Analyse des temps de réponse et de la demande processeur en ordonnancement temps réel de tâches périodiques

Pascal Richard
Laboratoire d'Informatique Scientifique et Industrielle
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Téléport 2 - 1 avenue Clément Ader
BP 40109
86961 Chasseneuil cedex
pascal.richard@ensma.fr

Abstract

La mise en œuvre de nombreux systèmes temps réel repose sur un exécutif temps réel. Il fournit un ordonnanceur à priorité pour attribuer le processeur aux tâches durant la vie du système. La validation d'un système temps réel revient alors à vérifier que l'ordonnancement respecte les contraintes temporelles des tâches pour tous les comportements possibles du système. Nous présentons deux techniques d'analyse d'ordonnançabilité des tâches périodiques à échéances strictes : l'analyse du temps de réponse et l'analyse de la demande processeur. Les principes de base de ces méthodes, leurs distinctions et leurs généralisations pour analyser des systèmes complexes sont présentés à travers des exemples simples dans le contexte de systèmes temps réel monoprocesseurs. Nous mettrons enfin en évidence les deux principaux problèmes à étudier durant la conception de nouveaux tests d'ordonnançabilité.

1 Introduction

Une tâche périodique est activée à intervalle régulier de temps. La longueur de cet intervalle définit la *période* d'activation de la tâche. Chaque occurence d'une tâche est appelée une *requête* (instance ou job). Dans un système temps réel dur, chaque requête est soumise à une échéance temporelle stricte. Le non respect d'une contrainte temporelle n'est pas admissible à l'exécution. L'ordonnancement de tâches périodiques est associé aux systèmes de contrôle-commande (p. ex., lecture échantillonnée d'un signal, etc.) et aux systèmes réactifs en général. La périodicité de l'activation des tâches est l'unique spécificité de l'ordonnancement temps réel par rapport à la théorie de l'ordonnancement classique. Cette nuance peut paraître mineure, mais en général la périodicité des tâches change la nature combinatoire des problèmes d'ordonnancement. Toutefois, l'ordonnancement temps réel nécessite de recourir aux mêmes techniques algorithmiques de résolution de problèmes combinatoires [20].

Le concepteur d'un système temps réel définit un ensemble (ou système) de tâches périodiques. Elles sont soumises à des contraintes temporelles dont le respect doit être validé (vérifié) avant la mise en service du système ou durant l'exécution des tâches si leurs caractéristiques ne sont pas connues a priori. La validation et l'ordonnancemement d'un système sont très souvent considérés comme des problèmes disjoints, bien qu'ils sont bien sûr fortement imbriqués en réalité. Lorsque l'ordonnancement est construit durant l'exécution (c-à-d., en-ligne), alors la validation consiste à analyser le comportement d'un algorithme fondé en général sur l'attribution de priorités aux tâches. L'ordonnancement et la validation sont alors clairement séparés et n'intéragissent pas (c.à.d., pas d'échange de données entre les étapes de validation et d'ordonnancement). Cette décomposition du problème d'ordonnancement consistant d'une part, à choisir un algorithme d'ordonnancement, et d'autre part, à analyser son comportement, permet de reduire la validation à un problème d'évaluation des performances. L'algorithme de validation est désigné sous le nom de *test d'ordonnançabilité* dans la littérature [3]. Un test positif indique que pour tous les comportements possibles du système alors les contraintes temporelles des tâches seront respectées.

Les tâches périodiques sont exécutées indéfiniment. Bien que l'ordonnancement soit infini, la périodicité des tâches permet de limiter la recherche des fautes temporelles dans un intervalle de temps borné, appelé intervalle de faisabilité ou d'étude (*feasibility interval*) [12]. La périodicité de l'ordonnancement est égale au ppcm des périodes des tâches et est désignée sous le nom d'*hyperpériode*. Simplifier le test

d'ordonnançabilité en un problème d'évaluation de performance suggère l'utilisation de techniques de simulation. Bien que cette technique soit très utile durant la conception préliminaire d'un système, nous rejettons son utilisation comme test d'ordonnançabilité pour les stratégies d'ordonnancement en-ligne. En effet, l'idée couramment rencontrée, consistant à simuler le comportement du système sur les bases des caractéristiques de l'ordonnanceur et du comportement pire cas de chaque tâche ne conduit pas nécessairement au pire comportement du système. L'instabilité d'un ordonnancement, liée à la variation des paramètres des tâches durant leur exécution, peut engendrer des situations qui n'auront jamais été simulées. La simulation, aussi bien que les tests logiciels classiques, ne permettent en aucun cas de conclure avec certitude que les tâches respecteront leurs contraintes temporelles. Seule une analyse pire cas, qui repose sur la caractérisation du pire comportement de l'application temps réel, permettra d'arriver à une conclusion avec un temps de calcul raisonnable.

Dans le premier exposé sur l'ordonnancement dans le cadre de l'école d'été ont été présentés les algorithmes et leurs propriétés de base [9]. Afin d'éviter les répétitions, nous conseillons aux lecteurs débutants en ordonnancement temps réel de lire ce premier article. Dans la suite nous considérons la validation d'un système reposant sur un ordonnanceur en-ligne. Trois techniques d'analyse pire cas existent dans la littérature (tests d'ordonnançabilité) : *l'analyse du facteur d'utilisation du processeur, l'analyse de la demande processeur et l'analyse du temps de réponse*. Ces techniques ne sont pas équivalentes. Dans ce second volet sur l'ordonnancement, nous nous limitons aux deux dernières techniques.

La majorité des monographies en temps réel présentent séparément l'ordonnancement des tâches à priorité fixe et à priorité dynamique. Nous choisissons ici une présentation différente afin de clairement dissocier la différence entre les tests exacts et approchés, comment ces méthodes se généralisent et enfin montrer les difficultés à surmonter pour constuire de nouveaux tests. Nous présentons paragraphe 2 la démarche générale de ces deux techniques tests et deux fonctions fondamentales pour calculer la demande processeur générée par les tâches. Le paragraphe 3 présente les tests exacts et le suivant les tests approchés. Enfin, dans le paragraphe 5, nous présentons quelles sont les difficultés à surmonter pour définir un nouveau test et les hypothèses simplificatrices couramment utilisées dans la littérature.

Nous utiliserons les notations suivantes : la première date de réveil de la tâche τ_i est notée r_i, sa pire durée d'exécution C_i, l'échéance relative à l'arrivée sera notée D_i et la période entre deux activations est T_i. Pour les systèmes à priorité fixe, nous considérons que les tâches sont triées selon leur priorité (τ_1 est la tâche la plus prioritaire). Nous nous limitons volontairement à l'ordonnancement monoprocesseur. La généralisation (de l'analyse du temps de réponse) aux systèmes distribués revient en pratique à valider les processeurs séparément en introduisant une gigue sur les activations des tâches comme variables de liaison entre les systèmes monoprocesseurs qui composent les systèmes distribués. Nous renvoyons aux références bibliographiques [16, 17] pour plus de détails.

2 Présentation des deux analyses

Dans le principe, l'analyse du temps de réponse et l'analyse de la demande processeur repose sur un même constat : *un dépassement d'échéance ne survient jamais lorsque le processeur est libre.* Ces analyses se restreignent à l'étude des intervalles de temps où le processeur exécute des tâches : *les périodes d'activité (busy periods).* L'analyse du temps de réponse et l'analyse de la demande processeur portent sur l'étude d'une ou plusieurs périodes d'activité. Chaque période d'activité à analyser va être caractérisée par *un scénario pire cas.* Bien que les deux analyses reposent sur le même constat, les principes de ces deux tests sont différents.

Définition 1 *L'analyse du temps de réponse est un test d'ordonnançabilité en deux étapes :*

- *tout d'abord le pire temps de réponse R_i (ou une borne supérieure) de chaque tâche est calculé,*
- *puis le respect des échéances est testé en vérifiant : $R_i \leq D_i$,$1 \leq i \leq n$ (complexité algorithmique en $O(n)$).*

L'analyse des temps de réponse se fait tâche par tâche et la complexité du test d'un système de tâches est principalement liée au calcul des pires temps de réponse. Par contre, l'analyse de la demande processeur est un test considérant toutes les tâches simultanément.

Définition 2 *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-à-d., n'excède pas la longueur de l'intervalle).*

Nous présentons maintenant les concepts nécessaires à l'étude de ces deux techniques d'analyse de l'ordonnançabilité d'un système de tâches.

2.1 Périodes d'activité et scénario de test

Définition 3 *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.*

Lorsque l'ordonnanceur est conservatif, c'est-à-dire qu'il n'insère pas de temps creux dans l'ordonnancement s'il existe une tâche prête à s'exécuter, le nombre de périodes d'activités différentes est fini puisque l'ordonnancement est périodique avec une période égale au ppcm des périodes des tâches. Précisément, l'ordonnanceur va être confronté exactement au même scénario de réveils des tâches à chaque début d'hyperpériode et prendra en conséquence exactement les mêmes décisions. Toutefois, l'analyse de toutes les périodes d'activité n'est en général pas possible puisque leur nombre est exponentiel. De plus, trouver dans quelle période d'activité une tâche ne respectera pas son échéance n'est pas un problème simple et nécessite dans la majorité des cas un temps de calcul exponentiel dans la taille du système de tâches à analyser. Pour des systèmes simples de tâches, nous allons présenter les résultats analytiques connus pour caractériser la période d'activité pour trouver les tâches ne respectant pas leurs échéances dans un système non ordonnançable.

Dans le cas d'un système à priorité fixe, tester l'ordonnançabilité d'une tâche τ_i ne nécessite pas de considérer les tâches moins prioritaires puisqu'elles n'engendreront pas d'interférence sur τ_i. Ceci revient à définir une période d'activité se limitant à un sous-ensemble de tâches prioritaires.

Définition 4 *(Systèmes à priorité fixe) 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).*

Une période d'activité va être caractérisée par les dates de réveils de requêtes qui la débutent. Nous pouvons maintenant définir la notion de scénario.

Définition 5 *Un scénario est l'ensemble des dates de réveils des requêtes permettant de caractériser* une *période d'activité du processeur.*

Deux cas se produiront suivant que les pires scénarios considérés durant le test se produiront ou non durant la vie du système :

- 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,
- 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.

Exemple 1 *Quelques exemples de scénarios conduisant soit à un test exact, soit à un test approché :*

- *Tests exacts : le scénario se produit toujours dans la vie de l'application.*
  - *Ordonnancement de tâches périodiques à départ simultané et à priorité fixe : le pire scénario est défini par le réveil simultané d'une requête de chaque tâche (c-à-d., l'instant critique). Ce scénario se produit au démarrage de l'application.*
  - *Ordonnancement de tâches à départ différé et gigue sur activation. On peut toujours définir les valeurs des gigues de façon à créer un instant critique (resynchroniser l'activation des tâches au début d'une période d'activité) et se ramener ainsi au pire scénario présenté ci-dessus.*
- *Tests approchés : le scénario peut ne pas se produire dans la vie de l'application.*
  - *Ordonnancement de tâches périodiques à départ différé et à priorité fixe : le pire scénario est défini par le réveil simultané des tâches. Savoir si ce scénario se produira dans la vie de l'application est co-NP-Complet (problème de congruences simultanées). En conséquence, la demande processeur calculée sera une borne supérieure.*
  - *Ordonnancement de tâches périodiques à priorité fixe en non-préemptif : le pire scénario pour la tâche τ_i survient lorsqu'elle est réveillée en même temps que les requêtes des tâches plus prioritaires $(\tau_1,\ldots,\tau_{i-1})$ et juste après de la plus longue tâche parmi les moins prioritaires $(\max_{j=i+1,\ldots,} C_j)$. Analyser si ce scénario surviendra ou non dans la vie de l'application n'est pas un problème simple a priori. En conséquence le test correspondant sera approché.*

Afin de faciliter la conception de tests exacts, deux hypothèses se rencontrent presque systématiquement dans la littérature. Elles permettent d'assurer que le pire scénario se produit dans la vie du système. Ces hypothèses sont d'introduire une *gigue sur activation* (permettant ainsi de traiter les systèmes distribués) ou des *tâches sporadiques* (la période entre deux requêtes est une durée minimum et non une durée exacte, comme dans le cas des tâches périodiques). Ces hypothèses supplémentaires permettent de considérer un problème d'ordonnancement plus général, mais dont les pires scénarios sont plus simples à caractériser.

Exemple 2 *Voici deux exemples introduisant une simplification en jouant sur la périodicité des activations*

des tâches :

- *Ordonnancement de tâches sporadiques à priorité fixe et à départ différé : le scénario où toutes les tâches se réveillent simultanément pourra se produire dans la vie du système. Dans le cas de tâches strictement périodiques, une telle caractérisation n'existe pas.*
- *Ordonnancement EDF de tâches sporadiques : le pire temps de réponse d'une tâche τ_i survient dans une periode d'activité où toutes les tâches autres que τ_i sont réveillées simultanément et où τ_i se réveille à une date d'échéance d'une autre tâche. Le test exact de Spuri [19] dans le cas sporadique ne serait pas exact si les tâches étaient strictement périodiques, car le pire scénario ne se produirait pas forcément dans la vie du système.*
- *Ordonnancement de tâches périodiques à priorité fixe et départ différé avec gigue sur activation des tâches : les gigues vont permettre de resynchroniser les tâches de façon à définir le pire scénario qui se produira dans la vie du système.*

2.2 Evaluation de la demande processeur

Caractériser l'activité du processeur revient à compter les requêtes activées dans un intervalle de temps (c-à-d., dans la période d'activité). Deux fonctions vont être particulièrement utiles par analyser l'activité du processeur associée aux exécutions des requêtes des tâches. Cette activité sera dans la suite désignée sous le nom de *demande processeur.* Deux fonctions permettent de définir la demande processeur sur un intervalle de temps [14, 2] :

- la demande processeur des tâches réveillées avant une date t , qui sera notée $rbf(t)$ (request bound function, parfois notée $G(t)$),
- la demande processeur des tâches devant se terminer avant la date t (l'échéance est avant ou à la date t), qui sera notée dbf (demand bound function, parfois notée $H(t)$)

La première fonction est particulièrement utile pour analyser les systèmes à priorité fixe, tandis que la seconde est utile pour analyser l'ordonnancement produit par EDF. Nous nous limitons aux tâches à échéance contrainte ($D_i \leq T_i$ - constrained deadline) et à départ différé (asynchronous) pour illustrer les définitions de ces deux fonctions.

2.2.1 La fonction de travail du processeur

La demande processeur des tâches réveillées dans l'intervalle de temps $[0,t[$ repose sur le nombre k de réveils d'une tâche τ_i dans cet intervalle de temps. Cela nécessite de connaître la dernière requête activée avant la date t. Les deux inégalités suivantes permettent de déterminer k :

$$r_i + (k-1)T_i < t \quad \Rightarrow \quad k < \frac{t - r_i}{T_i} + 1$$
$$r_i + kT_i \geq t \quad \Rightarrow \quad k \geq \frac{t - r_i}{T_i}$$

Nous devons de plus assurer $k \geq 0$. Ces inégalités seront respectées pour : $k = \max(0,\lceil \frac{t-r_i}{T_i} \rceil)$. Notons que les requêtes comptabilisées ne sont pas nécessairement terminées à la date t. La durée maximum cumulée de travail processeur associée aux réveils de τ_i sur l'intervalle de temps $[0,t[$ est notée rbf (request bound function) :

$$rbf(\tau_i,t) = \max\left(0, \left\lceil \frac{t - r_i}{T_i} \right\rceil\right) C_i$$

La fonction de travail du processeur sur l'intervalle $[0,t[$ tient compte des requêtes de toutes les tâches :

$$W(t) = \sum_{j=1}^{n} rbf(\tau_j,t) \tag{1}$$

Le travail processeur est une fonction en escalier. La droite affine $f(t) = t$ définit la capacité maximum de traitement du processeur (avec une vitesse unitaire). Ainsi, lorsque $W(t) = t$, alors à cette date, le processeur a terminé toutes les requêtes réveillées dans l'intervalle $[0,t[$.

Pour les sytèmes à priorité fixe, la fonction de travail $W_i(t)$ est la durée cumulée des tâches de priorité supérieure ou égale à i et réveillées dans l'intervalle $[0,t[$:

$$W_i(t) = \sum_{j=1}^{i} rbf(\tau_j,t) \tag{2}$$

2.2.2 La demande processeur

La fonction $dbf(t_1,t_2)$ (demand bound function) est la durée cumulée des requêtes dont la date de réveil et l'échéance sont dans l'intervalle $[t_1,t_2]$. Nous pouvons définir le nombre k de réveils d'une tâche τ_i. Pour déterminer les requêtes dont les échéances surviennent dans un intervalle $[0,t]$, nous identifions la dernière requête de la tâche τ_i avec les deux inégalités suivantes :

$$r_i + (k-1)T_i + D_i \leq t \quad \Rightarrow \quad k \leq \frac{t - r_i - D_i}{T_i} + 1$$
$$r_i + kT_i + D_i > t \quad \Rightarrow \quad k > \frac{t - r_i - D_i}{T_i}$$

Nous savons aussi que $k \geq 0$. Ces inégalités seront respectées pour la tâche τ_i : $k = \max\left(0,\lfloor \frac{t-r_i-D_i}{T_i} \rfloor + 1\right)$. Nous avons donc la demande processeur des requêtes

tâches	C_i	D_i	T_i	R_i
τ_1	2	10	10	2
τ_2	10	25	30	14
τ_3	55	100	120	119

TAB. **1 –.** *Système de tâches. La colonne R_i indique les temps de réponse des tâches avec un ordonnanceur à priorité fixe*

d'échéances inférieures ou égales t dans l'intervalle $[0,t[$:

$$dbf(0,t) = \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{t - r_i - D_i}{T_i} \right\rfloor + 1\right) \times C_i \quad (3)$$

Exemple 3 *Considérons un système de trois tâches à départ simultané dont les paramètres sont indiqués dans le tableau 1. La colonne R_i donne les temps de réponse des tâches lorsqu'elles sont ordonnancées avec des priorités fixes. Nous constatons que la tâche τ_3 n'est pas ordonnaçable (alors qu'elle le serait avec une échéance égale à la période). La figure 1 présente les fonctions $W_3(t) = rbf(t)$ et $dbf(t)$.*

2.2.3 Généralisations

En raisonnant sur des inégalités simples comme nous l'avons fait dans les paragraphes précédents, nous pouvons généraliser les formules précédentes pour traiter des systèmes plus complexes, comme ceux introduisant par exemple des paramètres supplémentaires dans la définition des tâches comme la gigue sur activation ou des temps de blocage associés à l'attente devant les ressources critiques (c-à-d., le facteur de blocage). Il convient toutefois d'être vigilent afin d'assurer que les valeurs calculées par les fonctions rbf ou dbf sont des bornes supérieures de la demande processeur, car sinon le test d'ordonnançabilité correspondant ne serait pas correct. Ceci nécessite de caractériser *le(s) pire(s) scénario(s)* d'arrivée des requêtes engendrant la plus grande activité du processeur.

Nous détaillons maintenant séparément les tests exacts et les tests approchés.

3 Tests exacts

Nous considérons pour simplifier la présentation les tâches à échéance contrainte (c-à-d., $D_i \leq T_i$) et qui sont à départ simultané (synchronous).

3.1 Analyse du temps de réponse

Le temps de réponse d'une requête est la différence entre sa date de fin et sa date de réveil. Le pire temps de réponse d'une tâche est le plus grand temps de réponse de ses requêtes. L'ordonnancement étant périodique, le nombre de valeurs différentes des temps de réponse des requêtes d'une tâche est fini et calculable.

3.1.1 Ordonnancement à priorité fixe

Comme nous l'avons vu précédemment, le calcul pratique du temps de réponse nécessite toujours de caractériser le(s) scénario(s) d'arrivée des tâches conduisant au pire temps de réponse de la tâche étudiée, déterminer une fonction d'analyse de la durée cumulée des tâches correspondant au(x) pire(s) scénario(s). Nous illustrons le calcul du pire temps de réponse d'une tâche τ_i dans un système à priorité fixe [**?**].

Lemme 1 *(Scénario) Le pire temps de réponse d'une tâche τ_i survient d'une période d'activité de niveau i débutant par un instant critique.*

Théorème 1 *(Test) Le pire temps de réponse de τ_i est défini par le plus petit point fixe de l'équation :*

$$W_i(t) = t$$

L'algorithme correspondant a été présenté dans le premier exposé sur l'ordonnancement et pour cette raison nous ne le détaillons pas (voir [9]). La complexité algorithmique du calcul du temps de réponse d'une tâche est pseudo-polynomiale : $O(n \sum_{i=1}^{n} C_i)$. Puisque les pires temps de réponse des tâches sont calculés en séquence, la complexité asymptotique du test d'ordonnançabilité est la même que celle pour analyser une tâche. Notons que l'existence d'un algorithme polynomial est un problème ouvert.

Exemple 4 *Sur le système de tâches présenté dans le tableau 1, les temps de réponse sont calculés en 3 itérations maximum et les temps de réponse correspondant sont indiqués dans la colonne R_i de ce même tableau.*

3.1.2 Ordonnancement EDF

Contrairement au système à priorité fixe, le pire temps de réponse d'une tâche ordonnancée par EDF ne survient pas nécessairement dans la première période d'activité du processeur [19]. Le pire temps de réponse d'une tâche peut être calculé en construisant (par simulation) l'ordonnancement EDF. Mais l'algorithme résultant sera exponentiel. A notre connaissance, aucun algorithme polynomial ou pseudo polynomial n'est encore connu pour calculer le pire temps de réponse exact des tâches ordonnancées par EDF.

3.2 Analyse de la demande processeur

L'analyse de la demande processeur est un test d'ordonnançabilité consistant à vérifier que toutes les requêtes devant s'exécuter dans tout intervalle de temps ne dépassent pas la capacité du processeur (c-à-d., la longueur de l'intervalle considéré). Contrairement

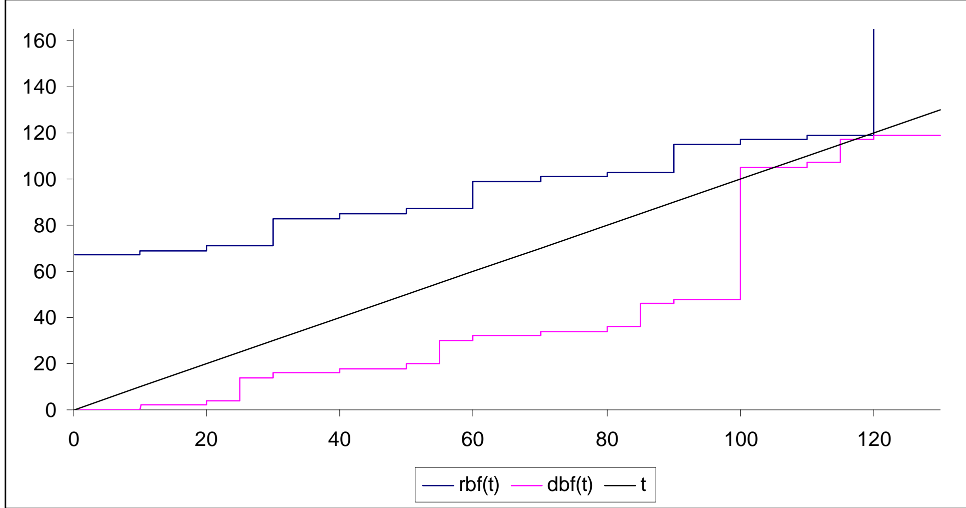

FIG. **1 –.** *Fonctions $rbf(t)$, $dbf(t)$ et $f(t) = t$ pour le système de tâches du tableau 1*

à l'analyse du temps de réponse, l'analyse de la demande processeur analyse toutes les tâches simultanément.

3.2.1 Ordonnancement à priorité fixe

Nous présentons dans la suite les principes de cette analyse à travers l'exemple des tâches à priorité fixe et à départ simultané [11].

Lemme 2 *(Scénario) Il est suffisant d'analyser l'intervalle de temps $[0,D_i)$ pour savoir si τ_i respectera ou non son échéance.*

Théorème 2 *(Test de Lehoczky, Sha et Ding) Dans un système de tâches à départ simultané, une tâche τ_i est ordonnançable si, et seulement si, il existe un instant $t \in (0,D_i]$ tel que $W_i(t) \leq t$.*

Cela revient donc à rechercher la valeur minimum de la fonction $W_i(t)/t$ dans l'intervalle $[0,D_i]$. La fonction de la demande processeur ne change de valeur qu'à des instant précis car l'équation 2 est une fonction en escalier. Le test va se limiter aux valeurs correspondant à des minima locaux de la fonction de la demande processeur. L'ensemble de ces valeurs définissent l'*ensemble des points d'ordonnancement* (Testing Set ou Scheduled Point Test) :

$$S_i = \{bT_j | j = 1 \dots i, b = 1 \dots \lfloor D_i/T_j \rfloor\} \qquad (4)$$

Ainsi, vérifier que la tâche τ_i est ordonnançable nécessite de calculer : $\min_{t \in S_i} \left(\frac{W_i(t)}{t}\right) \leq 1$. En conséquence, si une date $t \in S_i$ satisfait $W_i(t) \leq t$ alors τ_i est ordonnançable et il est inutile d'examiner d'autres points d'ordonnancement. Ainsi en pratique, seulement un sous-ensemble des points d'ordonnancement de S_i sera analysé. Mais, d'un point de vue complexité algorithmique, le nombre d'itérations du test est borné par le ratio: D_i/T_j. Donc, sa complexité algorithmique est pseudo-polynomiale. Le test complet se formule de la façon suivante :

$$\max_{i=1\dots n} \left\{ \min_{t \in S_i} \left(\frac{W_i(t)}{t} \right) \right\} \leq 1$$

Exemple 5 *Nous illustrons ce test sur l'analyse de la tâche τ_3 du système de tâches présenté dans le tableau 1. Les ensembles de tests sont :*

$$\begin{aligned} S_1 &= \{10\} \\ S_2 &= \{10,20,30\} \\ S_3 &= \{10,20,30,40,50,60,70,80,90,100\} \end{aligned}$$

Les calculs de $W_3(t)/t$ pour l'ensemble S_3 sont présentés dans le tableau 2. Le système est non ordonnançable puisqu'aucune date $t \in S_3$ ne conduit à vérifier la condition : $W_3(t)/t \leq 1$.

Nous renvoyons à [14] pour une présentation des algorithmes et une variante de ce test (permettant d'avoir une complexité indépendante des valeurs des paramètres, mais exponentielle - $O(n2^n)$). On pourra aussi consulter [5] pour avoir des informations complémentaires.

3.2.2 Ordonnancement EDF

Les tâches étant à départ simultané, la plus grande charge processeur survient au démarrage de l'application. Précisémment, la demande processeur dans le cas de tâches à départ simultané vérifie [4] :

Lemme 3 *(Scénario)*

$$dbf(0,t_2 - t_1) \geq dbf(t_1,t_2) \qquad \forall t_1,t_2 \quad et \quad t_1 \leq t_2$$

t	10	20	30	40	50	60	70	80	90	100
$W_1(t)/t$	0,2									
$W_1(t)/t$	1,2	0,7	0,5							
$W_3(t)/t$	6,7	3,4	2,3	2,0	1,7	1,4	1,4	1,2	1,14	1,1

TAB. **2 –.** *Exécution du test associé au théroème 2 pour analyser le système de tâches du tableau 1.*

Nous pouvons simplifier l'expression du calcul de la demande du processeur puisque nous supposons $r_i = 0$ et $D_i \leq T_i$, $1 \leq i \leq n$:

$$dbf(0,t) = \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) C_i \quad (5)$$

$$= \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i \quad (6)$$

Ceci conduit à définir un intervalle d'étude pour EDF débutant à l'instant critique 0, jusqu'à $H = ppcm(T_i)$, puisque l'ordonnancement est périodique de période H. En relachant la contrainte d'intégralité de l'équation 6, il est démontré dans [4] que l'intervalle d'étude peut se limiter dans le cas $U < 1$ à l'intervalle $[0,t_{lim}]$, avec :

$$t_{lim} = \frac{U}{1 - U} \max_{i=1..n}(T_i - D_i)$$

Ceci permet d'établir le test d'ordonnançabilité suivant :

Théorème 3 *(Test de Baruah, Howell et Rosier) Un système de tâches périodiques et à départ simultané est ordonnançable avec un facteur d'utilisation $U < 1$ si, et seulement si :*

$$dbf(0,t) \leq t \qquad \forall t, 0 < t < t_{lim}$$

Exemple 6 *Nous illustrons le fonctionnement de ce test sur le système de tâches du tableau 1. Le facteur d'utilisation du processeur est $U = 0{,}99$ et ceci conduit à la valeur $t_{lim} = 2380$ (On remarque que le ppcm des périodes est égal à $H = 120$. Cette valeur peut être utilisée comme limite de temps puisque les tâches sont à départ simultané). Nous devons donc tester toutes les dates t entre 0 et H pour vérifier la condition $dbf(t) \leq t$. Si pour une valeur de t, cette inégalité n'est pas vérifiée alors le système est non ordonnançable. Par contre, si le test est positif pour toutes les dates t entre 0 et H alors le système est ordonnançable. Pour les dates $t \in [0,100[$, toutes les inégalités sont respectées. Par contre à la date $t = 100$, nous avons : $dbf(0,t) = 105 > t$ (nous pouvons aussi directement le constater sur la figure 1 : la courbe $dbf(0,t)$ passe au dessus de la droite représentant la capacité du processeur $f(t) = t$). Le système de tâches n'est donc pas ordonnançable sous EDF.*

Sous l'hypothèse U constant, la complexité de ce test est $O\left(n \max_{i=1..n}(T_i - D_i)\right)$, l'algorithme est pseudo-polynomial. Ce test a été amélioré dans [18, 15], mais ces algorithmes sont eux aussi pseudo-polynomiaux. Notons que l'existence d'un algorithme fortement polynomial est un problème ouvert.

3.2.3 Généralisation

Les tests exacts présentés reposent sur l'analyse de la demande processeur, exploitent des propriétés permettant de limiter l'intervalle d'étude et enfin la localisent dans la première période d'activité du processeur (cas de tâches à départ simultané). En l'absence de caractérisation précise de la période d'étude, l'analyse de la demande processeur doit être généralisée afin d'établir un test d'ordonnançabilté. Ceci passe notamment par la définition plus générale de la fonction de la demande cumulée du processeur.

Définition 6 *Fonction de demande du processeur (Demand Bound Function). Soit τ_i une tâche, la fonction de demande du processeur $dbf(t_1,t_2)$ est la durée maximum cumulée d'exécution des tâches qui ont leurs réveils et échéances dans un intervalle de durée $[t_1,t_2]$.*

Cette définition de la fonction $dbf(t_1,t_2)$ permet de définir un test simple pour les tâches indépendantes, de façon analogue au paragraphe précédent. Un test général peut être formulé de la façon suivante :

Théorème 4 *Un système de tâches est ordonnançable si, et seulement si :*

$$\forall t_1 < t_2 \qquad dbf(t_1,t_2) \leq t_2 - t_1 \qquad (7)$$

Remarquons que dans le cas général, pour montrer que le système est non ordonnançable il est suffisant de trouver une valeur de t telle que l'inégalité 7 ne soit pas satisfaite. La généralisation à des tâches dépendantes avec des structures conditionnelles a été faite dans [2].

Toutefois, toute généralisation soulève deux questions :

- Comment calculer efficacement la fonction dbf ?
- Comment choisir un ensemble de points d'ordonnancement aussi petit que possible et qui soit suffisant pour garantir la correction du test défini par l'équation 7 ?

4 Tests approchés

L'objectif d'un test approché est de fournir une *décision approchée* au problème d'ordonnançabilité :

si le test approché renvoie Ordonnançable, alors les tâches respecteront leurs contraintes temporelles, sinon on ne peut pas conclure. La principale motivation pour concevoir un test approché est de définir des algorithmes de test avec une complexité algorithmique plus faible, soit parce que le test exact nécessite un temps de calcul exponentiel ; soit parce que le test doit être utilisé en-ligne pour contrôler l'admission de nouvelles tâches périodiques.

Les tests approchés vont donc utiliser des valeurs approchées de la demande processeur (c-à-d.., des bornes supérieures des fonctions présentées dans le paragraphe 2) et limiter le nombre d'itérations nécessaires pour prendre une décision. Ces deux opérations vont introduire du *pessimisme* dans les méthodes d'analyse.

4.1 Tests approchés sans garantie

Il est important (et peu rassurant) de constater qu'aucune estimation quantitative du pessimisme des tests approchés n'est en général rigoureusement proposée dans la littérature. Les évaluations reposent uniquement sur des simulations numériques qui ne comparent pas le test approché avec un test exact. De façon générale, les tests approchés d'ordonnançabilité conduisent à surdimensionner les systèmes afin de satisfaire la condition suffisante d'ordonnançabilité définie dans le test de validation !

Exemple 7 *Considérons l'ordonnancement de tâches périodiques à priorité fixe, à départ simultané et en mode non préemptif. Nous avons défini le pire scénario dans l'exemple 1. Considérons deux tâches τ_1,τ_2 de périodes identiques T et de durées C_1 et C_2, des échéances $D_1 = T/2$ et $D_2 = T$. Le scénario va conduire aux pires temps de réponse : $R_1 = R_2 = C_1 + C_2$. Puisque les tâches ont une période identique, alors la tâche τ_2 n'aura pas d'interférence avec τ_1. En conclusion le pire temps de réponse exact de τ_1 est $R_1^* = C_1$. Le rapport $\frac{R_1}{R_1^*} = 1 + \frac{C_2}{C_1}$ et on constate que le pire temps de réponse associé au scénario du test peut être arbitrairement éloigné de la valeur exacte R_1^*, puisque C_2/C_1 n'est borné par aucune constante. Le test classiquement utilisé pour valider des systèmes de tâches non-préemptifs (ou ordonnancement de messages dans un réseau CAN par exemple) est donc sans garantie de performance vis-à-vis d'un test exact.*

4.2 Tests approchés avec garantie

Nous rappelons tout d'abord quelques définitions sur l'approximation polynomiale, puis nous illustrons les tests avec garantie de performance sur les tâches à priorité fixe.

4.2.1 Approximation polynomiale

Une estimation du pessimisme des méthodes approchées peut être obtenue en utilisant des *algorithmes d'approximation* (ou approchés). Ces algorithmes sont utilisés pour résoudre de façon approchée des problèmes d'optimisation. L'intérêt d'un algorithme d'approximation est de posséder une *garantie de performance* vis-à-vis d'une méthode exacte. Précisément, soit A un algorithme approché et OPT une méthode exacte, alors la borne d'erreur ϵ $(0 < \epsilon < 1)$ de l'algorithme A pour toute instance I du problème d'optimisation est définie par :

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

Un algorithme approché a une garantie de performance bornée par le *ratio* suivant : $r_A = 1 + \epsilon$ (pour un problème de minimisation). 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. Une approximation polynomiale est un algorithme 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 $\mathcal{NP}$-Difficile. Nous renvoyons à [8] pour des compléments sur la complexité et l'approximabilité des problèmes.

Depuis plusieurs années, les algorithmes d'approximation intéressent les concepteurs de tests afin de garantir les performances dans le pire cas des tests approchés. Toutefois, tester la faisabilité d'un système de tâches est un problème de décision, alors que les algorithmes approchés concernent les problèmes d'optimisation. Bien que le temps de réponse soit un critère quantitatif, il n'existe pas à notre connaissance de calcul de pire temps de réponse approché avec une garantie constante de performance par rapport au pire temps de réponse exact. Mais des tests approchés reposant sur l'analyse de la demande processeur ont été proposés.

4.2.2 Approximation des temps de réponse

Un test d'ordonnançabilité fournit une réponse binaire : *ordonnançable* ou *non ordonnançable.* Par contre, le temps de réponse est un critère quantitatif. Celuici peut donc être utilisé pour définir le ratio d'approximation du calcul du pire temps de réponse d'une tâche. A notre connaissance il n'existe pas de résultat dans la littérature de calcul d'un pire temps de réponse approché avec une garantie de performance par

rapport au pire temps de réponse exact. Nous donnons ci-après un premier résultat négatif d'approximation du temps de réponse.

Un moyen classique pour établir des bornes est de relacher la contrainte d'intégralité dans la fonction de la demande processeur (plus précisément dans les fonctions $rbf_i(0,t)$). Nous notons R_i^* la valeur exacte du pire temps de réponse :

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

En utilisant cette expression, on obtient une borne inféreure et une borne supérieure calculables en temps linéaire (c-à-d., $O(n)$) :

$$R_i^* \geq \frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} = \overline{R}_i \quad (8)$$

$$R_i^* \leq \frac{\sum_{j=1}^{i} C_j}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} = \tilde{R}_i \quad (9)$$

Nous montrons que ces bornes n'ont pas de garantie de performance : c'est-à-dire qu'elles peuvent être très éloignées des valeurs exactes des pires temps de réponse des tâches. Précisément, nous notons $\overline{R}_i$ (respectivement $\tilde{R}_i$) la borne supérieure du temps de réponse (respectivement la borne inférieure), alors les ratios $\overline{R}_i/R_i^*$ et $R_i^*/\tilde{R}_i$ tendent vers l'infini (d'un point de vue asymptotique).

Théorème 5 *Soit r le ratio $\overline{R}_i/R_i^*$:*

a. Le ratio n'est pas borné pour les systèmes de tâches quelconques.

b. Si nous supposons qu'il existe une constante K telle que $K > \sum_i (C_i)/\min_i C_i$, alors $r \leq K$.

Preuve : Nous montrons tout d'abord la première assertion. Considérons l'instance suivante avec deux tâches: $\tau_1(1-\epsilon,1,1)$ et $\tau_2(K\epsilon,K,K)$, où ϵ respecte $0 < \epsilon < 1$ et K est un entier supérieur à 1. Notons que les périodes sont proportionnelles, en conséquence RM sera optimal et une condition nécessaire et suffisante d'ordonnançabilité $C_1/T_1 + C_2/T_2 \leq 1$. Le facteur d'utilisation est:

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

Le système de tâches est donc ordonnançable et le pire temps de réponse exact est:

$$\begin{aligned} R_1^* &= \overline{R}_1 = 1 - \epsilon \\ R_2^* &= K \qquad \overline{R}_2 = \frac{1 + (K-1)\epsilon}{\epsilon} \end{aligned}$$

La borne $\overline{R}_2$ pour τ_2 conduit au ratio suivant :

$$\lim_{\epsilon \to 0} \frac{\overline{R}_2}{R_2^*} = \lim_{\epsilon \to 0} \frac{1}{K\epsilon} + \frac{(K-1)}{K} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} = \infty$$

Maintenant, si nous supposons qu'il existe une constante K telle que $K > \sum_i (C_i)/\min_i C_i$ pour tout système de tâches alors en partant des équations 9, nous pouvons écrire :

$$\frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} \leq R_i^* \leq \frac{\sum_{j=1}^{i} C_j}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}}$$

ainsi,

$$\frac{\overline{R}_i}{R_i^*} \leq \frac{\sum_{j=1}^{i} C_j}{C_i} \leq K$$

Nous pouvons établir les mêmes types de résultat pour la borne inférieure du pire temps de réponse exact. L'existence d'algorithme approché (avec un ratio constant) pouvant être calculé en temps polynomial est un problème ouvert.

4.2.3 Approximation polynomiale de la demande processeur

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-à-d., 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 dans la suite le schéma d'approximation polynomiale de [1, 7] qui utilise 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 ordonnaçable*, 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.

Nous illustrons cette approche sur l'ordonnancement à priorité fixe [7]. La fonction $rbf(\tau_i,t)$ 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. Un moyen simple de définir un schéma d'approximation polynomiale est de ne considérer qu'un nombre borné k de paliers. Au delà, une fonction linéaire (continue) sera utilisée pour définir une borne supérieure de $rbf(\tau_i,t)$. Le nombre de paliers va être défini à l'aide du paramètre d'erreur ϵ :

$$k = \left\lceil \frac{1}{\epsilon} \right\rceil - 1$$

ϵ	0,01	0,1	0,2	0,3	0,4	0,5
k	101	11	6	5	4	3

TAB. **3 –.** *Nombre de paliers de la fonction $rbf(\tau_i,t)$ considérée avant la linéarisation de la fonction dans le test approché de Fisher et Baruah*

Nous pouvons maintenant définir l'approximation de la demande cumulée de la demande processeur de la tâche τ_i, qui sera notée $\overline{rbf}(\tau_i,t)$:

$$\overline{rbf}(\tau_i,t) = rbf(\tau_i t) \quad \text{Si} \quad t \leq (k-1)T_i \quad (10)$$

$$= C_i + t\frac{C_i}{T_i} \quad \text{Sinon} \quad (11)$$

La demande processeur approchée, est alors définie par :

$$\overline{W}_i(t) = C_i + \sum_{j=1}^{i-1} \overline{rbf}(\tau_i,t) \quad (12)$$

Pour terminer le test, Fisher et Baruah utilisent l'analyse de la demande processeur avec un ensemble de points d'ordonnancement (Testing Set) possédant un nombre polynomial d'entrées dans la taille du système de tâches à analyser et du paramètre de précision $1/\epsilon$:

$$\overline{S}_i = \{bT_j | j = 1 \ldots i-1, b = 1 \ldots k\} \quad (13)$$

Exemple 8 *Le tableau 3 donne la valeur de k en fonction de la borne d'erreur ϵ. Nous donnons figures 2 et 3, le graphique des fonctions $\overline{W}_2(t)$ et $\overline{W}_3(t)$ pour le système de tâches présenté dans le tableau 1 et des valeurs d'epsilon égales à 0,5 et 0,3.*

L'algorithme de test s'implémente très facilement en $O(n^2/\epsilon)$. Clairement, si ϵ est proche de 0, alors le nombre d'itérations effectuées par l'algorithme est très grand, mais le nombre d'itérations est polynomial en $1/\epsilon$. En conséquence, cet algorithme paramétrique est un FPTAS. Le choix du paramètre de précision ϵ est donc primordial pour obtenir un test rapide et avec une bonne garantie de performance.

Le cas des tâches avec des échéances arbitraires (c-à-d., telles que l'échéance D_i et la période T_i ne sont pas reliées par une contrainte) est présenté dans [7]. Par souci de concision, nous renvoyons à [1] pour l'approximation de la demande processeur pour EDF.

5 Difficulter pour concevoir les tests d'ordonnançabilité

Nous pensons que l'établissement d'un test d'ordonnançabilité devient difficile à concevoir lorsque l'algorithme d'ordonnancement n'est pas *robuste* (c-à-d., sujet aux anomalies d'ordonnancement) pour le problème considéré et que le problème d'ordonnançabilité est *difficile.* A notre connaissance, ces deux critères ne sont pas connus comme étant dépendants. Dans les deux cas, le calcul de la demande processeur (fonctions rbf ou dbf) doit donc reposer sur un scénario conduisant à la pire demande processeur. Un résultat analytique doit impérativement être construit pour garantir la correction du test.

Tâches	C_i	$D_i = T_i$	π_i
τ_1	1	3	1
τ_2	2	6	2
τ_3	4	12	3

TAB. **4 –.** *Tâches à échéance sur requête à ordonnancer sans préemption et avec des priorités fixes (π_i)*

5.1 Anomalies d'ordonnancement

Une anomalie d'ordonnancement est liée à l'instabilité de l'ordonnanceur. Ce problème, bien connu en environnement multiprocesseur, se formule de la façon suivante : *un système ordonnançable avec les pires durées d'exécution peut être non ordonnançable avec des durées d'exécutions plus petites.* Un ordonnanceur en-ligne sera dit *robuste* s'il n'est pas sujet à des anomalies d'ordonnancement pour le système de tâches considéré.

Exemple 9 *Voici quelques exemples de systèmes soumis à des anomalies d'ordonnancement en monoprocesseur :*

- *ordonnancement non préemptif (cas particulier d'ordonnancement avec ressources partagées en exclusion mutuelle),*
- *ordonnancement à priorité fixe avec contraintes de précédence,*
- *ordonnancement de tâches avec suspension (durant une opération d'entrée-sortie),*

Nous détaillons le premier exemple.

Exemple 10 *Nous présentons tableau 4 un système de tâches à ordonnancer sans préemption. La figure 4 présentent deux ordonnancements : le premier les tâches sont exécutées avec leurs pires durées d'exécution (a), tandis que dans le second la tâche 2 s'exécute en 1 seule unité de temps au lieu de 2 (b). Ainsi, réduire la durée d'exécution de τ_2 rend le système non-ordonnançable.*

5.2 Complexité

Déterminer la complexité du problème d'ordonnançabilité permet d'identifier quel type d'algorithme doit être construit pour analyser un système de tâches.

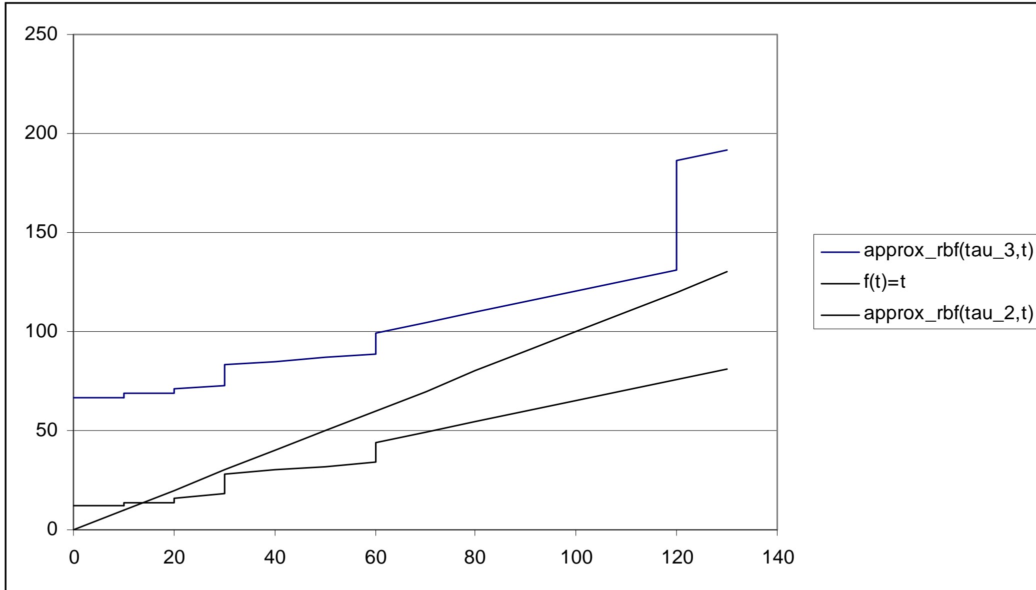

Fig. **2 –.** *Approximation de la demande processeur dans le test approché de Fisher et Baruah.* $\epsilon = 0{,}5$ $(k = 3)$*, le test conduit à l'ordonnançabilité de* τ_2*, mais* τ_3 *n'est pas ordonnançable sur un processeur de vitesse* $(1 - \epsilon)$*.*

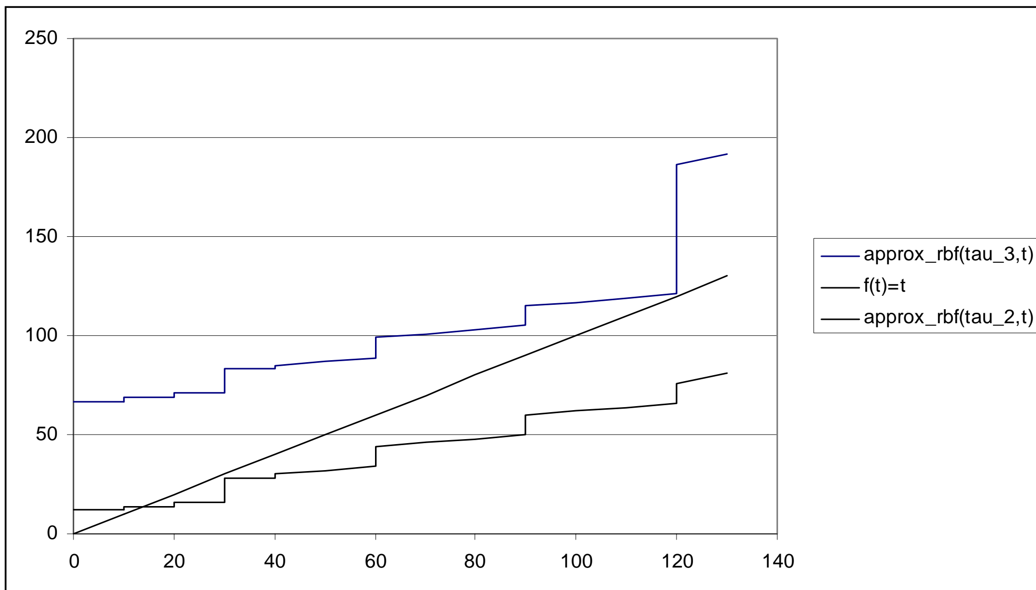

Fig. **3 –.** *Approximation de la demande processeur dans le test approché de Fisher et Baruah.* $\epsilon = 0{,}3$ $(k = 5)$*, le test conduit à l'ordonnançabilité de* τ_2*, mais* τ_3 *n'est pas ordonnançable sur un processeur de vitesse* $(1 - \epsilon)$*.*

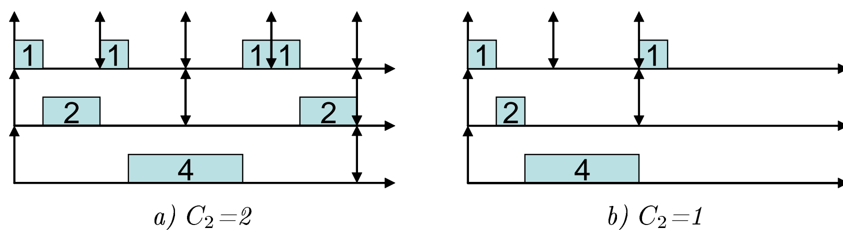

Fig. **4 –.** *Anomalies d'ordonnancement en non-préemptif pour le système de tâches du tableau 4*

Si le problème est $\mathcal{NP}$-difficile alors les tests seront généralement des tests approchés car établir un test exact sera trop coûteux en temps de calcul et le(s) pire(s) scénario(s) seront difficiles à caractériser.

Le paysage est paradoxalement compliqué en ordonnancement de tâches périodiques puisque les problèmes sont soit dans $\mathcal{NP}$, soit dans co-$\mathcal{NP}$, ou bien non connus comme étant dans l'une de ces deux classes de problèmes.

Exemple 11 *Nous donnons une caractérisation simple de ces deux classes de problèmes vis-à-vis de l'ordonnançabilité des tâches à départ simultané et à échéance contrainte (les tests exacts ont été présentés plus haut) :*

*- pour les systèmes ordonnancés avec des priorités fixes, il est possible de décider en temps polynomial qu'*une tâche est ordonnançable *(par un algorithme non-déterministe). Le test de Lehoczky, Sha et Ding (voir [14]) permet de résoudre ce problème en temps pseudo-polynomial. Supposons qu'un algorithme non-déterministe (c-à-d., l'oracle d'une machine de Turing non déterministe) nous donne un point d'ordonnancement t et une tâche τ_i, alors la tâche est ordonnançable si la condition $W_i(t) \leq t$ est vraie. Cette vérification s'effectue en temps polynomial, montrant ainsi que le problème d'ordonnançabilité est dans $\mathcal{NP}$.*

*- on sait pour les systèmes ordonnancés par EDF décider en temps polynomial qu'*une tâche n'est pas ordonnançable *(par un algorithme non déterministe). Supposons qu'un algorithme non-déterministe nous donne un point d'ordonnancement t, alors la condition $dbf(0,t) > t$ permet de conclure que le système est non-ordonnançable en temps polynomial. Ceci établit que le problème d'ordonnançabilité est dans co-$\mathcal{NP}$.*

Il est assez surprenant que l'analyse d'ordonnançabilité pour un système de tâches donné est soit dans $\mathcal{NP}$, soit dans co-$\mathcal{NP}$, en fonction de l'algorithme d'ordonnancement considéré (à priorité fixe ou EDF). Après tout, cela laisse un peu d'espoir sur l'existence d'un test polynomial ! Les deux exemples considérés précédemment sont ouverts du point de vue de leur complexité (ils ne sont pas connus $\mathcal{NP}$-difficiles et aucun algorithme polynomial n'est connu).

Notons par ailleurs que l'ordonnancabilité des tâches en mode non-préemptif est $\mathcal{NP}$-Difficile au sens fort et que l'ordonnancement de tâches à départ différé est co-$\mathcal{NP}$-Complet au sens fort (y compris en préemptif).

6 Conclusion

Nous résumons ici les principales caractéristiques des analyses présentées dans cet article. L'analyse du temps de réponse s'étend assez facilement pour tenir compte de contraintes supplémentaires (ressources, systèmes distribués,...). Mais les tests correspondants sont généralement approchés et sans garantie de performance vis-à-vis d'un calcul exact des pires temps de réponse des tâches. Par contre, l'analyse de la demande processeur conduit à des algorithmes de tests performants, mais les résultats analytiques sur lesquels ils reposent sont moins faciles à étendre. Pour cette seconde technique d'analyse, des tests approchés avec garantie de performance permettent d'envisager leur utilisation pour contrôler l'admission en-ligne de nouvelles tâches périodiques. Toutefois, à notre connaissance, aucune généralisation de l'analyse de la demande processeur n'est connue pour les systèmes distribués.

Par terminer, nous avons volontairement limité la bibliographie aux références les plus récentes et avec des présentations pédagogiques. Des nombreuses pointeurs bibliographiques seront trouvés dans ces articles. En complément sur l'analyse de la demande processeur, nous conseillons en première lecture l'article [18] pour l'analyse EDF et l'article [14] pour l'analyse des systèmes à priorité fixe. Pour l'analyse du temps de réponse, nous renvoyons aux monographies bien connues du domaine [10, 6, 13].

Remerciements

Je tiens à remercier très sincèrement Stéphane Pailler et Frédéric Ridouard, doctorants dans notre laboratoire, pour leurs multiples lectures de cet article et leurs nombreux commentaires qui ont permis d'améliorer la présentation de cet article.

Références

[1] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. *Euromicro Int. Conf. on Real-Time Systems (ECRTS'04)*, 2004.

[2] S. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Journal of Real-Time Systems*, 24(1):93–128, 2003.

[3] S. Baruah and J. Goossens. Scheduling real-time tasks: Algorithms and complexity. *In Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Joseph Y-T Leung (ed). Chapman Hall/CRC Press*, 2004.

[4] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Journal of Real-Time Systems*, 1:301–324, 1990.

[5] E. Bini and G. Buttazzo. Schedulability analysis of periodic fixed-priority systems. *IEEE Transactions on Computers*, 53(11): , nov 2004.

[6] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling, Algorithms and Applications.* Kluwer Academic Publishers, 1997.

[7] N. Fisher and S. Baruah. A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. *proc. Proceedings of the EuroMicro Conference on Real-Time Systems, (ECRTS'05)*, 2005.

[8] M. Garey and D. Johnson. *Computers and Intractability: a guide to the theory of NP-Completeness.* W.H. Freeman, San Francisco, 1979.

[9] L. George. Conditions de faisabilité pour l'ordonnancement temps réel. *Ecole d'Eté Temps Réel (ETR'05)*, 2005.

[10] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. Gonzales-Harbour. *A practitioner's handbook for real-time system analysis.* Kluwer Academic Publishers, 1993.

[11] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Extact characterization and average case behavior. *Real-Time Systems Symposium*, pages 166–171, 1989.

[12] J. Leung and M. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information processing letters*, 11(3):115–118, 1980.

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

[14] Y. Manabe and S. Aoyagi. A feasibility decision algorithm for rate monotonic and deadline monotonic scheduling. *Journal of Real-Time Systems*, 14(2):171–181, 1998.

[15] M. Park and Y. Cho. Feasibility analysis of hard real-time periodic tasks. *Journal of Systems and Softwares*, 73:89–100, 2004.

[16] P. Richard. Analyse du temps de réponse des systèmes temps réel. *actes de l'Ecole d'Eté Temps Réel (ETR'03), Toulouse*, (1):241–262, 2003.

[17] P. Richard, M. Richard, and F. Cottet. Analyse holistique des systèmes temps réel distribués: principes et algorithmes. *in: ordonnancement pour l'informatique parallèle, Traité IC2, Hermès*, page , juin 2003.

[18] I. Ripoll, A. Crespo, and A. Mok. Improvement in feasibility testing for real-time tasks. *Journal of Real-Time Systems*, 11(1):19–39, 1996.

[19] M. Spuri. Analysis of deadline scheduled real-time systems. *INRIA Reseach Report 2772*, page 34p, 1996.

[20] J. Stankovic, M. Spuri, M. DiNatale, and G. Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 28(6):16–25, 1995.