Real-Time Systems, RTS'2000, Paris, 28 et 29 mars 2000

Cyclicité des ordonnancements de systèmes de tâches périodiques différées

Emmanuel Grolleau, Annie Choquet-Geniet
Laboratoire d'Informatique Scientifique et Industrielle
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Téléport 2
1, avenue Clément Ader
BP 40109
86961 FUTUROSCOPE CHASSENEUIL Cedex
grolleau,ageniet@ensma.fr

Cyclicité des ordonnancements de systèmes de tâches périodiques différées

Emmanuel Grolleau, Annie Choquet-Geniet
LISI-ENSMA
Téléport 2
1, avenue Clément Ader
BP 40109
86961 FUTUROSCOPE CHASSENEUIL Cedex
grolleau,ageniet@ensma.fr

Résumé : Nous montrons que toute séquence d'ordonnancement valide construite par un algorithme conservatif (i.e. au plus tôt), pour un système de tâches périodiques (pouvant partager des ressources et communiquer) différées, est cyclique de période le *PPCM* des périodes des tâches à partir du dernier temps creux acyclique. Nous améliorons et généralisons ainsi la borne proposée par Leung et Merrill. Ce résultat permet de borner la durée de simulation d'ordonnancements en-ligne, et de connaître la longueur nécessaire de construction de séquences d'ordonnancement hors-ligne.

1. Introduction

Un système temps réel se distingue des autres systèmes informatiques par le fait qu'il est soumis à des contraintes temporelles, dues à la criticité de certaines actions qui lui permettent d'interagir avec l'environnement d'un procédé à contrôler [15]. Il peut être modélisé par un système de tâches périodiques [11], qui peuvent communiquer et partager des ressources comme dans la plupart des systèmes informatiques.

Valider un système temps réel consiste en une validation algorithmique et temporelle des tâches. Nous nous intéressons à l'aspect temporel de la validation des systèmes temps réel, c'est à dire à l'ordonnancement de tâches périodiques.

Dans le cas de systèmes de tâches indépendantes, il existe des algorithmes d'ordonnancement de complexité polynomiale basés sur des priorités allouées aux tâches, dont la validation temporelle peut être obtenue à l'aide de techniques analytiques ne nécessitant pas la construction de séquences d'ordonnancement [10, 11, 16]. Ces méthodes, dites en-ligne, ont été étendues aux systèmes de tâches soumises à des contraintes de précédence (i.e. tâches communicantes)[4, 7]. Cependant, dès lors que les tâches partagent des ressources critiques, le problème de l'ordonnancement est NP-difficile [12]. En effet, en dehors des risques d'interblocages dus aux ressources, l'un des phénomènes les plus caractéristiques engendré par l'adjonction de ressources dans un système est l'inversion de priorité, le temps perdu par une tâche à cause de l'inversion de priorité n'étant, dans le cas général, pas borné. Pour palier à ce problème, les algorithmes classiques d'ordonnancement ont été enrichis de protocoles de gestion de ressources, tels le protocole à priorité héritée [13], le protocole à priorité plafond [6, 13, 14] ou le protocole de gestion des ressources par piles [2]. La propriété de tels protocoles est de limiter, voire de borner la durée des blocages dus aux attentes de ressources.

La validation de telles approches s'effectue soit par la vérification de conditions suffisantes analytiques, soit par simulation, c'est à dire par la construction effective de la séquence d'ordonnancement. Cependant, si l'ordonnancement construit doit être implanté en-ligne (i.e. par ordonnanceur, qui implémente une politique d'ordonnancement, et non par séquenceur, qui suit une séquence prédéfinie), la durée des sections critiques des tâches doit être augmentée de la durée de blocage de celles-ci, qui dépend de l'algorithme d'ordonnancement et du protocole de gestion des ressources. Après cette intégration, si certaines sections critiques sont relativement longues, la charge théorique du système peut dépasser la capacité de traitement du processeur, ce qui rend le système non ordonnançable.

Une approche complémentaire de l'ordonnancement en ligne est donc requise, notamment pour les systèmes de tâches fortement interagissantes. En effet, si une séquence construite hors ligne, sans prise en compte de la durée de blocage des tâches lors de leur entrée en section critique, est valide, alors son implantation via un séquenceur respecte les contraintes temporelles. Différentes approches d'ordonnancement hors ligne ont été étudiées dans la littérature : algorithmes génétiques, recuit simulé [3], logique temporelle [1, 8], énumération arborescentes [5, 17] et réseaux de Petri [9].

Que l'approche choisie soit en-ligne avec une validation par simulation ou bien hors-ligne, il est nécessaire de connaître la durée de la séquence d'ordonnancement à construire. Cette durée est aussi nécessaire à l'étude de performance de certains algorithmes en-ligne même si ceux-ci peuvent être validés de manière analytique. Si toutes les tâches sont réveillées initialement au même instant (on parle de tâches simultanées), alors la durée de simulation est connue et est le *PPCM* des périodes des tâches. Par contre, lorsque certaines tâches sont réveillées tardivement (tâches différées), la durée de simulation n'est connue à ce jour que dans un cas particulier : si l'algorithme d'ordonnancement est *earliest deadline* et que le système étudié est composé de tâches indépendantes, alors la durée de simulation peut être bornée par la date de réveil la plus tardive des tâches plus deux fois le *PPCM* des périodes [10]. Nous généralisons et améliorons ce résultat en donnant une durée de simulation optimale pour

l'ensemble des algorithmes conservatifs (dits aussi au plus tôt) ordonnançant des systèmes de tâches pouvant partager des ressources et être soumis à des contraintes de précédences.

En section 2, nous abordons le concept de temps creux acyclique, qui nous le verrons, est la clé de la cyclicité des ordonnancements, et nous montrons la cyclicité des ordonnancements de systèmes de tâches indépendantes de charge processeur maximale pour tout algorithme d'ordonnancement conservatif, puis nous étendons le résultat aux systèmes de tâches de charge quelconque. En section 3, la preuve est étendue au cas des tâches communicantes et partageant des ressources.

2. Cyclicité des ordonnancements de tâches indépendantes

2.1. Modèle de tâches temps réel

Nous présentons le modèle temporel des tâches que nous utilisons dans la suite. Chaque tâche τ_i est caractérisée temporellement par quatre paramètres temporels :

- r_i sa date d'activation, ou première date de réveil ;
- C_i sa charge, c'est à dire la durée que le processeur doit lui consacrer pour exécuter chacune de ses occurrences ;
- D_i son délai critique, représente le temps qui lui est accordé après chacune de ses activations pour être exécutée. La date correspondante se nomme l'échéance ;
- P_i sa période d'activation ;

Le modèle utilisé est déterministe, et toutes les dates et les durées prennent des valeurs entières : en effet, le temps est discrétisé en quanta de temps correspondant à l'unité de préemption de l'architecture cible. Chaque tâche est (ré-)activée périodiquement aux dates r_i+kP_i pour $k \in \mathbb{N}$, chacune des occurrences doit respecter son échéance donnée par $r_i+kP_i+D_i$. La figure 1 représente graphiquement les paramètres temporels d'une tâche τ_i. Nous dénotons une tâche $\tau_i<r_i,C_i,D_i,P_i>$ et un système de tâches $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$. On note $P=PPCM_{i=1..n}(P_i)$ la méta période d'un système et $r=max_{i=1..n}\{r_i\}$ sa date de réveil la plus tardive.

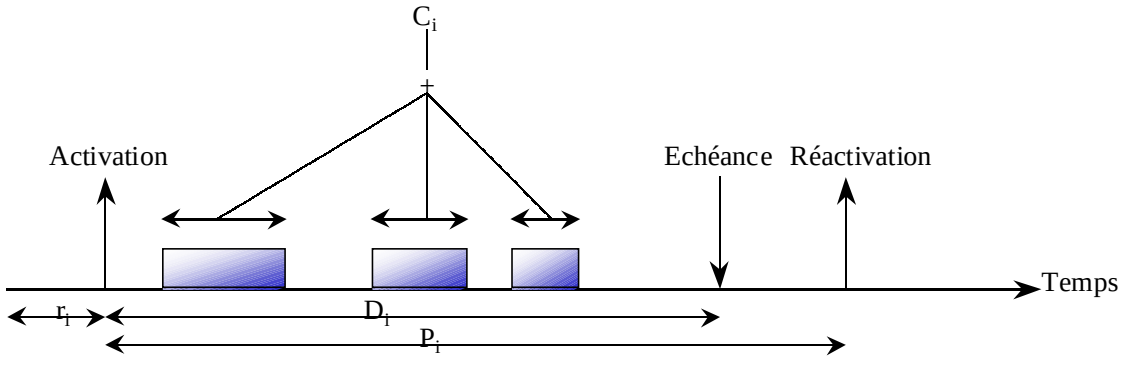

figure 1 : modèle temporel d'une tâche périodique

L'une des caractéristiques importantes d'un système de tâches S est sa charge processeur $U_S = \sum_{i=1}^{n} \frac{C_i}{P_i}$. En effet, puisque $\frac{C_i}{P_i}$ est la fraction du processeur requise par

la tâche τ_i, U_S est la fraction du processeur requise par le système de tâches. Si U_S>1, le système S ne peut pas être ordonnancé sur un seul processeur, et si U_S<1, alors le processeur n'est pas totalement employé par S, et il sera inactif (on dit aussi oisif) pendant $P(1-U_S)$ unités de temps toutes les P unités de temps. Ces temps creux apparaissent périodiquement à chaque méta période.

2.2. Exemple

Dans cette partie, nous abordons le problème de la cyclicité à travers un exemple. Soit $S_1=\{\tau_1<0,1,4,4>, \tau_2<1,3,6,6>, \tau_3<3,1,4,4>\}$ un système de tâches indépendantes. Sa charge est $U_{S_1}=1$, il n'y a donc pas de temps creux correspondant au fait que la charge du système est inférieure à 1 dans les ordonnancements de S.

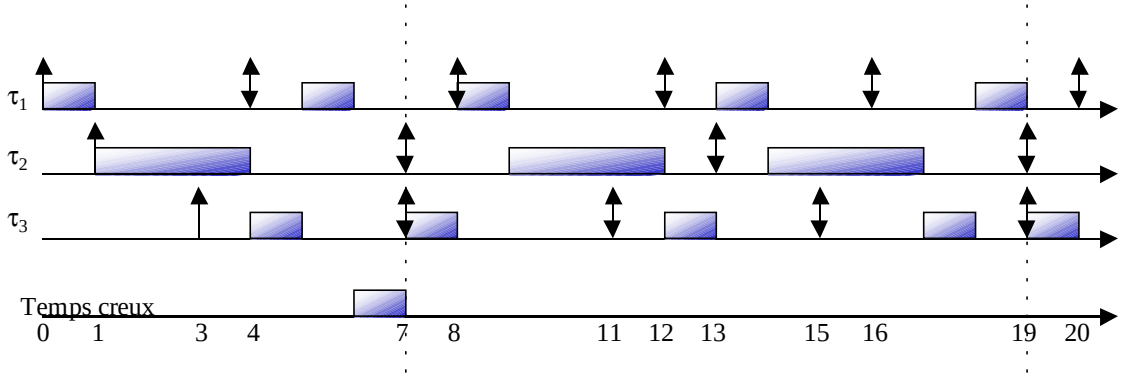

figure 2 : séquence d'ordonnancement de S_1 produite par earliest deadline

Cependant, la séquence d'ordonnancement de S_1 produite par *ealiest deadline* montre la présence d'un temps creux à l'instant *6*. De plus, l'étude de cette séquence montre qu'elle est dans le même état[1] juste après ce temps creux (à l'instant *7*) que P unités de temps plus tard (à l'instant *7+12=19*). La séquence d'ordonnancement infinie σ peut donc être exprimée par $\sigma=\sigma_{[0..7[}\sigma^*_{[7..19[}$ où $\sigma_{[a..b[}$ est la restriction de σ à l'intervalle $[a..b[$, et l'étoile est l'étoile de Kleene. Donc la durée minimale permettant d'exprimer la séquence produite par *earliest deadline* est *19* unités de temps (contre $3+2\times12=27$ unités de temps donné par [10]).

On peut constater sur cet exemple que le temps creux n'apparaît qu'une fois dans toute la séquence, nous le qualifions donc de temps creux acyclique. Par opposition, les temps creux correspondant au fait que la charge est inférieure à 1 sont nommés temps creux cycliques. De plus, la séquence est cyclique de période P après la date d'occurrence de ce temps creux acyclique. Dans la suite, nous montrons que pour tout système de tâches, et pour toute séquence produite par un algorithme conservatif, la date suivant le dernier temps creux acyclique correspond toujours au début du cycle de la séquence, et que le cycle dure P unités de temps.

[1] L'état du système à un instant regroupe le compteur ordinal des tâches (c'est à dire l'endroit où leur traitement en est) et leurs horloges locales (i.e. le temps restant avant leur prochaine activation, et de manière équivalente avant leur prochaine échéance)

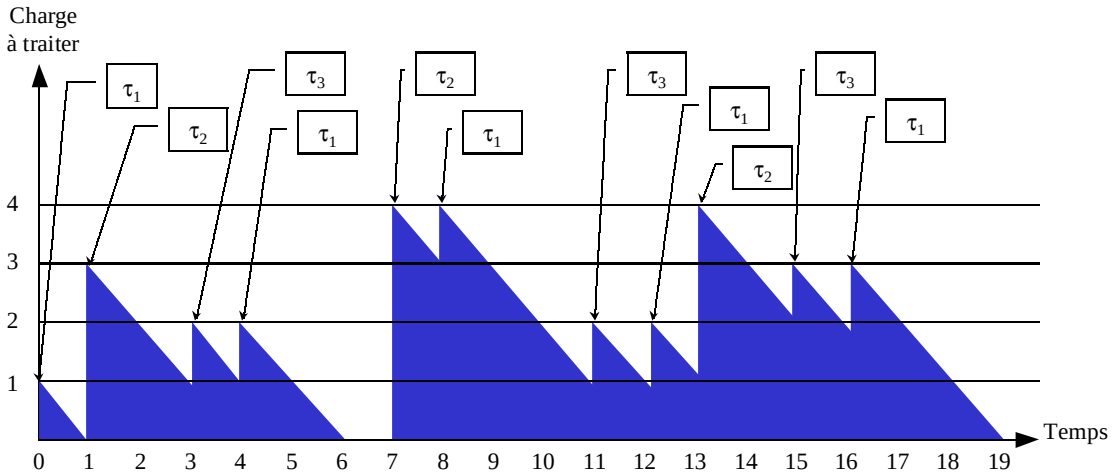

figure 3 : diagramme de charge représentant à chaque instant la charge restant à traiter pour le système S_1 lorsqu'un algorithme conservatif est utilisé : si la charge restant à traiter est nulle, alors il y a un temps creux (comme à l'instant 6), sinon, le processeur traite une unité de temps d'une des tâches actives du système

La figure 3 montre que la position des temps creux est indépendante de l'algorithme conservatif utilisé : en effet, pour les systèmes de tâches indépendantes, quel que soit l'algorithme conservatif choisi, un temps creux n'apparaît que lorsque toutes les tâches actives sont terminées, dans le cas contraire, le processeur traite un quantum de temps de tâche.

2.3. Etude des temps creux acycliques

Nous avons vu que les temps creux acycliques jouaient un rôle prédominant dans la cyclicité des ordonnancements. Nous les étudions donc de façon détaillée dans cette section. Dans un premier temps, nous montrons qu'ils sont en nombre borné, et qu'ils ne peuvent donc pas apparaître dans la partie cyclique d'un ordonnancement. Ensuite, nous montrons que leur date d'occurrence est bornée par $r+P$. Enfin, nous en déduisons le résultat de cyclicité des ordonnancements.

Afin de faciliter cette étude de prime abord, nous considérons des systèmes de tâches indépendantes de charge maximale (i.e. U_S=1). Nous généralisons ensuite la preuve.

2.3.1 Cas des systèmes de tâches de charge maximale

Lemme 1 : Soit $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$ un système de tâches avec U_S=1. Le nombre de temps creux dans une séquence valide de S est borné.

Preuve : Nous notons $k_i=\left\lfloor \frac{r-r_i}{P_i} \right\rfloor$ le nombre de requêtes de τ_i terminées dans l'intervalle [0..r[, où $\lfloor a \rfloor$ dénote la partie entière inférieure de a.

Soit $N \in \mathbb{N}$ un entier arbitrairement grand et voyons sur la figure 4 le nombre de requêtes de τ_i terminées dans l'intervalle [*0..r+NP*[: elle doit être exécutée entièrement $k_i + \frac{NP}{P_i}$ fois dans cet intervalle.

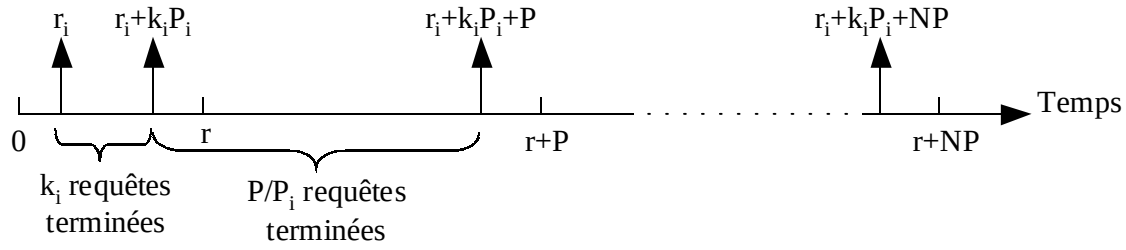

figure 4 : requêtes de τ_i au cours du temps

Donc le processeur doit lui consacrer au moins $C_i(k_i + \frac{NP}{P_i})$ unités de temps sur cet intervalle. En généralisant le raisonnement à l'ensemble des tâches, le processeur doit consacrer au moins $\sum_{i=1}^{n} k_i C_i + \frac{NPC_i}{P_i}$ unités de temps au système dans l'intervalle [*0..r+NP*[. Puisque U_S=1, le processeur doit consacrer au moins $NP + \sum_{i=1}^{n} k_i C_i$ unités de temps au système. Donc il ne peut pas rester oisif plus de $r - \sum_{i=1}^{n} k_i C_i$ unités de temps sur une durée infinie.

❒

Nous pouvons donc en déduire que sous l'hypothèse de charge maximale, aucun temps creux ne peut apparaître dans la partie cyclique d'un ordonnancement.

Lemme 2 : Soit $\sigma = \sigma_a \sigma_c^*$ une séquence d'ordonnancement infinie d'un système de tâches de charge maximale, où σ_a est la partie acyclique de σ et σ_c est sa partie cyclique. σ_c ne contient aucun temps creux.

Preuve : Découle du Lemme 1.

❒

Ce lemme montre que dans tous les cas, le début du cycle des ordonnancements ne peut pas commencer avant le dernier temps creux acyclique. Afin de raisonner sur les algorithmes d'ordonnancements conservatifs, nous les caractérisons dans les définitions suivantes.

Définition 1 : La requête instantanée d'une tâche τ_i est définie par :

$$\begin{cases} C_{req} : Temps \times Tâche \rightarrow Requête\ \mathrm{Pr}ocesseur \\ C_{req}(t, \tau_i) = \begin{cases} C_i \text{ si } (t - r_i) \equiv 0\,[P_i] \\ 0 \text{ sinon} \end{cases} \end{cases}$$

Où *Temps* est un ensemble de dates (entiers), *Tâches* est l'ensemble des tâches d'un système, *Requête Processeur* est un entier correspondant au temps nécessaire au traitement de la tâche τ_i à la date *t*, et $a \equiv b\ [c]$ signifie que *a* est congru à *b* modulo *c*.

La requête instantanée d'un système *S* est définie par :

$$\begin{cases} C_{req} : Temps \rightarrow Requête\,\text{Pr}\,ocesseur \\ C_{req}(t) = \sum_{i=1}^{n} C_{req}(t, \tau_i) \end{cases}$$

En d'autres termes, la requête instantanée d'une tâche τ_i est C_i lorsque celle-ci est activée à l'instant considéré, et *0* sinon.

Définition 2 : La charge instantanée d'un système lorsque les tâches sont indépendantes et qu'il est ordonnancé par un algorithme conservatif est définie par :

$$\begin{cases} C_{rest} : Temps \rightarrow Requête\,\text{Pr}\,ocesseur \\ C_{rest}(0) = C_{req}(0) \\ C_{rest}(t) = C_{req}(t) + \max\{C_{rest}(t-1) - 1, 0\} \end{cases}$$

Cette définition caractérise le fonctionnement des algorithmes d'ordonnancement conservatifs : si la charge restante à un instant est nulle, alors il y a un temps creux (i.e. le processeur ne traite pas de tâche), sinon, le processeur traite une unité de temps de charge jusqu'à l'instant suivant.

Lemme 3 : Soit $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$ un système de tâches avec U_S=1. $\forall\Delta\in I\!N$, $\sum_{t=0}^{P-1} C_{req}(r+\Delta+t) = P$.

Preuve : Le preuve se base sur la définition de la charge. Après la date *r*, chaque tâche τ_i est activée exactement $\frac{P}{P_i}$ fois toutes les *P* unités de temps. Il en résulte que le processeur doit traiter $\sum_{i=1}^{n} \frac{PC_i}{P_i} = P$ unités de charge en *P* unités de temps.

❒

Il en découle le lemme suivant :

Lemme 4 : Soit $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$ un système de tâches avec U_S=1. $\forall\Delta\in I\!N$, $C_{rest}(r+P+\Delta)\geq C_{rest}(r+\Delta)$.

Preuve : D'après le Lemme 3, à partir de la date *r*, le processeur doit traiter *P* unités de temps toutes les *P* unités de temps. La charge instantanée d'un système ne peut donc pas décroître.

❒

Lemme 5 : Soit $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$ un système de tâches avec U_S=1. Si il y a un temps creux après *r*, alors il n'y a pas de temps creux une méta-période plus tard, i.e. $\forall\Delta\in I\!N$, $C_{rest}(r+\Delta)=0 \Rightarrow C_{rest}(r+P+\Delta)>0$.

Preuve : en sommant la Définition 2 sur une méta-période, on obtient :

$$\sum_{i=1}^{P} C_{rest}(r+\Delta+i) = \sum_{i=1}^{P} C_{req}(r+\Delta+i) + \max\{0, C_{rest}(r+\Delta+i-1) - 1\}$$

Puisque U_S=1, d'après le Lemme 3,

$$\sum_{i=1}^{P} C_{rest}(r+\Delta+i) = P + \sum_{i=1}^{P} \max\{0, C_{rest}(r+\Delta+i-1) - 1\}$$

Puisque par hypothèse, $C_{rest}(r+\Delta)=0$,

$$\sum_{i=1}^{P} C_{rest}(r+\Delta+i) = P + \sum_{i=2}^{P} \max\{0, C_{rest}(r+\Delta+i-1)-1\}$$

$$\sum_{i=1}^{P} C_{rest}(r+\Delta+i) \geq P + \sum_{i=2}^{P} C_{rest}(r+\Delta+i-1)-1$$

$$\sum_{i=1}^{P} C_{rest}(r+\Delta+i) \geq P + \sum_{i=1}^{P-1} C_{rest}(r+\Delta+i)-1$$

$$\sum_{i=1}^{P} C_{rest}(r+\Delta+i) \geq P - P + 1 + \sum_{i=1}^{P-1} C_{rest}(r+\Delta+i)$$

$$C_{rest}(r+P+\Delta) \geq 1$$

❒

Lemme 6 : Soit $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$ un système de tâches avec $U_S=1$. Si il n'y a pas de temps creux après r, alors il n'y en a pas une méta-période plus tard, i.e. $\forall\Delta\in I\!N$, $C_{rest}(r+\Delta)>0 \Rightarrow C_{rest}(r+P+\Delta)>0$.
Preuve : découle du Lemme 4.
❒

Proposition 1 : Soit $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$ un système de tâches avec $U_S=1$. Il n'y a plus de temps creux à partir la date $r+P$.
Preuve : découle du Lemme 5 et du Lemme 6.
❒

Définition 3 : Nous notons t_c la date d'occurrence du dernier temps creux acyclique, et nous posons $t_c=-1$ si il n'y a pas de temps creux acyclique.

La Proposition 1 montre qu'il ne peut plus y avoir de temps creux après la date $r+P$, c'est à dire que $t_c<r+P$. La proposition suivante montre qu'après le dernier temps creux, le système se trouve dans le même état qu'une méta-période plus tard.

Proposition 2 : Soit $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$ un système de tâches avec $U_S=1$. Tout algorithme d'ordonnancement conservatif valide sur $[0..t_c+P+1[$ laisse le système dans le même état à la date t_c+P+1 qu'à la date t_c+1.
Preuve : D'après la Définition 3, $C_{rest}(t_c)=0$, donc $C_{rest}(t_c+1)=C_{req}(t_c+1)$. Puisque t_c est la date du dernier temps creux, aucun temps creux n'apparaît après t_c. Cela implique que le système de tâches requiert P unités de temps de traitement pendant chaque méta-période après la date t_c+1, car si il y avait moins de requête instantanée, il y aurait d'autres temps creux. Donc, la Définition 2 devient, $\forall\Delta\in I\!N$,

$$C_{rest}(t_c+1+\Delta) = C_{req}(t_c+1+\Delta) + C_{rest}(t_c+\Delta)-1$$

En sommant cette définition sur une méta-période, on obtient

$$\sum_{i=1}^{P} C_{rest}(t_c+1+i) = \sum_{i=1}^{P} C_{req}(t_c+1+i) + C_{rest}(t_c+i)-1$$

$$\sum_{i=1}^{P} C_{rest}(t_c+1+i) = P - P + \sum_{i=1}^{P} C_{rest}(t_c+i)$$

$$C_{rest}(t_c+P+1) = C_{rest}(t_c+1)$$

Puisqu'à la date t_c+1, toutes les tâches n'ont pas forcément été activées, $C_{req}(t_c+P+1) \geq C_{req}(t_c+1)$. De plus, par définition, $C_{rest}(t_c+P+1) \geq C_{req}(t_c+P+1)$, or

$C_{rest}(t_c + P + 1) = C_{rest}(t_c + 1)$, ce qui implique $C_{rest}(t_c+P+1)= C_{req}(t_c+P+1)= C_{req}(t_c+1)$. La charge restant à traiter à l'instant t_c+P+1 est donc donnée par les activations des tâches à l'instant t_c+1, comme c'est le cas à l'instant t_c+1.

❒

Théorème 1 : Soit $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$ un système de tâches avec $U_S=1$.

- Une séquence σ créée par un algorithme conservatif est valide si et seulement si elle est valide sur l'intervalle $[0..t_c+P+1[$. La séquence est alors donnée par $\sigma=\sigma_{[0..t_c+1[}\sigma^*_{[t_c+1..t_c+P+1[}$;
- $t_c<r+P$;
- La longueur $[0..t_c+P+1[$ pour σ est minimale.

Preuve : La première partie découle de la Proposition 1, la seconde partie est la Proposition 1, et la troisième est impliquée par le fait qu'aucun temps creux ne peut se trouver dans la partie cyclique de la séquence (d'après le Lemme 2), et que par définition du *PPCM*, *P* est la longueur minimale que peut avoir un cycle afin que les horloges locales des tâches se retrouvent dans le même état.

❒

La détermination du cycle d'un ordonnancement est donc sujette au calcul de t_c. Ce calcul peut être effectué à l'aide d'un diagramme de charge, et de façon équivalente à l'aide de l'algorithme suivant :

```
tc ← -1
Crest ← Creq(0)
Pour t allant de 0 à r+P faire
      Si Crest=0 alors tc ← t
      Sinon Crest ← Crest-1
      FinSi
      Crest ← Crest+Creq(t+1)
Fait
```

2.3.2 Cas des systèmes de tâches de charge quelconque

Le théorème de cyclicité n'est pas encore très utilisable car il nécessite que la charge du système soit maximale, ce qui est relativement rare. Dans cette partie, nous le généralisons donc au cas des systèmes de tâches de charge inférieure à 1. Pour cela, nous introduisons, de façon instrumentale, une tâche oisive prenant en compte les temps creux cycliques. En effet, étant donné que toute séquence d'ordonnancement d'un système $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$ possède exactement $P(1-U_S)$ temps creux cycliques dans chaque intervalle de la taille d'une méta-période, la séquence produite par un algorithme d'ordonnancement conservatif sur *S* est identique à la séquence d'ordonnancement produite par le même algorithme sur $S'=S \cup \{\tau_0<r_0,P(1-U_S),P,P>\}$ dans laquelle la priorité de la tâche oisive est minimale, quel que soit r_0 (remarquons que le fait de ne pas forcer sa priorité à être minimale permettrait d'obtenir des séquences non-conservatives, et que notre résultat de cyclicité reste valide dans ce cas). En effet, dans ce cas, la tâche oisive se voit affecter le processeur lorsqu'il n'y a aucune autre tâche à traiter, ce qui correspond exactement aux emplacements des temps creux

cycliques en l'absence de cette tâche. Pour le moment, nous laissons sciemment de côté la date de réveil de la tâche τ_0. Les preuves de la section précédente ne sont pas modifiées, et le résultat de cyclicité reste valide dans ce cas.

La difficulté supplémentaire, par rapport au cas où la charge est maximale, tient à la différentiation entre les temps creux cycliques et acycliques. En effet, qu'ils soient cycliques ou acycliques, les temps creux représentent le même concept : l'inactivité du processeur. Afin de conserver une longueur de séquence minimale, il convient de faire une interprétation correcte des temps creux. Une façon possible de procéder est la suivante : un diagramme de charge donne l'ensemble des temps creux présents dans l'intervalle $[0..r+P[$: si il existe des temps creux acycliques, alors ils se trouvent dans cet intervalle. Si cet intervalle contient plus de $P(1-U_S)$ temps creux, alors il est possible que certains d'entre eux soient acycliques. Afin de minimiser la longueur de la séquence à produire, nous choisissons de considérer que le premier temps creux cyclique suit le dernier temps creux acyclique. Soit t la date d'occurrence du premier temps creux dans la séquence. Si il y a plus de $P(1-U_S)$ temps creux dans l'intervalle $[t..t+P[$, alors ce temps creux est acyclique, et on recommence le processus sur le deuxième temps creux. Sinon, il n'y a pas de temps creux acyclique. L'algorithme suivant détaille formellement la méthodologie :

```
C_rest ← C_req(0)
temps_creux :liste ← [-1]
-- Liste servant à stocker les emplacements des temps creux
Pour t allant de 0 à r+P faire
      Si C_rest=0 alors ajouter t à temps_creux
      Sinon C_rest ← C_rest - 1
      FinSi
      C_rest ← C_rest+C_req(t+1)
Fait
-- La liste temps_creux contient les emplacements des temps creux
-- dans leur ordre chronologique d'apparition
t_c ← -1
Tant que longueur(temps_creux)>P(1-U_S) faire
      premier ← tête(temps_creux)
      -- Date du premier temps creux
      correspondance ← (P(1-U_S)+1)ème élément de temps_creux
      Si correspondance≤premier+P alors
      -- Il y a plus de P(1-U_S) temps creux dans [premier..premier+P[
      -- donc premier correspond à un temps creux acyclique
            t_c ← premier
            supprimer_tête(temps_creux)
      Sinon Sortir de la boucle
      FinSi
Fait
```

2.4. Exemple

Nous illustrons l'algorithme précédent sur un exemple.

Soit $S_2=\{\tau_1<0,3,10,10>,\tau_2<4,2,5,5>,\tau_3<3,3,15,15>\}$ un système de tâches indépendantes. Sa charge est $U_{S_2}=27/30$, il y a donc 3 temps creux cycliques par méta-période. Etudions le diagramme de charge de S_2 (de façon équivalente, le diagramme produit par la première partie de l'algorithme précédant) sur la figure 5.

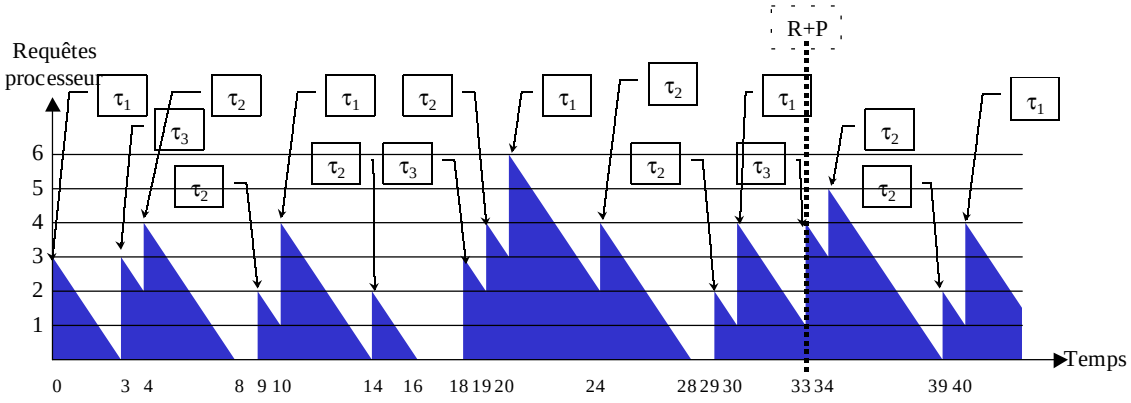

figure 5 : diagramme de charge du système S_2

Il y a 4 temps creux entre *0* et *r+P=33*, aux instants *8*, *16*, *17*, et *28*. Il y a moins de *P* unités de temps d'écart entre le premier et le quatrième, le temps creux situé à l'instant *8* est donc acyclique. Il ne reste plus alors que *3* temps creux, donc $t_c=8$. La longueur nécessaire à la représentation d'une séquence conservative de S_2 est donc de *39* unités de temps, et une séquence σ s'écrit $\sigma=\sigma_{[0..9[}\sigma^*_{[9..39[}$.

3. Cyclicité des ordonnancements de tâches dépendantes

Nous avons montré dans [9] que le résultat de cyclicité s'étendait au cas de tâches intéragissantes (i.e. partageant des ressources critiques et échangeant des messages) sous les hypothèses suivantes :

- Il n'y a pas d'interblocage, c'est à dire qu'un protocole de gestion de ressources prévenant ce type de problèmes est utilisé (protocole à priorité plafond, protocole de gestion des ressources par piles, allocation ordonnée des ressources) ;
- Les communications sont effectuées par des messages 1-1, c'est à dire qu'à chaque message correspond une et une seule tâche émettrice, et une et une seule tâche réceptrice ;
- Aucune tâche n'est susceptible d'attendre un message à l'intérieur d'une section critique ;
- Toute ressource prise par une tâche est libérée ;
- Si une tâche τ_i envoie n_i messages à une tâche τ_j à chacune de ses occurrences, et que τ_j en lit n_j à chacune de ses occurrences, alors les fréquences d'émission et de réception sont identiques, i.e. $\frac{n_i}{P_i}=\frac{n_j}{P_j}$.

Sous ces hypothèses, un temps creux peut apparaître si et seulement si il n'y a aucune tâche active ou bien si toutes les tâches actives sont en attente d'un message émis par une tâche qui n'est pas encore activée ou bien d'un message d'une tâche en attente. Les attentes forment ainsi des chaînes d'attente, que l'on montre périodiques de période *P*. Ces chaînes d'attente dépendent uniquement du système de tâches considérés et des contraintes de précédence induites par les communications qui sont mises en jeu.

Il est impossible que toutes les tâches actives soient en attente d'une ressource, sinon, ce serait un cas d'interblocage. La définition de la charge restante est modifiée pour

prendre en compte les chaînes d'attente de façon à ce qu'un temps creux apparaisse lorsque la charge restante est nulle ou bien égale à la charge de tâches toutes en attente.

Définition 4 : La charge en attente est définie par la somme des charges des tâches en attente, et est notée $C_{att}(t)$.

Le calcul de C_{att} et la preuve de cyclicité de cette fonction ne sont pas détaillés ici, le lecteur intéressé peut se référer à [9].

Définition 5 : La charge instantanée d'un système ordonnancé par un algorithme conservatif est définie par :

$$\begin{cases} C_{rest} : Temps \rightarrow Requête\ \Pr ocesseur \\ C_{rest}(0) = C_{req}(0) \\ C_{rest}(t) = C_{req}(t) + C_{rest}(t-1) - \gamma(t-1) \end{cases}$$

$$\text{Avec } \gamma(t) = \begin{cases} 1 \text{ si } C_{rest}(t) > C_{att}(t) \\ 0 \text{ sinon} \end{cases}$$

Cette définition caractérise le fonctionnement des algorithmes d'ordonnancement conservatifs : si la charge restante appartient à des tâches en attente alors il y a un temps creux, sinon, le processeur traite une unité de temps de charge jusqu'à l'instant suivant. La preuve de la cyclicité des séquences d'ordonnancement est alors similaire au cas où les tâches sont indépendantes, sachant que $\forall \Delta \in I\!N$, $C_{att}(r+\Delta) = C_{att}(r+P+\Delta)$ et $C_{att}(P+\Delta) \geq C_{att}(\Delta)$.

Nous illustrons le résultat à travers un exemple. Afin de simplifier la lecture, les tâches réceptrices attendent un message avant de commencer leur exécution, alors que les tâches émettrices émettent en fin d'exécution. Nous remplaçons donc les communications par des contraintes de précédence. Soit $S_3=\{\tau_1<0,1,4,4>, \tau_2<1,1,4,4>, \tau_3<2,1,4,4>, \tau_4<0,1,4,4>\}$ un système de tâches de charge maximale tel que τ_2 précède τ_1 et τ_1 précède τ_4. Le diagramme de charge (voir figure 6) montre l'évolution de la charge de S_3 lorsqu'il est ordonnancé par un algorithme conservatif, le trait gras représente la charge des tâches en attente.

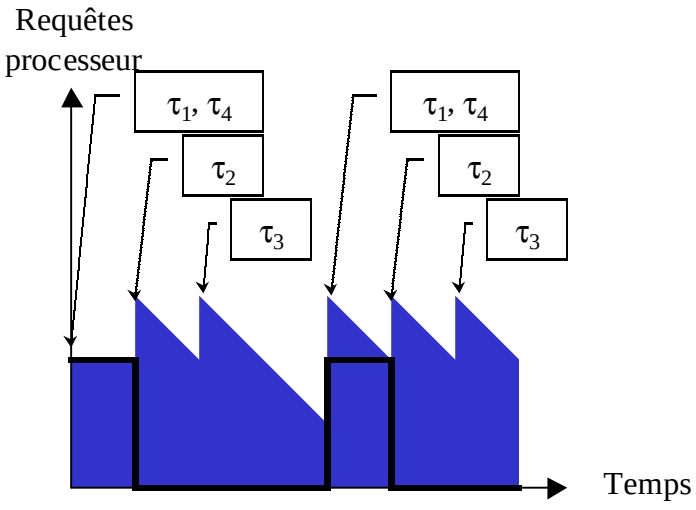

figure 6 : diagramme de charge de S_3

Il y a un temps creux si la charge restant à traiter est égale à la charge en attente (comme à l'instant *0*). La charge en attente correspond, à l'instant *0+kP* pour $k \in \mathbb{N}$, à la charge de la tâche τ_1, en attente du réveil de τ_2, plus la charge de la tâche τ_2, en attente de la tâche τ_1. Toute séquence d'ordonnancement conservative de S_3 s'écrit donc $\sigma=\sigma_{[0..1[}\sigma^*_{[1..5[}$.

4. Conclusion

Nous avons montré que toute séquence d'ordonnancement conservative était cyclique de période *P* après le dernier temps creux acyclique. Celui-ci peut apparaître dans un intervalle [*0..r+P*[. On peut donc limiter la construction de tout système de tâches à *r+2P* unités de temps. Cependant, la détermination de la date du dernier temps creux acyclique permet d'obtenir une longueur minimale pour la séquence à construire.

Ce résultat améliore la borne de [10] qui est de *r+2P*, puisque cette borne est maintenant une borne supérieure. De plus, le résultat est généralisé à tout algorithme conservatif, et aux systèmes de tâches pouvant partager des ressources et communiquer. Des investigations dans le domaine des algorithmes non conservatifs ont été menées, démontrant que le même résultat s'y appliquait. Les perspectives immédiates de ce résultat concernent l'ordonnancement multiprocesseur et distribué.

5. Bibliographie

[1] R. Alur and T. A. Henzinger, *Real-time logics: complexity and expressiveness, , 5th Annual IEEE Symposium on Logic in Computer Science*, 1990, pp. 390-401.

[2] T. P. Baker, *Stack-based scheduling of real-time processes*, The Journal of Real-Time Systems, 3 (1991), pp. 67-99.

[3] J. P. Beauvais and A. M. Déplanche, *Affectation de tâches dans un système temps réel réparti*, Technique et Science Informatiques, 17 (1998).

[4] J. Blazewicz, *Deadline scheduling of tasks - a survey*, Foundations of Control Engineering, 1 (1976), pp. 203-216.

[5] P. Bratley, M. Florian and P. Robillard, *Scheduling with earliest start and due date constraints on multiple machines*, Naval Research Logistic Quarterly, 22 (1975), pp. 165-173.

[6] M. Chen and K. Lin, *Dynamic priority ceilings : a concurrency control protocol for real-time systems*, Real-Time Systems, 2 (1990), pp. 325-346.

[7] H. Chetto, M. Silly and T. Bouchentouf, *Dynamic scheduling of real-time tasks under precedence constraints*, Journal of Real-Time Systems, 2 (1990), pp. 181-194.

[8] E. M. Clarke, E. A. Emerson and A. P. Sistla, *Automatic verification of finite-state concurrent systems using temporal logic specifications*, ACM Transactions on Programming Languages and Systems, 8 (1986), pp. 244-263.

[9] E. Grolleau, *Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de Petri en environnement monoprocesseur et réparti, , ENSMA*, Université de Poitiers, Futuroscope, 1999.

[10] J. Leung and M. Merrill, *A note on preemptive scheduling of periodic real-time tasks*, Information Processing Letters, 11 (1980), pp. 115-118.

[11] C. L. Liu and J. W. Layland, *Scheduling algorithms for multiprogramming in real-time environment*, Journal of the ACM, 20 (1973), pp. 46-61.

[12] A. K. Mok, *Fundamental design problems of distributed systems for the hard real-time environment*, , *Department of Electrical Engineering and Computer Science*, Massachussets Institute of Technologie, Cambridge, 1983.

[13] L. Sha, R. Rajkumar and J. P. Lehoczky, *Priority inheritance protocols : an approach to real-time synchronization*, IEEE Transactions on Computers, 39 (1990), pp. 1175-1185.

[14] M. Spuri and J. A. Stankovic, *How to integrate precedence constraints and shared resources in real-time scheduling*, IEEE Transactions on Computers, 43 (1994), pp. 1407-1412.

[15] J. A. Stankovic, *Misconceptions about real-time computing*, Computer, 21 (1988), pp. 10-19.

[16] J. A. Stankovic, M. Spuri, M. D. Natale and G. Buttazzo, *Implications of classical scheduling results fo real-time systems*, IEEE Computer, 28 (1995), pp. 1-24.

[17] J. Xu and D. Parnas, *Scheduling processes with release times, deadlines, precedence, and exclusion relations*, IEEE Transactions on Software Engineering, 16 (1990), pp. 360-369.