^* \stackrel{\text{def}}{=} \min\{t \in S_i \mid W_i(t) \leq t\}$$

Alors, nous définissons la durée cumulée de la demande processeur pour ce point d'ordonnancement critique de τ_i pour en déduire son pire temps de réponse.

Théorème 11 ([31]) *Le pire temps de réponse d'une tâche τ_i, telle que $W_i(t^*) \leq t^*$ (c-à-d., la tâche est faisable), est défini par $R_i = W_i(t^*) + J_i$.*

2.2.b. Analyse approchée des temps de réponse

Deux approches ont été définies dans la littérature pour calculer des bornes supérieures des pires temps réponse à l'aide d'algorithmes polynomiaux. Ces deux approches reposent sur l'approximation de la fonction RBF par des fonctions linéaires.

tel-00554265, version 1 - 10 Jan 2011

Approximation linéaire des pires temps de réponse. Nous avons vu dans le chapitre précédent la borne définie par Bini et Baruah [6]. Nous rappelons qu'ils ont montré pour les tâches sans gigue d'activation que la pire charge d'activité du processeur, qui est la durée maximum que le processeur exécute τ_i dans tout intervalle de longueur t, peut-être bornée par une fonction linéaire (voir [6] pour plus de détails) :

$$\text{LA}^{BB}(\tau_i, t) \stackrel{\text{def}}{=} U_i\, t + C_i(1 - U_i) \tag{4.4}$$

En utilisant cette fonction linéaire, [6] présente une borne supérieure du pire temps de réponse de τ_i :

$$R_i \leq \frac{C_i + \sum_{j=1}^{i-1} C_j(1 - U_j)}{1 - \sum_{j=1}^{i-1} U_j} = ub_i^{BB} \tag{4.5}$$

Nous avons montré dans le chapitre précédent que cette borne du pire temps de réponse définie dans l'Eq. (4.5) n'a pas de borne constante d'erreur par rapport à la valeur exacte du pire temps de réponse de τ_i (c-à-d., il existe des tâches telles que la borne supérieure est c fois supérieure à R_i, où c est un nombre positif arbitrairement grand). Ainsi, l'algorithme linéaire correspondant (c-à-d., de complexité $\mathcal{O}(n)$) n'est pas un algorithme d'approximation puisqu'aucun ratio constant ne peut être établi. Mais, nous avons montrer à l'aide de la technique d'augmentation de ressource que cette borne linéaire est une borne supérieure du pire temps de réponse sur un processeur de capacité unitaire et une borne inférieure du temps de réponse si un processeur de capacité $1/2$ est considéré. En d'autres termes, une accélération du processeur d'un facteur 2 est le prix à payer pour pouvoir utiliser cette borne linéaire dont le calcul s'effectue de façon très efficace (en temps linéaire).

Récemment Davis et Burns [33] ont étendu la formule précédente pour tenir compte des facteurs de blocage introduit par la protocole de gestion de ressources et la gigue sur activation. La fonction linéaire d'approximation devient alors une simple extension de l'Equation 4.5 :

$$\text{LA}_4(\tau_i, t) \stackrel{\text{def}}{=} (t + T_i + J_i - C_i)\frac{C_i}{T_i} \tag{4.6}$$

Schémas complets d'approximation. Les techniques d'approximation ont été récemment utilisées pour définir des tests approches d'ordonnançabilité. Ces tests s'exécutent en temps polynomial dans la taille du système de tâches et d'un paramètre de précision $1/\epsilon$ (c-à-d., ce sont des schémas d'approximation complets ou FPTAS). L'idée est d'exécuter une analyse exacte durant seulement les k premières étapes de l'algorithme (k est défini à partir d'ϵ) et d'utiliser ensuite une fonction linéaire pour approximer ensuite la fonction de demande processeur (Request Bound Function).

Dans la Table 4.1 est présentée une brève description des fonctions linéaires approchées qui ont été utilisées dans la littérature, ainsi que celles présentés dans la suite de ce chapitre.

En utilisant le schéma d'approximation présenté par Fisher and Baruah dans [15] pour établir un test booléen approché d'ordonnançabilité, nous avons montré qu'il était possible de déduire des bornes supérieures des pires temps de réponse des tâches dans [16, 31]. Dans le paragraphe suivant, nous présentons un schéma d'approximation nouveau qui améliore les meilleurs résultats connus (c-à-d. ceux présentés dans [15, 16]) pour l'analyse de la faisabilité et étendons ce schéma d'approximation

tel-00554265, version 1 - 10 Jan 2011

TABLE 4.1 – Fonctions d'approximation linéaires de la demande processeur

Paramètres entiers	Approximation linéaire	Auteurs et articles
Non	$\text{LA}_1(\tau_i, t) \stackrel{\text{def}}{=} (t + T_i)\frac{C_i}{T_i}$	Fisher and Baruah [14, 15]
Oui	$\text{LA}_2(\tau_i, t) \stackrel{\text{def}}{=} (t + T_i - 1)\frac{C_i}{T_i}$	Richard and Goossens [31]
Oui	$\text{LA}_3(\tau_i, t) \stackrel{\text{def}}{=} (t + T_i + J_i - 1)\frac{C_i}{T_i}$	Fisher, Nguyen, Goossens, Richard [16]
Non	$\text{LA}_4(\tau_i, t) \stackrel{\text{def}}{=} (t + T_i + J_i - C_i)\frac{C_i}{T_i}$	ce chapitre

pour les gigues sur activations. Enfin, nous présentons deux nouvelles méthodes permettant de déduire des bornes supérieures des temps de réponse qui améliorent les résultats présentés dans nos travaux antérieurs [16, 31, 32].

3. Bornes des pires temps de réponse

Dans ce paragraphe, nous introduisons un test approché d'ordonnançabilité et présentons comment déduire une borne du pire temps de réponse d'une tâche dont le test montre la faisabilité.

1. Depuis une extension de la fonction linéaire de Bini et Baruah (Eq. (4.4)), nous définissons un nouveau FPTAS pour analyser la faisabilité de systèmes de tâches. Si le FPTAS retourne un point d'ordonnancement critique (c.f., Définition 15), nous pouvons conclure que la tâche analysée est faisable. Si un tel point n'existe pas, alors la tâche τ_i est conclue non faisable sur processeur de vitesse $(1 - \epsilon)$ (où ϵ est le paramètre de précision en entrée du schéma d'approximation).
2. Nous proposons deux nouvelles "méthodes de déduction" qui dérive une borne supérieure du pire temps de réponse de τ_i et prend en entrée le point d'ordonnancement critique déterminé à l'étape (1) (sous l'hypothèse que cette étape en retourne un, sinon l'équation Eq. (4.5) peut être utilisée pour déterminer une borne supérieure du pire temps de réponse de τ_i).

3.1. Schéma d'approximation

La fonction RBF est une fonction discontinue avec une "marche" de hauteur C_i toutes les T_i unités de temps. Pour approximer cette fonction avec une marge d'erreur de $1 + \epsilon$ (paramètre de précision ϵ, $0 < \epsilon < 1$), nous utilisons les principes de l'algorithme présenté dans [14] : nous considérons les $(k-1)$ premières "marches" de la fonction $\text{RBF}(\tau_i, t)$, où k est défini comme $k \stackrel{\text{def}}{=} \lceil 1/\epsilon \rceil - 1$ et une fonction linéaire d'approximation ensuite. Depuis cette définition, nous vérifions que $(k + 1) \geq 1/\epsilon$.

tel-00554265, version 1 - 10 Jan 2011

Dans [16], et *sous l'hypothèse que tous les paramètres des tâches sont des entiers*, la fonction d'approximation de RBFa été définie ainsi :

$$\delta(\tau_i,t) \stackrel{\text{def}}{=} \begin{cases} \text{RBF}(\tau_i,t) & \text{for } t \leq (k-1)T_i - J_i, \\ \text{LA}_3(\tau_i,t) & \text{sinon.} \end{cases} \tag{4.7}$$

Ainsi, jusqu'à la date $(k-1)T_i - J_i$, aucune approximation n'est effectuée pour évaluer la demande processeur de τ_i, et ensuite elle est évaluée par une fonction linéaire dont la pente est égale au facteur d'utilisation de la tâche τ_i.

Nous proposons dans la suite une approximation linéaire fondée sur une extension de la fonction d'approximation $\text{LA}_4(\tau_i,t)$ qui définira une amélioration du test approché d'ordonnançabilité par rapport aux précédents résultats connus dans la littérature, d'une part, et n'impose pas d'avoir des paramètres entiers, d'autre part. Dans ce but, nous mettons en évidence des propriétés sur les points d'ordonnancement critiques qui permettront de limiter la recherche aux points pour lesquels la fonction linéaire est plus grande (ou égale) à la fonction RBF.

Tout d'abord, nous prouvons la propriété suivante, qui est valide pour toutes les périodes d'activité des tâches :

Lemme 5 *Une période d'activité de niveau-i ne peut se terminer dans tout intervalle* $(mT_j - J_j, mT_j + C_j - J_j), 1 \leq j \leq i$, *où* m *est un entier arbitraire.*

Preuve : Nous provons le résultat par contradiction. Supposons que la période d'activité est terminée à la date $t \in (mT_j - J_j, mT_j + C_j - J_j)$, alors à la date $mT_j - J_j$, cette période d'activité n'est pas encore terminée. De plus, à cette date, une instance de τ_j est réveillée. Ainsi, la période d'activité doit inclure l'exécution de cette nouvelle instance. Puisque cette instance ne peut pas se terminer avant la date $mT_j + C_j - J_j$, et ceci même si aucune préemption ne survient, alors la période d'activité ne peut pas être terminée avant la date $mT_j + C_j - J_j$, ce qui contredit l'hypothèse initiale. ■

Le corollaire suivant d'obtient directement du Lemme 5 :

Corollaire 1 *Si* t *est la longueur de la période d'activité synchrone de niveau-i, alors* $\forall j, 1 \leq j \leq i, \exists m \in N, t \in [mT_j + C_j - J_j, (m+1)T_j - J_j]$.

A partir du corollaire précédent, nous obtenons la propriété suivante sur les points d'ordonnancement critiques :

Corollaire 2 *Si* t^* *est un point critique de la tâche* τ_i *alors* $\forall j, 1 \leq j \leq i, \exists m \in N, t^* \in [mT_j + C_j - J_j, (m+1)T_j - J_j]$.

Preuve : Nous prouvons ce résultat par contradiction. Si $\exists j, 1 \leq j \leq i, \nexists m \in N, t^* \in [mT_j + C_j - J_j, (m+1)T_j - J_j] \Rightarrow \exists h \in N, t^* \in (hT_j - J_j, hT_j + C_j - J_j)$. Soit t^p le point d'ordonnancement qui est juste avant t^* dans S_i et t^{bp} l'instant auquel la période d'activité synchrone de niveau -i se termine. Dans [31], nous avons montré que $t^{bp} \in (t^p, t^*]$. Puisque t^p et t^* sont deux points adjacents dans S_i, nous vérifions nécessairement $t^p \geq hT_j - J_j$. En conséquence, $t^{bp} \in (hT_j - J_j, hT_j + C_j - J_j)$, ce qui contredit le Corollaire 1. ■

Maintenant, nous pouvons réduire l'ensemble des points d'ordonnancement défini dans l'Eq. (4.3) (l'ensemble des points d'ordonnancement dans lequel le point critique est recherché) de la façon suivante :

tel-00554265, version 1 - 10 Jan 2011

Corollaire 3 *Pour une tâche faisable τ_i, il est suffisant de vérifier les points d'ordonnancement de l'ensemble suivant :*

$$\begin{aligned} S_i^{(0)} &\stackrel{\text{def}}{=} \{aT_j - J_j \mid j = 1 \dots i-1, a = 1 \dots \lfloor \frac{D_i - J_i + J_j}{T_j} \rfloor\} \cup \{D_i - J_i\}, \\ S_i &\stackrel{\text{def}}{=} S_i^{(0)} \backslash \{t \in S_i^{(0)} \mid t \in (aT_j - J_j, aT_j + C_j - J_j), j = 1, \dots, i,\ a \geq 0\}. \end{aligned} \tag{4.8}$$

τ_i est faisable si, et seulement si, $\exists t \in S_i, W_{i,l}(t) \leq t$.

Considérons maintenant la fonction linéaire de Bini et Baruah définie dans Eq. (4.4) :

$$\text{LA}^{BB}(\tau_i, t) = \text{LA}_3(\tau_i, t) \stackrel{\text{def}}{=} (t + T_i - C_i)\frac{C_i}{T_i}.$$

Nous étendons cette fonction avec les gigues sur activitions :

$$\text{LA}_4(\tau_i, t) \stackrel{\text{def}}{=} (t + T_i + J_i - C_i)\frac{C_i}{T_i}. \tag{4.9}$$

Nous montrons que cette fonction est aussi une borne supérieure de la demande processeur sous l'hypohtèse d'une certaine condition sur la date t.

Lemme 6 *$\forall \tau_j, \forall t \in [mT_j + C_j - J_j; (m+1)T_j - J_j]$, où m est un entier positif arbitraire, nous avons :*

$$\text{RBF}(\tau_j, t) = \left\lceil \frac{t + J_j}{T_j} \right\rceil C_j \leq \text{LA}_4(\tau_j, t).$$

Preuve : Pour $t \in [mT_j + C_j - J_j, (m+1)T_j - J_j]$, nous avons :

$$\begin{array}{ccc} mT_j + C_j - J_j \leq t & & \leq (m+1)T_j - J_j \\ \left\lceil \frac{mT_j + C_j - J_j + J_j}{T_j} \right\rceil \leq \left\lceil \frac{t + J_j}{T_j} \right\rceil & & \leq \left\lceil \frac{(m+1)T_j - J_j + J_j}{T_j} \right\rceil \end{array}$$

Comme $C_j \leq T_j$, nous avons $m+1 \leq \left\lceil \frac{t+J_j}{T_j} \right\rceil \leq m+1 \Rightarrow \left\lceil \frac{t+J_j}{T_j} \right\rceil = m+1 \Rightarrow \text{RBF}(\tau_j, t) = (m+1)C_j$. De plus,

$$\begin{array}{c} t \geq mT_j + C_j - J_j \\ \frac{t + T_j + J_j - C_j}{T_j} C_j \geq \frac{mT_j + C_j - J_j + T_j + J_j - C_j}{T_j} C_j \\ \text{LA}_4(\tau_j, t) \geq (m+1)C_j = \text{RBF}(\tau_j, t). \end{array}$$

Le lemme est prouvé. ■

A partir de ce lemme et du Corollaire 3, nous obtenons que $\text{LA}_4(\tau_i, t)$ peut être plus petite que la fonction $\text{RBF}(\tau_i, t)$ pour certains instances mais pas pour celles qui comptent pour vérifier l'ordonnançabilité de τ_i (les points d'ordonnancement n'appartiennent pas à ces intervalles).

En conséquence, il est possible de définir une RBF améliorée :

tel-00554265, version 1 - 10 Jan 2011

TABLE 4.2 – Ensemble de tâches à priorité fixe

Tâches	C_i	D_i	T_i	J_i
τ_1	2	4	4	0
τ_2	3	8	8	0

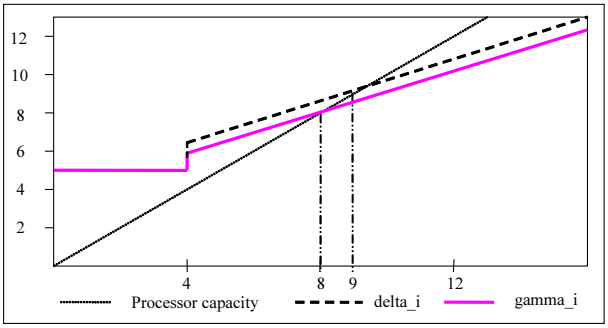

FIGURE 4.1 – Approximation de la fonctione RBF pour le système de tâches présenté Table 4.2

$$\gamma(\tau_i, t) \stackrel{\text{def}}{=} \begin{cases} \text{RBF}(\tau_i, t) & \text{for } t \leq (k-1)T_i - J_i, \\ \text{LA}_4(\tau_i, t) & \text{sinon.} \end{cases} \tag{4.10}$$

Depuis les définitions de $\gamma(\tau_i, t)$ et de $\delta(\tau_i, t)$, il est simple de vérifier que $\gamma(\tau_i, t)$ peut seulement améliorer la fonction approchée $\delta(\tau_i, t)$, et ainsi les résultats du FPTAS présenté dans [16].

Théorème 12 *Sous l'hypothèse que les paramètres des tâches sont des entiers (ainsi,* $\forall i, 1 \leq i \leq n, C_i \geq 1$*),* $\gamma(\tau_i, t)$ *définit une borne supérieure plus serrée de la* $\text{RBF}(\tau_i, t)$ *en comparaison avec* $\delta(\tau_i, t)$ *:*

$$\forall t > 0, \delta(\tau_i, t) \geq \gamma(\tau_i, t).$$

Nous allons montrer qu'un FPTAS peut être basé sur la fonction $\gamma(\tau_i, t)$. Notons aussi que $\delta(\tau_i, t)$ ne peut être utilisé que pour des systèmes de tâches dont les paramètres sont des entiers, alors que $\gamma(\tau_i, t)$ peut être utilisé même lorsque les paramètres sont des nombres réels.

Pour définir le test approché de faisabilité selon les principes de l'analyse de la demande processeur (Processor Demand Analysis [20]), nous définissons la fonction approchée de la demande processeur cumulée comme :

$$\widehat{W_i}(t) \stackrel{\text{def}}{=} C_i + \sum_{j=1}^{i-1} \gamma(\tau_j, t).$$

En utilisant le Corollaire 3, et en considérant la borne de précision ϵ conduisant à $k = \lceil 1/\epsilon \rceil - 1$, et en appliquant la technique d'approximation présentée dans [15], nous définissons l'ensemble des points

tel-00554265, version 1 - 10 Jan 2011

d'ordonnancement $\widehat{S}_i \subseteq S_i$:

$$\begin{aligned}
\widehat{S_i^{(0)}} &\stackrel{\text{def}}{=} \{bT_a - J_a \mid a = 1, \ldots, i-1, b = 1, \ldots, k-1\} \\
&\quad \cup \{D_i - J_i\}, \\
\widehat{S}_i &\stackrel{\text{def}}{=} \widehat{S_i^{(0)}} \backslash \{t \in \widehat{S_i^{(0)}} \mid t \in (aT_j - J_j, aT_j + C_j - J_j), \\
&\quad j = 1, \ldots, i,\ a \geq 0\}.
\end{aligned} \tag{4.11}$$

Nous établissons maintenant les principes de l'algorithme :
- S'il existe un instant $t \in \widehat{S}_i$ tel que $\widehat{W}_i(t) \leq t$, alors τ_i est faisable (sur un processeur de vitesse 1),
- sinon, τ_i n'est pas faisable sur un processeur de vitesse $(1 - \epsilon)$.

Une implémentation simple de cette méthode conduit à un algorithme d'ordonnançabilité en $\mathcal{O}(n^2/\epsilon)$. C'est un schéma d'approximation complet (FPTAS) puisque l'algorithme est polynomial en la taille de l'entrée n et le paramètre de précision en entrée du schéma $1/\epsilon$.

Nous présentons dans la Table 4.2 un ensemble de tâches dans lequel la tâche τ_2 n'est pas montrée faisable en utilisant la fonction $\text{LA}_3(\tau_i, t)$ de [16] (qui utilise la fonction $\delta(\tau_i, t)$ pour approximer $\text{RBF}(\tau_i, t)$) pour la valeur $k = 2$. En utilisant la nouvelle fonction approchée $\text{LA}_4(\tau_i, t)$ (c'est-à-dire l'approximation de la RBF par la fonction approchée $\gamma(\tau_2, t)$), τ_2 est maintenant prouvée ordonnançable puisque $\widehat{W}_2(8) = 8$ (voir Figure 4.1 qui présente ces deux fonctions pour la tâche τ_2).

3.2. Correction du schéma d'approximation

Nous prouvons dans ce paragraphe la correction du test d'ordonnançabilité présenté dans le paragraphe précedent. Le point clé pour démontrer la correction de ce schéma d'approximation est : $1 \leq \gamma(\tau_i, t)/\text{RBF}(\tau_i, t) \leq (1 + \epsilon)$, qui montre que la déviation de l'approximation par rapport à la valeur exacte de la fonction RBF est bornée. Ce résultat est utilisé ensuite pour prouver que si une tâche est montrée non ordonnançable par le test approchée alors elle ne sera pas ordonnançable sur un processeur de vitesse $(1 - \epsilon)$.

Le premier résultat montre que notre approximation est une borne supérieure de la fonction RBF pour tout point d'ordonnancement.

Théorème 13 $\forall j, 1 \leq j \leq i-1, \forall t \in S_i,\ \gamma(\tau_j, t) \geq \text{RBF}(\tau_j, t)$.

Preuve : Trivial depuis le Lemme 6 et la définition de S_i (Eq. (4.8)). ■

Le second théorème donne le ratio d'approximation :

Théorème 14 $\forall j, 1 \leq j \leq i-1, \forall t \in (0, D_i - J_i],\ \gamma(\tau_j, t) \leq (1 + \frac{1}{k})\text{RBF}(\tau_j, t)$.

Preuve : D'après l'Eq. (4.9) :

$$\text{LA}_4(\tau_j, t) \stackrel{\text{def}}{=} (t + T_j + J_i - C_j)\frac{C_j}{T_j}$$

Pour $t \leq (k-1)T_j - J_j$, il est évident que les inégalités tiennent puisque $\gamma(\tau_j, t) = \text{RBF}(\tau_j, t)$. Nous considérons maintenant le cas $t > (k-1)T_j - J_j$, où $\gamma(\tau_j, t) = \text{LA}_4(\tau_j, t)$.

tel-00554265, version 1 - 10 Jan 2011

Puisque $\text{RBF}(\tau_j, t)$ est une fonction en escalier et $\text{LA}_4(\tau_j, t)$ est une fonction linéaire strictement croissante, il est clair que le ratio $\text{LA}_4(\tau_j, t)$ / $\text{RBF}(\tau_j, t)$ est une fonction monotone croissante dans tout intervalle $(mT_j - J_j, (m+1)T_j - J_j], m \in N, m \geq (k-1)$, c'est-à-dire, $\text{LA}_4(\tau_j, t)$ / $\text{RBF}(\tau_j, t)$ est strictement croissante hormis à l'instant t qui est un multiple de la période T_j où ce ratio atteint un maximum local, et la fonction est continue à gauche. Puisque les maxima locaux de $\text{LA}_4(\tau_j, t)$ / $\text{RBF}(\tau_j, t)$ sont atteints aux instants $(m+1)T_j - J_j\ , m \geq (k-1)$, alors trouver son maximum global pour $t > (k-1)T_j - J_j$, nous avons juste à considérer les instants $t = hT_j - J_j$ avec $h \in N, h \geq k$.

Nous avons finalement :

$$\begin{aligned}
\frac{\text{LA}_4(\tau_j, hT_j - J_j)}{\text{RBF}(\tau_j, hT_j - J_j)} &= \frac{hT_j - J_j + T_j + J_j - C_j}{hT_j} \\
&= 1 + \frac{1}{h} - \frac{C_j}{hT_j} \\
&\leq 1 + \frac{1}{h} \leq 1 + \frac{1}{k}.
\end{aligned}$$

Le théorème est prouvé. ■

En utilisant une approche similaire à celle présentée dans [15], nous pouvons maintenant établir la correction de l'approximation.

Tout d'abord, nous prouvons que si la tâche τ_i est prouvée non ordonnançable par le test approché, alors elle est non ordonnançable avec certitude sur un processeur de capacité $(1 - \epsilon)$.

Théorème 15 *Si $\widehat{W}_i(t) > t$ pour tout $t \in (0, D_i - J_i]$, alors τ_i est non ordonnançable sur un processeur de capacité $(1 - \epsilon)$.*

Preuve : Nous allons prouver que si $\forall t \in (0, D_i - J_i]$, $\widehat{W}_i(t) > t$, alors $\forall t \in (0, D_i - J_i]$, $W_{i,l}(t) > (1-\epsilon)t$:

$$\begin{aligned}
\widehat{W}_{i,l}(t) &> t \\
C_i + \sum_{i-1} \gamma(\tau_j, t) &> t.
\end{aligned}$$

A partir du Théorème 14, pour tout instant $t \in (0, D_i - J_i]$, nous avons :

$$\begin{aligned}
C_i + \sum_{i-1} \frac{k+1}{k} \text{RBF}(\tau_j, t) &> t \\
\frac{k+1}{k} \left(C_i + \sum_{i-1} \text{RBF}(\tau_j, t) \right) &> t \\
\frac{k+1}{k} W_{i,l}(t) &> t \\
W_{i,l}(t) &> \frac{k}{k+1} t.
\end{aligned}$$

Nous avons

$$\frac{k}{k+1} = 1 - \frac{1}{k+1}.$$

tel-00554265, version 1 - 10 Jan 2011

TABLE 4.3 – Inégalités entre les points d'ordonnancement exacts, approchés critiques et intersections

Inégalités	Présentés dans
$t_i^{int} \leq t_i^*$	[31]
$\hat{t}_i^{int} \leq \hat{t}_i^*$	Ce chapitre
$t_i^* \leq \hat{t}_i^*$	[16]
$t_i^{int} \leq \hat{t}_i^{int}$	Ce chapitre

Notons que $k \stackrel{\text{def}}{=} \lceil \frac{1}{\epsilon} \rceil - 1$ et $\lceil \frac{1}{\epsilon} \rceil \geq \frac{1}{\epsilon}$, nous obtenons

$$\frac{k}{k+1} \geq 1 - \epsilon.$$

Ainsi pour tout $t \in (0, D_i - J_i]$,

$$W_{i,l}(t) > (1 - \epsilon)\, t.$$

Le théorème est prouvé. ■

Maintenant, si le test approché conclut que la tâche τ_i est faisable, alors elle est faisable sur un processeur de capacité unitaire.

Théorème 16 *S'il existe un instant $t \in \widehat{S}_i$ tel que $\widehat{W}_i(t) \leq t$, alors $W_i(t) \leq t$.*

Preuve : Puisque $t \in \widehat{S}_i$, alors $t \in S_i$. De plus, le Théorème 13 autorise de conclure que $\forall t \in S_i, W_i(t) \leq \widehat{W}_i(t)$. Ainsi, $W_i(t) \leq t$ et τ_i est ordonnançable. ■

Pour terminer la preuve de correction du test, nous devons prouver que l'ensemble des points d'ordonnancement est suffisant.

Théorème 17 *Si $\forall t \in \widehat{S}_i, \widehat{W}_i(t) > t$, alors nous vérifions aussi que $\forall t \in (0, D_i - J_i], \widehat{W}_i(t) > t$.*

Preuve : (Sketch) Soient t_1 et t_2 deux points d'ordonnancement *adjacents* dans $\widehat{S}_i$ (i.e., $\nexists\, t \in \widehat{S}_i$ tel que $t_1 < t < t_2$). Puisque $\widehat{W}_i(t_1) > t_1$, $\widehat{W}_i(t_2) > t_2$ et $\widehat{W}_i(t)$ est une fonction en escalier non décroissante et continue à gauche, nous concluons que $\forall t \in (t_1, t_2), \widehat{W}_i(t) > t$. Le théorème suit. ■

Théorème 18 *S'il existe un instant $t \in \widehat{S}_i$ tel que $\widehat{W}_i(t) \leq t$, alors il existe un instant $t' \in (0, D_i - J_i]$ tel que $W_i(t') \leq t'$.*

Preuve : Depuis le Théorème 18, s'il existe un instant $t \in (0, D_i - J_i]$ tel que $\widehat{W}_i(t) \leq t$, alors il existe un instant $t' \in \widehat{S}_i$ tel que $\widehat{W}_i(t') \leq t'$. De plus, Théorème 13 implique que $\forall t' \in \widehat{S}_i, \widehat{W}_i(t') \geq W_i(t')$. ■

3.3. Pires temps de réponse approchés (méthodes de déduction)

Dans le paragraphe précédent, nous avons proposé un test d'ordonnançabilité que vérifie qu'une tâche est faisable sur un processeur de capacité 1 ou non faisable sur un processeur de capacité $(1 - \epsilon)$. Si une tâche est conclue faisable par ce test alors il existe un point d'ordonnancement qui vérifie la

tel-00554265, version 1 - 10 Jan 2011

TABLE 4.4 – Méthodes de déduction des bornes des temps de réponse des tâches depuis les tests approchés (FPTAS)

Méthodes de déduction	Auteurs et articles
$\hat{R}_i \stackrel{\text{def}}{=} \widehat{W}_i(\hat{t}^*) + J_i$	Fisher, Nguyen, Goossens, Richard [16]
$\hat{R}_i^w \stackrel{\text{def}}{=} W_i(\hat{t}^*) + J_i$	Ce chapitre
$\hat{R}_i^{wInt} \stackrel{\text{def}}{=} W_i(\hat{t}^{int}) + J_i$	Ce chapitre

condition d'ordonnançabilité. La méthode permettant de déduire une borne approchée du pire temps de réponse d'une tâche depuis un tel point est appellée dans la suite une "méthode de déduction". Si le test approché ne fournit pas une réponse positive pour une tâche, alors notre approche ne permet pas de déduire une borne approchée du pire temps de réponse de cette tâche (toutefois, une telle borne peut être calculée par la formule 4.5 (initialement présentée dans [6]), mais sans garantie de performance reliée au paramètre ϵ). Notez que les méthodes de déduction sont en fait totalement indépendantes des fonctions d'approximation linéaires utilisées dans les tests approchés d'ordonnançabilité (c.f. les fonctions présentées dans la Table 4.1) et définissent une approche orthogonale pour améliorer la qualité des bornes supérieures des pires temps de réponse des tâches.

Tout d'abord, nous introduisons les notions sur lesquelles reposent les méthodes de déduction des bornes des pires temps de réponse. Comme nous l'avons présenté dans le paragraphe 2.2.a, l'analyse exacte d'ordonnançabilité d'une tâche faisable τ_i permet de déterminer le point d'ordonnancement critique t^*, comme le premier point d'ordonnancement $t \in S_i$ tel que $W_i(t) \leq t$. Soit t^{int} un point d'ordonnancement $t \leq t^*$ tel que $W_i(t) = t$. Dans [31] est prouvé qu'il existe seulement un instant t^{int}, qui correspond à la première intersection entre la fonction de demande cumulée et la fonction affine représentant la capacité du processeur. En conséquence, nous avons forcément $W_i(t^{int})+J_i = W_i(t^*)+J_i$ puisque ces deux valeurs sont égales au pire temps de réponse exact R_i de la tâche τ_i.

Nous utilisons les mêmes principes pour définir les méthodes de déduction applicables pour *toutes* les analyses approchées connues pour les tâches à priorité fixe (celles présentées dans [15,16] et dans ce chapitre). Soient $\widehat{W}_i(t)$ et $\widehat{S}_i$ respectivement la fonction de demande processeur approchée et l'ensemble des points d'ordonnancement utilisés dans l'analyse approchée d'une tâche τ_i donnée. Soit $\hat{t}^*$ le premier point d'ordonnancement $t \in \widehat{S}_i$ tel que le test approché conclut que τ_i est faisable, c'est-à-dire, $\widehat{W}_i(\hat{t}^*) \leq \hat{t}^*$. Si un tel point d'ordonnancement existe nous pouvons alors définir $\hat{t}^{int} \leq \hat{t}^*$ tel que $\widehat{W}_i(\hat{t}^{int}) = \hat{t}^{int}$. Fondé sur $\hat{t}^*$, $\hat{t}^{int}$, et les points d'ordonnancement exacts correspondants (c-à-d., t^* and t^{int}), nous proposons deux nouvelles méthodes de déduction, qui conduiront à des bornes supérieures des pires temps de réponse. Ces inégalités sont présentées dans la Table 4.3 et illustrées dans la Figure 4.3. Enfin, un résumé des méthodes existantes et celles proposées dans ce chapitre est présenté dans la Table 4.4.

Dans [16] est présentée une méthode de déduction qui conduit à la borne supérieure suivante du temps de réponse d'une tâche :

tel-00554265, version 1 - 10 Jan 2011

Définition 16 ([16]) *Soit une tâche τ_i telle qu'il existe un instant $t \in \widehat{S}_i$ vérifiant $\widehat{W}_i(t) \leq t$, alors une borne supérieure approchée de son pire temps de réponse est définie par :*

$$\begin{aligned} \hat{t}^* &\stackrel{\text{def}}{=} \min\left\{t \in \widehat{S}_i \middle| \widehat{W}_i(t) \leq t\right\}, \\ \hat{R}_i &\stackrel{\text{def}}{=} \widehat{W}_i(\hat{t}^*) + J_i. \end{aligned} \tag{4.12}$$

Nous proposons maintenant une nouvelle méthode qui conduit à une nouvelle borne supérieure $\hat{R}_i^w$ du temps de réponse de τ_i. Notons que dans cette définition, le point d'ordonnancement critique approché est défini à partir de la fonction de demande cumulée *approchée* (the approximate Workload function), mais la borne supérieure du pire temps réponse est quant à elle définie en utilisant la demande cumulée *exacte* dans le but d'améliorer la qualité de la borne.

Définition 17 *Soit une tâche τ_i telle qu'il existe un instant $t \in \widehat{S}_i$ vérifiant $\widehat{W}_i(t) \leq t$, alors une borne supérieure approchée de son pire temps de réponse est définie par :*

$$\begin{aligned} \hat{t}^* &\stackrel{\text{def}}{=} \min\left\{t \in \widehat{S}_i \middle| \widehat{W}_i(t) \leq t\right\}, \\ \hat{R}_i^w &\stackrel{\text{def}}{=} W_i(\hat{t}^*) + J_i. \end{aligned} \tag{4.13}$$

Nous allons prouvé que cette méthode définie une borne plus serrée du pire temps de réponse en comparaison avec la borne $\hat{R}_i$ qui avait été obtenue avec la précédente méthode de déduction.

Théorème 19 *Pour toute tâche τ_i telle qu'il existe un instant $t \in \widehat{S}_i$ vérifiant $\widehat{W}_i(t) \leq t$, nous avons*

$$R_i \leq \hat{R}_i^w \leq \hat{R}_i.$$

Proof: Soit t^* un point critique correspondant au pire temps de réponse de τ_i (c-à-d., le premier instant dans S_i tel que $W_i(t) \leq t$). Soit $\hat{t}^*$ le premier instant dans $\widehat{S}_i$ tel que $\widehat{W}_i(t) \leq t$ (c-à-d., $\hat{t}^*$ est un point critique). Puisque $\widehat{S}_i \subseteq S_i$, nous avons $\hat{t}^*$ qui est aussi le premier instant dans S_i vérifiant $\widehat{W}_i(t) \leq t$.

Puisque le Théorème 18 montre que $\forall t \in S_i, \widehat{W}_i(t) \geq W_i(t)$, nous avons nécessairement $t^* \leq \hat{t}^*$. Pusique $W_i(t)$ est une fonction non-décroissante, $W_i(t^*) \leq W_i(\hat{t}^*)$. De façon équivalente, $R_i \leq \hat{R}_i^w$.

Finallement, $\forall t \in S_i, \widehat{W}_i(t) \geq W_i(t)$, alors $W_i(\hat{t}^*) \leq \widehat{W}_i(\hat{t}^*)$, nous avons ainsi obtenu le membre droit de l'inégalité. ■

Nous proposons dans la suite une autre méthode de déduction $\hat{R}_i^{wInt}$ qui améliore la précédente. Nous conservons toujours l'hypothèse que la tâche τ_i est classée ordonnançable par le test approché, il doit alors exiter un point d'intersection entre la fonction de demande cumulée du processeur par les tâches et la droite affine représentant la capacité du processeur comme illustré Figure 4.2. Ce point d'intersection permet de définir une borne supérieure du pire temps de réponse d'une tâche de la façon suivante :

Définition 18 *Soit une tâche τ_i telle qu'il existe un instant $t \in \widehat{S}_i$ vérifiant $\widehat{W}_i(t) \leq t$, alors une borne supérieure approchée de son pire temps de réponse est donnée par :*

$$\begin{aligned} \hat{t}^{int} &\stackrel{\text{def}}{=} \min\left\{t \geq 0 \middle| \widehat{W}_i(t) = t\right\}, \\ \hat{R}_i^{wInt} &\stackrel{\text{def}}{=} W_i(\hat{t}^{int}) + J_i \end{aligned} \tag{4.14}$$

tel-00554265, version 1 - 10 Jan 2011

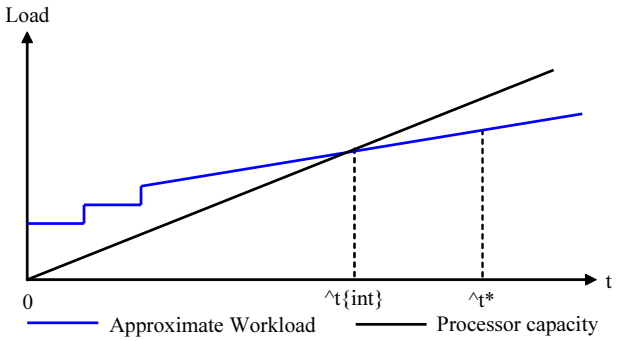

FIGURE 4.2 – Point d'intersection approchée $\hat{t}^{int}$ vs. point critique approché $\hat{t}^*$

Nous montrons tout d'abord que cette nouvelle méthode de déduction $\hat{R}_i^{wInt}$ définit une borne supérieure de temps de réponse qui est valide, puis nous comparons cette nouvelle méthode de déduction avec les deux premières présentées au début de ce paragraphe.

Théorème 20 *$\hat{R}_i^{wInt}$ est une borne supérieure du pire temps de réponse de la tâche τ_i :*

$$\hat{R}_i^{wInt} \geq R_i \tag{4.15}$$

Preuve : Rappelons que $\hat{R}_i^{wInt} \stackrel{\text{def}}{=} W_i(\hat{t}^{int}) + J_i$ et $R_i = W_i(t^{int}) + J_i$. Puisque $W_i(t)$ est une fonction non-décroissante, l'inégalité définie dans l'Eq. (4.15) tient si, et seulement si, nous avons $\hat{t}^{int} \geq t^{int}$ (obligation de preuve).

Dans ce but, nous définissons une condition suffisante pour qu'un instant t soit plus petit que $\hat{t}^{int}$, puis nous montrerons que t^{int} respecte bien cette condition.

Lemme 7 *Soit t un instant arbitraire tel que $t \in [0, \hat{t^*}]$, alors nous avons :*

$$\widehat{W}_i(t) \geq t \Rightarrow t \leq \hat{t}^{int}$$

Preuve : Soit $\hat{t}^{pred}$ le point d'ordonnancement juste avant le point d'ordonnancement critique approché $\hat{t}^*$ dans $\hat{S}_i$. Nous prouvons le lemme sur deux intervalles : $[0, \hat{t}^{pred}]$ and $(\hat{t}^{pred}, \hat{t^*}]$.

a) Si $t \in [0, \hat{t}^{pred}]$:

A partir de la définition d'un point d'ordonnancement critique, nous avons $\forall t \in \hat{S}_i, t \leq \hat{t}^{pred}, \hat{W}_i(t) > t$ (1).

Soient $t_1, t_2 \leq \hat{t}^{pred}$ deux points *adjacents* dans $\hat{S}_i$ (c-à-d., $\nexists\, t \in \hat{S}_i$ tels que $t_1 < t < t_2$). D'après (1) $\Rightarrow \widehat{W}_i(t_1) > t_1$, $\widehat{W}_i(t_2) > t_2$. Puisque $\widehat{W}_i(t)$ est une fonction en escalier non-décroissante et continue à gauche $\Rightarrow \forall t \in (t_1, t_2)\ \hat{W}_i(t) > t$. En conséquence, $\forall t \leq \hat{t}^{pred}, \widehat{W}_i(t) > t \Rightarrow \hat{t}^{int} \notin [0, \hat{t}^{pred}] \Rightarrow \forall t \in [0, \hat{t}^{pred}], t < \hat{t}^{int}$

b) Si $t \in (\hat{t}^{pred}, \hat{t^*}]$:

Par le fait que $\hat{W}_i(\hat{t}^{pred}) > t$, $\hat{W}_i(\hat{t^*}) \leq t$, et $\hat{W}_i(t)$ est une fonction en escalier non-décroissante et continue à gauche : il existe un, et un seul, instant $t \in (\hat{t}^{pred}, \hat{t^*}]$ tel que $\hat{W}_i(t) = t$, que nous nommerons

tel-00554265, version 1 - 10 Jan 2011

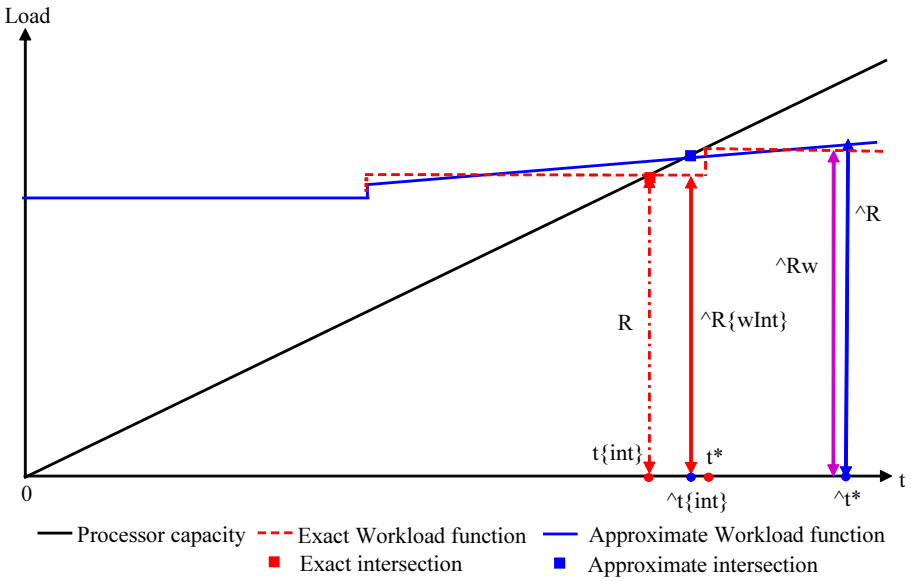

FIGURE 4.3 – Temps de réponse exact et borne supérieure du temps de réponse de la tâche τ_2 de l'ensemble de tâches défini dans la Table 4.5

$\hat{t}^{int}$. De plus, puisque $\hat{W}_i(t)$ est une fonction linéaire par morceau dont la pente est strictement inférieure à 1, nous obtenons $\forall t \in (\hat{t}^{pred}, \hat{t}^{int}), \hat{W}_i(t) > t$ et $\forall t \in (\hat{t}^{int}, \hat{t^*}], \hat{W}_i(t) < t$.

En considérant les cas a) et b), le lemme est prouvé. ■

Nous pouvons maintenant établir l'inégalité $t^{int} \leq \hat{t}^{int}$ de la façon suivante :

Considérons t^{int}, l'instant correspondant à l'intersection exacte entre la demande et la capacité du processeur, ou de façon équivalente, l'instant correspondant à la longueur de la période d'activité synchrone de niveau-i. En utilisant la propriété sur la longueur de la période d'activité synchrone de niveau-i dans le Corollaire 1, puis le Lemme 6, alors nous avons $\hat{W}_i(t^{int}) \geq W_i(t^{int})$. Puisque $W_i(t^{int}) = t^{int} \Rightarrow \hat{W}_i(t^{int}) \geq t^{int}$.

De plus, puisque $t^{int} \leq t^*$ (depuis [31]) et $t^* \leq \hat{t}^*$ (depuis [16])$\Rightarrow t^{int} \in [0, \hat{t^*}] \Rightarrow$ nous pouvons directement appliquer le Lemme 7, nous avons donc $t^{int} \leq \hat{t}^{int}$ et le théorème est prouvé. ■

Nous prouvons que cette méthode de déduction conduit à une borne plus serrée du temps de réponse de la tâche τ_i en comparaison avec la borne $\hat{R}_i^w$.

Théorème 21 *Pour chaque tâche τ_i telle qu'il existe un instant $t \in \hat{S}_i$ vérifiant $\hat{W}_i(t) \leq t$, alors nous avons :*

$$\hat{R}_i^{wInt} \leq \hat{R}_i^w.$$

Preuve : En utilisant le Lemme 7, $\hat{t}^{int} \leq \hat{t^*}$. Puisque $W_{i,l}(t)$ est une fonction non décroissante, $W_i(\hat{t}^{int}) \leq W_i(\hat{t^*})$, et le théorème est prouvé. ■

Nous illustrons les trois méthodes de déduction Figure 4.3 en considérant l'ensemble de tâches présenté dans la Table 4.5. Nous analysons seulement l'ordonnançabilité de la tâche τ_2. En utilisant l'algorithme Deadline Monotonic, nous obtenons $t^{int} = 14, t^* = 15$ et $R_2 = W_2(t^*) = 14$.

tel-00554265, version 1 - 10 Jan 2011

TABLE 4.5 – Ensemble de tâches à priorité fixe et son analyse d'ordonnançabilité

Tâches	C_i	D_i	T_i	t^{int}	t^*	$\hat{t}^{int}$	$\hat{t}^*$	R_i	$\hat{R}_i^{wInt}$	$\hat{R}_i^w$	$\hat{R}_i$
τ_1	1	7.5	7.5								
τ_2	12	18	14	14	15	$14 + \frac{11}{13}$	18	14	14	15	$15 + \frac{4}{15}$

Nous choisissons $\epsilon = 0.4$ et ainsi $k = 2$ et $\widehat{S_2} = \{7.5, 18\}$. Puisque $\widehat{W_2}(7.5) > 7.5$, nous considérons seulement $t > 7.5$. Puisque, l'analyse du temps de réponse approchée conduit à :

$$\begin{aligned} \widehat{W_2}(t) &= C_2 + (t + T_1 + C_1 + J_1)\frac{C_1}{T_1} \\ &= \frac{193 + 2t}{15} \end{aligned}$$

En calculant le point d'intersection : $\widehat{W_2}(t) = t$, nous avons $\hat{t}^{int} = 14 + \frac{11}{13} \Rightarrow \hat{R}_i^{wInt} = W_2(\hat{t}^{int}) = 14$

Puisque $\widehat{W_2}(18) < 18 \Rightarrow \hat{t}^* = 18. \Rightarrow \hat{R}_2^w = W_2(\hat{t}^*) = 15$ et $\hat{R}_2 = \widehat{W_2}(\hat{t}^*) = 15 + \frac{4}{15}$. Comme prévu, nous avons bien $R_2 \leq \hat{R}_2^{wInt} \leq \hat{R}_2^w \leq \hat{R}_2$. Tous les résultats sont résumés dans la Table 4.5.

4. Analyse pire cas des bornes d'erreur

4.1. Analyse du ratio d'approximation

Nous savons que le pire temps de réponse ($wcrt$) est non approximable au regard du ratio d'approximation classique (voir [13] pour les détails), mais les preuves présentés dans [13] sont très complexes et nous choisissons de présenter ici que notre meilleure borne (c-à-d., $\hat{R}_i^{WInt}$) ne conduit pas à une garantie de performance avec le ratio d'approximation classique à l'aide d'un contre exemple.

La garantie de performance de notre borne supérieure peut être analysée à l'aide du *ratio d'approximation* . Soit a une valeur obtenue par un algorithme A, et opt la valeur exacte (c-à-d., optimale) ; l'algorithme A a un ratio d'approximation c, où $c \geq 1$, si et seulement si, $opt \leq a \leq c \times opt$ pour toutes les entrées de l'algorithme A (si un tel c, n'existe pas alors l'algorithme A n'a pas de ratio approximation).

Le prochain résultat établit que la borne approchée du temps de réponse $\hat{R}_i^{wInt}$ n'a pas de ratio d'approximation constant.

Théorème 22 *Pour un paramètre de précision donné ϵ, il existe des systèmes de tâches pour lesquels $cR_i \leq \hat{R}_i^{wInt}$ pour tout entier c.*

Preuve : Soit k défini par $k = \lceil 1/\epsilon \rceil - 1$.

Nous prouvons ce théorème en démontrant qu'un système de tâche et une tâche τ_i par laquelle $\hat{R}_i^{wInt}/R_i$ tend vers ∞. Toutes les tâches de notre système vérifient $D_i = T_i, J_i = 0, B_i = 0$; nous présentons les paramètres de la tâche τ_i par une paire ordonnée (C_i, T_i). Considérons l'ensemble de tâches suivant : $\tau_1 = (K, 2K + \lambda)$, $\tau_2 = (K, 2K + \lambda)$ et $\tau_3 = (\lambda K, K(2K + \lambda) + \frac{2K(K+\lambda)}{\lambda})$, où λ est un nombre entier positif arbitrairement petit $\frac{1}{\lambda}$ et K est un nombre entier arbitraire qui est strictement

tel-00554265, version 1 - 10 Jan 2011

plus grand que $k-1$. Notons que par construction, $D_3 = T_3 > (k-1)(2K+\lambda)$. Ainsi, depuis l'Eq. (4.11), $\widehat{S_3}$ contient le point $t = D_3$.

En utilisant l'algorithme d'ordonnancement Deadline Monotonic, la tâche τ_3 ne peut être exécutée que λ unités de temps dans tout intervalle de temps de longueur $2K+\lambda$. En conséquence, le temps de réponse exact de τ_3 est : $R_3 = K(2K+\lambda)$. L'approximation devient une fonction linéaire à la date $(k-1)(2K+\lambda)$ qui est strictement plus petite que $K(2K+\lambda)$. Nous considérons l'instant $t > (k-1)(2K+\lambda)$. Ainsi, dans $\widehat{S_3}$, seul l'instant $t = D_3$ sera considéré.

L'analyse approchée du temps de réponse conduit à :

$$\widehat{W_3}(t) \quad = \quad \lambda K + 2(t+K+\lambda)\frac{K}{2K+\lambda}$$

En résolvant la condition sur l'instant d'intersection $\widehat{W_3}(t) = t$, nous avons :

$$t = K(2K+\lambda) + \frac{2K(K+\lambda)}{\lambda}$$

$\Rightarrow$ l'instant $t = T_3 = D_3$ est exactement l'instant d'intersection de τ_3.

En appliquant la Définition 18 et en remplaçant $\hat{t}^{int} = D_3$, nous obtenons que le temps de réponse approché de la tâche t_3 est :

$$\hat{R}_3^{wInt} = W_3(\hat{t}^{int}) = K(2K+2+\lambda) + \frac{2K^2}{\lambda}$$

Ainsi,

$$\lim_{\lambda \to 0} \frac{\hat{R}_3^{wInt}}{R_3} = \lim_{\lambda \to 0} \frac{K(2K+2+\lambda) + \frac{2K^2}{\lambda}}{K(2K+\lambda)} = \infty$$

et le théorème est prouvé. ■

4.2. Analyse de l'augmentation de ressource

Le théorème 22 précédent révèle que l'approximation des temps de réponse avec la borne $\hat{R}_i^{wInt}$ n'offre aucune garantie de performance quantifiable, en utilisant la mesure classique du ratio d'approximation utilisée dans la théorie de l'optimisation. Toutefois, une approche alternative pour contourner les limitations de cette approche, appelée la technique d'augmentation de ressource, devient de plus en plus populaire dans la théorie de l'ordonnancement temps réel. Dans cette technique, la performance de l'algorithme analysé est comparée avec un algorithme optimal qui s'exécute sur un processeur plus lent. Dans ce paragraphe, nous appliquons la technique d'augmentation de ressource pour quantifier la déviation de $\hat{R}_i^{wInt}$ par rapport au temps de réponse exact.

Tout d'abord, nous supposons que le facteur de ralentissement du processeur est appliqué de façon équivalent à toutes les tâches du système (nous avons conscience que cette hypothèse est simplificatrice pour certaines tâches dont la durée d'exécution n'est pas forcément reliée à la capacité du processeur). Sous cette hypothèse, nous pouvons considérer une tâche s'exécutant sur un processeur de capacité s comme une tâche dont le WCET est augmenté de $1/s$ fois par rapport un son temps d'exécution sur un processeur de capacité unitaire (mais tous les autres paramètres restent identiques lorsqu'un

tel-00554265, version 1 - 10 Jan 2011

processeur de capacité s est considéré). Nous considérons que C_i^s, D_i^s, T_i^s, sont respectivement la pire durée d'exéctuion, l'échéance relative et la période de la tâche τ_i sur un processeur de capacité s, nous avons alors :

$C_i^s = \frac{C_i}{s}$
$D_i^s = D_i$
$T_i^s = T_i$
$J_i^s = J_i$

Nous introduisons des notations supplémentaires pour la borne de la demande processeur et pour la demande cumulée de processeur par une tâche τ_i sur un processeur de vitesse s :

$$\mathrm{RBF}^s(\tau_i, t) \stackrel{\text{def}}{=} \left\lceil \frac{t + J_i}{T_i^s} \right\rceil C_i^s \tag{4.16}$$

$$W_i^s(t) \stackrel{\text{def}}{=} C_i^s + \sum_{j=1}^{i-1} \mathrm{RBF}^s(\tau_j, t) \tag{4.17}$$

Clairement nous avons la propriété suivante que est satisfaite :

Lemme 8 $\forall i, 1 \leq i \leq n, \forall t, \mathrm{RBF}^s(\tau_i, t) = \frac{\mathrm{RBF}(\tau_i,t)}{s}$, $W_i^s(t) = \frac{W_i(t)}{s}$.

En utilisant la relation entre la fonction de demande processeur, la demande cumulée de processeur dont la capacité est s et celle sur un processeur de capacité unitaire, tout en exploitant les propriétés sur nos fonctions approchées, nous pouvons déterminer que le plus grand facteur de ralentissement associé à notre borne supérieure du temps de réponse $\hat{R}_i^{wInt}$.

Théorème 23 *La borne $\hat{R}_i^{wInt}$ (Eq. (4.14)) est*

1. *une borne supérieure du pire temps de réponse de τ_i ; et*
2. *une borne inférieure du pire temps de réponse de τ_i* si le système est implémenté sur un processeur de vitesse $k/(k+1)$, $k > 0$.

Preuve : La première proposition est directement vérifiée depuis le Théorème 20. Nous prouvons la seconde.

Nous notons $s = \frac{k}{k+1} < 1$

Par Théorème 14, $\forall j, 1 \leq j \leq i-1, \forall t \in (0, D_i - J_i]$, nous avons :

$$\begin{aligned}
\gamma(\tau_j, t) &\leq \frac{k+1}{k} \mathrm{RBF}(\tau_j, t) \\
\sum_{i-1} \gamma(\tau_j, t) &\leq \sum_{i-1} \frac{k+1}{k} \mathrm{RBF}(\tau_j, t) \\
C_i + \sum_{i-1} \gamma(\tau_j, t) &\leq \frac{k+1}{k} C_i + \sum_{i-1} \frac{k+1}{k} \mathrm{RBF}(\tau_j, t) \\
C_i + \sum_{i-1} \gamma(\tau_j, t) &\leq C_i^s + \sum_{i-1} \mathrm{RBF}^s(\tau_j, t) \\
\widehat{W}_{i,l}(t) &\leq W_i^s(t).
\end{aligned} \tag{4.18}$$

tel-00554265, version 1 - 10 Jan 2011

Soit t^{int} le point d'intersection correspondant au pire temps de réponse de τ_i sur le processeur de vitesse-s (c-à-d., le premier instant tel que $W_i^s(t) = t$). Soit $\hat{t}^{int}$ le premier instant tel que $\widehat{W_i}(t) = t$ (c-à-d., $\hat{t}^{int}$ est un point le d'intersection approché).

En utilisant le Théorème 20 nous avons $\forall t \in (0, \hat{t}^{int})$ $\widehat{W_i}(t) > t$. Depuis (4.18), nous avons nécessairement $\forall t \in (0, \hat{t}^{int})$ $W_i^s(t) > t \Rightarrow t^{int} \notin (0, \hat{t}^{int}) \Rightarrow t^{int} \geq \hat{t}^{int}$.

Puisque $W_i^s(t) = \frac{W_i(t)}{s} > W_i(t)$ et le fait que $W_i(t)$ et $W_i^s(t)$ sont des fonctions non-décroissantes, nous avons $W_i(\hat{t}^{int}) \leq W_i^s(t^{int})$. De façon équivalente, $\hat{R}_i^{wInt}$ est une borne inférieure du temps de réponse exacte $W_i^s(t^{int})$. ■

Comment interpréter le théorème 23 précédent ? Premièrement, il est garanti que $\hat{R}_i^{wInt}$ est de fait une borne supérieure R_i ; ainsi, cette borne peut être utilisée de façon sûre pour estimer le temps de réponse d'une tâche. Puisque le théorème 23 n'est pas utilisable pour borner la quantité avec laquelle $\hat{R}_i^{wInt}$ excède la valeur de R_i, elle assure toutefois au concepteur qu'une borne supérieure de temps de réponse inférieure à $\hat{R}_i^{wInt}$ si le système était implémenté sur un vitesse $k/(k+1)$ fois plus rapide. Formulé de façon différente, *une accélération du processeur d'un facteur* $(k+1)/k$ *est une borne supérieure du prix à payer pour utiliser cette borne du temps de réponse calculée en temps polynomial en la taille du système de tâche et la constante* $1/\epsilon$.

5. Experimentations Numériques

Dans ce paragraphe, nous décrivons les expérimentations numériques réalisées pour comparer notre borne supérieure des temps de réponse à celles connues afin de mettre en évidence les garanties de performance moyenne.

Modèle Stochastique. Nous avons généré aléatoirement des systèmes de tâches à échéances contraintes. Des facteurs d'utilisation non biaisés ont été générés à l'aide de la méthode UUniFast [8]. Les périodes T_i sont générées aléatoirement dans l'intervalle $[1, 2500]$ et les pires durées d'exécution C_i sont calculées par $C_i = U_i T_i$. Les échéances relatives D_i sont générées aléatoirement dans l'intervalle $[C_i, T_i]$. Une loi uniforme de probabilité a été utilisée pour générer ces nombres. C_i, D_i, T_i ont été arrondis à l'entier le plus proche. Le facteur d'utilisation varie de 0.5 à 0.9 (avec un pas de 0.1) et pour chaque valeur, le même nombre de configurations de tâches a été généré.

Les paramètres de l'expérimentation sont le nombre de tâches n, le facteur d'utilisation U et le paramètre de précision ϵ (qui permet de déterminer k, le nombre d'étapes avant de débuter l'approximation linéaire de la demande processeur). Pour chaque jeu de paramètres, nous avons répliqué 400 fois l'expérience afin d'obtenir des résultats statistiquement corrects.

Nous présentons ci-après les notions qui seront utilisées dans nos expérimentations. $supB_i$ est la borne supérieure de Bini et Baruah :

$$supB_i = \frac{C_i + \sum_{j=1}^{i-1} C_j(1 - U_j)}{1 - \sum_{j=1}^{i-1} U_j}.$$

Dans la Table 4.6 sont présentées les fonctions d'approximations liniares utiliséees durant nos expérimentations (c-à-d., $\text{LA}_3(\tau_i, t), \text{LA}_4(\tau_i, t)$, voir la Table 4.1) et les trois méthodes de déduction des bornes de temps de réponse $\hat{R}_i, \hat{R}_i^w, \hat{R}_i^{wInt}$ (voir la Table 4.4).

tel-00554265, version 1 - 10 Jan 2011

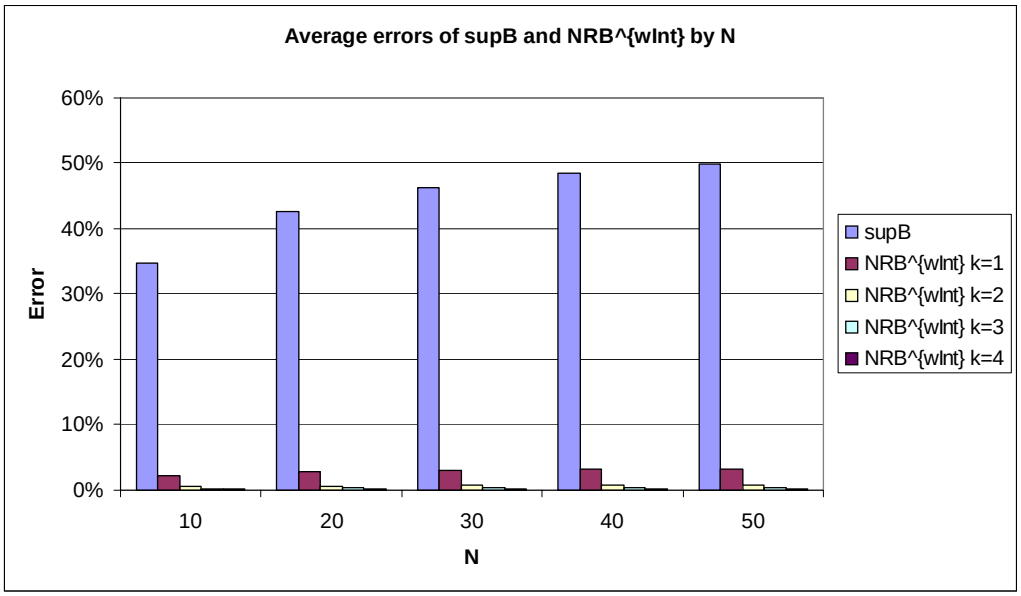

FIGURE 4.4 – Average error of $supB_i$ and NRB_i^{wInt}

Comparaison avec les bornes linéaires. Nous comparons notre meilleure borne supérieure NRB_i^{wInt}, obtenue avec l'approximation linéaire $\text{LA}_4(\tau_i, t)$ et la méthode de déduction $\hat{R}_i^{wInt}$ avec la meilleure borne supérieure des temps de réponse obtenue par un algorithme linéaire $supB_i$.

Pour pouvoir comparer ces bornes, seules les tâches conclues faisables par notre test approché ont été considérées (sinon, aucune borne supérieure ne peut être calculée). Nous avons considéré deux indicateurs pour comparer les résultats : l'erreur moyenne par rapport à la valeur exacte du temps de réponse (c-à-d., $(ub_i - R_i)/R_i$, où ub_i est une borne supérieure et R_i est la valeur exacte du temps de réponse) et le taux de tâches statuées "non ordonnançables" en utilisant une borne supérieure (c-à-d., $ub_i > D_i$) pour des tâches faisables (c-à-d., $R_i \leq D_i$).

Les résultats numériques sont présentés Figure 4.4 et Figure 4.5. Dans le premier graphique, l'erreur moyenne de l'algorithme approché est présentée pour différentes valeurs de k ; ces résultats montrent

TABLE 4.6 – Schéma d'approximation et méthodes de déduction comparés dans les expérimentations numériques

Approximation linéaire/ Méthode de déduction	$\hat{R}_i$	$\hat{R}_i^w$	$\hat{R}_i^{wInt}$
$\text{LA}_3(\tau_i, t)$ by Fisher, Nguyen, Richard and Goossens in [16]			$FNRG_i^{wInt}$
$\text{LA}_4(\tau_i, t)$ — Ce chapitre	NRB_i	NRB_i^w	NRB_i^{wInt}

tel-00554265, version 1 - 10 Jan 2011

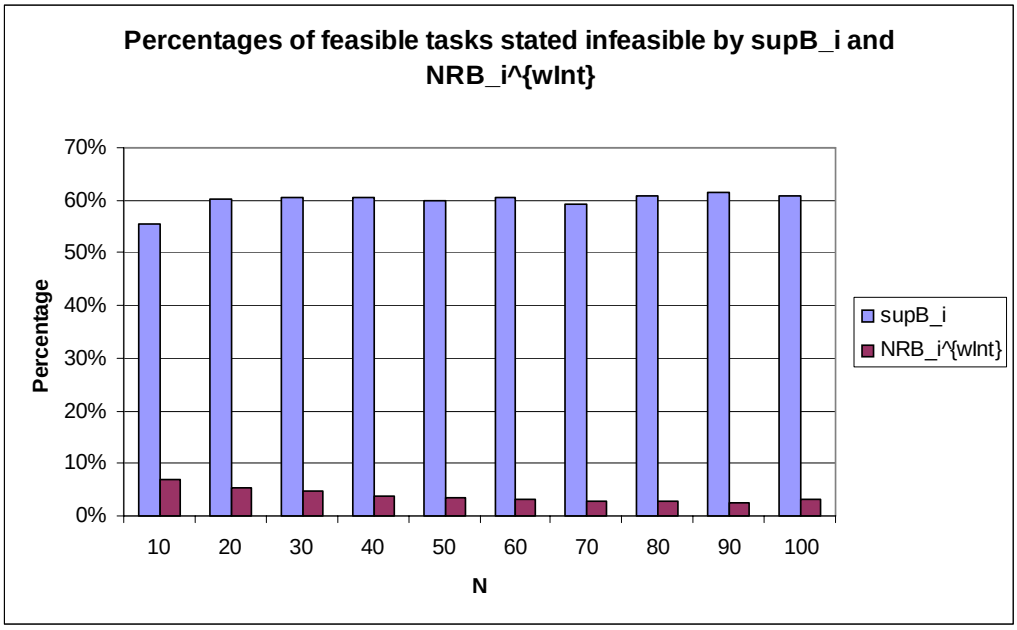

FIGURE 4.5 – Tâches faisables classées non ordonnançable par sup_iB et NRB_i^{wInt}

que notre méthode améliore clairement les résultats avec les précédentes bornes connues, même lorsque $k = 1$ (c-à-d., la plus petite valeur possible puisque $k = \lceil 1/\epsilon \rceil - 1$ et $0 < \epsilon < 1$). L'erreur moyenne est inférieure à 1% lorsque $k = 3$ (c-à-d., $0.25 \leq \epsilon < 0.33$). Dans le second graphique, nous voyons que notre approche est moins pessimiste puisque seules quelques tâches ordonnançable n'ont pas été acceptées avec notre méthode approchée. Ce qui est plus intéressant est que lorsque le nombre de tâches augmente, l'erreur moyenne de $supB_i$ augmente alors que celle de notre borne diminue.

Comparaison avec la meilleure borne approchée. Dans la Figure 4.6, nous calculons et comparons les erreurs moyennes des deux bornes supérieure par rapport à k : la borne $FNRG_i^{wInt}$ obtenue avec l'approximation linéaire $\text{LA}_3(\tau_i, t)$ ([16]) et la borne NRB_i^{wInt} résultant de l'approximation linéaire $\text{LA}_4(\tau_i, t)$ présentée dans ce chapitre, toutes les deux basées sur la même méthode de déduction $\hat{R}_i^{wInt}$ (Définition 18). Notre borne fournit une petite amélioration autour de 2 % en comparaison avec la meilleure borne connue auparavant $FNRG_i^{wInt}$.

Comparaison des bornes obtenues avec les méthodes de déduction existantes. En utilisant le même schéma d'approximation, présenté dans ce chapitre (c-à-d., l'approximation linéaire $\text{LA}_4(\tau_i, t)$), pour chaque tâche vérifiée "faisable", nous calculons et comparons trois bornes supérieures : NRB_i, NRB_i^w et NRB_i^{wInt} respectivement calculées avec les méthodes de déduction $\hat{R}_i$, $\hat{R}_i^w$ et $\hat{R}_i^{wInt}$.

Dans la Figure 4.7, nous reportons l'erreur moyenne de ces trois bornes en comparaison avec le temps de réponse exact, lorsque le nombre de tâches varient de 10 à 100 (avec un pas de 10). Nous pouvons voir que comparativement à la méthod présentée dans [16,31] (c-à-d., NRB_i), notre première méthode (i.e., NRB_i^w) réduit l'erreur moyenne de près de 50 % et que pour notre seconde méthode (c-à-d., $\hat{R}_i^{wInt}$), l'erreur moyenne diminue de moins de 20%.

Analyse de l'augmentation de ressource. Pour chaque ensemble de tâches généré, nous cal-

tel-00554265, version 1 - 10 Jan 2011

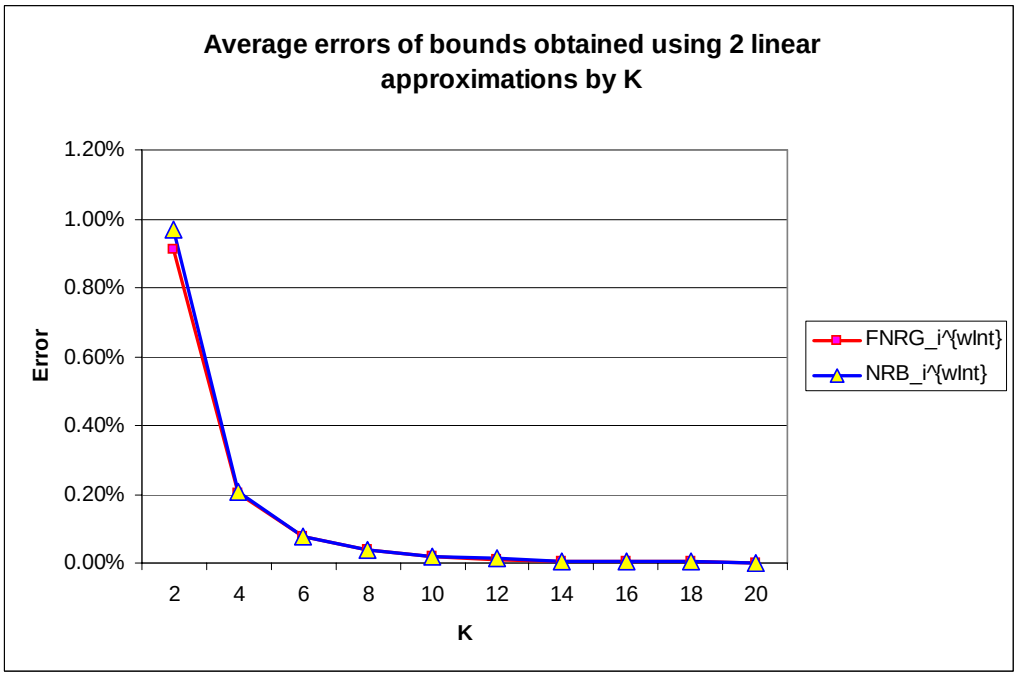

FIGURE 4.6 – Erreur moyenne de $FNRG_i^{wInt}$ et NRB_i^{wInt} selon k

culons (à l'aide d'une recherche dichotomique) le facteur de ralentissement exact s tel que la borne NRB^{wInt} devienne l'actuel temps de réponse de la tâche sur un processeur de vitesse s. Nous avons vu d'un point de vue théorique que le pire facteur de ralentissement est borné par $k/(k+1)$, où k est relatif à paramètre de précision de l'algorithme approché.

Dans la Figure 4.8, nous indiquons les facteurs de ralentissement moyens et minimum en fonction du paramètre de précision k. On peut voir que l'erreur moyenne est toujours entre $k/(k+1)$ et 1 ; très proche de 1, ce qui signifie que la capacité de processeur "perdue" en utilisant cette approche est très faible en pratique. Comme prévu, la valeur minimum correspond à la borne théorique $(k/(k+1))$; ainsi, le pire facteur de ralentissement a été atteint dans chacune de nos simulations pour un jeu de paramètres fixés.

Dans la Figure 4.9, nous comparons le facteur de ralentissement moyen (slowdown factor SDF) de $supB_i$ et celle de notre borne. Nous pouvons noter que pour le SDF moyen, nos bornes apportent une amélioration d'approximativement 28% par rapport à la borne $supB_i$ et ceci même lorsque $k = 2$. Le SDF moyen pour $k = 4$ est supérieur à 97%, et donc que la capacité de processeur perdue avec notre méthode est inférieure à 3%.

6. Conclusions

Nous avons présenté dans ce chapitre un nouveau test d'ordonnançabilité approché et deux nouvelles méthodes de déduction de bornes supérieures des pires temps de réponse des tâches à priorité fixe, échéance contrainte et assujettis à des gigues sur activation. Si le paramètre de précision de l'algorithme

tel-00554265, version 1 - 10 Jan 2011

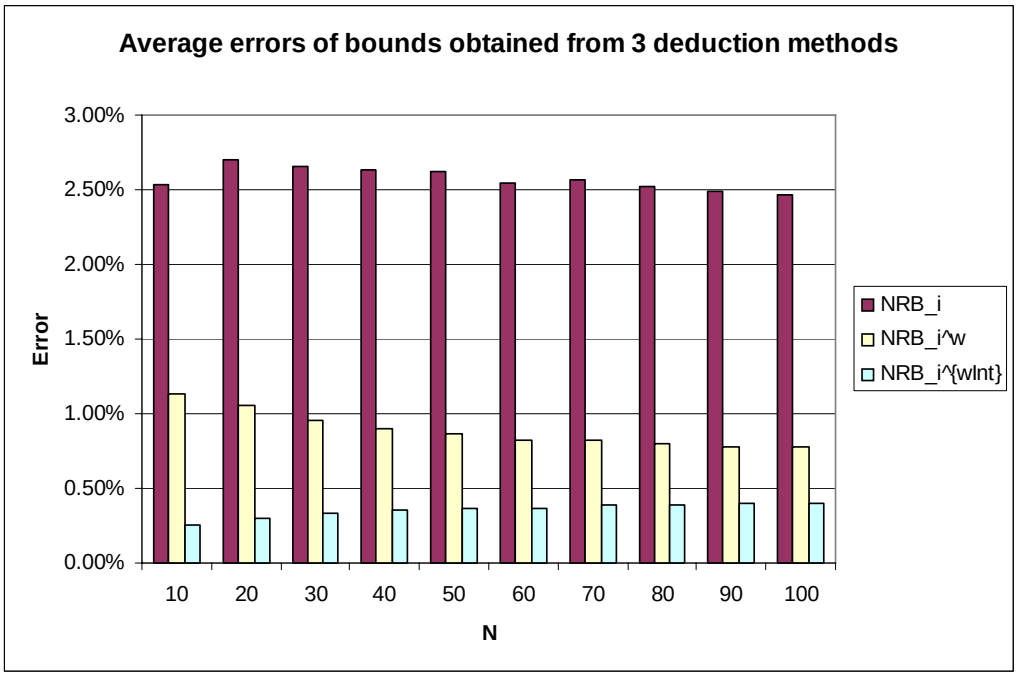

FIGURE 4.7 – Erreurs moyennes des bornes obtenues avec les trois méthodes de déduction

approché, ϵ, qui est utilisé pour fixer le nombre d'étapes avant de commencer l'approximation linéaire de la demande processeur est très petit, alors nos expérimentations montrent que les bornes calculées sont très proches des temps de réponse exacts, mais tout en restant calculable en temps polynomial en fonction de la taille du système de tâches analysé et du paramètre de précision $1/\epsilon$.

Nous avons quantifié la qualité de l'approximation réalisée. Nous avons montré que les bornes des temps de réponse n'offrent pas de garantie de performance si le ratio d'approximation conventionnel est considéré. Toutefois, en utilisant la technique d'augmentation de ressource, nous avons obtenu des garanties de performance suivante : notre borne est supérieure à la valeur exact du temps de réponse lorsqu'un processeur de capacité unitaire est considéré et la borne est inférieure au temps de réponse exacte si un processeur de capacité $k/(k+1)$ plus rapide est considéré (où $k \stackrel{\text{def}}{=} \lceil 1/\epsilon \rceil - 1$).

Notons enfin, que les méthodes présentées dans ce chapitre peuvent être facilement étendues (analyses d'ordonnançabilité et calculs de temps de réponse) pour tenir compte des facteurs de blocage associés à protocole de gestion de ressources.

tel-00554265, version 1 - 10 Jan 2011

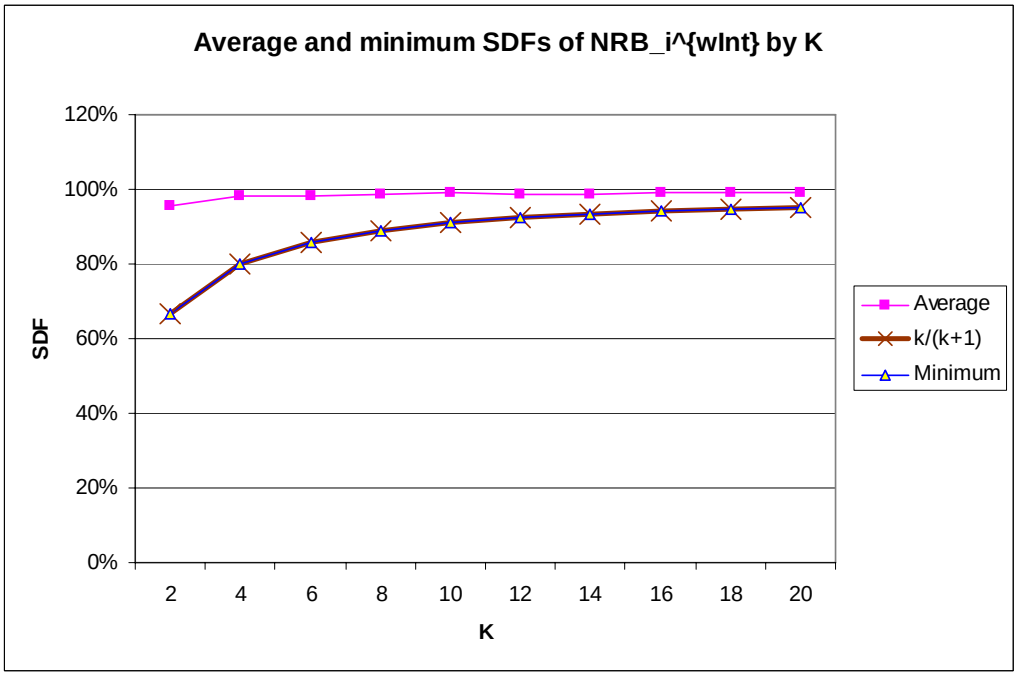

FIGURE 4.8 – Facteurs de ralentissement moyen et minimum de NRB_i^{wInt}

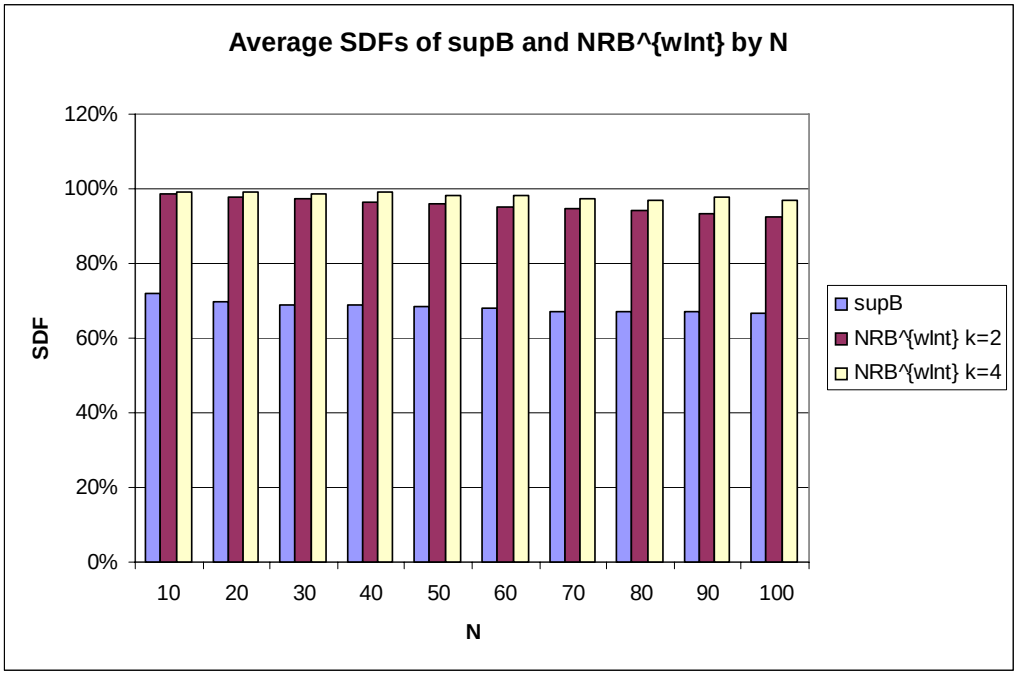

FIGURE 4.9 – Facteur de ralentissement moyen de $supB_i$ et NRB_i^{wInt}

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Chapitre 5

Tâches à échéances arbitraires

1. Introduction

Dans ce chapitre nous étendons les résultats précédents aux cas de tâches à échéance arbitraire (l'échéance relative D_i n'est pas reliée à la période T_i). Nous considérons des tâches sporadiques à priorité fixe devant s'exécuter sur une plateforme monoprocesseur, sans gigue d'activation. Pour ce modèle de tâches RM et DM ne sont plus des algorithmes optimaux, toutefois l'algorithme d'Audsley permet de définir une assignation des priorités afin de définir un ordonnancement à priorité fixe respectant toutes les échéances des tâches s'il en existe un. L'inconvénient de cette procédure d'assignation des priorités est qu'un test de faisabilité doit être exécuté pour chaque niveau de priorité (du niveau le plus faible, au niveau le plus élévé).

Des tests exacts et approchés sont connus pour ce modèle de tâche. L'analyse exacte de la demande processeur et le calcul du temps de réponse ont été étudiés dans [19]. L'analyse ne peut se limiter comme dans le cas d'échéance contrainte à l'étude de la première instance de la tâche analysée. Au contraire, toutes les instances contenues dans la période d'activité synchrone de niveau i doivent être considérées afin de déterminer le respect des échéances d'une tâche τ_i. Le nombre d'instances à considérer est donc borné par la longueur de cette période d'activité synchrone et conduit à un test de complexité pseudo-polynomiale lorsque le facteur d'utilisation du processeur est strictement inférieur à 1. Un test approché, reposant sur l'approximation de la fonction de demande processeur (c.f., chapitre précédent) a été proposé par Fisher et Baruah en 2005 [15].

Dans un premier temps, nous revisitons dans ce chapitre le test approché de Fisher et Baruah dans le même contexte. Ce test repose sur un algorithme polynomial en la taille du système de tâches et du paramètre de précision ϵ (c-à-d., c'est un schéma d'approximation complet ou FPTAS). Nous avons détecté dans cet algorithme des problèmes que nous mettrons tout d'abord en évidence. Nous montrerons notamment que l'algorithme originel de Fisher et Baruah n'est pas valide dans le cas d'une assignation quelconque des priorités aux tâches, mais l'est seulement lorsque les priorités respectent l'ordonnancement selon RM. Nous proposons ensuite une modification de cet algorithme pour le rendre valide quelle que soit l'assignation des priorités. Nous détaillerons le fonctionnement de ce nouvel algorithme sur un exemple détaillé complet.

Dans un deuxième temps, nous étendons la méthode pour permettre le calcul de bornes supérieures

tel-00554265, version 1 - 10 Jan 2011

des pires temps de réponse des tâches à échéance arbitraire. La méthode contourne l'écueil rencontré dans le chapitre précédent, à savoir, la méthode pourra calculer les temps de réponse approchés des tâches non ordonnançables, avec une garantie de performance de l'approximation affectuée grace à la technique d'augmentation de ressource déjà utilisée dans le chapitre précédent. Nous présenterons des résultats numériques d'expérimentation. Le travail présenté dans la suite est le fruit d'une collaboration avec Nathan Fisher [29].

Afin de faciliter la lecture non linéaire de ce document, nous redéfinissons rapidement les caractéristiques et hypothèses associées au modèle de tâche, ainsi que les principaux résultats connus dans la littérature.

2. Modèle de tâches et résultats connus

2.1. Modèle de tâche

Nous considérons un système de tâches sporadiques s'exécutant sur un processeur et qui ne se suspendent pas elles-mêmes par des opérations de synchronisation. Les tâches ont des priorités fixes initialisées au démarrage de l'application et qui ne sont jamais modifiées ensuite. Une tâche en exécution peut être préemptée à tout instant par une tâche plus prioritaire, sans surcoût. A chaque instant, la tâche prête dont la priorité est la plus élevée obtient le processeur pour s'exécuter. Sans perte de généralité, nous supposons que les tâches sont indexées dans l'ordre décroissant des priorités : τ_1 est la tâche la plus prioritaire et τ_n est assignée au niveau de priorité le plus faible.

Une tâche sporadique τ_i, $1 \leq i \leq n$, est définie par son pire temps d'exécution (WCET) C_i, une échéance relative D_i et une période T_i qui définit l'intervalle minimum entre deux occurrences successives de τ_i. Nous supposons dans la suite que $\forall i C_i \neq 0$. Le facteur d'utilisation d'une tâche τ_i est la fraction de temps où elle requiert le processeur : $U_i \stackrel{\text{def}}{=} C_i/T_i$. Le facteur d'utilisation d'un système de tâches est : $U \stackrel{\text{def}}{=} \sum_{j=1}^{n} U_j$.

2.2. Résultats connus sur le calcul du pire temps de réponse

2.2.a. Analyse exacte

A notre connaissance, aucun test de faisabilité polynomial n'est connu pour le modèle de tâche considéré dans ce chapitre. Des algorithmes pseudo-polynomiaux en temps sont connus et reposent sur l'analyse de la demande processeur (Request Bound Function ou RBF) d'une tâche τ_i à un instant t (notée $\text{RBF}(\tau_i, t)$) :

$$\text{RBF}(\tau_i, t) \stackrel{\text{def}}{=} \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{5.1}$$

Cette fonction calcule une borne supérieure du temps d'exécution requit par une tâche τ_i sur l'intervalle de temps $[0, t)$. La première approche utilisée pour déterminer si une tâche τ_i est ordonnançable ou non consiste à calculer son pire temps de réponse R_i. Pour la tâche τ_i, son pire temps de réponse surviendra toujours dans une période d'activité initiée par un instant critique [23], c'est-à-dire que la tâche analysée s'active au même instant que ses tâches plus prioritaires. Ce scénario d'activation des tâches définit la *période d'activité synchrone de niveau i*. Ainsi, dans une telle période d'activité, seules

des tâches de priorité au moins égale à i s'exécutent de façon continue. Dans le cas de tâches à échéance contrainte, il est suffisant de considérer la première instance de τ_i pour déterminer son pire temps de réponse. Mais lorsque des tâches sont à échéance arbitraire, Lehoczky [19] a montré qu'il est nécessaire d'étudier toutes les instances de la tâche analysée dans la période d'activité synchrone de niveau i.

Pour calculer le temps de réponse d'une instance de τ_i, nous définissons la *fonction de demande processeur cumulée*, basée sur la fonction RBF, de la façon suivante :

$$W_{i,l}(t) \stackrel{\text{def}}{=} lC_i + \sum_{j<i} \text{RBF}(\tau_j, t) \tag{5.2}$$

Cette fonction est simplement la durée cumulée d'exécution de toutes les tâches de priorités supérieure i sur l'intervalle $(0, t]$ et l'exécution des l premières instances de τ_i. Avec cette définition, soit $R_{i,l}$ le point d'intersection entre cette fonction de demande et la droite affine représentant la capacité de traitement du processeur, en utilisant des approximations successives nous pouvons calculer de façon itérative $R_{i,l}$ comme le plus petit point fixe de $W_{i,l}(t) = t$. Nous en déduisons ainsi le pire temps de réponse de $\tau_{i,l}$. Puisque le nombre d'instances de τ_i dans la période d'activité de niveau i peut être pseudo-polynomial et que le calcul pour chaque instance est lui même un calcul pseudo-polynomial en temps, la complexité totale de la méthode de calcul du pire temps de réponse de τ_i est pseudo-polynomiale.

Dans [20] a été introduit une autre approche pour résoudre le problème d'ordonnançabilité dans le cas d'échéance contrainte, connue sous le nom d'analyse de la demande processeur (Processor Demand Analysis ou PDA). Cette approche est étendue dans [19] au cas des tâches à priorité fixe et échéance arbitraire, que est rappelée ci-après :

Théorème 24 ([19]) *Un système de tâches sporadiques τ est ordonnançable si, et seulement si, $\forall \tau_i \in \tau, l(>0) \in N, \exists t \in ((l-1)T_i, (l-1)T_i + D_i]$ tel que $W_{i,l}(t) \leq t$*

Ce résultat peut se reformuler ainsi :

Théorème 25 ([19]) *Soit $N_i \stackrel{\text{def}}{=} min\{m \in N \mid \exists t \in ((m-1)T_i, (m-1)T_i + T_i], W_{i,m}(t) \leq t\}$, alors une système de tâches sporadiques τ est ordonnançable si, et seulement si, $\forall \tau_i \in \tau, l(>0) \in N, l \leq N_i, \exists t \in ((l-1)T_i, (l-1)T_i + D_i]$ tel que $W_{i,l}(t) \leq t$.*

2.2.b. Analyse approchée des temps de réponse

Deux approches principales ont été développées dans la littérature pour déterminer des bornes supérieures des pire temps de réponse approchés. Ces deux approches sont fondées sur une approximation linéaire de la fonction de demande processeur (c-à-d., la fonction RBF).

Bornes calculables en temps linéaire. Il est bien connu que le pire scénario qu'affronte une tâche τ_i survient dans le plus grand intervalle de temps de longueur t durant lequel la tâche τ_i s'exécute. La longueur de cet intervalle peut être bornée en relaxant la contrainte d'intégralité de la fonction RBF (c-à-d., en relaxant la fonction plafond dans cette fonction). Ceci conduit à : $LA(\tau_i, t) \stackrel{\text{def}}{=} (\frac{t}{T_i} + 1)C_i = C_i + tU_i$. En utilisant cette fonction linéaire, une borne supérieure de la fonction cumulant la durée d'exécution des tâches de priorité supérieure ou égale à $\tau_{i,l}$ peut être calculée comme : $\widehat{W}_{i,l}(t) =$

tel-00554265, version 1 - 10 Jan 2011

$lC_i + \sum_{j<i} LA(\tau_i, t)$. En résolvant l'équation $\widehat{W}_{i,l}(t) = t$, une borne supérieure du pire temps de réponse de $\tau_{i,l}$ est obtenue :

$$R_{i,l} \leq \frac{lC_i + \sum_{j=1}^{i-1} C_j}{1 - \sum_{j=1}^{i-1} U_j} \tag{5.3}$$

Récemment, cette borne a été améliorée dans [6] et étendue ensuite pour tenir compte des gigues sur activation et facteur de blocage associés à un protocole de gestion de ressource dans [33]. Nous avons montré dans le chapitre 3 (voir aussi [10]) que cette borne supérieure n'a pas une borne constante d'erreur vis-à-vis du pire temps de réponse exact (c-à-d., il existe un ensemble de tâches tel que la borne supérieure est c fois plus grande que R_i où c est un nombre arbitrairement grand). Ainsi, l'algorithme linéaire en temps (de complexité $\mathcal{O}(n)$) n'est pas un algorithme approché au sens classique de la théorie de l'optimisation. Mais, en utilisant la technique d'augmentation de ressource, nous avons montré que cette formule conduit à une borne supérieure si un processeur de capacité unitaire est considéré et une borne inférieure sur un processeur de capacité $1/2$. Ainsi, un facteur d'accélaration du processeur de 2 est le prix à payer pour utiliser cette borne supérieure de R_i qui se calcule en temps linéaire.

Schéma d'approximation. Les techniques d'approximation ont été récemment étudiées pour définir des tests de faisabilité d'un système de tâches à échéance contrainte [15] (voir chapitre précédent). L'approximation est fondée sur l'approche suivante : si une tâche est prouvée faisable par le test, alors celle-ci est ordonnançable sur un processeur de capacité unitaire, tandis que si le test conclut que la tâche n'est pas ordonnançable, alors la tâche n'est pas faisable sur un processeur de capacité $(1 - \epsilon)$ [11, 14]. L'idée principale est de réaliser un calcul exact durant seulement k étapes (où k est fonction du paramètre de précision ϵ en entrée de l'algorithme) et d'utiliser ensuite une fonction linéaire pour approximer efficacement la fonction de demande processeur. Ce test vérifie la faisabilité d'une tâche en temps polynomial en le nombre de tâches n, et k le nombre d'étapes fixé en entrée de l'algorithme. Cette méthode définit donc un schéma d'approximation complet ou FPTAS, en considérant la technique d'augmentation de ressource comme indicateur de garantie de performance.

Une première idée pour généraliser cette approche au cas d'échéance arbitraire serait d'appliquer cette méthode pour chaque instance de la tâche τ_i à analyser dans la période d'activité synchrone de niveau i. Toutefois, le nombre d'instances à considérer est pseudo-polynomial :

- Le nombre d'instances d'une tâche τ_i à considérer dans une période d'activité synchrone de niveau i est de l'ordre de $\mathcal{O}(\frac{T_i}{C_i})$ et donc ce nombre n'est pas borné polynomialement dans la taille du système de tâches. En conséquence, utiliser l'approche développée pour les tâches à échéance contrainte au cas de tâches à échéance arbitraire conduirait donc à une complexité pseudo-polynomiale en temps puisque le test serait exécuté pour chaque instance de τ_i dans la période d'activité synchrone de niveau i.
- Pour chaque point d'ordonnancement, il y a au plus $\mathcal{O}(\frac{D_i}{T_i})$ instances actives. En conséquence, même si le test s'appliquait sur les points d'ordonnancement, la complexité du test approché demeurerait pseudo-polynomiale en temps.

Dans le paragraphe suivant nous présentons la méthode développée par Fisher et Baruah [14], qui permet de contourner les problèmes mentionnés plus haut et définit un FPTAS pour résoudre le problème de faisabilité considéré dans ce chapitre. De plus la complexité de ce test pour les tâches à

tel-00554265, version 1 - 10 Jan 2011

échéance arbitraire est asymptotiquement la même que dans le cas de tâches à échéance contrainte. Nous présentons dans le paragraphe suivant cet algorithme et nous l'analyserons afin de mettre en évidence des problèmes montrant que celui ci n'est correct que lorsque les tâches sont ordonnancées avec l'algorithme d'ordonnancement RM.

3. Analyse du schéma d'approximation de Fisher et Baruah

Nous présentons tout d'abord les idées principales du test de Fisher et Baruah [14] qui vérifie la faisabilité des tâches sporadiques à échéance arbitraire. Nous donnerons ensuite des contre-exemples de sa validité dans le cas d'assignation arbitraire des priorités aux tâches et nous montrerons qu'il est valide sous l'assignation RM. Nous pensons que ces problèmes ne peuvent pas être simplement corrigés sans modification de nombreuses formules et preuves. Toutefois nous proposerons des modifications de cet algorithme et des formules utilisées qui conserve la structure globale de l'algorithme (décomposition en deux étapes).

3.1. L'algorithme de Fisher et Baruah

Dans le but d'approximer la fonction RBF avec une borne d'erreur de $1+\epsilon$ (paramètre de précision du schéma d'appoximation complet, $0 < \epsilon < 1$), l'algorithme présenté dans [14] utilise les mêmes fondements que ceux utilisés dans le cas à échéance contrainte [15] : $(k-1)$ étapes durant lesquelles la fonction exacte de $\text{RBF}(\tau_i, t)$ est utilisée, où k est défini par $k \stackrel{\text{def}}{=} \lceil 1/\epsilon \rceil - 1$ et une approximation linéaire ensuite dans les étapes suivantes. On vérifie que $(k+1) \geq 1/\epsilon$.

Afin de faciliter la lecture non linéaire du mémoire, nous rappelons la définition de la fonction d'approximation de la demande processeur pour la tâche τ_i est :

$$\delta(\tau_i, t) \stackrel{\text{def}}{=} \begin{cases} \text{RBF}(\tau_i, t) & \text{si } t \leq (k-1)T_i, \\ (t+T_i)U_i & \text{sinon.} \end{cases} \tag{5.4}$$

Donc, jusqu'à la date $(k-1)T_i$, aucune approximation n'est effectuée pour évaluer le demande totale de de τ_i, et ensuite elle est approximée par une fonction linéaire de pente égale au facteur d'utilisation de τ_i.

La fonction approchée de demande cumulée du processeur pour chaque instance $\tau_{i,l}$ est définie de la façon suivante :

$$\widehat{W}_{i,l}(t) = lC_i + \sum_{j<i} \delta(\tau_j, t) \tag{5.5}$$

L'algorithme de Fisher et Baruah [14] est fondé sur l'analyse d'un sous-ensemble de taille polynomiale de l'ensemble des points d'ordonnancement considérés dans le test exact de Lehoczky [19]. Dans la première étape de cet algorithme, l'ensemble des points d'ordonnancement est exploré pour tester les instances de τ_i (c-à-d., les instances se terminant avant ou à la date $\max_{j \in 1,\ldots,i-1} \{(k-1)T_j\}$).

$$\tilde{S}_i \stackrel{\text{def}}{=} \{t = bT_a : a = 1 \ldots i, b = 1 \ldots k-1\} \cup \{0\} \tag{5.6}$$

tel-00554265, version 1 - 10 Jan 2011

Algorithm 1: FBApproxFirstStage

input :
$\tau = C[n], T[n], D[n]$: tableau d'entiers /* Paramètres des tâches */
i : entier /* Index de la tâche analysée */
k : entier /* The FPTAS paramètre de précision */
output: lowest_active : integer /* dernière instance active */

Construire $\tilde{S}_i$ en utilisant l'équation 5.6 /* Ensemble ordonné des points d'ordonnancement */
lowest_active=1 /* index de l'instance de τ_i à analyser */
foreach $t_a \in \tilde{S}_i - \{0\}$ **do**
  if $t_a > (lowest\_active-1)T_i + D_i$ **then** /* une instance se termine dans $(t_{a-1}, t_a]$ */
    Soit t_{a-1} l'élément adjacent avant t_a dans $\tilde{S}_i$;
    Soit y l'exécution cumulée des instances de tâches de priorité supérieure à τ_i et réveillée à l'instant t_a;
    $a = (t_{a-1}, \widehat{W}_{i,\text{lowest\_active}}(t_{a-1}) + y)$;
    $b = (t_a, \widehat{W}_{i,\text{lowest\_active}}(t_a))$;
    Déterminer si la droite (a, b) coupe la droite $f(t) = t$;
    Soit t' ce point d'intersection;
    if $t' > (lowest\_active - 1)T_i + D_i$ **then return** *"non faisable"* **else**
    lowest_active=lowest_active+1 ;
  end
  $x = \max\left(0, \left\lceil \frac{t_a}{T_i} \right\rceil - Z_i(t)\right)$;
  lowest_active $= \max(x + 1, \text{lowest\_active})$;
end
return *lowest_active*;

En utilisant cet ensemble de test, la première étape détermine sur toutes les instances de la tâche τ_i avec des échéances plus petite que $t = \max_{\{j \in \{1 \ldots i\}}\{(k-1)T_j\}$ sont ordonnançables ou non. Si aucune échéance n'est dépassée, jusqu'à la date t, alors la première étape de l'algorithme retourne l'index (noté h) de la première instance de τ_i qui n'est pas terminée à la t.

La tâche τ_i peut générer une nombre pseudo-polynomial d'instances dans la période d'activité synchrone de niveau i. *Toutefois, Fisher et Baruah indique que dans tout intervalle $(t_1, t_2]$ défini par deux points d'ordonnancement adjacents dans l'ensemble de test, seule une instance de τ_i peut avoir son échéance au sein de cet intervalle.* Ils définissent un moyen efficace de calculer l'index de cette première instance de τ_i. Pour cela, une approximation de la demande processeur est définie pour déterminer les instances ayant leur exécution terminée à la date t (Equation 5 dans [14]) :

$$\widetilde{W}_i(t) = \delta(\tau_i, t) + \sum_{j<i} \delta_j(t) \tag{5.7}$$

Fisher et Baruah utilise cette fonction pour déterminer l'index de la première instance de τ_i se

tel-00554265, version 1 - 10 Jan 2011

Algorithm 2: FBApproxSecondStage

input :
- $\tau = C[n], T[n], D[n]$: tableau d'entiers — /* Paramètres des tâches */
- i : entier — /* Index de la tâche analysée */
- k : entier — /* The FPTAS paramètre de précision */
- h : entier — /* Sortie de FBApproxFirstStage */

Soit $t_h = \frac{hC_i}{1-\sum_{j<i} U_j} + \frac{\sum_{j<i} C_i}{1-\sum_{j<i} U_j}$ — /* Step 0 */

if $t_h > (h-1)T_i + D_i$ **then** — /* Step 1: l'instance h dépasse son échéance */
  return τ_i *non faisable*;
end
if $\frac{C_i}{1-\sum_{j<i} U_j} > T_i$ **then** — /* Step 2:τ_i est non faisable */
  return τ_i *non faisable*;
end
return τ_i *est non faisable* — /* Step 3 */

terminant avant la date t :

$$\left\lceil \frac{t}{T_i} \right\rceil - Z_i(t) \tag{5.8}$$

où $Z_i(t)$ détermine le nombre d'instances $l \leq \left\lceil \frac{t}{T_i} \right\rceil$ telles que $\widetilde{W_i}(t) > t$, et est défini par :

$$Z_i(t) = \max\left(\left\lceil \frac{\widetilde{W_i}(t) - t}{C_i} \right\rceil, 0 \right) \tag{5.9}$$

En conséquence, dans tout intervalle constitué par deux points adjacents de l'ensemble $\tilde{S}_i$ des points d'ordonnancement, au plus une instance de τ_i se verra tester son ordonnançabilité. La première étape de l'algorithme, que nous venons de décrire, est présenté dans l'Algorithme 1.

Puisque la période d'activité synchrone de niveau i n'est pas nécessairement totalement explorée dans la première étape de l'algorithme (c-à-d., lorsque l'algorithme retourne la valeur lowest_active), le statut d'ordonnançabilité de la tâche peut être un problème ouvert. Pour traiter ce problème, une seconde étape est exécutée, en temps constant. Elle utilise une approximation linéaire (précisément, l'équation 5.3). La seconde étape prend en entrée l'index h retourné par la première étape de l'algorithme (c-à-d., la première instance non terminée à la date t). L'algorithme correspondant est présenté dans l'algorithme 2.

L'algorithme principal appelle tout d'abord la première étape. Si l'index h d'une instance de τ_i est retourné après l'exécution de la première étape, alors la seconde étape de l'algorithme est appelée pour tester en temps constant si l'instance $\tau_{i,h}$ est ordonnançable ou non. L'algorithme principal est présenté dans l'Algorithme 3.

tel-00554265, version 1 - 10 Jan 2011

Algorithm 3: FBApprox

input :
$\tau = (C[n], T[n], D[n])$: tableau d'entiers `/* Paramètres des tâches */`
ϵ : réel `/* Paramètre de précision` $0 < \epsilon < 1$ `*/`

$k = \lceil \frac{1}{\epsilon} \rceil - 1$;
foreach $t_a \in \tilde{S}_i - \{0\}$ **do**
  if *ApproxFirstStage*(τ, i, k, h) *retourne non faisable* **then**
    return τ_i *non faisable*;
  end
  if *FBApproxSecondStage*(τ, i, k, h) *retourne non faisable* **then**
    return
  end
  τ_i non faisable;
end
return τ *est faisable*

3.2. Analyse de l'algorithme de Fisher et Baruah

Nous montrons tout d'abord que l'algorithme de Fisher et Baruah n'est pas correct si une assignation quelconque des priorités est considérée. Nous montrerons ensuite que l'algorithme est valide si les tâches sont ordonnancées avec RM.

Problèmes dans le cas général. Nous montrons ici plusieurs problèmes dans l'algorithme présenté dans le paragraphe précédent. Premièrement, l'ensemble des points d'ordonnancement utilisé $\tilde{S}_i$ dans la première étape de l'algorithme suppose qu'une seule instance de tâche peut se terminer entre deux points successifs d'ordonnancement. Nous montrerons cela à travers un contre exemple. De la même façon, la fonction définissant le nombre d'instances se terminant à chaque point d'ordonnancement (c.f., Equation 6 dans [14]) doit être modifiée pour corriger les preuves de correction du test (c-à-d., le lemme 4 dans [14]).Finallement, l'étape 2 dans la seconde phase du test approché (ApproxSecondStage) n'est en pratique pas utile car le test logique correspondant est toujours faux et ainsi aucune tâche ne peut être détectée non ordonnançable à cette étape. Nous fournirons une preuve simple de cet invariant.

Nous pensons que les problèmes mentionnés ne peuvent pas être corrigés facilement sans modifier les formules présentées dans le paragraphe précédent et réviser les preuves les utilisant. Toutefois, ces modifications préserverons la structure à 2 phases de l'algorithme. Nous proposerons aussi d'améliorer l'algorithme en introduisant un test vérifiant si la période d'activité de niveau i est approximativement terminée ou non. Une telle modification diminuera les efforts de calcul durant le test approché de faisabilité d'une tâche.

Nous montrons tout d'abord que la première observation qui est la base de la première étape de l'algorithme approché proposé dans [14] est fausse : plusieurs instances de la tâche analysée peuvent se terminer entre deux points adjacents de l'ensemble de test.

Théorème 26 *L'observation " Pour tout couple de points d'ordonnancement adjacents* $t_1, t_2 \in \widehat{S}_i, |t_2 - t_1| \leq T_i$*, au plus une instance de* τ_i *peut voir son échéance survenir entre deux points adjacents de* $\widehat{S}_i$*"*

tel-00554265, version 1 - 10 Jan 2011

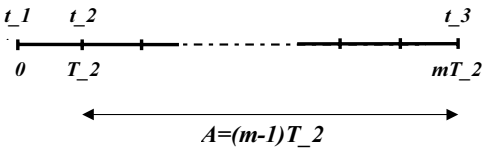

FIGURE 5.1 – L'intervalle entre deux points d'ordonnancement dans $\widehat{S_i}$ peut-être plus grand que T_i

est fausse.

Preuve : Considérons le contre exemple qui est illustré dans la Figure 3.2 :
$\tau_i = \{\tau_1, \tau_2\}, Prio(\tau_1) > Prio(\tau_2)$
$T_1 = mT_2$
$k = 2$ où k est le nombre d'étape avant de débuter l'approximation linéaire.
D'après l'Eq. (5.6) nous avons :

$$\begin{aligned} \widehat{S_2} &\stackrel{\text{def}}{=} \{t = bT_a \mid a = 1, \ldots, 2, b = 1, \ldots, 1\} \bigcup \{t = 0\} \\ &= \{t_1 = 0, t_2 = T_1, t_3 = T_2\} \end{aligned}$$

Considérons t_2, t_3 qui sont adjacents dans l'ensemble de test $\widehat{S_2}$, nous avons :
$|t_2 - t_3| = |T_1 - T_2| = |(m-1)T_2| = A$
En augmentant m, nous pouvons facilement obtenir $A > T_2$ ce qui contredit l'affirmation mentionnée. ■

Nous provons maintenant qu'avec la définition définie dans l' Eq. (5.6), le Lemme 4 défini dans [15] n'est pas correct.

Théorème 27 *L'expression "'Si* $l > \left\lceil \frac{t}{T_i} \right\rceil - Z_i(t)$ *et* $t \geq (l-1)T_i$*, alors* $\widehat{W_{i,l}}(t) > t$*"' n'est pas correcte.*

Preuve : Nous prouvons à l'aide d'un contre exemple satisfaisant les 3 conditions suivantes :

1. $t \in ((l-1)T_i, (l-1)T_i + D_i]$
2. $l = \left\lceil \frac{t}{T_i} \right\rceil - Z_i(t) + 1$
3. $\widehat{W_{i,l}}(t) \leq t$

Premièrement, nous choisissons arbitrairement un entier strictement plus grand que 2 comme étant le nombre d'étapes avant l'approximation linéaire (c-à-d., la valeur k).

Puis, nous choisissons un ensemble de tâches comme suit (chaque tâche est présentée avec la notation $\tau_i =< C_i, D_i, T_i >$) :
$\tau_i = \{\tau_1, \tau_2\}, Prio(\tau_1) > Prio(\tau_2)$
$\tau_2 =< (k+1)\alpha T_2, kT_2, T_2) >$ avec un nombre réel arbitraire α tel que $0 < \alpha < \frac{1}{k+2}$
$\tau_1 =< k(1-(k+2)\alpha)T_2, D_1, kT_2) >$
Puis, nous choisissons un instant $t = T_1 = kT_2$. Notons que $t > (k-1)T_2$ ($\Rightarrow \delta(\tau_2, t) = (t + T_2)U_2$) et que puisque $k > 2 \Rightarrow t < (k-1)kT_2 = (k-1)T_1$ ($\Rightarrow \delta(\tau_1, t) = \text{RBF}(\tau_1, t) = \left\lceil \frac{t}{T_1} \right\rceil C_1$).

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Finalement, nous étudions la tâche 2 (c-à-d., $i = 2$).
Nous prouvons la condition 1 :

$$\begin{aligned}
\widetilde{W_2}(t) &= \delta(\tau_2, t) + \sum_{j<2} \delta_j(t) \\
&= \delta(\tau_2, t) + \delta(\tau_1, t) \\
&= (t + T_2)U_2 + \left\lceil \frac{t}{T_1} \right\rceil C_1 \\
&= (kT_2 + T_2)(k+1)\alpha + k(1 - (k+2)\alpha)T_2 \\
&= T_2(k+1)^2\alpha + k(1 - (k+2)\alpha)T_2 \\
&= T_2(k + \alpha) \\
&= kT_2 + \alpha T_2 > kT_2 = t.
\end{aligned}$$

Ainsi $\widetilde{W_2}(t) > t \Rightarrow max\left(\left\lceil \frac{\widetilde{W_2}(t)-t}{C_2} \right\rceil, 0\right) = Z_2(t) \geq 1 \Rightarrow$ si nous choisissons $l = \left\lceil \frac{t}{T_2} \right\rceil - Z_2(t) + 1$ alors $l \leq \left\lceil \frac{t}{T_2} \right\rceil - 1 + 1 = \left\lceil \frac{t}{T_2} \right\rceil = \left\lceil \frac{kT_2}{T_2} \right\rceil = k \Rightarrow (l-1) < k \Rightarrow t = kT_2 > (l-1)T_2$ (1).
De plus, $t = kT_2 \leq (l-1)T_2 + kT_2 = (l-1)T_2 + D_2$ (2)
Depuis (1) et (2), nous avons satisfait la condition 1. La condition 2 est trivialement satisfaite par construction de l. Nous considérons maintenant la dernière condition :

$$\begin{aligned}
\widehat{W_{i,l}}(t) &= lC_2 + \sum_{j<2} \delta_j(t) \\
&= \left(\left\lceil \frac{t}{T_2} \right\rceil - \left\lceil \frac{\widetilde{W_2}(t) - t}{C_2} \right\rceil + 1 \right) C_2 + \delta(\tau_1, t) \\
&\leq \left(\left\lceil \frac{t}{T_2} \right\rceil - \frac{\widetilde{W_2}(t) - t}{C_2} + 1 \right) C_2 + \delta(\tau_1, t) \\
&= \left\lceil \frac{t}{T_2} \right\rceil C_2 + C_2 - \widetilde{W_2}(t) + \delta(\tau_1, t) + t \\
&= \left\lceil \frac{t}{T_2} \right\rceil C_2 + C_2 - (\delta(\tau_2, t) + \delta(\tau_1, t)) + \delta(\tau_1, t) + t \\
&= \left\lceil \frac{t}{T_2} \right\rceil C_2 + C_2 - \delta(\tau_2, t) + t = RHS
\end{aligned}$$

En remplaçant $t = kT_2$, nous avons :

$$\begin{aligned}
RHS &= (k+1)C_2 - (kT_2 + T_2)\frac{C_2}{T_2} + t \\
&= t
\end{aligned}$$

La condition 4 est prouvée. Nous avons ainsi défini un contre exemple du Lemme 4 défini dans [14]. ■

Maintenant, considérons l'étape 2 dans l'algorithme ApproxSecondStage qui vérifie : si $\frac{C_i}{1-\sum_{j<i} U_j} > T_i$ alors la tâche τ_i est conclue non ordonnançable. Nous montrons ci-après que l'inégalité $\frac{C_i}{1-\sum_{j<i} U_j} \leq T_i$ est toujours vraie (et que donc l'étape 2 de l'algorithme n'est pas utile en pratique). En utilisant le fait que $\sum_{j\leq i} U_j \leq 1$ puisque le facteur d'utilisation est toujours inférieur ou égal à 1 :

$$\begin{aligned} \frac{C_i}{T_i} &\leq (1 - \sum_{j<i} U_j) \\ \frac{C_i}{1-\sum_{j<i} U_j} &\leq T_i \end{aligned}$$

Correction sous RM**.** Nous prouvons maintenant que si on restreint l'utilisation de l'algorithme approché aux systèmes de tâches ordonnancés par l'algorithme Rate Monotonic (c-à-d., $\forall i \in [1,n] :: (T_1 \leq T_2 \leq \ldots \leq T_i)$) alors le Lemme 4 n'est pas valide dans sa présente forme. Toutefois, dans ce contexte, l'adjonction d'une condition addtionnelle et conforme aux hypothèse de l'algorithme ApproxFirstStage, permet d'obtenir une version du Lemme 4 correct.

Théorème 28 *L'expression "Si* $l > \left\lceil \frac{t}{T_i} \right\rceil - Z_i(t)$ *et* $t > (l-1)T_i$*, alors* $\widehat{W}_{i,l}(t) > t$ *" n'est pas correct.*

Preuve : Nous montrons ceci avec un contre exemple dont les 3 conditions suivantes sont satisfaites :

1. $t > (l-1)T_i$
2. $l = \left\lceil \frac{t}{T_i} \right\rceil - Z_i(t) + 1$
3. $\widehat{W}_{i,l}(t) \leq t$

Premièrement, nous choisissons arbitrairement un entier positif k (pour déterminer le nombre d'étape avant le début de l'approximation linéaire) et un entier positif m.

Alors, nous choisissons un système de tâche comme suit (les tâches sont définies à l'aide du format $\tau_i =< C_i, D_i, T_i >$) :

$\tau_i = \{\tau_1, \tau_2\}, Prio(\tau_1) > Prio(\tau_2)$ ainsi $T_1 \leq T_2$

$\tau_2 =< C_1, D_1, T >$

$\tau_1 =< C_2, D_2, T >$

où C_1, C_2, T sont choisis pour respecter :

$$\frac{k+m-1}{k+m} < \frac{C_1+C_2}{T} = U_1 + U_2 = U < 1 \tag{5.10}$$

Puis, nous choisissons un instant $t = (k-1+m)T$. Notons que $t > (k-1)T = (k-1)T_1 = (k-1)T_2$ ($\Rightarrow \delta(\tau_1, t) = (t+T_1)U_1$ et $\delta(\tau_2, t) = (t+T_2)U_2$).

Maintenant, nous choisissons d'analyser la seconde tâche (c-à-d., $i = 2$) et $l = \left\lceil \frac{t}{T_i} \right\rceil - Z_i(t) + 1 = \left\lceil \frac{t}{T_2} \right\rceil - Z_2(t) + 1$ (ainsi, par construction la seconde condition est respectée).

Nous prouvons maintenant la condition 1 :

tel-00554265, version 1 - 10 Jan 2011

$$\begin{aligned}
\widetilde{W}_2(t) &= \delta(\tau_2,t) + \sum_{j<2} \delta_j(t) \\
&= \delta(\tau_2,t) + \delta(\tau_1,t) \\
&= (t+T_2)U_2 + (t+T_1)U_1 \\
&= (t+T)U_2 + (t+T)U_1 \\
&= (t+T)(U_1+U_2) \\
&= (t+T)U \\
&= ((k-1+m)T+T)U \\
&= ((k+m)T)U \\
&> ((k+m)T)\frac{k+m-1}{k+m} \text{ depuis Eq. (5.10)} \\
&> (k+m-1)T = t.
\end{aligned}$$

Par définition, $Z_i(t) = \max\left\{\left\lceil \frac{\widetilde{W}_i(t)-t}{C_i}\right\rceil, 0\right\}$. Comme prouvé au-dessus, $\widetilde{W}_2(t) > t \Longrightarrow \left\lceil \frac{\widetilde{W}_2(t)-t}{C_2}\right\rceil \geq 1 \Longrightarrow Z_2(t) = \left\lceil \frac{\widetilde{W}_2(t)-t}{C_2}\right\rceil \geq 1$. Puisque $l = \left\lceil \frac{t}{T_2}\right\rceil - Z_2(t) + 1$, nous avons :

$$\begin{aligned}
l &\leq \left\lceil \frac{t}{T_2}\right\rceil \\
l &< \frac{t}{T_2} + 1 \\
l-1 &< \frac{t}{T_2} \\
(l-1)T_2 &< t.
\end{aligned}$$

La condition 1 est satisfaite. Nous considérons maintenant la dernière condition :

$$\begin{aligned}
\widehat{W}_{2,l}(t) &= lC_2 + \sum_{j<2} \delta_j(t) \\
&= \left(\left\lceil \frac{t}{T}\right\rceil - Z_2(t) + 1\right) C_2 + \delta(\tau_1,t) \\
&= \left(\left\lceil \frac{t}{T}\right\rceil - \left\lceil \frac{\widetilde{W}_2(t)-t}{C_2}\right\rceil + 1\right) C_2 + \delta(\tau_1,t) \\
&\leq \left(\left\lceil \frac{t}{T}\right\rceil - \frac{\widetilde{W}_2(t)-t}{C_2} + 1\right) C_2 + \delta(\tau_1,t) \\
&= \left\lceil \frac{t}{T}\right\rceil C_2 + C_2 - \widetilde{W}_2(t) + \delta(\tau_1,t) + t
\end{aligned}$$

tel-00554265, version 1 - 10 Jan 2011

$$
\begin{aligned}
&= \left\lceil \frac{t}{T} \right\rceil C_2 + C_2 - (\delta(\tau_2, t) + \delta(\tau_1, t)) + \delta(\tau_1, t) + t \\
&= \left\lceil \frac{t}{T} \right\rceil C_2 + C_2 - \delta(\tau_2, t) + t = RHS
\end{aligned}
$$

En remplaçant $t = (k-1+m)T$, nous avons :

$$
\begin{aligned}
RHS &= (k-1+m)C_2 + C_2 - ((k-1+m)T\frac{C_2}{T} + C_2) + t \\
&= t
\end{aligned}
$$

Ainsi, $W_{2,l}(t) = t$ respecte la condition 3 et contredit le Lemme 4 dans [14]. ■

Maintenant, nous prouvons qu'une légère modification du Lemme 4 qui prend en compte $t \leq \max_{j<i}\{(k-1)T_j\}$ permet de corriger ce lemme et rend ainsi l'algorithme d'approximation valide sous l'hypothèse qu'un ordonnanceur RM est utilisé.

Lemme 9 *Si* $l > \left\lceil \frac{t}{T_i} \right\rceil - Z_i(t)$, *if* $\exists t : (l-1)T_i < t \leq \max_{j<i}\{(k-1)T_j\}$ *alors* $\widehat{W}_{i,l}(t) > t$.

Preuve : Soit $a = \left\lceil \frac{t}{T_i} \right\rceil - Z_i(t) + 1 \Longrightarrow l \geq a, \widehat{W}_{i,l}(t) \geq \widehat{W}_{i,a}(t)$

Pour tout l, nous choisissons un instant t arbitrairement tel que $(l-1)T_i < t \leq \max_{j<i}\{(k-1)T_j\} \Longrightarrow (a-1)T_i < t \leq \max_{j<i}\{(k-1)T_j\}$.

Puisque $t > (a-1)T_i \Longleftrightarrow \frac{t}{T_i} > a-1 \Longleftrightarrow \left\lceil \frac{t}{T} \right\rceil \geq a$. D'après la définition de a, nous obtenons $Z_i(t) \geq 1$. En conséquence puisque $Z_i(t) = \max\left\{ \left\lceil \frac{\widetilde{W}_i(t)-t}{C_i} \right\rceil, 0 \right\}$, nous obtenons $Z_i(t) = \left\lceil \frac{\widetilde{W}_i(t)-t}{C_i} \right\rceil \geq 1$.

Nous avons :

tel-00554265, version 1 - 10 Jan 2011

$$
\begin{aligned}
\widehat{W}_{i,a}(t) &= aC_i + \sum_{j<i} \delta(\tau_j, t) \\
&= \left(\left\lceil \frac{t}{T_i} \right\rceil - Z_i(t) + 1 \right) C_i + \sum_{j<i} \delta(\tau_j, t) \\
&= \left(\left\lceil \frac{t}{T_i} \right\rceil - \left\lceil \frac{\widetilde{W}_i(t) - t}{C_i} \right\rceil + 1 \right) C_i + \sum_{j<i} \delta(\tau_j, t) \\
&> \left(\left\lceil \frac{t}{T_i} \right\rceil - \left(\frac{\widetilde{W}_i(t) - t}{C_i} + 1 \right) + 1 \right) C_i + \sum_{j<i} \delta(\tau_j, t) \text{ Notons que } C_i \neq 0 \forall i \\
&> \left\lceil \frac{t}{T_i} \right\rceil C_i - (\widetilde{W}_i(t) - t) + \sum_{j<i} \delta(\tau_j, t) \\
&> \left\lceil \frac{t}{T_i} \right\rceil C_i - \widetilde{W}_i(t) + \sum_{j<i} \delta(\tau_j, t) + t \\
&> \left\lceil \frac{t}{T_i} \right\rceil C_i - \left(\delta(\tau_i, t) + \sum_{j<i} \delta(\tau_j, t) \right) + \sum_{j<i} \delta(\tau_j, t) + t \\
&> \left\lceil \frac{t}{T_i} \right\rceil C_i - \delta(\tau_i, t) + t = RHS
\end{aligned}
$$

Sous RM nous avons $\forall i \in [1, n] :: (T_1 \leq T_2 \leq \ldots \leq T_i)$. En conséquence, $t \leq \max_{j<i} \{(k-1)T_j\} \Longrightarrow t \leq (k-1)T_i \Longrightarrow \delta(\tau_i, t) = \text{RBF}(\tau_i, t) = \left\lceil \frac{t}{T_i} \right\rceil$. En remplaçant ceci dans RHS nous avons $\widehat{W}_{i,a}(t) > t$. Le lemme est prouvé. ■

Ainsi, en utilisant la version modifiée du lemme 3 et sous l'hypothèse que RM ordonnance les tâches, alors l'algorithme ApproxFirstStage est correct.

4. Version révisée de l'algorithme de Fisher et Baruah

4.1. Résultats préliminaires

Fondé sur la fonction approchée $\widehat{W}_{i,l}(t)$, nous définissons pour chaque instance job $\tau_{i,l}$, un temps de réponse approché $\widehat{R}_{i,l}$ comme :

Définition 19 *Soit $\widehat{R}_{i,l}$ la première intersection entre $\widehat{W}_{i,l}(t)$ et $f(t) = t$, alors $\widehat{R}_{i,l}$ peut être défini par :*

$$\widehat{R}_{i,l} \stackrel{\text{def}}{=} min \left\{ t \mid \widehat{W}_{i,l}(t) = t \right\} \tag{5.11}$$

Puis, nous définissons $\hat{N}_i$ comme l'index de la dernière instance dans la période d'activité de niveau i qui sera considérée dans notre test de faisabilité :

tel-00554265, version 1 - 10 Jan 2011

Définition 20

$$\hat{N}_i \stackrel{\text{def}}{=} min\left\{l \in N \mid \widehat{R}_{i,l} \leq (l-1)T_i + T_i\right\} \tag{5.12}$$

Enfin, nous définissons pour une tâche donnée τ_i, l'ensemble des points d'ordonnancement qui seront testés par notre algorithme approché :

$$\widetilde{S}_i \stackrel{\text{def}}{=} \{t = bT_a \mid a = 1, \ldots, i-1; b = 1, \ldots, k-1\} \bigcup \{0\} \bigcup \{\infty\} \tag{5.13}$$

Deux éléments t_1 et t_2 ($t_1 < t_2$) dans cet ensemble seront *adjacents* s'il n'y a aucun élément t dans cet ensemble tel que $t_1 < t < t_2$. Nous définissons la notion d'*intervalle primitif* de l'ensemble comme l'intervalle borné entre deux points adjacents dans cet ensemble.

Afin de simplifier la terminologie, nous utiliserons dans toute la suite de ce chapitre le terme "l'instance $\tau_{i,l}$ est terminée" pour indiquer que "la fonction approchée de demande processeur $\widehat{W}_{i,l}(t)$ coupe la droite affine définissant la capacité du processeur", le terme "respecte son échéance" traduira que"$\widehat{R}_{i,l} \in [(l-1)T_i, (l-1)T_i + D_i]$ ", le terme "l'instant d'intersection" indiquera "un instant approché d'intersection", et ainsi de suite.

L'expression $\widehat{R}_{i,l}$ définie plus haut est le premier instant tel que $\widehat{W}_{i,l}(t) = t$ et ne représente pas nécessairement la date de fin de $\tau_{i,l}$. Toutefois, pour toute instance $\tau_{i,l}$ terminée dans la période d'activité de niveau i, son $\widehat{R}_{i,l}$ correspond à sa date de fin (c-à-d., le premier instant après le réveil tel que $W_{i,l}(t) = t$), comme le stipule le lemme suivant :

Lemme 10

$$\forall i \leq n, l \leq \hat{N}_i, \widehat{R}_{i,l} > (l-1)T_i$$

Tout d'abord, nous présentons la propriété suivante sur $\widehat{R}_{i,l}$:

Lemme 11 *$\forall i \leq n$, soient l, h des nombres naturels tels que $l < h$, alors $\widehat{R}_{i,l} < \widehat{R}_{i,h}$*

Preuve : Par définition $\widehat{W}_{i,h}(\widehat{R}_{i,h}) = \widehat{R}_{i,h}$. De plus, $l < h \Longrightarrow W_{i,l}(t) < W_{i,h}(t)$. Enfin :

$$\widehat{W}_{i,l}(\widehat{R}_{i,h}) < \widehat{R}_{i,h} \Longrightarrow \exists t^* < \widehat{R}_{i,h} :: (\widehat{W}_{i,l}(t^*) = t^*)$$

Puisque $\widehat{R}_{i,l} \stackrel{\text{def}}{=} \min\{t \mid W_{i,l}(t) = t\}$, alors $\widehat{R}_{i,l} \leq t^*$ et en conséquence $\widehat{R}_{i,l} < \widehat{R}_{i,h}$. ■

Nous prouvons maintenant le Lemme 10 :

Preuve : Si $\hat{N}_i = 1$ alors le lemme tient puisque le premier point d'intersection est nécessairement après 0. Si $\hat{N}_i > 1$ alors par la Définition 5.12, $\forall i \leq n, l \leq \hat{N}_i - 1, \widehat{R}_{i,l}\widehat{R}_{i,l} > (l-1)T_i + T_i > (l-1)T_i$. Depuis Lemme 11, nous savons que $\widehat{R}_{i,\hat{N}_i} > \widehat{R}_{i,Nh-1} = (\hat{N}_i - 1 - 1)T_i + T_i = (\hat{N}_i - 1)T_i$. Le lemme est prouvé. ■

Ainsi, nous avons une condition nécessaire et suffisante pour vérifier si une instance de tâche τ_i est approximativement faisable comme suit :

Corollaire 4

$$\forall l \leq \hat{N}_i :: (\widehat{R}_{i,l} \leq (l-1)T_i + D_i \Longleftrightarrow (\exists t \in ((l-1)T_i, \leq (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t))$$

tel-00554265, version 1 - 10 Jan 2011

Preuve : La direction "Seulement Si" est triviale depuis le Lemme 10 et la définition de $\widehat{R}_{i,l}$. Pour la direction "Si", nous choisissons un instant arbitraire $t^* \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t$. Comme $\widehat{W}_{i,l}(t)$ est une fonction en escalier non décroissante continue à gauche et que $\widehat{W}_{i,l}(0) = lC_i > 0$, alors $\exists t^{int} \leq t^* :: (\widehat{W}_{i,l}(t^{int}) = t^{int})$ Depuis la définition de $\widehat{R}_{i,l} \Longrightarrow \widehat{R}_{i,l} \leq t^{int} \leq t^* \Longrightarrow \widehat{R}_{i,l} \in ((l-1)T_i, (l-1)T_i + D_i]$. Le corollaire est prouvé. ■

4.2. Propriétés des intervalles primitifs

Nous pouvons utiliser $\widehat{R}_{i,l}$ pour vérifier à la volée que la période d'activité de niveau i n'est pas terminée. En utilisant les résultats établis dans le paragraphe précédent, nous proposons un algorithme qui teste la faisabilité de τ_i en analysant les instants d'intersection $\widehat{R}_{i,l}$ localisés dans un intervalle primitif de $\widetilde{S}_i$. Pour réduire la complexité de ce calcul, nous définissons quelques propriétés permettant de ne tester que la première instance dans chaque intervalle primitif. Précisément, la première instance de τ_i dans un intervalle primitif aura un temps de réponse plus grand que toute autre instance se terminant dans cet intervalle.

Tout d'abord, nous prouvons la propriété suivante sur toutes les instances de τ_i qui se terminent dans un même intervalle primitif de $\widetilde{S}_i$:

Lemme 12 *Pour tout $i \leq n$, l et h deux nombres naturels tels que $l < h$ et pour tout couple de points adjacents d'ordonnancement $t_1 < t_2$ dans $\widetilde{S}_i$ tels que $\widehat{R}_{i,l}, \widehat{R}_{i,h} \in (t_1, t_2]$ alors $\widehat{R}_{i,h} < \widehat{R}_{i,l} + (h-l)T_i$.*

Preuve : Soit $\mathbf{T}_{-t_1,t_2}$ l'ensemble de toutes les tâches τ_j tel que $j < i$ et $(k-1)T_j \leq t_1$. En d'autres termes, $\mathbf{T}_{-t_1,t_2}$ est l'ensemble de toutes les tâches dont la fonction RBF aprpochée devient linéaire après t_1. Notons que $\mathbf{T}_{-t_1,t_2} \subseteq 1..i-1$.

Nous pouvons reformuler $\widehat{W}_{i,l}(t)$ dans $(t_1, t_2]$ comme suit :

$$\begin{aligned}
\widehat{W}_{i,l}(t) &= lC_i + \sum_{j<i} \delta(\tau_j, t) \\
&= lC_i + \sum_{j<i, j \notin \mathbf{T}_{-t_1,t_2}} \text{RBF}(\tau_j, t) + \sum_{j \in \mathbf{T}_{-t_1,t_2}} (t + T_j)U_j \\
&= lC_i + \sum_{j<i, j \notin \mathbf{T}_{-t_1,t_2}} \text{RBF}(\tau_j, t) + \sum_{j \in \mathbf{T}_{-t_1,t_2}} T_j U_j + t \sum_{j \in \mathbf{T}_{-t_1,t_2}} U_j
\end{aligned}$$

$$\begin{aligned}
\text{Let } A &= lC_i + \sum_{j<i, j \notin \mathbf{T}_{-t_1,t_2}} \text{RBF}(\tau_j, t) + \sum_{j \in \mathbf{T}_{-t_1,t_2}} T_j U_j, \\
B &= \sum_{j \in \mathbf{T}_{-t_1,t_2}} U_j.
\end{aligned}$$

Nous avons alors :

$$\widehat{W}_{i,l}(t) = A + Bt$$

Notons que pour tout instant $t \in (t_1, t_2]$, A et B sont constants.

tel-00554265, version 1 - 10 Jan 2011

En utilisant la définition de $W_{i,l}(t)$, $\widehat{W}_{i,h}(t) = W_{i,l}(t) + (h-l)C_i \Longrightarrow$ nous obtenons :

$$\widehat{W}_{i,h}(t) = (h-l)C_i + A + Bt$$

Nous calculons maintenant $\widehat{R}_{i,l}, \widehat{R}_{i,h}$ avec les deux formules de $\widehat{W}_{i,l}(t)$ et $\widehat{W}_{i,h}(t)$.
Comme $\widehat{R}_{i,l}$ est l'intersection de $\widehat{W}_{i,l}(t)$ avec $y = t$ dans l'intervalle $(t_1, t_2]$

$$\Longrightarrow \widehat{R}_{i,l} = \frac{A}{1-B}$$

.

De façon similaire,

$$\widehat{R}_{i,h} = \frac{A + (h-l)C_i}{1-B}$$

Nous avons ainsi :

$$\begin{aligned}
\widehat{R}_{i,h} - \widehat{R}_{i,l} &= \frac{A + (h-l)C_i}{1-B} - \frac{A}{1-B} \\
&= (h-l)\frac{C_i}{1-B} \\
&= (h-l)T_i\frac{\frac{C_i}{T_i}}{1-B} \\
&= (h-l)T_i\frac{u_i}{1-B} = RHS
\end{aligned}$$

En reportant $B = \sum_{j \in \mathbf{T}_{-t_1,t_2}} U_j$, nous avons :

$$\begin{aligned}
1-B &= 1 - \sum_{j \in \mathbf{T}_{-t_1,t_2}} U_j \\
&\geq 1 - \sum_{j<i} U_j \\
&> \sum_{1 \leq j \leq n} U_j - \sum_{j<i} U_j \text{(puisque le facteur d'utilisation est strictement inférieur à 1)} \\
&> \sum_{i \leq j \leq n} U_j \\
&> U_i
\end{aligned}$$

Nous obtenons :

$$\frac{U_i}{1-B} < 1$$

$\Longrightarrow RHS < (h-l)T_i$. Le lemme est prouvé. ■

tel-00554265, version 1 - 10 Jan 2011

Grace au lemme précédent, nous obtenons une propriété fondamentale sur les temps de réponse des instances qui se terminent dans un même intervalle primitif : étant donné deux points adjacents t_1, t_2 de $\widetilde{S}_i$, en supposant que la période d'activité de niveau i n'est pas terminée avant t_1, alors le théorème suivant est vérifié :

1. Pour vérifier si toute instance est terminée dans $(t_1, t_2]$ respecte son échéance, il est suffisant de tester l'échéance de la première instance terminée dans $(t_1, t_2]$.
2. Pour vérifier qu'une instance ne se termine pas avant le réveil de l'instance suivante dans l'intervalle primitif, il est suffisant de tester la dernière instance terminée dans l'intervalle primitif. Cette propriété permet de tester si la période d'activité de niveau i est approximativement terminée ou non.

Théorème 29 *Pour tout $i \leq n, l, h \in N, l < h$ et tout couple de points adjacents $t_1 < t_2$ de $\widetilde{S}_i$ tel que $\widehat{R}_{i,l}, \widehat{R}_{i,h} \in (t_1, t_2]$ alors nous avons :*

$$\widehat{R}_{i,l} \leq (l-1)T_i + D_i \Longrightarrow \widehat{R}_{i,h} \leq (h-1)T_i + D_i, \tag{5.14a}$$

$$\widehat{R}_{i,h} > (h-1)T_i + T_i \Longrightarrow \widehat{R}_{i,l} > (l-1)T_i + T_i \tag{5.14b}$$

Preuve : Nous prouvons tout d'abord Eq. (5.14a) :

$$\begin{aligned} \widehat{R}_{i,l} &\leq (l-1)T_i + D_i \\ \widehat{R}_{i,h} - (h-l)T_i &\leq (l-1)T_i + D_i \text{ (pour } \widehat{R}_{i,h} - (h-l)T_i < \widehat{R}_{i,l} \text{ (depuis le Lemme 12)} \\ \widehat{R}_{i,h} &\leq (h-1)T_i + D_i \end{aligned}$$

Maintenant, nous prouvons l'Eq. (5.14b) :

$$\begin{aligned} \widehat{R}_{i,h} &> (h-1)T_i + T_i \\ \widehat{R}_{i,l} + (h-l)T_i &> (h-1)T_i + T_i \text{ (pour } \widehat{R}_{i,h} < \widehat{R}_{i,l} + (h-l)T_i \text{ depuis le Lemme 12)} \\ \widehat{R}_{i,l} &> (l-1)T_i + T_i \end{aligned}$$

Le théorème est prouvé. ■

Le résultat suivant est une conséquence directe du Théorème 29 pour la condition nécessaire (la condition suffisante est triviale) :

Corollaire 5 *$\forall t_1, t_2 \in \widetilde{S}_i$, Soient first et last respectivement définis comme :*

$$\begin{aligned} first &\stackrel{\text{def}}{=} \min\left\{l \in N \mid \widehat{R}_{i,l} \in (t_1, t_2]\right\} \\ last &\stackrel{\text{def}}{=} \max\left\{l \in N \mid \widehat{R}_{i,l} \in (t_1, t_2]\right\} \end{aligned}$$

alors les inégalités suivantes sont toujours respectées :

$$\widehat{R}_{i,first} \leq (first-1)T_i + D_i \Longleftrightarrow \forall l \in [first, last] : \widehat{R}_{i,l} \leq (l-1)T_i + D_i, \tag{5.15a}$$

$$\widehat{R}_{i,last} > (last-1)T_i + T_i \Longleftrightarrow \forall l \in [first, last] : \widehat{R}_{i,l} > (l-1)T_i + T_i \tag{5.15b}$$

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Du corollaire précédent, nous obtenons que : dans chaque intervalle primitif il peut y avoir un nombre pseudo-polynomial d'instances de la tâche analysée qui se terminent, mais sous l'hypothèse que la période d'activité de niveau i n'est pas terminée dans l'intervalle primitif précédent, alors il est suffisant de vérifier la première instance terminée dans l'intervalle pour tester si toutes les instances de cet intervalle respectent leur échéance. De plus, pour vérifier que la période d'activité de niveau i se termine dans un intervalle primitif, il est suffisant de tester la dernière instance de τ_i réveillée dans cet intervalle et de tester si celle-ci se termine avant le réveil de l'instance suivante. Si tel est le cas, il ne sera pas nécessaire de considérer les intervalles primitifs suivants (c-à-d., la période d'activité de niveau i a été pleinement explorée).

Enfin, nous proposons une méthode pour identifier simplement les index de la première instance et la dernière instance (c-à-d., first et last) dans un intervalle primitif de $\widetilde{S}_i$. Dans ce but, nous introduisons tout d'abord une fonction qui sera utilisée pour déterminer l'ensemble des instances ayant leurs demande d'exécution satisfaite à la date t,

$$\widehat{W}_i(t) \stackrel{\text{def}}{=} \text{RBF}(\tau_i, t) + \sum_{j<i} \delta(\tau_j, t) \tag{5.16}$$

$\widehat{W}_i(t)$ represente une approximation "proche" de la fonction de demande cumulée de la tâche τ_i et de toutes les tâches plus prioritaires pour une date *quelconque* donnée t. En comparaison , $\widehat{W}_{i,l}(t)$ est seulement une approximation "proche" lorsque t appartient à l'intervalle $((l-1)T_i, lT_i]$. Notons enfin, que cette fonction est une légère modification de la l'Equation 13 présentée dans [14].

Remarquons que le nombre d'instances *actives* à la date t est en $\mathcal{O}(D_i/T_i)$. La fonction suivante est utilisée pour identifier l'index de la dernière instance réveillée avant la date τ_i et ayant son exécution approximativement terminée à la date t.

$$I_i(t) \stackrel{\text{def}}{=} \left\lceil \frac{t}{T_i} \right\rceil - \left\lceil \frac{\widehat{W}_i(t) - t}{C_i} \right\rceil \tag{5.17}$$

$I_i(t)$ est une modification de la fonction $Z_i(t)$ définie dans [14]. Précisément, nous avons supprimé la fonction maximum pour prouver les deux lemmes suivants. Si la valeur de $I_i(t)$ n'est pas négative, alors elle sera utilisée pour déterminer l'index de la dernière instance ayant son exécution approximativement satisfaite à la date t. Puisqu'uniquement cette instance sera testée parmi toutes les instances terminées dans l'intervalle primitif, le test correspondant (pour un intervalle primitif donné) s'effectue en temps constant.

Lemme 13 *Si* $I_i(t) > 0$ *alors* $\forall l \in \{1, \ldots, I_i(t)\}, \widehat{W}_{i,l}(t) \leq t$

Preuve :

Soit $b = I_i(t) \Longrightarrow l \leq b, \widehat{W}_{i,l}(t) \leq \widehat{W}_{i,b}(t)$

Nous avons :

$$\begin{aligned}
\widehat{W}_{i,b}(t) &= bC_i + \sum_{j<i} \delta(\tau_j, t) \\
&= \left(\left\lceil \frac{t}{T_i} \right\rceil - \left\lceil \frac{\widehat{W}_i(t) - t}{C_i} \right\rceil \right) C_i + \sum_{j<i} \delta(\tau_j, t) \\
&= \left\lceil \frac{t}{T_i} \right\rceil C_i - \left\lceil \frac{\widehat{W}_i(t) - t}{C_i} \right\rceil C_i + \sum_{j<i} \delta(\tau_j, t) \\
&\leq \left\lceil \frac{t}{T_i} \right\rceil C_i - \frac{\widehat{W}_i(t) - t}{C_i} C_i + \sum_{j<i} \delta(\tau_j, t) \\
&\leq \left\lceil \frac{t}{T_i} \right\rceil C_i + \sum_{j<i} \delta(\tau_j, t) - \widehat{W}_i(t) + t \\
&\leq \widehat{W}_i(t) - \widehat{W}_i(t) + t = t
\end{aligned}$$

Le lemme est prouvé. ■

Lemme 14 *Si* , $t \in ((l-1)T_i, (l-1)T_i + D_i]$ *et* $l > I_i(t)$ *alors* $\widehat{W}_{i,l}(t) > t$

Preuve : Soit $a = I_i(t) + 1 \Longrightarrow l \geq a, \widehat{W}_{i,l}(t) \geq \widehat{W}_{i,a}(t)$

Nous avons :

$$\begin{aligned}
\widehat{W}_{i,a}(t) &= aC_i + \sum_{j<i} \delta(\tau_j, t) \\
&= \left(\left\lceil \frac{t}{T_i} \right\rceil - \left\lceil \frac{\widehat{W}_i(t) - t}{C_i} \right\rceil + 1 \right) C_i + \sum_{j<i} \delta(\tau_j, t) \\
&> \left(\left\lceil \frac{t}{T_i} \right\rceil - \left(\frac{\widehat{W}_i(t) - t}{C_i} + 1 \right) + 1 \right) C_i + \sum_{j<i} \delta(\tau_j, t) \text{ (Notons que } C_i > 0 \forall i) \\
&> \left\lceil \frac{t}{T_i} \right\rceil C_i - (\widehat{W}_i(t) - t) + \sum_{j<i} \delta(\tau_j, t) \\
&> \left\lceil \frac{t}{T_i} \right\rceil C_i + \sum_{j<i} \delta(\tau_j, t) - \widehat{W}_i(t) + t \\
&> \widehat{W}_i(t) - \widehat{W}_i(t) + t = t
\end{aligned}$$

Le lemme est prouvé. ■

Remarquons que la fonction $I_i(t)$ ne peut pas être appliquée pour $t = \infty$. Par ailleurs, pour le dernier intervalle primitif de $\widetilde{S}_i$, contenant le point d'ordonnancement ∞ en borne droite, il n'est pas nécessaire de vérifier la terminaison de la période d'activité de niveau i (puisque $U < 1$, cette période

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

d'activité ne peut pas être infinie). Durant le test, nous divisons les instances de τ_i en deux sous-ensembles : celles dont les instances se termine avant ou à la date $\max_{j\in 1,\ldots,i-1}\{(k-1)T_j\}$ et celles se terminant dans l'intervalle $(\max_{j\in 1,\ldots,i-1}\{(k-1)T_j\},\infty)$ (c-à-d., terminées dans le dernier intervalle de $\widetilde{S}_i$). Dans ce dernier cas, toutes les fonctions de demande processeur qui seront utilisées sont des fonctions strictement linéaires. Correspondant à ces deux sous-ensembles, l'algorithme se divise en deux étapes ou phases.

4.3. Première étape : instances terminées à la date $\max_{j\in 1,\ldots,i-1}\{(k-1)T_j\}$

Nous construisons un algorithme qui détermine, pour tout couple de points adjacents $t_1, t_2 \in \widetilde{S}_i \backslash \infty$, et qui se terminent dans l'intervalle $(t_1, t_2]$. Ainsi, nous testons en temps constants si toutes ces instances respectent ou non leurs échéances ainsi que la terminaison de la période d'activité de niveau i. Dans le Section 5, nous prouverons la correction de cet algorithme qui est présenté dans l'algorithme 4.

Nous résumons les principales différences entre la nouvelle version d'ApproxFirstStage et la version originelle présentée dans [14], désignée sous le nom FBApproxFirstStage :

- Les fonctions suivantes ont été modifiées : $Z_i(t)$ et $\tilde{W}_i(t)$,
- Dans FBApproxFirstStage, l'algorithme teste si *une instance a son échéance dans un intervalle primitif*. Mais, il ne test pas si cette instance se termine ou non dans cet intervalle. Dans la version révisée, la fonction $I_i(t)$ est calculée au début de chaque itération et comparée avec la valeur de la précédente intération (c-à-d., last_active value), alors que dans FBApproxFirstStage la fonction $Z_i(t)$, qui joue un rôle équivalent à $I_i(t)$ est appelé à la fin de l'itération. Ainsi, dans la version révisée nous testons si une *instance se termine dans l'intervalle primitif.*

4.4. Seconde phase : instances terminées après $\max_{j\in 1,\ldots,i-1}\{(k-1)T_j\}$

Dans ce paragraphe, nous décrivons comment tester en temps constant la faisabilité des instances de τ_i qui ne sont pas terminées avant la date $\max_{j\in 1,\ldots,i-1}\{(k-1)T_j\}$ selon notre schéma d'approximation. Notons que depuis la définition de $\widehat{W}_{i,l}(t)$:

$$\forall l \in N, \forall t \in \left(\max_{j\in 1,\ldots,i-1}\{(k-1)T_j\},\infty\right) :: \left(\widehat{W}_{i,l}(t) = lC_i + \sum_{j<i}(t+T_j)U_j\right) \tag{5.18}$$

Supposons que l'algorithme ApproxFirstStage (τ, i, k) retourne l'index h comme étant la dernière instance de τ_i qui n'a pas son exécution satisfaite à la date $\max_{j\in 1,\ldots,i-1}\{(k-1)T_j\}$. Alors pour tout $l \in N, (l \geq h)$, nous pouvons résoudre l'Eq. (5.18) pour trouver le point $\widehat{R}_{i,l}$ auquel $\widehat{W}_{i,l}(t) = t$

$$\widehat{R}_{i,l} \stackrel{\text{def}}{=} \frac{lC_i}{1-\sum_{j<i}U_j} + \frac{\sum_{j<i}C_j}{1-\sum_{j<i}U_j} \tag{5.19}$$

Comme ApproxFirstStage (τ, i, k) retourne h, il n'existe pas d'itération dans la boucle principale d'ApproxFirstStage dans laquelle $last \leq (l-1)T_i + T_i$. Par le Corollaire 5, ceci implique que la période d'activité de niveau i ne s'est pas terminée durant la première phase d'analyse. De façon équivalente, $h \leq \hat{N}_i$ and $\forall l \in [h, \hat{N}_i], t = \widehat{R}_{i,l}$ est actuellement l'instant auquel les instances $\tau_{i,l}$ ont approximativement leurs exécutions terminées.

tel-00554265, version 1 - 10 Jan 2011

En conséquence, depuis le Théorème 29 et depuis le fait que $(\max_{j\in 1,\ldots,i}\{(k-1)T_j\},\infty)$ est aussi un intervalle primitif de $\widetilde{S}_i \Longrightarrow$ pour tester la faisabilité de toutes les instances $\tau_{i,l}$ qui se terminent après $\max_{j\in 1,\ldots,i}\{(k-1)T_j\}$ et sont dans la première période d'activité de niveau i, nous devons seulement déterminer l'instant d'intersection de la première instance, c'est-à-dire $\widehat{R}_{i,h}$. L'algorithme révisé de ApproxSecondStage est présenté Algorithme 5. Cette phase est quasi-identique à l'algorithme originel présenté par Fisher et Baruah (l'unique différence est la suppression de l'étape 2, qui a été démontrée inutile en pratique).

4.5. Exemple détaillé

Nous illustrons le fonctionnement du schéma d'approximation proposé dans ce chapitre sur l'ensemble des tâches présenté dans la Table 5.1 tiré de [19]. En utilisant Deadline Monotonic comme algorithme d'ordonnancement, nous assignons à τ_1 la plus forte priorité. Nous analysons maintenant sa faisabilité en utilisant la méthode exacte et la méthode approchée. Les pires temps de réponse exacts sont présentés dans la Table 5.2. Dans cette analyse exacte, 7 instances de tâches sont testées dans la première période d'activité de niveau i : pour $\tau_{2,7}$ le temps de réponse est $R_{2,7} = 94 < 100 = T_2$. Puisque toutes les instances respectent leur échéance (c-à-d., $\widehat{R}_{i,l} \leq (l-1)T_2 + D_2$) alors τ_2 est conclue faisable.

Nous analysons maintenant la faisabilité de τ_2 en utilisant le schéma d'approximation. Nous choisissons $\epsilon = 0.25$ soit $k = 3$ et est alors $\widetilde{S}_i = \{0, 70, 140, \infty\}$. Les deux phases du schéma d'approximation seront exécutées.

Dans la première phase, nous considérons les instances terminées avant ou à la date $t = 140$ à l'aide d'ApproxFirstStage, les résultats sont présentés dans la Table 5.4 :

1. Avant la première itération de la boucle dans ApproxFirstStage, $last\_active = 0$.
2. Dans la première itération, nous considérons l'intervalle primitif $(0, 70], t_a = t_1 = 70$. Puisque $I_2(70) = \lceil \frac{70}{100} \rceil - \left\lceil \frac{\widehat{W}_2(70)-70}{62} \right\rceil = 0 = last\_active$ nous sautons à la prochaine itération. $last\_active$ reste à 0.
3. Nous considérons maintenant l'intervalle primitif $(70, 140], t_a = t_2 = 140$. Puisque $I_2(140) = \lceil \frac{140}{100} \rceil - \left\lceil \frac{\widehat{W}_2(140)-140}{62} \right\rceil = 1 > last\_active$ et les affectations suivantes sont effectuées : $first = last\_active + 1 = 0 + 1 = 1, next = I_2(140) = 1$.
   En calculant l'instant d'intersection dans l'intervalle primitif $(70, 140]$ auquel $f(t) = t$ coupe $\widehat{W}_{2,first}(t)$, nous obtenons le point d'intersection $t' = \widehat{R}_{2,first} = \widehat{R}_{2,1} = 114 < (1-1).100+140 \Longrightarrow$ Il n'existe dont pas d'instance terminée dans cet intervalle qui ne respecte son échéance.
   De même dans l'intervalle $(70, 140]$, l'instant auquel $f(t) = t$ coupe $\widehat{W}_{2,last}(t)$ permet de déterminer le point d'intersection $t'' = \widehat{R}_{2,last} = \widehat{R}_{2,1} = 114 > (1-1).100+100 \Longrightarrow$ Ainsi, la première période d'activité de niveau i n'est pas terminée dans cette intervalle.
   En conséquence $last\_active = last = 1$ et le test continue.
4. Puisqu'il n'y a plus de t_a à tester, l'algorithme retourne $h = last\_active + 1 = 1 + 1 = 2$.

Nous considérons maintenant les instances terminées après $t = 140$ avec la seconde phase de l'algorithme, ApproxSecondStage. Les réstultats sont présentés dans la Table 5.5. La valeur h utilisée dans cet algorithme est celle retournée par ApproxFirstStage. Donc $h = 2 \Longrightarrow \widehat{R}_{i,h} = \widehat{R}_{2,2} = 238, (63) <$

$(2-1).100+140$. Ainsi, nous concluons que τ_2 *est faisable*. Ce résultat respecte le Théorème 31 et le test approchés conclut que τ_2 est faisable sur un processeur de capacité unitaire.

Dans le prochain paragraphe, nous prouvons la correction de ce schéma d'approximation.

5. Correction du schéma d'approximation

Dans ce paragraphe, nous prouvons la correction de l'algorithme Approx. Le but est ici de démontrer :

Si Approx(τ, ϵ) retourne "faisable" alors l'ensemble de tâche s'ordonnance fiablement sur un processeur de capacité unitaire. Si Approx(τ, ϵ) retourne "non faisable", alors l'ensemble de tâches n'est pas ordonnançable sur un *processeur plus lent*, dont la capacité est $(1-\epsilon)$ fois la capacité du processeur pour lequel l'ensemble de tâches a été spécifié.

Le paragraphe est organisé comme suit : nous mettons tout d'abord en évidence un invariant dans l'algorithme ApproxFirstStage dans le Lemme 15. Cet invariant quantifie l'ensemble des instances de la tâche τ_i qui ont été conclues faisables par le schéma d'approximation, et ceci avant chaque intération de la boucle ApproxFirstStage. En utilisant cet invariant, nous analysons les sorties d'ApproxFirstStage et d'ApproxSecondStage respectivement dans le Lemme 16 et le Lemme 17. Nous montrons dans le Théorème 30 qu'Approx retourne "τ est faisable" si, et seulement si, les instances de la tâche analysée respectent leurs échéances respectives. Dans ce but, nous prouvons le Théorème 25.

Soient $last\_active_a, first_a, last_a$, et respectivement $last_a\_active, first, last$ les valeurs après a itérations de la boucle principale ($forloop$) de l'algorithme ApproxFirstStage. Le lemme suivant identifie l'ensemble des instances de τ_i qui sont garanties d'être faisables après a itérations. Précisémment, $last\_active_a$ premières instances de τ_i ont terminé leurs exécutions avant ou à la date t_a et elles respectent leurs échéances respectives. Toutes les instances suivantes ne se sont pas terminées avant la date t_a et ainsi le prochain intervalle primitif devra être testé (c-à-d., la période d'activité de niveau i n'est pas terminée).

Lemme 15 *Pour tout $a \geq 0$, après a itérations et avant l'itération $a+1$ de for...loop d'ApproxFirstStage, si l'algorithme n'a pas encore retourné soit τ_i est non faisable, soit τ_i est faisable alors les conditions suivantes sont vérifiées :*

$$\forall l \in \{1, \ldots, last\_active_a\} :: (\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t) \tag{5.20a}$$

$$\forall l > last\_active_a, \forall t \leq t_a :: (\widehat{W}_{i,l}(t) > t) \tag{5.20b}$$

Preuve :

Base : $last\_active_a = last\_active_0 = 0$ (valeur initiale avant la boucle). L'Eq. (5.20a) est trivialement satisfaite.

Comme $t_a = t_0 = 0$, la seconde condition est elle aussi satisfaite.

Induction : Par construction, si l'algorithme n'a retourné ni τ_i *n'est pas faisable*, ni τ_i *est faisable*, alors nous avons $last\_active_a = max(last\_active_{a-1}, I_i(t_a))$, donc :

1.

   a) $\forall l > last\_active_{a_1}, \forall t \leq t_{a-1} :: \widehat{W}_{i,l}(t) > t$ (hypothèse d'induction)$\Longrightarrow \forall l > last\_active_a, \forall t \leq t_{a-1} :: \widehat{\widehat{W}}_{i,l}(t) > t$.

tel-00554265, version 1 - 10 Jan 2011

b) $\forall l > I_i(t_a), \widehat{W}_{i,l}(t_a) > t_a$ (Depuis Lemme 14) $\Longrightarrow \forall l > last\_active_a, \widehat{W}_{i,l}(t_a) > t_a$.

Depuis (a), (b) et le fait que $\widehat{W}_{i,l}(t)$ est une fonction en escalier non-décroissante continue à gauche, l'Eq. (5.20b) de l'hypothèse d'induction est prouvée.

2. Si $last\_active$ ne change pas (c-à-d., $\max(last\_active_{a-1}, I_i(t_a)) = last\_active_{a-1}$) alors par l'hypothèse de recurrence la première condition est vérifiée. Si $last\_active$ augmente à $I_i(t_a)$, l'hypothèse de recurrence couvre aussi l'Eq. (5.20a) pour les instances dont les index appartiennent à $[1, \ldots, last\_active_{a-1}]$. Nous avons pour terminer à tester la condition pour les instances $l \in [last\_active_{a-1} + 1, I_i(t_a)]$ (en d'autres termes, pour $l \in [first_a, last_a]$).

   Depuis le Lemme 13, $\forall l \in [last\_active_{a-1} + 1, I_i(t_a)], \widehat{W}_{i,l}(t)(t_a) \leq t_a$. De plus, en utilsant l'hypothèse de recurrenceEq. (5.20b), $\forall l \in [last\_active_{a-1} + 1, I_i(t_a)], \forall t \leq t_{a-1} :: \widehat{W}_{i,l}(t) > t$. Puisque $\widehat{W}_{i,l}(t)$ est une fonction en escalier non décroissante et continue à gauche, nous obtenons $\forall l \in [last\_active_{a-1}+1, I_i(t_a)]$, $\widehat{R}_{i,l}$ (c-à-d., le premier instant satisfaisant l'équation $\widehat{W}_{i,l}(t) = t$) appartient à l'intervalle $(t_{a-1}, t_a]$.

   De plus, soit y la demande totale des intances de tâches plus prioritaires que τ_i réveillées à la date t_{a-1}, alors pour $\forall l \in [last\_active_{a-1} + 1, I_i(t_a)]$, $\widehat{R}_{i,l}$ est aussi l'intersection entre deux droites : le segment de droite reliant $(t_{a-1}, \widehat{W}_{i,l}(t_{a-1}) + y)$ à $(t_a, \widehat{W}_{i,l}(t_a))$ et la droite affine $f(t) = t$.

   En conséquence, par construction d'ApproxFirstStage, nous avons $\widehat{R}_{i,last\_active_{a-1}+1} = \widehat{R}_{i,first} = t'$ and $\widehat{R}_{i,I_i(t_a)} = \widehat{R}_{i,last} = t''$.

   a) L'algorithme n'a pas encore retourné τ_i *n'est pas faisable* $\Longrightarrow \widehat{R}_{i,first} = t' \leq (first - 1)T_i + D_i \Longrightarrow \forall l \in [first, last] :: \widehat{R}_{i,l} \leq (l-1)T_i + D_i$ (depuis Corollaire 5).

   b) L'algorithme n'a pas encore retourné τ_i *est faisable* $\Longrightarrow \widehat{R}_{i,last} = t'' > (last - 1)T_i + T_i \Longrightarrow \forall l \in [first, last] :: \widehat{R}_{i,l} > (l-1)T_i + T_i$ (depuis Corollaire 5).

   Depuis les cas (a) et (b), nous obtenons $\forall l \in [first, last] :: ((l-1)T_i + T_i < \widehat{R}_{i,l} \leq (l-1)T_i + D_i) \Longrightarrow \forall l \in [first, last] :: (\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t)$. L'hypothèse de récurrence est prouvée.

■

Pour montrer que l'algorithme Approx identifie correctement les tâches faisables, nous analysons les sorties respectives des deux phases (ApproxFirstStage et ApproxSecondStage). Nous montrons dans le prochain lemme :

1. les instances $1..h$ respectent leurs échéances,
2. une instance de τ_i ne respecte pas son échéance,
3. toutes les instances respectent leur échéance dans la période d'activité de niveau i.

Lemme 16 *Après a itérations, ApproxFirstStage retourne :*

1. $h \Longleftrightarrow \forall l < h :: (\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t)$
2. τ_i *n'est pas faisable* $\Longleftrightarrow \exists l :: (\forall t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) > t)$
3. τ_i *est faisable* $\Longleftrightarrow \forall l \leq \hat{N}_i :: (\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t)$

tel-00554265, version 1 - 10 Jan 2011

Preuve :

1. Trivial depuis le Lemme 15 et comme $h = last\_active_a + 1$.
2. τ_i *n'est pas faisable* $\Longrightarrow t' = \widehat{R}_{i,first_a} > (l-1)T_i + D_i$. Comme ApproxFirstStage ne doit pas avoir retourné τ_i *est faisable* avant $\Longrightarrow first_a \leq \hat{N}_i$. Nous pouvons alors appliquer le Corollaire 4 et le résutlat est prouvé.
3. τ_i *est faisable* $\Longrightarrow$ nous avons :
   (a) ApproxFirstStage ne doit pas avoir retourné τ_i *est faisable* avant $\Longrightarrow t' = \widehat{R}_{i,first_a} \leq (l-1)T_i + D_i \Longrightarrow \in [first_a, last_a]\widehat{R}_{i,l} \leq (l-1)T_i + D_i$
   (b) Comme ApproxFirstStage ne doit pas avoir retourné τ_i *est faisable* avant $\Longrightarrow first_a \leq \hat{N}_i$.
   (c) $\widehat{R}_{i,last_a} < (l-1)T_i + D_i \Longrightarrow \hat{N}_i \leq last_a$

   En conséquence, $\forall l \in [first_a, \hat{N}_i] :: (\widehat{R}_{i,l} \leq (l-1)T_i + D_i) \Longrightarrow \forall l \in [first_a, \hat{N}_i] :: (\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t)$. De plus, comme $first_a = last\_active_{a-1} + 1$, depuis le Lemme 15 nous avons $\forall < first_a :: (\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t)$. La dernière assertion est donc prouvée.

■

Lemme 17 *ApproxSecondStage retourne :*

1. τ_i *n'est pas faisable* $\Longleftrightarrow \exists l :: (\forall t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) > t)$
2. τ_i *est faisable* $\Longleftrightarrow \forall l \leq \hat{N}_i :: (\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t)$

Preuve :

1. trivial depuis la construction d'ApproxSecondStage et le Corollaire 4.
2. ApproxSecondStage retourne τ_i *est faisable* $\Longrightarrow \widehat{R}_{i,h} \leq (h-1)T_i + D_i$. Comme nous avons rasionné avant durant la construction d'ApproxSecondStage, et depuis le Corollaire 4 ceci conduit à $\forall l \in [h, \hat{N}_i) :: ((\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t)$. De plus, comme ApproxSecondStage est appelé, ApproxFirstStage doit avoir retourné une valeur h et ainsi $\forall l \in [1, h-1) :: ((\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t)$. Le résultat est prouvé.

■

Depuis les deux lemmes ci-dessus, nous pouvons maintenant prouver qu'Approx(τ, i, k) retournera"τ_i est faisable" si, et seulement si, pour chaque tâche τ_i de τ et pour toutes les instances l de τ_i, il existe un instant t entre le réveil de l'instance l et son échéance tel que $\widehat{W}_{i,l}(t) \leq t$. Le théorème suivant prouve ce résutlat.

Théorème 30 *Approx (τ, i, k) retourne τ_i est faisable si, et seulement si :*

$$\forall \tau_i \in \tau, l \in [1, \hat{N}_i](l < 0) :: (\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t). \tag{5.21}$$

tel-00554265, version 1 - 10 Jan 2011

Preuve : Nous prouvons tout d'abord la direction "Seulement si". Premièrement, nous supposons qu'Approx (τ, k) retourne "τ est faisable" $\Longleftrightarrow$ Soit ApproxFirstStage ou ApproxSecondStage doit avoir retourné "τ est faisable", on utilise alors soit le Lemme 16 ou le Lemme 17 respectivement pour terminer la preuve.

Pour la direction "Si", nous montrons la propriété contraposée. En d'autres mots, nous supposons qu'Approx (τ, k) retourne "τ n'est pas faisable". Une nouvelle fois, nous devons considérer deux cas :

1. ApproxFirstStage retourne "τ_i n'est pas faisable"
2. ApproxSecondStage retourne "τ_i n'est pas faisable"

Dans les deux cas, nous utilisons respectivement le Lemme 16 ou le Lemme 17 pour prouver $\exists l \in [1, \hat{N}_i] :: (\forall t \in ((l-1)T_i, (l-1)T_i + D], \widehat{W}_{i,l}(t) > t$. Ainsi, nous obtenons la négation de l'Eq. (5.21). ■

Nous rappelons maintenant le résultat suivant tiré de [14] :

Lemme 18 ([14]) *Pour toute tâche τ_i et pour tout instant t :*

$$\forall t, i, W_{i,1}(t) \leq \widehat{W}_{i,1}(t)_i(t) \leq \frac{k+1}{k} W_{i,1}(t)$$

Nous étendons ce résultat pour tout $\widehat{W}_{i,l}(t)$ comme suit :

Lemme 19 *Pour toute tâche τ_i, tout nombre naturel l et tout instant t :*

$$\forall t, i, l, \qquad W_{i,l}(t) \leq \widehat{W}_{i,l}(t) \leq \frac{k+1}{k} W_{i,l}(t)$$

où $k = \lceil \frac{1}{\epsilon} \rceil - 1$ est le nombre d'étapes avant de débuter l'approximation linéaire le la fonction demande processeur (c-à-d., la fonction RBF*).*

Preuve : Dans [14] est prouvé l'inégalité :

$$\text{RBF}(\tau_i, t) \leq \delta(\tau_i, t) \leq \frac{k+1}{k} \text{RBF}(\tau_i, t)$$

Ainsi, nous avons :

$$\begin{aligned}
\sum_{j<i} \text{RBF}(\tau_j, t) \leq \sum_{j<i} \delta(\tau_j, t) &\leq \frac{k+1}{k} \sum_{j<i} \text{RBF}(\tau_j, t) \\
lC_i + \sum_{j<i} \text{RBF}(\tau_j, t) \leq lC_i + \sum_{j<i} \delta(\tau_j, t) &\leq lC_i + \sum_{j<i} \frac{k+1}{k} \text{RBF}(\tau_j, t) \\
lC_i + \sum_{j<i} \text{RBF}(\tau_j, t) \leq lC_i + \sum_{j<i} \delta(\tau_j, t) &\leq \frac{k+1}{k} lC_i + \frac{k+1}{k} \sum_{j<i} \text{RBF}(\tau_j, t) \\
lC_i + \sum_{j<i} \text{RBF}(\tau_j, t) \leq lC_i + \sum_{j<i} \delta(\tau_j, t) &\leq \frac{k+1}{k} \left(lC_i + \sum_{j<i} \text{RBF}(\tau_j, t) \right) \\
W_{i,l}(t) \leq \widehat{W}_{i,l}(t) &\leq \frac{k+1}{k} W_{i,l}(t)
\end{aligned}$$

tel-00554265, version 1 - 10 Jan 2011

Le lemme est prouvé. ■

Nous pouvons maintenant prouver qu'Approx déclare 'τ est faisable", alors τ is faisable sur un processeur de capacité unitaire :

Théorème 31 *Un ensemble de tâches sporadiques, τ est faisable si Approx (τ, ϵ) retourne "τ est faisable" (où $0 < \epsilon < 1$).*

Preuve :

Par définition de N_i in Théorème 25 $\Longrightarrow \forall l < N_i :: (\forall t \in ((l-1)T_i, (l-1)T_i + D_i] : W_{i,l}(t) > t)$. Par le Lemme 19 $\Longrightarrow \forall l < N_i :: (\forall t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) > t)$. En conséquence, nous avons $N_i \leq \hat{N}_i$. De plus, si Approx (τ, ϵ) retourne 'τ est faisable" alors par le Théorème 30,

$$\begin{aligned}
& \forall \tau_i \in \tau, l \in [1, \hat{N}_i] :: (\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t) \\
\Longrightarrow \quad & \forall \tau_i \in \tau, l \in [1, N_i] :: (\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) \leq t) \\
\Longrightarrow \quad & \forall \tau_i \in \tau, l \in [1, N_i] :: (\exists t \in ((l-1)T_i, (l-1)T_i + D_i] : W_{i,l}(t) \leq t) \text{ by Lemme 19}
\end{aligned}$$

Le théorème suit en appliquant le Théorème 25. ■

Dans le théorème suivant, nous étudions le cas Approx retourne "τ n'est pas faisable".

Théorème 32 *Pour une système de tâches sporadiques τ and $\epsilon \in (0, 1)$, Si Approx (τ, ϵ) retourne "τ n'est pas faisable" alors τ n'est pas faisable sur un processeur de capacité $(1 - \epsilon)$.*

Preuve : Supposons qu'Approx (τ, k) retourne "τ n'est pas faisable", alors par le Théorème 30,

$$\begin{aligned}
& \exists \tau_i \in \tau, l \in [1, \hat{N}_i] :: (\forall t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) > t) \\
\Longrightarrow \quad & \exists \tau_i \in \tau, l \in N :: (\forall t \in ((l-1)T_i, (l-1)T_i + D_i] : \widehat{W}_{i,l}(t) > t) \\
\Longrightarrow \quad & \exists \tau_i \in \tau, l \in N :: (\forall t \in ((l-1)T_i, (l-1)T_i + D_i] : W_{i,l}(t) > \tfrac{k}{k+1} t \text{ par le Lemme 19}
\end{aligned}$$

Nous avons $\frac{k}{k+1} = 1 - \frac{1}{k+1}$. Notons que $k \stackrel{\text{def}}{=} \lceil \frac{1}{\epsilon} \rceil - 1$ et $\lceil \frac{1}{\epsilon} \rceil \geq \frac{1}{\epsilon}$, nous obtenons $\frac{k}{k+1} \geq 1 - \epsilon$.

Alors τ_i est non ordonnançable sur un processeur dont la capacité est $(1 - \epsilon)$. ■

Il est simple de vérifier que la complexité de l'algorithme révisé est la même que l'algorithme originel de Fisher et Baruah, présenté dans [15] (c-à-d., $\mathcal{O}(n^2 k)$). Toutefois, notons que notre ensemble de points d'ordonnancement est strictement plus petit que celui utilisé dans FBApprox et de plus nous avons complété ApproxFirstStage par un test qui détecte la terminaison de la période d'activité de niveau i (si tel est le cas l'algorithme ApproxFirstStage s'arrête, et l'algorithme ApproxSecondStage sera exécuté).

6. Conclusions

Dans ce chapître, nous avons revisité l'algorithme de Fisher et Baruah présenté dans [14] pour analyser l'ordonnançabilité des systèmes de tâches à priorités fixes et à échéances arbitraires. Nous avons prouvé à l'aide de contre-exemples que ce test n'est pas correct dans le cas général.

tel-00554265, version 1 - 10 Jan 2011

Pour régler ces problèmes, nous avons tout d'abord montré le contexte restrictif dans lequel le test peut être applicable (assignation des priorités à l'aide de l'algorithme d'ordonnancement Rate Monotonic). Afin de pouvoir étendre le test au cas d'assignation de priorité fixe quelconque, nous avons redéfini des fonctions d'approximation de la demande processeur, corrigé les preuves en conséquence, éliminé une étape rédondante dans la seconde partie du test ainsi qu'introduit de nouveaux résultats pour obtenir un test d'ordonnançabilité plus efficace. Pour ce dernier point, nous avons modifié la structure de l'algorithme afin de déterminer la terminaison de la periode d'activité. L'ensemble de ces modifications conduit un test approché avec une erreur de faisabilité quantifiable et d'une compléxité polynomiale (c-à-d., FPTAS). Les contributions de ce chapitre seront présentées à la conférence RTNS en novembre 2010 [29].

Ce nouveau schéma d'approximation peut être facilement étendu pour calculer des bornes supérieures des pires temps de réponse des tâches. Nous lèveraiont la limitation du calcul de bornes de temps de réponse seulement garanti pour les tâches statuées faisables par le test d'ordonnançabilité. Nous seraiont donc ainsi en mesure de calculer des bornes supérieures des pires temps de réponse pour les tâches ordonnançable et non ordonnançable, avec la même garantie de performance (avec la technique d'augmentation de ressources comme nous l'avons effectué dans le chapitre précédent). La mise en oeuvre et les expérimentations numériques de cette méthode seront traitées comme perspectives de recherche.

tel-00554265, version 1 - 10 Jan 2011

Algorithm 4: RevisedApproxFirstStage

input :
$\tau = C[n], T[n], D[n]$: tableau d'entiers /* Paramètres des tâches */
i : entier /* Index de la tâche analysée */
k : entier /* The FPTAS Paramètre de précision */
output: lowest_active : entier /* dernière instance analysée */

Construire $\tilde{S}_i$ en utilisant l'équation 5.13 /* Ensemble ordonné des points d'ordonnancement */
last_active=0 ;
foreach $t_a \in \tilde{S}_i - \{0\} - \{\infty\}$ **do**
  last=$I_i(t_a)$;
  if *last*>*last_ active* **then** /* une instance se termine dans $(t_{a-1}, t_a]$ */
    first=last_active+1;
    /* Vérifier l'échéance de la première instance terminée dans l'intervalle primitif */
    Soit t_{a-1} l'élément avant t_a dans $\tilde{S}_i$;
    Soit y le demande d'exécution totale des tâches plus prioritaires que τ_i se réveillant à la date t_{a-1};
    $a = (t_{a-1}, \widehat{W}_{i,\text{first}}(t_{a-1}) + y)$;
    $b = (t_a, \widehat{W}_{i,\text{first}}(t_a))$;
    Déterminer si la droite (a, b) coupe la droite affine $f(t) = t$;
    Soit t' le point d'intersection;
    if $t' > (first - 1)T_i + D_i$ **then return** *"non faisable"*;
    /* Vérifier si la période d'activité de niveau i est terminée dans l'intervalle primitif (en testant la dernière instance de τ_i) */
    $c = (t_{a-1}, \widehat{W}_{i,\text{last}}(t_{a-1}) + y)$;
    $d = (t_a, \widehat{W}_{i,\text{last}}(t_a))$;
    Déterminer si la droite (c, d) coupe la droite affine $f(t) = t$;
    Soit t'' le point d'intersection;
    if $t'' \leq (last - 1)T_i + T_i$ **then return** *"faisable"*;
    last_active := *last*;
  end
end
return *last_ active*;

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Algorithm 5: RevisedApproxSecondStage

input :
$\tau = C[n], T[n], D[n]$: tableau d'entiers `/* Paraètres de tâches */`
i : entier `/* Index de la tâche analysée */`
k : entier `/* Le paramètre de précision du FPTAS */`
h : entier `/* sortie d'ApproxFirstStage */`

$\widehat{R}_{i,h} = \frac{hC_i}{1-\sum_{j<i} U_j} + \frac{\sum_{j<i} C_j}{1-\sum_{j<i} U_j}$ `/* Temps de réponse approché de` $\tau_{i,h}$ `*/`
if $\widehat{R}_{i,h} > (h-1)T_i + D_i$ **then** `/* Step 1: l'instance` h `dépasse son échéance */`
  return τ_i *non faisable*;
end
return τ_i *est faisable*;

TABLE 5.1 – Ensemble de tâches à priorité fixe

Tâches	C_i	D_i	T_i
τ_1	26	40	70
τ_2	62	140	100

TABLE 5.2 – Analyse exacte de τ_2 de la Table 5.1

Réveils τ_2 instance	Date de fin	Temps de réponse exact	Statut
0	114	114	Continue
100	202	102	Continue
200	316	116	Continue
300	404	104	Continue
400	518	118	Continue
500	606	106	Continue
600	696	$94 < 100 = T_2$	Stop
		max=118	Return τ_2 *est faisable*

TABLE 5.3 – Paramètres approchés, Points d'ordonnancement et étapes pour τ_2 de la Table 5.1

ϵ	k	$\widetilde{S}_i$	Première étape	Seconde étape
0.25	3	$\{0, 70, 140, \infty\}$	$\tau_{i,l}$ terminée $(0, 140]$	$\tau_{i,l}$ terminée dans $(140, \infty)$

TABLE 5.4 – Analyse approchée de faisabilité de la tâche τ_2 dans la Table 5.1 avec ApproxFirstStage

$(t_{a-1}, t_a]$	$last\_active_{a-1}$	$I_i(t_a)$	$first$	$last$	t'	t''	$last\_active_a$	Statut
$(0, 70,]$	0	0					0	Continue
$(70, 140]$	0	1	1	1	114	114	1	Continue
								Return 2

tel-00554265, version 1 - 10 Jan 2011

TABLE 5.5 – Analyse approchée de fasiabilité de τ_2 dans la Table 5.1 avec ApproxSecondStage

Intervalle testé	h	$\widehat{R}_{i,h}$	Statut
$(140, \infty)$	2	$238,63 < (2-1).100 + 140$	Retourne τ_2 *est faisable*

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Conclusion générale

La conception et l'analyse de systèmes temps réels, tels que dans les systèmes de contrôle-commande, nécessite de disposer de plus d'information qu'un test booléen sur le respect des échéance des tâches (c.f., analyse RMA des systèmes temps réel à priorité fixe). Les temps de réponse des tâches fournissent aux concepteurs de systèmes temps réel un indicateur fondamental dans la compréhension de la dynamique globale du système de tâches temps réel. La conception intéractive et l'analyse des systèmes temps réel nécessitent de calculer les pires temps de réponse à de nombreuses reprises. Nous nous sommes limités dans ce mémoire à l'étude des systèmes monoprocesseurs à priorité fixe. Le calcul des pires temps de réponse exacts pour ces systèmes nécessitent un temps de calcul pseudo-polynomial en la taille du système à analyser (ce problème est connu NP-Difficile au sens faible) et une telle charge de calcul est souvent trop importante pour être intégrée dans un processus de conception. Pour de telles applications, il peut être acceptable d'utiliser un algorithme plus rapide qui fournit une analyse approchée au lieu d'utiliser des analyses exactes coûteuse en temps de calcul. Finalement, ces bornes supérieures introduisent du pessimisme dans les tests d'ordonnançabilité effectués, et il est en conséquence souhaitable de pouvoir le quantifier avec précision. C'est la raison pour laquelle, dans ce travail, nous proposons des algorithmes pour calculer efficacement des bornes supérieures des pires temps de réponse et faire une comparaison entre elles ainsi qu'analyser leur qualité pour déterminer leur compétitivité par rapport à un calcul exact des pires temps de réponse des tâches.

Dans ce but, nous avons tout d'abord étudié les bornes supérieures linéaires de temps de réponse de Bini et Baruah et de Sjödin et Hansson dans Chapitre 3. Nous avons analysé la qualité de ces bornes en utilisant la technique de la théorie d'optimisation et nous avons établi qu'aucun ratio d'approximation constant ne peut être défini. Puis, nous avons essayé de quantifier le pessimisme de ces bornes en utilisant la technique d'augmentation de ressource. Pour ces deux bornes calculables en temps linéaire, nous avons établi qu'elles étaient des bornes supérieures du pire temps de réponse lorsqu'un processeur de capacité unitaire est considéré et qu'elles étaient des brones inférieures du pire temps de réponse lorsqu'un processeur de capacité $1/2$ est considéré. Ainsi ce facteur d'augmentation de 2 de la capacité du processeur est la borne supérieure du prix à payer pour utiliser ces approximations des pires temps de réponse.

Dans un second temps dans Chapitre 4, nous avons étudié la technique d'approximation polynomiale introduite dans [15]. Nous avons défini deux nouveaux schémas d'approximation complet (FPTAS) de test d'ordonnançabilité pour les systèmes de tâches à priorité fixe qui sont meilleurs que les tests

de [15]. Puis, nous avons proposé une nouvelle méthode pour déduire deux bornes supérieures du pire temps de réponse à partir un schéma d'approximation établi dans le test de faisabilité. Les algorithmes correspondants sont paramétriques dans le sens qu'un paramètre de précision ϵ est introduit pour contrôler le temps passé à faire une analyse exacte avant de débuter l'approximation dans les calculs des pires temps de réponse. Si le paramètre de précision est un nombre très petit, alors nos expérimentations prouvent que les bornes supérieures sont très près des pires temps de réponse exacts, mais encore calculables dans un temps polynomial en fonction des paramètres des tâches et de la constante $1/\epsilon$.

Nous avons étudié la qualité de ces bornes à travers deux métriques quantitatives : la ratio d'approximation classique et le facteur d'augmentation de ressource. A l'aide de contre-exemples, nous avons prouvé dans un premier temps que les bornes supérieures que nous avons proposées n'offrent pas des garanties de performance selon la métrique conventionnelle (ratio d'approximation classique). Néanmoins, une garantie de performance peut être obtenue en utilisant une augmentation de la capacité de calcul d'un facteur de $\frac{k+1}{k}$ (où k est le nombre d'étapes avant de débuter la phase d'approximation dans les calculs du pire temps de réponse de la tâche analysée). Précisément, nous avons démontré que ces bornes offrent la garantie quantitative suivante : les bornes calculées constituent des supérieures du temps de réponse exact lorsqu'un processeur de capacité unitaire est considéré, et elles définissent des bornes inférieures au temps de réponse exact si le système était implémenté sur un processeur de vitesse $\frac{k}{k+1}$.

Finalement, nous avons étendu la technique de schéma d'approximation aux systèmes de tâches à échéances arbitraires dans le Chapitre 5. Dans ce travail, nous avons dans un en premier temps revisité le schéma présenté par Fisher et Baruah dans [14] et à l'aide de contre-exemples, nous avons montré que le test correspondant n'était pas correct dans le cas général d'une part, et pouvait être amélioré, d'autre part. Puis, nous avons modifié les définitions et les preuves de validité du test qui préservent la structure globale de l'algorithme de Fisher et Baruah. Grace à ces modifications, nous avons obtenu un nouveau FPTAS pour tester l'ordonnançabilité des tâches avec une erreur quantifiable à l'aide de la technique d'augmentation de ressource. Précisement, nous avons obtenu un test de compléxité polynomiale dans le taille de système n et le borne d'erreur $\frac{1}{\epsilon}$ et qui a la propriété suivante : si le test conclut qu'une tâche est faisable alors elle est effectivement faisable sur un processeur de vitesse unitaire. Sinon, elle est infaisable sur un processeur plus lent de vitesse $(1 - \epsilon)$.

Les perspectives à court terme de ce travail sont d'étendre le dernier schéma d'approximation d'ordonnançabilité des tâches à priorité fixe et échéance arbitraire pour déterminer des bornes supérieures des pires temps de réponse des tâches. Nous pensons obtenir ainsi de nouvelles bornes supérieures et établir une garantie de performance à l'aide de la technique d'augmentation de ressource. Il serait de plus intéressant de comparer les résultats obtenus avec les résultats des bornes proposées dans le chapitre Chapitre 4. Nous souhaitons aussi appliquer nos techniques d'approximation pour déterminer des bornes inférieures des pires temps de réponse des tâches. Finalement, nous souhaitons aussi étendre cette approche à systèmes non-préemptifs et au calcul des pires temps de réponse des messages transmis sur un réseau CAN (Controller Area Network) dont l'attribution du canal repose sur l'arbitrage de priorité fixe. Ces derniers points permettraient alors d'étendre notre approche à l'analyse de grands systèmes distribués.

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Bibliographie

[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] K Albers and F. Slomka. Efficient feasibility analysis for real-time systems with edf scheduling. *proc. of Design, Automation and Test in Europe Conference (Date'05)*, 2005.

[3] N.c. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. *YCS, University of york*, 164, nov 1991.

[4] Neil C. Audsley, Alan Burns, Mike Richardson, Ken W. Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5) :284–292, September 1993.

[5] S. Baruah, E. Bini, T.H.C. Nguyen, and P Richard. Continuity and approximability of response time bounds. *Euromicro Conf. on Real-Time Systems (ECRTS'07), Work-in Progress*, 2007.

[6] E. Bini and S. Baruah. Efficient computation of response time bounds under fixed-priority scheduling. *proc. Int. Real-Time and Network Systems (RTNS'07)*, 2007.

[7] E. Bini and G. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53, Nov 2004.

[8] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Journal of Real-Time Systems*, 30(1-2) :129–154, 2005.

[9] E. Bini, G. Buttazzo, and G.M. Buttazzo. Rate monotonic scheduling : The hyperbolic bound. *IEEE Transactions on Computers*, 52(7) :933–942, 2003.

[10] Enrico Bini, THC Nguyen, P Richard, and Sanjoy Baruah. A response-time bounds in fixed-priority scheduling with arbitrary deadlines. *IEEE Transactions on Computers (TOC'09)*, 58(2) :279–286, feb 2009.

[11] Samarjit Chakraborty, Simon Kunzli, and Lothar Thiele. Approximate schedulability analysis. *proc. Int. Symposium on Real-Time Systems (RTSS'02)*, 2002.

[12] A. Choquet-Geniet and E. Grolleau. Minimal schedulability interval for real-time systems of periodic tasks with offsets. *Theoretical of Computer Sciences*, 310 :117–134, 2004.

[13] F.Eisenbrand and T.Rothvoss. Static-priority real-time scheduling : Response time computation is np-hard. *proc. IEEE Int. Symposium on Real-Time Systems (RTSS'08)*, 2008.

tel-00554265, version 1 - 10 Jan 2011

[14] 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.

[15] N. Fisher and S.K. Baruah. A polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. *Proc. Int. Conf. on Real-Time Systems (RTNS'05)*, pages 233–249, 2005.

[16] N. Fisher, T.H.C. Nguyen, J. Goossens, and P. Richard. Parametric polynomial-time algorithms for computing response-time bounds for static-priority tasks with release jitters. *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'07)*, 2007.

[17] Laurent George, Nicolas Rivierre, and Marco Spuri. Preemptive and non-preemptive real-time uni- processor scheduling. *INRIA Research Report (2966), 55p.*, 1996.

[18] M. Joseph and P. Pandya. Finding response times in a real-time systems. *The Computer Journal*, 29(5) :390–395, 1986.

[19] J.P. Lehoczky. Fixed priority scheduling of periodic tasks with arbitrary deadlines. *proc. IEEE Int. Real-Time System Symposium (RTSS'90)*, pages 201–209, 1990.

[20] J.P. Lehoczky, L Sha, and Y Ding. The rate monotonic scheduling algorithm : exact characterization and average case behavior. *proc. IEEE Int. Real-Time System Symposium (RTSS'89)*, pages 166–171, 1989.

[21] J. Leung and M. Merrill. A note on preemptive scheduling of periodic real-time tasks. *Information Processing Letters*, 11(3) :115–118, 1980.

[22] J Y T Leung and J Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2 :273–250, 1982.

[23] 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.

[24] Jukka Mäki-Turja and Mikael Nolin. Efficient implementation of tight response-times for tasks with offsets. *Real-Time Systems Journal*, February 2008.

[25] Y. Manabee and S. Aoyagi. A feasible decision algorithm for rate monotonic and deadline monotonic scheduling. *Real-Time Systems Journal*, pages 171–181, 1998.

[26] T.H.C. Nguyen. Approximation de temps de réponse des tâches à priorité fixe. *Rapport de Master, LISI, ENSMA*, 2007.

[27] Thi Huyen Chau NGuyen, Pascal Richard, and Enrico Bini. Improved approximate response time bounds for static priority tasks. *Real-Time and Network Systems (RTNS'08)*, 2008.

[28] Thi Huyen Chau NGuyen, Pascal Richard, and Enrico Bini. Approximation techniques for response-time analysis of static-priority tasks. *Real-Time Systems*, 43 :147–176, 2009.

[29] Thi Huyen Chau NGuyen, Pascal Richard, and Nathan Fisher. The fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines revisited. *proc. Int. Conf. on Real-Time and Networked Systems*, 2010.

[30] P. Richard. Polynomial time approximate schedulability tests for fixed-priority real-time tasks : some numerical experimentations. *proc. Int. Real-Time and Network Systems (RTNS'06)*, 2006.

[31] P. Richard and J. Goossens. Approximating response times for static-priority tasks with release jitters. *WIP, Euromicro Int. Conf. on Real-Time Systems (ECRTS'06),*, 2006.

[32] P. Richard, J. Goossens, and N. Fisher. Approximate feasibility analysis and response-time bounds of static-priority tasks with release jitters. *proc. Int. Real-Time and Network Systems (RTNS'07)*, 2007.

[33] R.I.Davis and A.Burns. Response time upper bounds for fixed priority real-time system. *proc. IEEE Int. Symposium on Real-Time Systems (RTSS'08)*, 2008.

[34] M. Sjodin and H. Hansson. Improved response time analysis calculations. *proc. IEEE Int. Symposium on Real-Time Systems (RTSS'98)*, 1998.

[35] Marco Spuri. Analysis of deadline scheduled realtime systems. *INRIA Research Report (2772), 31p.*, 1996.

[36] K.W. Tindell. *Fixed Priority Scheduling of Hard Real-Time Systems.* PhD thesis, University of York, 1994.

tel-00554265, version 1 - 10 Jan 2011

tel-00554265, version 1 - 10 Jan 2011

Résumé

Deux techniques sont utilisées pour vérifier que des tâches temps réel respectent bien leurs échéances temporelles : les tests d'ordonnançabilité qui renvoient un résultat binaire (ordonnançable ou non) et les calculs de temps de réponse (Response Time Analysis - RTA) qui déterminent la longueur du plus long intervalle de temps entre le réveil et la terminaison d'une tâche. Ces deux approches ont une complexité pseudo-polynomiale et notons qu'aucun algorithme polynomial n'est connu. Dans ce contexte, elles ne sont pas particulièrement appropriées pour la conception intéractive des systèmes temps réel ou pour analyser des systèmes distribués à l'aide d'une analyse holistique. Dans de tels scénarios, un algorithme pseudo-polynomial est lent, puisque les calculs des temps de réponse des tâches sont exécutées à de nombreuses reprises. De plus, pour certains systèmes temps réels, tels que dans les systèmes de contrôle-commande, il est nécessaire de connaître le pire temps de réponse des tâches et non seulement la décision binaire sur l'ordonnançabilité des tâches. Dans ce contexte, il peut être acceptable d'utiliser un algorithme plus rapide qui fournit une analyse approchée au lieu d'utiliser des analyses reposant sur des calculs exacts. Comme cette approximation va introduire du pessimisme dans le processus de décision, il est souhaitable de le quantifier d'une manière à définir un compromis entre le temps de calcul et l'exigence de ressource du processeur. C'est la raison pour laquelle, dans ce travail, nous proposons des algorithmes pour calculer efficacement des bornes supérieures des pires temps de réponse et nous présentons des résultats sur leurs qualités dans le pire cas (analyse de compétitivité avec augmentation de ressource) et en moyenne (simulations).

Mots-clés : systèmes temps réel, ordonnancement à priorité fixe, analyse d'ordonnançabilité, systèmes monoprocesseurs, approximation polynomiale

Abstract

Two techniques are used to verify if real-time tasks meet their deadlines : schedulability tests that return a binary result (schedulable or not) and the calculation of the response times (Response Time Analysis - RTA) which determines the length of the longest interval between the release instant and the completion of a task. Both approaches have a pseudo-polynomial time complexity and please note that no polynomial time algorithm is known. In this context, they are not particularly suitable for interactive design of real-time systems or to analyze distributed systems using an holistic analysis. In such scenarios, a pseudo-polynomial time algorithm is slow, since the computations of the task response times are performed many times. Moreover, for some real time systems, such as the control-command systems, it is necessary to know the task worst-case response times and not only the binary decision on the tasks schedulability. In this context, it may be acceptable to use a faster algorithm which provides an approximate analysis instead of the analysis based on exact calculations. As these approximations will produce pessimism in the decision process, it is desirable to quantify this pessimism in such a way as to define a compromise between computation time and resource requirement of processor. That is the reason why, in this work, we propose efficient algorithms to compute upper bounds of worst-case response times and we present results on their qualities in the worst-case (competitive analysis resource augmentation) and in the average (simulations).

Keywords : real-time systems, fixed-priority scheduling, schedulability analysis, uniprocessor systems, polynomial approximation

Secteur de recherche : Informatique

LABORATOIRE D'INFORMATIQUE SCIENTIFIQUE ET INDUSTRIELLE
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Téléport 2 – 1, avenue Clément Ader – BP 40109 – 86961 Chasseneuil du Poitou Cedex
Tél : 05.49.49.80.63 – Fax : 05.49.49.80.64