UTILISATION DES RÉSEAUX DE PETRI POUR L'ORDONNANCEMENT HORS-LIGNE OPTIMAL DES SYSTÈMES TEMPS RÉEL

E. GROLLEAU, A. CHOQUET-GENIET
LISI/ENSMA
1 avenue Clément Ader, BP 40109,
86960 Futuroscope CEDEX - FRANCE
e-mail : grolleau,ageniet@ensma.fr

Résumé :
Nous présentons une méthode d'ordonnancement hors-ligne de systèmes de tâches périodiques temps réel. Elle se base sur une énumération de séquences d'ordonnancement valides obtenue par la construction du langage d'un réseau de Petri via son graphe des marquages, dont il nous faut calculer à l'avance la profondeur (i.e. la durée de simulation nécessaire). Une fois le graphe des marquages obtenu, il est très simple d'en extraire des séquences optimales pour certains critères.

Mots clés :
Cyclicité des ordonnancements, réseaux de Petri, règle de tir maximal, ordonnancement temps réel

I. 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 [13]. Il peut être modélisé par un système de tâches périodiques [9], 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 optimaux 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 [8, 9, 14]. Ces algorithmes sont dits en-ligne car les priorités sont calculées pendant la vie du système temps réel. Certains d'entre eux ont été étendues aux systèmes de tâches soumises à des contraintes de précédence (i.e. tâches communicantes)[2, 5]. Cependant, dès lors que les tâches partagent des ressources critiques, le problème de l'ordonnancement est NP-difficile [10]. 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 ce problème, les algorithmes classiques d'ordonnancement ont été enrichis de protocoles de gestion de ressources, tels le protocole à priorité héritée [11], le protocole à priorité plafond [4, 11, 12] ou le protocole de gestion des ressources par piles [1]. La propriété de tels protocoles est d'éliminer le phénomène d'inversion de priorité et de limiter, voire de borner la durée des blocages dus aux attentes de ressources. Cependant, ils ne permettent pas de valider certains systèmes de tâches fortement contraints en terme de ressources à cause du problème de non régularité des ordonnancements [6]. Il est en effet nécessaire d'augmenter de façon artificielle la charge processeur des tâches accédant à des ressources par la pire durée de blocage que celles-ci peuvent subir. Cette augmentation de la charge du système peut rendre certains systèmes non ordonnançables par des techniques d'ordonnancement en-ligne.

C'est dans le but de valider de tels systèmes que les approches hors-ligne ont été développées. Ces approches consistent à créer à l'avance une séquence (ou un ensemble de séquences) d'ordonnancement valide puis de l'implanter dans le noyau temps réel cible sous la forme d'une table consultée par un séquenceur. Dans le domaine de l'ordonnancement temps réel, l'approche la plus connue est celle de [16] qui permet d'ordonnancer des systèmes de tâches périodiques simultanées pouvant communiquer (sous la contrainte que les communications puissent être mises sous forme normale) et partager des ressources dont les accès peuvent être gérés par sémaphores binaires. Le but de leur approche est de minimiser le retard maximal des tâches par rapport à leurs échéances. Notre approche permet la prise en compte de tâches périodiques différées, pouvant communiquer par boîtes aux lettres (le modèle de communication choisi est donc plus large que celui nécessitant la mise sous forme normale des tâches communicantes), partager des ressources multi-instances, des ressources pouvant être accédées en lecture seule ou en écriture, et avoir des parties de code non préemptibles.

Dans le second chapitre, nous présentons le modèle de tâches étudié et ses propriétés de cyclicité. Ensuite, nous présentons le modèle réseau de Petri permettant l'obtention des séquences d'ordonnancement valides du système modélisé. Enfin, l'étude des propriétés du langage (via son graphe des marquages) du modèle nous permet de présenter un algo-

rithme d'extraction de séquences d'ordonnancement optimales.

II. MODÈLE DE TÂCHES ÉTUDIÉ

Les tâches considérées sont périodiques, nous supposons que les tâches sporadiques à contraintes strictes sont prises en compte à l'aide d'un serveur périodique, et que les tâches non périodiques à contraintes relatives sont ordonnancées en-ligne dans les temps creux laissés dans la séquence d'ordonnancement créée.

Chaque tâche τ_i est caractérisée temporellement à l'aide de quatre paramètres temporels [9] :

- r_i sa première date d'activation,
- C_i sa charge, c'est à dire le pire temps d'exécution requis par chacune de ses occurrences,
- D_i son délai critique, c'est à dire le temps qui lui est donné pour s'exécuter après chacune de ses activations. Les dates correspondant à la fin d'un délai critiques se nomment échéances.
- P_i sa période d'activation. τ_i est donc activée aux instants r_i+kP_i pour $k\in \mathbb{N}$. Nous supposons dans la suite que $D_i \leq P_i$.

Les paramètres temporels de τ_i sont donnés sous la forme $\tau_i<r_i,C_i,D_i,P_i>$, et un système de tâches est noté $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$. Par convention, $min_{i=1..n}\{r_i\}=0$. Les tâches d'un système S sont dites simultanées si $\forall i\in\{1..n\}$, $r_i=0$, et différées si $\exists i/\ r_i>0$.

Le problématique de l'ordonnancement hors-ligne est de produire une séquence infinie (en fait avec une partie cyclique) permettant de respecter toutes les contraintes temporelles des tâches, et en particulier leurs échéances. De plus, lorsque les tâches interagissent, leurs contraintes structurelles doivent être respectées (exclusions mutuelles induites par des accès aux ressources ou des parties non préemptibles, contraintes de précédence induites par la prise en compte des communications).

Les communications se font à l'aide de boîtes aux lettres sous la contrainte que la fréquence d'émission et la fréquence de réception pour une boîte aux lettres sont identiques.

L'accès aux ressources se fait soit en écriture (dans ce cas, tout autre accès à la ressource est prohibé), soit en lecture, ce qui permet plusieurs lectures concurrentes.

II.1 Gestion des temps creux

Il est possible de déterminer au préalable combien de temps creux peuvent être présents dans une séquence d'ordonnancement. Soit $U_S=\sum_{i=1}^{n}\frac{C_i}{P_i}$ la charge d'un système S. On peut montrer qu'en régime permanent (i.e. dans la partie cyclique d'une séquence), toute séquence a exactement $P(1-U_S)$ temps creux toutes les P unités de temps où $P=PPCM_{i=1..n}(P_i)$ est appelé la méta-période du système. Ces temps creux, nommés temps creux cycliques, peuvent donc être pris en compte par une tâche périodique appelée tâche oisive $\tau_0<r_0,P(1-U_S),P,P>$. La date de réveil de la tâche oisive est choisie comme étant le début du régime permanent de toute séquence d'ordonnancement.

Lorsque l'on considère les algorithmes classiques d'ordonnancement en-ligne, cette tâche n'a que peu d'intérêt, puisqu'elle n'est jamais prioritaire, et ne s'exécute que lorsqu'elle est la seule prête. Par contre, si l'on souhaite élargir le cadre de l'étude hors ligne aux algorithmes non conservatifs (i.e. dans lesquels le processeur peut rester inactif bien qu'il y ait des tâches prêtes), elle devient nécessaire si l'on veut parer à certaines situations de blocage en la traitant avant d'autres tâches. Par exemple, il peut être nécessaire de ne rien faire plutôt que de permettre à une tâche active d'entrer en section critique, afin d'éviter de bloquer une tâche urgente réveillée ultérieurement. Un autre avantage de l'introduction explicite de cette tâche est qu'elle permet de contrôler le nombre de temps creux placés au sein d'une séquence, ce qui garantit qu'il y en aura le nombre requis exactement.

II.2 Cyclicité des ordonnancements

Lorsque les tâches sont simultanées, à l'instant 0, elles sont toutes activées simultanément, et d'après la propriété du *PPCM*, une méta-période plus tard, elles sont à nouveau activées simultanément. L'état du système est donc identique à l'instant P et à l'instant 0. Il en résulte qu'une séquence σ valide de longueur P peut être infiniment répétée, conduisant à un ordonnancement valide. Dans ce cas, le régime permanent d'une séquence commence dès l'instant 0, d'où le choix de la date de réveil de la tâche oisive $r_0=0$.

Dans le cas des systèmes de tâches différées nous avons montré dans [6, 7] que le régime permanent commençait pour toute séquence d'ordonnancement (quel que soit l'algorithme l'ordonnancement considéré) à la date t_c+1 où t_c est la date du dernier temps creux acyclique. De plus, nous avons montré que t_c était indépendant de l'algorithme considéré, et que dans tous les cas, $t_c<max_{i=1..n}\{r_i\}+P$. La preuve ne peut malheureusement pas être présentée ici, par manque de place, mais nous donnons une idée de ce qu'est un temps creux acyclique sur un exemple. Soit $S_2=\{\tau_1<0,1,4,4>, \tau_2<1,3,6,6>, \tau_3<3,1,4,4>\}$ un système de tâches indépendantes. $U_{S2}=1$, donc il n'y a pas de temps creux dans le régime permanent d'un ordonnancement de S_2. Sur la figure 1, chaque activation d'une tâche τ_i se traduit par l'augmentation de la charge à traiter de C_i unités de temps. Lorsque la charge à traiter est non nulle, tout algorithme conservatif traite une unité de temps de charge par unité de temps. Lorsque la charge à traiter est nulle, il y a un temps creux. C'est le cas à l'instant 6 : un temps creux, résidu de la montée en charge, est présent. A l'instant 7, la charge à traiter n'est donnée que par les activations des tâches, état que l'on retrouve une méta-période plus tard à l'instant 19. Toute séquence d'ordonnancement σ conservative de S_2 valide sur [0..19[permet d'obtenir une séquence d'ordonnancement valide infinie $\sigma_{[0..7[}\sigma^{*}_{[7..19[}$.

Le résultat de cyclicité a été étendu au cas de systèmes de tâches interagissantes, et reste valide même si les temps creux ne sont pas placés de façon conservative (i.e. lorsqu'il n'y a rien à faire).

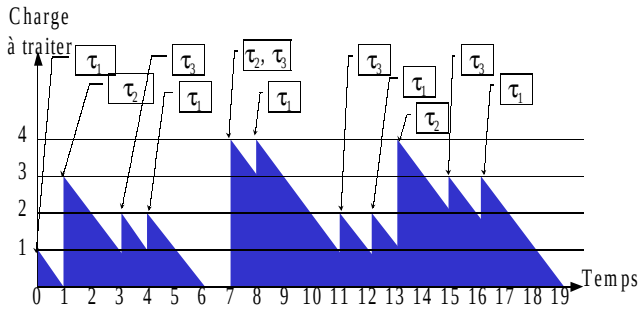

figure 1. *Diagramme de charge de S_2.*

Pour un système de tâches donné, il est donc possible a priori de savoir à partir de quelle date toute séquence d'ordonnancement entre dans un régime permanent en calculant la date du dernier temps creux acyclique t_c. La date d'activation de la tâche oisive est donc déduite à partir de t_c et est $r_0=t_c+1$. De plus, l'état atteint aux dates t_c+1 et t_c+P+1 est identique pour toute séquence d'ordonnancement valide.
Ce résultat permet la construction de séquence d'ordonnancement hors-ligne pour les systèmes de tâches périodiques dont certaines peuvent être différées.

III. Modélisation par réseaux de Petri

La démarche adoptée consiste à modéliser le système de tâches, avec leurs interactions, par un réseau de Petri (RdP) prenant en compte le temps. Des démarches similaires ont été adoptées dans le domaine de l'ordonnancement cyclique [3] en se basant sur des RdP temporisés. Puisque en ordonnancement temps réel, les tâches peuvent être préemptées à intervalles de temps réguliers, chaque tâche de charge C_i peut être représentée par la mise en séquence de C_i transitions de durée unitaire utilisant une ressource processeur. Puisque les durées sont toutes unitaires, nous pouvons utiliser un modèle de RdP simplifié par rapport aux RdP temporisés : les RdP autonomes avec la règle de tir maximal.

III.1 Réseaux de Petri avec la règle de tir maximal

Les RdP avec la règle de tir maximal peuvent être vus comme des RdP temporisés à vitesse maximale où toutes les transitions sont de durée unitaire [15]. Cependant, leur implémentation est simplifiée par le fait qu'il n'est pas nécessaire de prendre en compte le vecteur des durées résiduelles de tir en plus du marquage courant pour connaître l'état du RdP. Ils ont la puissance d'expression d'une machine de Turing, ce qui nous permettra de modéliser par la suite toute interaction possible entre les tâches. En contrepartie, l'analyse de tels RdP est délicate, mais nous nous intéressons à leur langage.

III.1.1 Définition

Un RdP autonome avec la règle de tir maximal est défini par :

- Q un ensemble fini de places,
- T un ensemble fini de transitions, avec $Q\cap T=\varnothing$,
- $F\subseteq (Q\times T)\cup(T\times Q)$ le flot du réseau,
- $W : F\rightarrow \mathbb{N}^+$ la fonction de pondération du flot,
- $M : Q\rightarrow \mathbb{N}$ est le marquage du réseau, on note M_0 le marquage initial.

On dit qu'une transition t est franchissable pour un marquage M si, comme pour les RdP autonomes, $\forall p\in Q$ tel que $F(p,t)$, $W(F(p,t))\leq M(p)$.
La règle de tir maximal est la suivante : soit $I\subseteq T$ l'ensemble des transitions franchissables à partir d'un marquage M, un sous-ensemble maximal $I'\subseteq I$ est franchi simultanément. Un ensemble maximal $I'\subseteq I$ est tel que pour toute transition $t\in I\setminus I'$, l'ensemble $I'\cup\{t\}$ ne peut être franchi simultanément, c'est à dire que t ne peut pas être tirée en même temps que les transitions de I' à cause de conflits.

III.1.2 Modélisation d'une action périodique

Les tâches modélisées étant périodiques, nous présentons dans cette partie de quelle façon une action périodique peut être modélisée à l'aide d'un RdP avec la règle de tir maximal. Soit une action e périodique de durée unitaire et de période P_e. La périodicité de e peut être modélisée comme sur la figure 2.

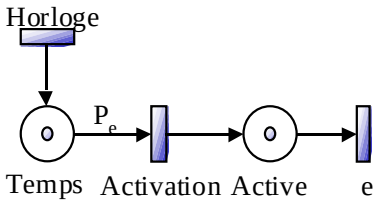

figure 2. *Modélisation d'une action périodique.*

La transition *Horloge* est une transition source, elle est donc toujours franchissable, et, par respect de la règle de tir maximal, étant donné qu'elle n'est pas en conflit avec aucune autre transition, elle est franchie à chaque fois qu'un ensemble de transitions est franchi. Il en résulte qu'elle produit à chaque tir d'un ensemble de transitions un jeton dans la place *Temps*, qui fonctionne comme un horloge locale de l'action périodique. La transition *Horloge* marque donc le temps, et chaque tir d'un ensemble maximal de transitions franchissables dure une unité de temps sur cette échelle de temps. Dès lors que *Temps* contient P_e jetons, c'est à dire aux instants kP_e-1 pour $k\in \mathbb{N}^+$, la transition *Activation* est tirée, produisant un jeton dans la place *Active*, dont la sémantique est de traduire le fait que l'action e peut être effectuée. Si la transition e n'est en conflit avec aucune autre transition (ce qui le cas ici), elle est tirée aux instants kP_e pour $k\in \mathbb{N}$.
Puisque les tâches considérées peuvent être différées (cas où la date de première activation r_e de l'action est différente de 0), remarquons que lorsque $r_e\leq P_e+1$, une modification du marquage initial M_0 suffit : $M_0(Temps)=P_e-r_e+1$, et $M_0(Active)=0$. Dans le cas où $r_e>P_e+1$, le modèle est augmenté d'une place et d'une transition permettant d'obtenir le comportement adéquat.
Par hypothèse, la période à laquelle e doit être tirée est plus grande que 1 (en effet, dans le cas contraire, e pourrait être modélisée par une boucle implicite avec *Active*).

III.2 Modélisation d'un système de tâches

Le principe de modélisation des actions périodiques est utilisé afin de modéliser la périodicité des tâches. Ainsi, la figure 3 donne le modèle RdP utilisé pour modéliser le système de

tâches indépendantes $S_1=\{\tau_1<1,2,3,4>, \tau_2<0,1,2,2>\}$. Le modèle est présenté en deux parties : la partie horlogerie, et la partie système de tâches.

III.2.1 Système d'horlogerie

Le système d'horlogerie permet de modéliser la périodicité des tâches, ainsi que leur première date d'activation. La modélisation utilisée est similaire à celle évoquée dans la partie précédente, avec une horloge globale *Horloge*, modélisée par une transition source. Elle produit à chaque unité de temps (i.e. à chaque tir d'un ensemble maximal) un jeton dans chacune des horloges locales $Temps_i$ associées au tâches. La pondération associée à l'arc reliant une horloge locale $Temps_i$ à la transition d'$Activation_i$ associée est la période de la tâche. Nous utilisons une coloration des jetons afin de simplifier la représentation du RdP : la sémantique associée à la couleur *a* est une notion d'activation, et la transition $Activation_i$ produit un jeton de couleur *a* dans la place $Active_i$ associée. Sans entrer pour le moment dans le détail de la modélisation du corps des tâches, remarquons que le tir de la dernière transition composant une tâche produit un jeton de couleur *b* dans la place $Active_i$, dont la sémantique est de signifier que la tâche τ_i a terminé sa dernière exécution. Pour pouvoir commencer son exécution, une tâche doit donc avoir un jeton *a* (elle est active) et un jeton *b* (elle a terminé sa dernière exécution) dans la place $Active_i$. Ce procédé interdit la réentrance des tâches.

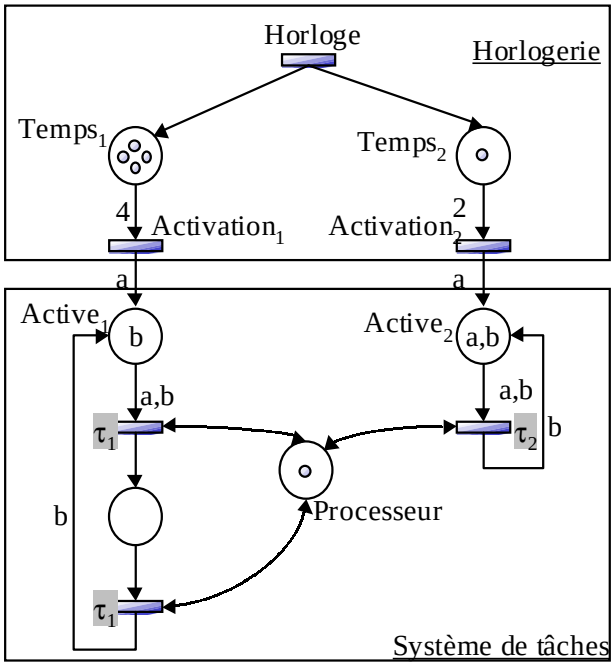

figure 3. *Modélisation d'un système $S_1=\{\tau_1<1,2,3,4>, \tau_2<0,1,2,2>\}$ par RdP.*

III.2.2 Système de tâches

La durée C_i de chaque tâche τ_i est modélisée par C_i transitions mises en série (nous modélisons cependant les blocs de durée supérieure à 3 par une série de trois transitions). Ces transitions sont labellées par le nom de la tâche à laquelle elles appartiennent. Chacune des transitions composant les tâches nécessite le jeton processeur : en effet, les tâches sont concurrentes. Dans la suite, nous ne représentons pas la place processeur afin d'alléger le graphique, mais il est important de remarquer que c'est une ressource utilisée par toutes les transitions de la partie système de tâches. Une exception cependant : les parties de tâches non-préemptibles sont modélisées par le fait que la première transition d'une partie non-préemptible prend la ressource processeur mais ne la rend pas, le jeton processeur n'est rendu que par la dernière transition de la partie non préemptible. Cela a pour effet d'interdire à toute autre tâche de s'exécuter lorsqu'une tâche est en mode non préemptif. Les ressources critiques accessibles uniquement en écriture sont modélisées à l'aide de places ressources de façon classique, dont le marquage initial est donné par le nombre d'instances disponibles de la ressource. Les ressources mono-instance du type lecteur/écrivain sont elles aussi modélisées à l'aide de places ressources dont le marquage initial est donné par le nombre maximal n_l de lecteurs simultanés. Un accès en lecture est modélisé par l'utilisation d'un jeton, et un accès en écriture par l'utilisation de n_l jetons.

Enfin, les communications sont modélisées à l'aide de places boîtes aux lettres : l'envoi d'un message se traduit par la production d'un jeton dans la boîte aux lettres, et la réception d'un message par la consommation d'un jeton de la boîte.

Les contraintes de précédence et d'exclusion mutuelle inhérentes à la structure des tâches modélisées sont donc directement prises en compte par le modèle.

L'ensemble des comportements du RdP est donc l'ensemble des comportements conservatifs[1] possibles du système modélisé, sans pour l'instant respecter les contraintes temporelles des tâches.

III.2.3 Prise en compte des contraintes temporelles

Il reste à modéliser les contraintes temporelles des tâches par la prise en compte de leur délai critique. Pour cela, étudions le comportement des horloges locales des tâches :

- Cas où $D_i<P_i$: lorsque le marquage $M(Temps_i)$ de l'horloge locale associée à τ_i est égal à D_i+1, cela signifie soit que la tâche n'a jamais encore été activée, soit que la tâche τ_i est activée depuis D_i unités de temps. La tâche τ_i respecte son échéance courante si et seulement si elle a terminé son exécution courante, i.e. $M(Active_i)=\{b\}$.
- Cas où $D_i=P_i$: lorsque le marquage $M(Temps_i)=1$, soit la tâche n'a jamais été activée, soit elle est activée, et son échéance tombe au même instant. La tâche τ_i respecte donc son échéance courante si et seulement si $M(Active_i)=\{b\}$ (cas où la tâche n'a pas encore été activée) ou si $M(Active_i)=\{a,b\}$ (cas où la tâche vient d'être réactivée).

Pour un marquage donné du RdP, les contraintes temporelles des tâches ne sont pas violées si et seulement si :

$M(Temps_i)=D_i+1 \Rightarrow M(Active_i)=\{b\}$

et $M(Temps_i)=1 \Rightarrow M(Active_i)=\{a,b\}$ ou $M(Active_i)=\{b\}$

[1] Un comportement conservatif est tel qu'à tout instant, si il existe un traitement pouvant être effectué, alors celui-ci est effectué (i.e. il n'y a pas de temps creux lorsqu'il y a du travail à effectuer)

Nous utilisons la notion de contraintes terminales sur les marquages du système. Tout comportement du RdP pour lequel chaque marquage rencontré respecte les contraintes terminales correspond à un comportement du système modélisé où aucune échéance n'est violée. Les transitions correspondant au corps des tâches étant étiquetées par le nom de la tâche à laquelle elles appartiennent, et les autres transitions étant étiquetées par le mot vide, l'ensemble des ordonnancements conservatifs valide du système modélisé est exactement le centre du langage terminal du RdP (ensemble des mots infinis pour lesquels chaque marquage rencontré est terminal). Le centre du langage terminal du RdP (voir figure 3) modélisant S_1 est donné sur la figure 4. Il correspond à l'ensemble des ordonnancements valides de S_1. Tout chemin du graphe des marquages est une séquence d'ordonnancement valide de S_1. Par exemple, $(\tau_2,\tau_1,\tau_1,\tau_2)^*$ est une séquence d'ordonnancement valide.

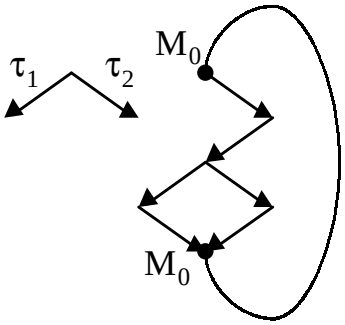

figure 4. Centre du langage terminal du RdP modélisant S_1.

Cependant, dès lors que des ressources ou des parties non préemptibles sont présentes, les ordonnancements conservatifs ne sont pas dominants, i.e. il existe des systèmes de tâches pour lesquels il existe des séquences d'ordonnancement valides mais pour lesquels aucune séquence conservative n'est valide. Il est donc nécessaire de permettre la génération des séquences non conservatives par l'adjonction d'une tâche oisive τ_0. De plus, afin d'obtenir les séquences d'ordonnancement non conservatives, une autre tâche, non périodique, modélisant les temps creux acycl,iques est introduite.

IV. Extraction de séquences d'ordonnancement optimales

IV.1 Etude du graphe des marquages

L'étude de la cyclicité des séquences d'ordonnancement permet de connaître la profondeur du graphe des marquages correspondant à l'ensemble des séquences d'ordonnancement valides, conservatives ou non (grâce à l'adjonction de la tâche oisive), d'un système de tâches modélisé. En effet, nous pouvons limiter la construction du graphe des marquages à la profondeur P pour des tâches simultanées, et à la profondeur t_c+P+1 pour des tâches différées. De plus, dans le cas des tâches simultanées, le marquage initial est un marquage d'accueil puisque le système de tâches est dans le même état à l'instant P qu'à l'instant 0 (voir figure 5(a)). Dans le cas des systèmes de tâches différées, il y a un seul marquage à la hauteur t_c+1, et ce marquage est un marquage d'accueil qui se retrouve à la hauteur t_c+P+1 (voir M_c sur la figure 5(b)).
De plus, les graphes des marquages ont une forme de diamant car tout chemin (de façon équivalente toute séquence d'ordonnancement) est une permutation d'un autre, et un marquage d'accueil existe dans tous les cas.

IV.2 Algorithme d'extraction de séquences

Nous proposons ici une méthode d'extraction de séquences d'ordonnancement optimales en fonction de différents critères (importance, minimisation du temps de réponse, maximisation du temps de retard admissible, de certaines tâches ou ensembles de tâches). L'ensemble des séquences extraites par cette méthode contient toutes les séquences optimales vis à vis des critères choisis.

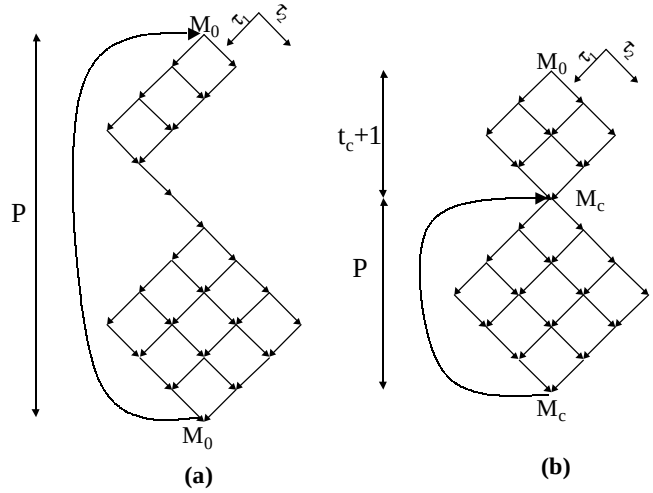

figure 5. *(a) Graphe des marquages pour un système de tâches simultanées. (b) Pour un système de tâches différées.*

IV.2.1 Critères d'optimisation

Nous présentons ici la liste des critères que nous avons étudiés. Soit E un ensemble de tâches d'un système de tâches S :

- Maximisation de l'importance : les tâches de E sont exécutées le plus tôt possible.
- Minimisation du temps de réponse maximal (ou moyen) : le temps maximal (ou moyen) entre l'activation et la terminaison des tâches de E est minimisé.
- Maximisation du temps de retard admissible minimal (ou moyen) : le temps minimal (ou moyen) entre la terminaison et l'échéance des tâches de E est maximisé.
- Minimisation du taux de réaction maximal (ou moyen) : le temps maximal (ou moyen) du temps de réponse divisé par l'échéance des tâches de E est minimisé.

IV.2.2 Algorithmes de choix de séquence

La méthode consiste à pondérer le graphe des marquages à l'aide d'une fonction qui dépend du critère à optimiser, puis à y effectuer une recherche des plus courts chemins.
Par exemple, pour minimiser le temps de réponse moyen d'un ensemble de tâches E, les arcs du graphe sont pondérés

implicitement de la façon suivante : le coût de tous les arcs est nul, sauf pour les arcs étiquetés par une transition de $\tau_i \in E$ correspondant à une terminaison de τ_i. Ces arcs sont étiquetés par le temps de réponse de τ_i correspondant. Cette pondération est effectuée en temps constant pour chaque arc. Les séquences optimales sont alors données par l'ensemble des plus courts chemins allant du noeud initial au noeud final.

La méthode utilisée est appliquée en deux phases.

- Chaque noeud est pondéré de telle sorte que sa pondération donne le coût minimal des chemins allant de ce noeud au noeud final. Cette opération est effectuée à l'aide d'un parcours topologique allant du noeud final au noeud initial. Elle a une complexité linéaire par rapport à la taille du graphe des marquages.
- Une fois les noeuds pondérés, l'ensemble des plus courts chemins est donné par un parcours du graphe. Ce parcours part du noeud initial M_0, et ne choisit dans ses fils que les noeuds M_i de pondération minimale. Il procède récursivement de la même façon sur chacun des noeuds M_i choisis.

Le sous-graphe obtenu contient l'ensemble des séquences d'ordonnancement minimisant le temps de réponse moyen des tâches de E.

La même technique peut être utilisée pour rechercher les séquences optimales en fonction de critères basés sur le temps de réponse des tâches.

V. CONCLUSION

Nous avons présenté une méthode d'ordonnancement hors-ligne de systèmes de tâches périodiques simultanées ou différées, pouvant partager des ressources en mono ou multi-instances, en lecture/écriture, pouvant avoir des parties non préemptibles, et communiquer par boîtes aux lettres. C'est à notre connaissance la seule méthode permettant d'ordonnancer des systèmes de tâches aussi généraux. La méthode se base sur la construction d'un réseau de Petri avec la règle de tir maximal, dont le langage terminal est l'ensemble des séquences valides du système modélisé. Les séquences obtenues sont conservatives ou non, grâce à l'adjonction au système modélisé d'une tâche oisive permettant la prise en compte des temps creux.

Une fois le graphe des marquages construit, il est possible d'en extraire des séquences optimales au vu de plusieurs critères permettant à un concepteur de systèmes temps réel de choisir les séquences en fonction de besoins précis. Cette extraction se fait en un temps linéaire en fonction de la taille du graphe.

L'étude de la cyclicité des séquences d'ordonnancement permet de borner a priori la profondeur du graphe des marquages et de fixer la date de réveil de la tâche oisive.

Le problème de l'ordonnancement étant NP-difficile dans le cas général, l'approche est exponentielle en temps et en espace. Nous employons donc différentes heuristiques permettant de limiter l'explosion combinatoire due à l'énumération. Cette approche a été mise en œuvre dans un outil d'ordonnancement nommé PeNSMARTS.

RÉFÉRENCES

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

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

[3] J. Carlier and P. Chretienne, *Timed Petri net schedules*, Advances in Petri Nets 1988, Lecture Notes in Computer Science, 340 (1988), pp. 62-84.

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

[5] 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.

[6] E. Grolleau, *Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de Petri en environnement monoprocesseur et multiprocesseur*, , 1999, pp. 235.

[7] E. Grolleau and A. Choquet-Geniet, *Cyclicité des ordonnancements de systèmes de tâches périodiques différées*, in Teknea, ed., *Real-Time Systems, RTS'2000*, Paris, 2000.

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

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

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] P. H. Starke, *Some properties of timed nets under the earliest firing rule*, Advances in Petri nets 1989, Venice in Lecture Notes in Computer Science, 424 (1990), pp. 418-432.

[16] 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.