RenPar'10
Strasbourg, 9 juin - 12 juin 1998

Cyclicité des ordonnancements au plus tôt des systèmes de tâches temps réel

Emmanuel Grolleau, Annie Choquet-Geniet et Francis Cottet

LISI/ENSMA
Téléport 2
BP 109
86960 FUTUROSCOPE Cedex, FRANCE
grolleau,ageniet,cottet@ensma.fr

Résumé

Nous nous intéressons aux ordonnancements au plus tôt des systèmes de tâches temps réel à contraintes strictes qui peuvent se synchroniser et partager des ressources. Lorsque la validation d'un tel système est obtenue par simulation, la durée de simulation requise doit être connue. Dans le but de connaître cette durée, nous étendons et affinons les résultats connus concernant la cyclicité des ordonnancements de systèmes de tâches dans le cas général de tâches pouvant se synchroniser et partager des ressources, ceci sans restriction sur l'algorithme d'ordonnancement utilisé pourvu qu'il soit au plus tôt.

Mots-clés : Temps réel, ordonnancement, cyclicité, validation

1. Introduction

Un système temps réel se distingue des autres systèmes informatiques par le fait qu'il est soumis à des contraintes temporelles fortes, dues à la criticité de certaines actions, typiquement des actions qui permettent au système d'interagir avec l'environnement. Une application temps réel peut être vue comme un système de tâches se synchronisant et partageant des ressources telles que le processeur dans les systèmes monoprocesseurs, les canaux de transmission dans les systèmes répartis, ou la sortie vers un terminal de contrôle [11] dans le but de contrôler un procédé.

Nous nous plaçons ici dans un contexte monoprocesseur, et nous supposons que les tâches composant le système sont toutes périodiques. Nous nous intéressons à la correction du système. Dans le cadre temps réel, on entend par correction du système non seulement la correction algorithmique classique, avec en particulier l'assurance qu'aucun blocage n'interviendra, mais aussi la correction temporelle : chaque tâche doit s'exécuter à l'intérieur d'une plage de temps dont la taille dépend des caractéristiques physiques du procédé. L'étude de la correction temporelle du système passe par l'étude des ordonnancements possibles. On peut utiliser soit des techniques d'ordonnancement en ligne : *Earliest Deadline* [8], *Rate Monotonic* [9] avec ou sans protocole de gestion de ressources, soit des techniques hors ligne, ce qui revient à construire un séquenceur, où le choix de la séquence utilisée dépend de critères fixés par le concepteur, critères qui peuvent ne pas être exprimables par les seuls paramètres temporels, comme par exemple des critères d'importance des tâches [6]. Dans l'approche en-ligne, et sous des hypothèses restrictives [3][9][12], un calcul analytique permet de d'affirmer ou d'infirmer la validité du système. Cependant dans le cas général, seule la simulation permet de conclure. Dans une approche hors-ligne, le problème de la longueur de la séquence à construire se pose de façon immédiate. Donc dans les deux cas, la connaissance de la durée de la séquence à construire est primordiale.

Lorsque toutes les tâches sont synchrones au démarrage (toutes les tâches sont activées dès le lancement de l'application), la durée de la simulation à construire est le plus petit commun multiple ($PPCM$) des périodes des tâches que nous appelons métapériode et notons P. En effet le système doit se retrouver dans le même état global (mêmes horloges et mêmes compteurs ordinaux) à chaque métapériode. Cependant, lorsque les tâches ne sont pas synchrones au démarrage, le problème est moins trivial et il n'existe une preuve de cyclicité que dans le cas où les tâches sont indépendantes et ordonnancées par *Earliest Deadline*

[8]. Nous étudions ici les systèmes de tâches interagissantes (synchronisations, ressources) en supposant simplement que l'on utilise un ordonnancement au plus tôt : le système n'est inactif que si aucune tâche ne peut s'exécuter.

L'article est organisé de la façon suivante : nous présentons tout d'abord le modèle temporel utilisé, ainsi que les hypothèses structurelles faites. Nous montrons ensuite que si le système a une charge de 100%, le nombre de temps creux (temps d'inactivité) est borné, puis nous donnons une borne supérieure à l'occurence du dernier temps creux dans le cas où la charge est de 100%, enfin nous établissons la cyclicité des ordonnancements au plus tôt. Le lecteur pourra se référer à [7] pour des preuves formelles de ces résultats.

2. Définitions et notations

2.1. Modèle temporel

Un système de tâches temps réel est composé d'un ensemble de tâches cycliques $\tau_{i,i=1..n}$ caractérisées par quatre paramètres temporels définis initialement dans [9] :

- r_i la date de première activation
- C_i la charge, durée de traitement nécessaire à chaque exécution
- R_i le délai critique, temps donné à la tâche pour se terminer après chacune de ses activations
- T_i période d'activation

Une tâche est donc caractérisée temporellement par $\tau_i =< r_i, C_i, R_i, T_i >$. Nous appelons $P = PPCM_{i=1..n}(T_i)$ la métapériode du système, et $r = max_{i=1..n}(r_i)$ la date de réveil la plus tardive.

2.2. Hypothèses structurelles

Nous adoptons les hypothèses suivantes :

- Les tâches sporadiques sont prises en compte par des serveurs périodiques. En effet rappelons que nous nous intéressons à la validation hors-ligne et que celle-ci n'a de sens que si toutes les occurrences des tâches sont connues à l'avance.

- Les tâches peuvent se synchroniser et partager des ressources.

- Lorsque deux tâches se synchronisent, les fréquences de la tâche émettrice et de la tâche réceptrice doivent être corrélées. Par exemple si une tâche τ_i attend 2 messages de τ_j pendant chacune de ses exécutions et que τ_j ne lui envoie qu'un message, alors $T_j = 2 \times T_i$.

- Une tâche ne peut pas attendre de message de synchronisation à l'intérieur d'une section critique. En effet il est préférable qu'une tâche sorte le plus vite possible de sa section critique.

3. Les temps creux

3.1. Temps creux cycliques

La charge totale d'un système de tâches est donnée par $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$. Lorsque celle-ci est inférieure à 100% ($U < 1$), il y a au moins $P(1-U)$ temps creux (temps pendant lequel le processeur ne fait rien) lors de chaque métapériode [9]. Ces temps creux sont donc cycliques et peuvent être "absorbés" par une tâche de fond $\tau_0 =< 0, P(1-U), P, P >$. On peut alors toujours se ramener à un système de tâche de charge 100% en augmentant ce dernier d'une tâche de fond. Nous nous limiterons donc dans la suite à étudier ce cas.

3.2. Temps creux acycliques

Lorsque les tâches ne sont pas synchrones au démarrage, il peut apparaître des temps creux acycliques. En effet considérons le système de tâches $S = \{\tau_1 =< 0, 1, 4, 4 >, \tau_2 =< 1, 3, 6, 6 >, \tau_3 =< 3, 1, 4, 4 >\}$ dont l'ordonnancement par *Earliest deadline* est donné sur la figure 1.

Bien que la charge de ce système soit maximale, un temps creux apparaît au temps 6, résidu des perturbations engendrées par la monté en charge du système. Lorsqu'un système est à charge maximale, P unités de temps de traitement dûes aux activations des tâches sont générées à chaque métapériode (de durée P), donc pour une entier $N \in \mathcal{N}$ arbitrairement grand, à la date $r + NP$, le processeur aura dû traiter au moins NP unités de temps du système. Ce qui borne naturellement le nombre de temps creux

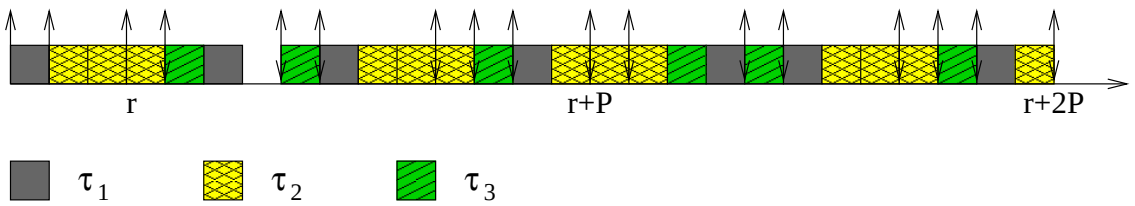

FIG. 1 – *Temps creux acyclique dans un ordonnancement*

acycliques d'un ordonnancement par r. On pourrait affiner facilement cette borne mais là n'est pas notre propos.

L'existence de cette borne permet de dégager une remarque importante. Pour une séquence d'ordonnancement MC^*, où M est la partie non cyclique et où C est la partie cyclique, aucun temps creux acyclique ne peut se trouver dans le cycle C. On sait de plus que le cycle a une durée multiple de la métapériode. D'où une question primordiale: le dernier temps creux a-t-il une date d'occurence limite?

On peut remarquer qu'un temps creux acyclique peut avoir lieu soit parce-que le processeur n'a rien à faire car toutes les tâches sont terminées en attente d'une nouvelle activation, soit parce-que toutes les tâches actives sont bloquées à cause d'une (ou plusieurs) attente de message, qui doit être émis par une tâche non réveillée. Typiquement une tâche réceptrice est bloquée en attente de message d'une tâche émettrice qui a été réveillée tardivement par rapport à elle. Dans ce cas il se crée des chaînes de blocage dans lesquelles la source est une tâche émettrice tardive, et dont chaque tâche τ_i est reliée à la tâche précédente τ_j dans la chaîne par une attente de message de τ_j correspondant à une émission suivant le point de blocage de τ_j. En aucun cas l'utilisation d'une ressource ne pourra influer sur une chaîne de blocage car d'après les hypothèses de départ une tâche ne peut pas attendre de message à l'intérieur d'une section critique. Etant donné que les tâches se synchronisant ont des périodes corrélées, les chaînes de blocage à un instant donné se retrouvent une métapériode plus tard et elles sont identiques à celles existant une métapériode plus tôt (sauf si les tâches la composant ne sont pas réveillées à cet instant).

3.3. Date d'occurence du dernier temps creux acyclique

Nous raisonnons sur la charge instantanée du système (charge effective à traiter par le processeur à un instant donné). Puisque la charge est maximale, P unités de temps (u.t.) de charge à traiter sont générées pendant chaque métapériode à partir de la date r, donc à un instant donné, la charge instantanée est inférieure ou égale à ce qu'elle sera P unités de temps plus tard. En effet on ne peut traiter avec un processeur plus de P u.t. en P u.t.

Lemme 1 : Si il existe un temps creux à la date $r + \Delta$ ($\Delta \geq 0$), il n'y en a pas en $r + P + \Delta$.

Preuve : Supposons qu'il existe un temps creux à la date $r+\Delta$, alors soit la charge instantanée en $r+\Delta$ est nulle, soit elle fait partie de tâches bloquées dans une chaîne de blocage. Or le temps creux implique que le processeur ne peut traiter au maximum que $P-1$ u.t. entre $r+\Delta$ et $r+P+\Delta$, ce qui signifie que la charge instantanée en $r + P + \Delta$ est strictement supérieure à la charge instantanée en $r + \Delta$. Comme les chaines de blocages se retrouvent toutes les métapériodes, la charge excédentaire ne fait pas partie d'une tâche bloquée, et il n'y a pas de temps creux en $r + P + \Delta$.□

Lemme 2 : Si il n'y a pas de temps creux à la date $r + \Delta$ ($\Delta \geq 0$), il n'y en a pas en $r + P + \Delta$.

Preuve : Supposons qu'il n'y ait pas de temps creux en $r+\Delta$, comme la charge instantanée est au moins égale P u.t. plus tard, et qu'il n'y a pas plus de tâches bloquées (périodicité des chaînes de blocage), le processeur pourra travailler en $r + P + \Delta$ ce qui signifie qu'il n'y aura pas non plus de temps creux à cette date.□

Il découle des lemmes 1 et 2 qu'il ne peut pas y avoir de temps creux à partir de la date $r + P$.

4. Cyclicité des ordonnancements

Posons $t_c < r + P$ la date du dernier temps creux acyclique. Si il n'y a aucun temps creux acyclique nous posons $t_c = -1$. Puisqu'en t_c il y a un temps creux, il y a deux configurations possibles au temps t_c :

– La charge instantanée est nulle. Entre t_c et $t_c + P$, $P - 1$ u.t. sont traitées, et P u.t. de traitement

sont induites par les activations des tâches. Il reste donc une u.t. à traiter à la date $t_c + P$, et cette u.t. ne peut faire partie d'une chaîne de blocage sinon elle aurait fait partie d'une chaîne de blocage à l'instant t_c. Elle est donc traitée et on se retrouve dans la même configuration à la date $t_c + P + 1$ qu'à la date $t_c + 1$.

- La charge instantanée fait partie de chaînes de blocage. Etant donné la cyclicité des chaînes de blocage, le même raisonnement que pour le cas précédent peut être appliqué.

Les ordonnancements au plus tôt sont donc cycliques de période P à partir du dernier temps creux acyclique qui a lieu avant $r + P$. Le temps de simulation requis d'un système de tâches est donc au mieux $r + P$ et au pire $r + 2P$ suivant les cas.

5. conclusion

Nous avons montré que tout ordonnancement valide au plus tôt d'un système de tâches temps réel était cyclique de période P à partir du dernier temps creux acyclique. Pour ordonnancer un système avec une politique d'ordonnancement au plus tôt, on commence donc par l'ordonnancer jusqu'à r+P. A cette date, la date t_c du dernier temps creux est connue : on compte le nombre de temps creux, s'il n'y en a pas, on considère que $t_c = -1$, sinon c'est la date du dernier temps creux trouvé qui est t_c. Il ne reste alors plus qu'à ordonnancer, si ce n'est déjà fait (cas où $t_c < r$), jusqu'à $t_c + P + 1$.

Comme seules les charges et dates d'activations sont prises en compte pour la démonstration, le résultat s'applique aussi sur les systèmes de tâches partageant des ressources, quel que soit le protocole de gestion de ressources utilisé (protocole à priorité plafond [2][5], protocole à priorité héritée [10], gestion des ressources par pile [1]). Ceci s'explique par le fait que le blocage d'une tâche par l'attente d'une ressource ne peut pas globalement bloquer le système de tâches.

Ce résultat est intéressant à double titre : d'une part il offre une durée de simulation généralisée à tout algorithme au plus tôt, inférieure à celle utilisée jusqu'à maintenant [8]. D'autre part il permet, dans le cadre de l'ordonnancement hors-ligne [4][6], de construire une séquence d'ordonnancement plus petite, en général, que $r + 2P$, ce qui diminue sensiblement la taille de la séquence d'ordonnancement à stocker pour le séquenceur.

Biblographie

1. T.P. Baker – Stack-Based Scheduling of Realtime Processes – Journal of Real-Time Systems, 3, 1991.
2. M. Chen, K. Lin – Dynamic Priority Ceilings : A Control Protocol for Real-Time Systems – Journal of Real-Time Systems, 2, 1990.
3. H. Chetto, M. Silly, T. Bouchentouf – Dynamic Scheduling of Real-Time Tasks Under Precedence Constraints – Journal of Real-Time Systems, 2, 1990.
4. A. Choquet-Geniet, D. Geniet, F. Cottet – Exhaustive Computation of the Scheduled Task Execution Sequences of a Real-Time Application – proc. 4th International Symposium on FTRTFTS, Uppsala, Sweden, 09/11-13/96.
5. J.B. Goodenough, L. Sha – The Priority Ceiling Protocol : A Method for Minimizing the Blocking of High Priority Ada Tasks – Proc. 2nd ACM Int. Workshop Real-Time Ada Issues, 1988.
6. E. Grolleau, A. Choquet-Geniet, F. Cottet – Ordonnancement Optimal de Systèmes de Tâches Temps Réel à l'Aide de Réseaux de Petri – proc. AGIS'1997, 12/9-11/1997.
7. E. Grolleau, A. Choquet-Geniet, F. Cottet – Cyclicité des Séquences d'Ordonnancement au Plus Tôt des Systèmes de Tâches Temps réel à Contraintes Strictes – rapport de recherche LISI/ENSMA n^o 97007.
8. J.Y.T. Leung, M.L. Merrill – A Note on Preemptive Scheduling of Periodic Real-Time Tasks – Information Processing Letters 11 (3), pp 115-118, 1980.
9. C.L. Liu, J.W. Layland – Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment – journal of the ACM, 20 (1), pp 46-61, 1973.
10. L. Sha, R. Rajkumar, J.P. Lehoczky – Priority Inheritance Protocols : An Approach to Real-Time Synchronisation – IEEE Transactions on Computers, Vol. 39, No. 9, pp. 1175-1185, September 1990.
11. J.A. Stankovic – Misconception about Real-Time Computing – IEEE Computer Magazine, 21 (10), 1pp 0-19, 1988.

12. J.A. Stankovic, M. Spuri, M. Di Natale, G. Buttazo – Implications of Classical Results for Real-Time Systems – IEEE Computer, vol. 28, n. 6, pp1-25, 1995.