Affectation optimale des priorités des tâches et des messages dans les systèmes distribués temps-réel

M. Richard {richardm@ensma.fr}
P. Richard {richardp@ensma.fr}
F. Cottet {cottet@ensma.fr}

Laboratoire d'Informatique Scientifique et Industrielle
École Nationale de Mécanique et d'Aérotechnique
Téléport 2 – BP 109 F-86960 Futuroscope, France

Résumé : Toute tâche ordonnancée dans un système informatique temps réel doit respecter ses contraintes temporelles (échéances, périodicités). Dans un système distribué, les ordonnancements des tâches et des messages sont fortement dépendants. Nous présentons une méthode arborescente affectant les priorités aux tâches et aux messages afin d'obtenir un système ordonnançable. Les tests d'ordonnançabilité mis en œuvre dans la méthode reposent sur le calcul des pires temps de réponse des tâches et des messages (analyse holistique).

Mots clés : Temps réel, systèmes distribués, affectation de priorités, analyse holistique, branch and bound.

1 Ordonnancement des systèmes temps-réel distribués

Nous étudions des systèmes temps réel distribués, composés de calculateurs communiquant à l'aide de messages via un réseau, sans mémoire partagée. Les applications temps réel possèdent des contraintes temporelles strictes. La conception d'un tel système nécessite donc une validation temporelle de son comportement, afin de vérifier le respect des échéances. Chaque tâche τ_i est représentée par un triplet (C_i, D_i, T_i) où C_i est le pire temps d'exécution, D_i l'échéance relative au réveil de la tâche et T_i la période des activations de τ_i. Les tâches sont toutes activées au démarrage de l'application.

Lorsque le système est distribué, un réseau informatique constitue l'unique moyen de communication entre les processeurs et constitue ainsi un ressource partagée par les tâches. Plusieurs niveaux de problèmes peuvent être considérés en fonction des caractéristiques du système distribué. Par exemple, le placement des tâches sur les processeurs, les éventuelles migrations des tâches durant leurs exécutions, l'ordonnancement des tâches sur les processeurs et des messages sur le réseau sont autant de paramètres bouleversant les performances du système. Dans cette article, nous faisons l'hypothèse que les tâches et les messages sont affectés sur un site ou un réseau de manière définitive (i.e. durant toute la vie de l'application).

Le réseau est vu comme une ressource critique partagée par toutes les tâches. La transmission des messages peut alors engendrer des problèmes *"d'inversion de priorité"* [But97] puisque la transmission d'un message quelconque ne peut pas être interrompue par l'émission d'un message prioritaire. De plus, seuls les pires temps d'exécution des tâches sont connus, et non leurs durées exactes. L'exécution plus rapide d'une tâche peut lui permettre d'émettre plus tôt son message, bloquant ainsi le message d'une tâche prioritaire. Ce problème est illustré figure 1, où la tâche τ_1 émet un message m_1 pour la tâche τ_3, et τ_2 émet un message m_2 pour la tâche τ_4. Dans le cas (a), les tâches s'exécutent avec leurs pires temps d'exécution, et chaque tâche respecte son échéance. Dans le cas (b), la tâche τ_2 s'exécute plus vite entraînant l'émission de m_2 avant m_1 sur le réseau. La tâche τ_3 manque alors son échéance (représentée par une flèche descendante). En conséquence, seule une analyse du *"pire cas"* peut être effectuée pour valider l'application. Elle fournit une condition suffisante d'ordonnançabilité.

Dans la section suivante, nous présentons brièvement une méthode de validation considérant les priorités des tâches et messages comme des paramètres : l'analyse holistique. Dans le paragraphe 3 nous présentons une méthode d'affectation des priorités des tâches et des messages qui est optimale vis-à-vis de l'analyse holistique.

Notations :

τ_i identificateur de tâche

C_i pire temps d'exécution de τ_i pour chaque activation (hors préemption)

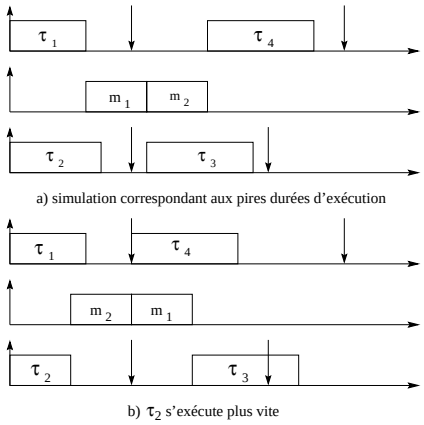

FIG. 1 – Simulation et validation d'une configuration de tâches

T_i période d'activation de τ_i

D_i échéance de τ_i, mesurée relativement à la date d'activation de la tâche et non reliée à T_i

J_i le pire temps de gigue sur l'activation de τ_i (différence entre l'activation et le réveil de la tâche)

$Prio_i$ priorité de la tâche τ_i, fixe durant toute la vie de l'application. Nous considérons la valeur 1 comme la priorité la plus forte.

Tr_i pire temps de réponse de τ_i : durée entre l'activation d'une tâche et sa fin d'exécution

Γ_i^{-1} ensemble des prédécesseurs immédiats dans le graphe de communication.

$[i]$ numéro de la tâche affectée à la priorité i

2 Analyse Holistique des systèmes à priorités fixes

Le terme d'analyse holistique a été introduit par K. Tindell [Tin94, TC94], et symbolise la prise en compte de la dépendance entre l'ordonnancement des tâches et des messages dans les systèmes temps réel distribués. Précisément, la date d'émission d'un message dépend du temps de réponse de la tâche émettrice, et la date de réveil d'une tâche réceptrice dépend du temps de réponse du message à recevoir. Ceci permet d'élaborer une analyse plus réaliste et plus fine du pire cas pouvant survenir dans la vie du système.

La figure 2 présente l'ordonnancement de deux tâches τ_i et τ_j communiquant par messages sur le réseau. L'émission du message est décalée par l'attente de la fin d'exécution de τ_i , et de même le début de τ_j est lié à l'arrivée du message sur son site. La gigue du message est notée J_m, et celle de la tâche réceptrice J_j. Les gigues sont les variables du problème modélisant la dépendance de l'ordonnancement conjoint des tâches et des messages. L'analyse holistique va permettre de les calculer par la résolution de systèmes d'équations récurrentes estimant le temps de réponse

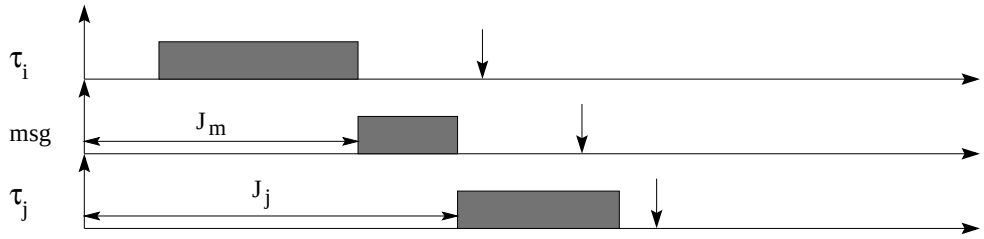

FIG. 2 – La gigue modélise l'attente due à la fin de la tâche précédente.

des tâches et des messages, et déterminant ainsi les valeurs des gigues. Nous obtenons ainsi les équations suivantes :

$$\begin{array}{ll} \text{Pour les tâches :} & J_i = \max\limits_{j \in \Gamma_i^{-1}} (Tr_j) \\ \text{Pour les messages :} & J_m = \max\limits_{j \in \Gamma_i^{-1}} (Tr_j) \end{array} \tag{1}$$

Toute modification des gigues va entraîner des modifications des temps de réponse des tâches dépendantes ou ceux des tâches ordonnancées sur le même processeur. Le principe de l'analyse holistique est de recommencer les calculs jusqu'à ce qu'un point fixe soit atteint. En pratique, les messages sont considérés comme des tâches et le réseau comme un processeur supplémentaire. Les contraintes de communication sont alors considérées comme des relations de précédence entre tâches. Nous définissons les fonctions *ResponseTime* qui calculent le pire temps de réponse d'une tâche τ_i en fonction des gigues des tâches et des messages et du processeur où τ_i est affectée. Soit n le nombre de tâches et de messages, le système d'équations à résoudre est :

$$1 \leq i \leq n \qquad \left\{ \begin{array}{l} \left\{ \begin{array}{lcl} Tr_i^{(0)} & = & C_i \\ J_i^{(0)} & = & 0 \end{array} \right. \\ \left\{ \begin{array}{lcl} Tr_i^{(k)} & = & ResponseTime\left(J_i^{(k-1)}\right) \\ J_i^{(k)} & = & \max\limits_{j \in \Gamma_i^{-1}} \left(Tr_j^{(k)}\right) \end{array} \right. \end{array} \right. \tag{2}$$

Le point fixe est atteint lorsque pour k $\in \mathbb{N}$ on vérifie :

$$Tr_i = Tr_i^{(k)} = Tr_i^{(k-1)}, 1 \leq \ i \leq n \tag{3}$$

Notons que les messages sont vus comme des tâches ne pouvant être interrompues. Ainsi l'affectation des priorités aux messages revient à considérer le réseau comme un processeur régi par une politique d'ordonnancement non-préemptive. Les paramètres temporels des messages tels que l'échéance D_i ou la période T_i sont hérités des tâches émettrices ou réceptrices. Les relations de communication sont ainsi transformées en contraintes de précédence. Pour plus de détails sur cette méthode de validation, le lecteur intéressé pourra se reporter à [RRC00].

3 Méthode d'affectation des priorités des tâches et des messages

L'analyse holistique permet de valider un système distribué si les priorités des tâches et des messages sont connues. Le choix de ces priorités est déterminant afin d'utiliser au mieux les ressources du système. Fixer les priorités s'avère long et fastidieux dès lors que le nombre de tâches et de messages augmente. Dans le but de faciliter cette étape, et surtout afin d'optimiser l'utilisation du système informatique, nous présentons une méthode de recherche arborescente réalisant l'affectation des priorités aux tâches et aux messages composant une application distribuée. Cette technique de paramétrage utilise l'analyse holistique comme outil de validation de l'ordonnançabilité du système. La méthode présentée est optimale vis-à-vis de cette analyse. Précisément, s'il existe un ensemble de priorités permettant de valider l'application par une analyse holistique, alors notre méthode doit le trouver. Ceci constitue une différence par rapport aux travaux présentés dans [GH95, TBW92], qui ne parcourent pas toutes les solutions potentielles, mais seulement un sous-ensemble (i.e. méthodes heuristiques).

Une procédure par séparation et évaluation (Branch and Bound) permet d'énumérer implicitement l'ensemble des affectations possibles des priorités aux tâches et aux messages. La procédure construit progressivement une solution en affectant à chaque étape une priorité à une tâche. Dès qu'une affectation des priorités conduit à un ordonnancement validé par l'analyse holistique, la procédure est arrêtée. Cette dernière repose sur deux fonctions : le principe de séparation et le principe d'évaluation.
Le principe de séparation indique pour un nœud donné la construction des nœuds fils (i.e. successeur immédiat dans l'arbre de recherche). Enfin le principe d'évaluation permet de déterminer si le nœud permettra de conduire à une solution réalisable. Si ce n'est pas le cas, le nœud est effacé de l'arbre de recherche et ses successeurs ne seront pas considérés dans la suite. Nous détaillons dans les parties suivantes les principes de séparation et d'évaluation.

3.1 L'arbre de recherche : principe de séparation

Dans cette section, nous présentons la structure et la construction de l'arbre de recherche dans lequel sont mémorisés tous les ensembles de priorités possibles. Chaque nœud de l'arbre représente l'affectation d'une priorité à une tâche.

3.1.1 Structure de l'arbre de recherche

Une énumération simple de toutes les affectations possibles des priorités produit un arbre possédant des branches redondantes. La figure 3 présente ce phénomène de redondance pour un problème à trois tâches et deux sites (τ_1 et τ_2 sur le site m_1 et τ_3 sur le site m_2). Un nœud de l'arbre est un triplet (τ_i, m_i, P_i) qui signifie que la tâche τ_i, s'exécutant sur la machine m_i, se voit affecter la priorité P_i. Ainsi les branches

du type (1) ont une structure différente, mais ont toutes la même sémantique (i.e. τ_1 possède la priorité P_1, τ_2 la priorité P_2 et τ_3 la priorité P_1). Il en est de même pour les branches du type (2). Dans ce problème, il existe, dans notre cas, deux solutions différentes possibles. Or l'arbre de la figure 3 énumère un espace de 12 solutions.

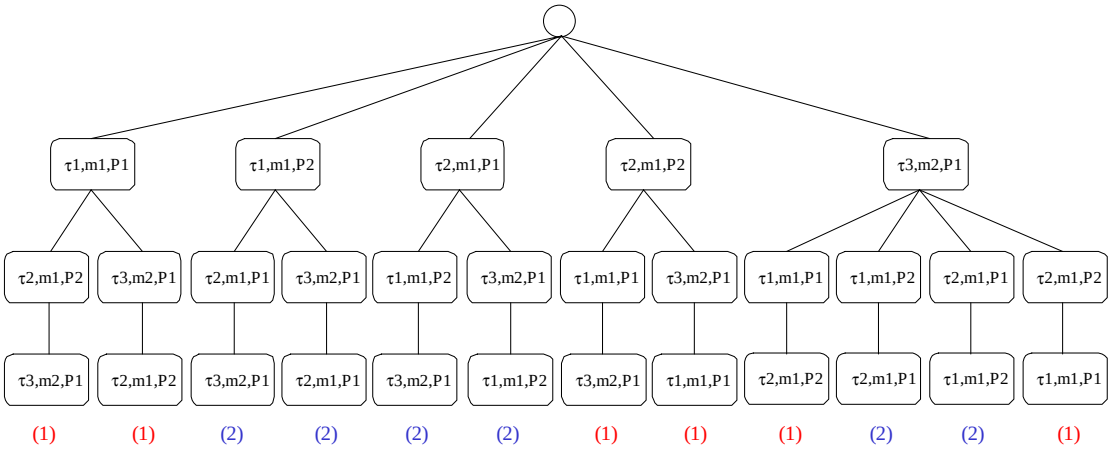

FIG. 3 – Phénomène de redondance lors d'une énumération simple

Afin d'éviter ce phénomène lors de la construction, nous proposons une technique d'énumération *site par site*. Notons que le ou les réseau(x) des applications que nous nous proposons de valider est(sont) vu(s) comme un(des) site(s) supplémentaire(s), conformément à l'analyse holistique.
La figure 4 présente notre technique d'énumération sur le même exemple que précédemment (i.e. figure 3). Les différents sites sont reliés par des nœuds fictifs représentant la racine du sous-arbre associé à un site ou un réseau. L'affectation des priorités est réalisée pour chaque site indépendamment des autres. Le parcours de l'arbre est un parcours *en profondeur d'abord*. Avec cette technique le nombre de feuilles de l'arbre de recherche correspond bien aux deux solutions possibles.

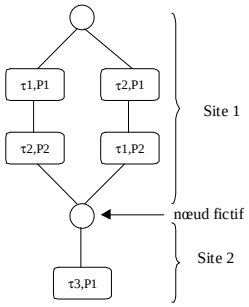

FIG. 4 – Structure générale de l'arbre de recherche

Dans le but d'améliorer la recherche, c'est-à-dire augmenter les chances de trouver une solution rapidement, nous affinons la structure de l'arbre en ce qui concerne les sites et les nœuds (i.e. les tâches).

3.1.2 Ordre des sites

La première amélioration porte sur les sites. Précisément, les sites sont classés selon leur *coefficient de charge* $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$. Deux cas se présentent alors : les sites peuvent-être classés selon la charge croissante ou bien selon la charge décroissante. Nous n'écartons à ce jour aucune de ces deux stratégies. Lors de la phase de test de notre procédure de recherche, nous expérimenterons ces deux méthodes de classement des sites dans la structure arborescente.

Classement selon la charge croissante : dans ce cas, le site possédant le coefficient de charge le plus élevé se situera en bas de l'arbre. Or, l'action de *retour arrière* implique que le site de niveau (i) est plus souvent parcouru que celui de niveau $(i-k)$. Le dernier site de la structure arborescente est donc celui le plus visité. Ainsi la chance de trouver une solution rapidement est plus grande.

Classement selon la charge décroissante : contrairement, le site possédant le coefficient de charge le plus grand sera ici placé au sommet de l'arbre. Puisqu'il s'agit du site le plus chargé, le nombre de feuilles de ce site susceptible de contenir une solution valide est moins important que dans le cas d'un site possédant un coefficient de charge faible. La chance de trouver une bonne séquence étant faible, le nombre de coupes (i.e. retours arrières) augmente. Or une coupe au sommet de la structure élimine de l'arbre de recherche tout le sous-arbre correspondant et diminue ainsi grandement le nombre de solutions potentielles à explorer.

3.1.3 Ordre des tâches

Les priorités des tâches sur un site sont attribuées dans l'ordre inverse des priorités. Ainsi, le premier nœud d'un sous-arbre (i.e. d'un site) correspond à la tâche τ_i la moins prioritaire sur le site modélisé par le sous-arbre. Nous montrons après que si la tâche τ_i correspondante n'est pas ordonnançable, alors aucun ordonnancement ayant τ_i au niveau de priorité le plus faible (parmi l'ensemble des priorités du site sur lequel elle s'exécute) conduira à un ordonnancement valide. Nous généralisons ainsi le résultat de [Aud91] au cas de tâches soumises à des gigues sur activations.

Dans un contexte de tâches indépendantes en monoprocesseur, l'instant critique survient lorsque toutes les tâches sont réveillées à la même date t. Ainsi, pour une configuration de n tâches, l'interférence subie par une tâche τ_i de priorité i est définie comme la charge engendrée par les tâches de priorité supérieure à i.

Si l'on se place dans le contexte d'application distribuée, les tâches peuvent être en relation avec d'autres tâches par l'intermédiaire de messages. Toutes ces dépendances sont modélisées par les gigues sur l'activation des tâches. Dans un tel contexte, le pire cas ne survient plus lorsque toutes les tâches sont réveillées à une même date t mais lorsque toutes les tâches reçoivent leurs messages au même instant.

Les gigues sur l'activation des tâches doivent donc être prises en compte dans le calcul de l'interférence induite par les tâches de plus forte priorité. Dans [Tin94], Tindell montre que la plus grande interférence engendrée par des tâches plus prioritaires

que τ_i dans une période d'activité de longueur t est bornée par :

$$\sum_{j=1}^{i-1} \left\lceil \frac{J_j + t}{T_j} \right\rceil C_j \tag{4}$$

De plus, l'accroissement des gigues sur l'activation des tâches τ_j ne peut faire décroître l'interférence engendrée par les tâches plus prioritaires.

Audsley [Aud91] montre, dans un contexte monoprocesseur avec tâches indépendantes, que l'accroissement de la priorité d'une tâche ne peut que diminuer ou laisser inchanger le temps de réponse de cette tâche. Ce résultat s'étend facilement au contexte monoprocesseur avec tâches soumises à la gigue sur l'activation des tâches.

Lemme 1 : *Pour un ensemble de gigues sur l'activation supposées fixes, dans un contexte de tâches indépendantes s'exécutant sur un processeur, diminuer la priorité d'une tâche τ_i (i.e. rendre la tâche τ_i moins prioritaire) ne peut pas augmenter le temps de réponse des autres tâches τ_k, avec $k \neq i$.*

Théorème 1 : *Soit τ une configuration de n tâches soumises aux gigues sur l'activation J_i s'exécutant sur un processeur, et ϕ_x une fonction d'affectation valide des priorités des niveaux $i,\ i+1,\ ...,\ n$. Il existe alors une affectation valide des priorités pour les tâches $\tau_i,\ 1 \leq i \leq n$ respectant l'ordre de ϕ_x.*

Preuve :
Considérons une affectation de priorité valide ϕ_y, et montrons que l'on peut transformer ϕ_y en affectant les mêmes tâches que ϕ_x aux niveaux de priorité $i,\ i+1,\ ...,\ n$ tout en respectant l'ordonnançabilité de ϕ_y. La preuve est faite par induction en transformant ϕ_y successivement par l'affectation des tâches $\phi_x^{-1}(n),\ \phi_x^{-1}(n-1),\ ...,\ \phi_x^{-1}(i)$ aux niveaux de priorité $n,\ n-1,\ ...,\ i$ dans l'affectation ϕ_y.
Base : Soit $\phi_x^{-1}(n) = \tau_A$ et $\phi_y(\tau_A) = m$, avec $m \leq n$. L'interférence due aux tâches plus prioritaires que τ_i est bornée par (4). L'interférence maximale est donc indépendante de l'ordre des priorités des tâches plus prioritaires que τ_i.
Hypothèse : nous faisons l'hypothèse que les tâches affectées aux niveaux de priorité $n-1,\ n-2,\ ..,\ i+1$ par ϕ_x sont affectées aux mêmes niveaux de priorité par la fonction ϕ_y, en gardant l'ordonnançabilité de la configuration τ.
Induction : Soit $\phi_x^{-1}(i) = \tau_B$ et $\phi_y(\tau_B) = m$, avec $m \leq i$. Les deux fonctions d'affectation ϕ_x et ϕ_y produisent l'affectation des mêmes tâches aux mêmes priorités pour les niveaux de priorités $n,\ n-1,\ ...,\ i+1$. La tâche τ_B se voit affecter la priorité i dans la fonction ϕ_y. Or l'on sait que τ_B est ordonnançable à ce niveau de priorité puisque $\phi_x^{-1}(i) = \tau_B$.
Après la modification de la fonction ϕ_y, les tâches affectées aux niveaux de priorité $1,\ 2,\ ...,\ i-1$ restent donc ordonnançables puisque, par le lemme 1, l'interférence induite par les tâches plus prioritaires n'a pas augmenté.

□

Afin d'améliorer la recherche, nous trions les indices des tâches selon la politique d'ordonnancement *"Deadline Monotonic"* qui donne la plus forte priorité à la tâche possédant la plus petite échéance relative ($D_i > D_j \Rightarrow Prio_i > Prio_j$). Cette heuristique impose que la première branche parcourue en profondeur d'abord respecte DM.

3.1.4 Structure d'un nœud

Un nœud de l'arbre de recherche représente l'affectation d'une priorité à une tâche. En pratique, chaque nœud mémorise une branche de l'arbre (i.e. une affectation ϕ_x). Ceci permet de stocker l'arbre dans une liste. Le parcours en *profondeur d'abord* revient à gérer cette liste selon la politique *FIFO*. La gigue sur l'activation est nécessaire pour le calcul du temps de réponse de la tâche par une analyse holistique puisqu'il permet de prendre en compte les relations de dépendances entre les différentes tâches et messages du système. Ainsi l'ensemble des estimations des bornes inférieures des gigues sur l'activation de toutes les tâches est également contenu dans un nœud, puisque ces gigues sur l'activation sont différentes selon l'assignation (i.e. la branche de l'arbre). De plus, la gigue sur l'activation des tâches permet de décider plus tôt si l'ensemble de priorités d'une branche incomplète de l'arbre de recherche ne produira pas d'ordonnancement valide, améliorant ainsi la rapidité de la procédure de recherche.

3.2 Arbre de recherche : Principe d'évaluation

A chaque création d'un nœud, il faut pouvoir décider si la tâche correspondante est ordonnançable avec le niveau de priorité assigné. Cette section décrit les évaluations réalisées dans les différents cas rencontrés lors de la procédure de recherche.

Le principe d'évaluation mis en place dans la méthode d'affectation des priorités repose sur le même principe que le système d'équations (2). Précisément, la structure générale de l'analyse holistique est reprise, et la principale différence réside dans le fait que toutes les priorités n'ont pas encore été affectées aux tâches et messages de l'application. En conséquence, seules des bornes inférieures des pires temps de réponse des tâches et messages seront évaluées ($LB(Tr_i)$). Les gigues sur l'activation des tâches et des messages qui en découlent sont donc elles-aussi des bornes inférieures ($LB(J_i)$).

Dans le but d'améliorer les performances de la procédure de recherche, une borne inférieure des gigues sur l'activation des tâches est calculée au préalable par la méthode présentée dans le paragraphe suivant pour toutes les tâches et tous les messages de l'application. Ces bornes inférieures sont injectées dans la première itération du calcul des bornes inférieures des pires temps de réponse.

3.2.1 Évaluation d'une borne inférieure des gigues sur l'activation des tâches

Dans l'analyse holistique, Tindell initialise les gigues sur l'activation des tâches à 0. La gigue sur l'activation modélise la dépendance entre les tâches et les messages de l'application. Pour toutes les tâches indépendantes, la gigue sur l'activation (J_i) est donc nulle. Pour les tâches possédant des relations de précédence le calcul de la gigue correspond au calcul d'une *date d'activation au plus tôt* (problème central de l'ordonnancement) [GM95].
Considérons le graphe présenté sur la figure 5 : pour qu'une tâche puisse débuter son exécution, toutes les tâches la reliant à α doivent être terminées. Le problème posé ici consiste donc à calculer la *date au plus tôt* de la tâche τ_4, c'est-à-dire la première date t à laquelle celle-ci va pouvoir commencer son exécution. Ce problème revient à chercher la longueur du plus long chemin de α à τ_4 dans un graphe sans circuit.

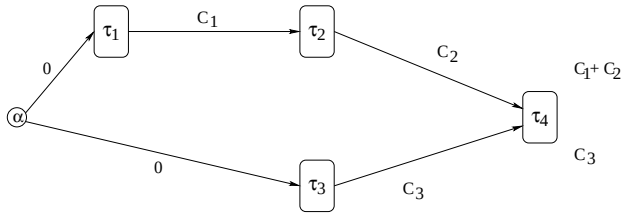

FIG. 5 – Graphe de calcul des dates d'activations au plus tôt

Au début de la procédure de recherche, toutes les bornes inférieures de gigue sur l'activation de toutes les tâches sont calculées en utilisant la méthode présentée ci-dessus. Soit Γ_i^{-1} l'ensemble des prédécesseurs immédiats de τ_i, le calcul (5) s'effectue selon l'ordre topologique du graphe de précédence. Les valeurs obtenues par les deux équation ci-dessous (5) vont initialiser le système (6).

$$\begin{array}{lcl} LB\left(J_i^{(0)}\right) & = & \max\limits_{j \in \Gamma_i^{-1}} \left(LB^{(0)}\left(J_j\right) + C_j\right) \\ LB\left(Tr_i^{(0)}\right) & = & LB\left(J_i^{(0)}\right) + C_i \end{array} \tag{5}$$

3.2.2 Évaluation d'une borne inférieure des pires temps de réponse

Lorsqu'une priorité est affectée à une tâche, un nœud contenant les informations présentées plus haut est créé. Afin de déterminer si cette tâche est ordonnançable à ce niveau de priorité, nous effectuons une estimation $LB(Tr_i)$ de son pire temps de réponse Tr_i. Pour ce faire, le système suivant est résolu :

$$1 \leq i \leq n \quad \left\{ \begin{array}{lcl} LB\left(Tr_i^{(k)}\right) & = & Evaluate\left(LB\left(J_i^{(k-1)}\right)\right) \\ LB\left(J_i^{(k)}\right) & = & \max\limits_{j \in \Gamma^{-1}} \left(LB\left(Tr_j^{(k)}\right)\right) \end{array} \right. \tag{6}$$

Le principe, qui est le même que celui de l'analyse holistique, est de recommencer les calculs jusqu'à ce qu'un point fixe soit trouvé. Le point fixe est obtenu lorsque pour le plus petit $k \in \mathbb{N}$:

$$LB\left(Tr_i\right) = LB\left(Tr_i^{(k-1)}\right) = LB\left(Tr_i^{(k)}\right), \quad 1 \leq i \leq n \tag{7}$$

La fonction *Evaluate* calcule une borne inférieure du temps de réponse d'une tâche en fonction des bornes inférieures des gigues sur l'activation. Cette évaluation porte sur un nœud donné, décrivant une branche (i.e. affectation) incomplète de l'arbre. Les calculs dépendent donc :

- du type du site. Si le site est régi par une politique d'ordonnancement préemptive, les tâches peuvent être interrompues. Si le site est régi par une politique d'ordonnancement non-préemptive, ou si le site est un réseau, les tâches modélisant les messages ne peuvent pas être interrompues.
- si l'affectation des priorités aux tâches a été effectuée ou non.

L'intérêt du système (6) est de propager les résultats de la fonction *Evaluate* immédiatement à l'ensemble des tâches via la mise à jour des gigues sur l'activation.
Dans la suite, nous détaillons la fonction *"Evaluate"*. La section 3.2.3 présente les calculs effectués dans le cas où la tâche analysée possède une priorité, tandis que la section 3.2.4 décrit l'analyse effectuée dans le cas d'une tâche non encore affectée.

3.2.3 Estimation pour une tâche ou un message possédant une priorité

Dans la suite, nous notons $[i]$ le numéro de la tâche affectée à la priorité i sur le processeur sur lequel elle s'exécute pour une affectation donnée (i.e. une branche).

Le calcul du temps de réponse d'une tâche et d'un message repose sur des résultats classiques de la littérature [LL73, GRS96, Tin94]. Plus précisément, le temps de réponse d'une tâche τ_i est basée sur la plus grande période de temps où le processeur est pleinement occupé par des tâches de priorités supérieures ou égales à i (*"i-level Busy Period"*)[Leh90]. La charge processeur d'une tâche $\tau_{[i]}$ est le rapport $\frac{C_{[i]}}{T_{[i]}}$. La longueur de cette période d'activité est alors la charge processeur engendrée par les tâches lorsque celles-ci sont activées simultanément lors de la réception de leurs premiers messages et les exécutions suivantes sont exécutées sans attente [Tin94]. Ainsi une période d'activité de longueur t, comportant q exécutions de $\tau_{[i]}$, est terminée si et seulement si : $t < (q+1)T_{[i]}$ (i.e. il n'existe plus de tâche à exécuter de priorité supérieure ou égale à i). Calculer le plus grand temps de réponse $Tr_{[i]}$ revient à examiner les Q exécutions de $\tau_{[i]}$ dans la période d'activité de niveau i et de longueur t. Une borne inférieure du pire temps de réponse $Tr_{[i]}$ affectée à la priorité i est alors donnée par :

$$LB\left(Tr_{[i]}\right) = \max_{q=1,2,\ldots,Q}\left(t + LB\left(J_{[i]}\right) - qT_{[i]}\right) \tag{8}$$

Selon la politique d'ordonnancement du site s'il s'agit d'un processeur ou si le site est un réseau, la longueur de la période d'activité se définit différemment.

Ordonnancement préemptif : (Cas d'un processeur) pour une tâche τ_t telle que $t = [i]$ par ψ_x, la longueur de la période d'activité se définit comme le point fixe de l'équation (10). Le principal facteur intervenant dans le calcul de la longueur de cette période d'activité de niveau i est l'interférence engendrée par les tâches plus prioritaires que τ_t dans le pire cas (9). Le point fixe est obtenu lorsque $t = t^{(k+1)} = t^{(k)}$.

$$\sum_{j=1}^{i} \left\lceil \frac{LB\left(J_{[j]}\right) + t}{T_{[j]}} \right\rceil C_{[j]} \tag{9}$$

$$t^{(k+1)} = (q+1)C_{[i]} + \sum_{j=1}^{i-1} \left\lceil \frac{LB\left(J_{[j]}\right) + t^{(k)}}{T_{[j]}} \right\rceil C_{[j]} \tag{10}$$

Ordonnancement non-préemptif :(Cas d'un réseau) les résultats conçus pour l'ordonnancement préemptif peuvent être aisément adaptés à l'ordonnancement non préemptif de tâches. L'unique différence provient du décalage pouvant être engendré par une tâche moins prioritaire, utilisant le processeur à l'instant critique. Les tâches ne pouvant être préemptées, l'instant critique est décalé d'au maximum la durée de la plus longue tâche, parmi les tâches moins prioritaires. Le pire cas survient alors lorsque cette dernière débute son exécution une unité de temps avant l'instant critique, c'est-à-dire à l'instant -1 en imposant que l'instant critique survienne à la date zéro (changement de l'origine des temps) [GRS96].

Théorème 2 : *[GRS96] En ordonnancement non préemptif, à priorité fixe, le pire temps de réponse d'une tâche τ_i est trouvé dans la période d'activité de niveau i et survient lorsque toutes les tâches plus prioritaires sont réveillées à un instant critique, et en réveillant la plus lente parmi les moins prioritaires (s'il en existe) à la date -1.*

L'interférence subit par une tâche de priorité i est bornée par la valeur suivante (11) :

$$\sum_{j=1}^{i-1} \left(\left\lfloor \frac{LB(J_{[j]}) + t}{T_{[j]}} \right\rfloor + 1 \right) C_{[j]} + \max_{i<k\leq n} (C_{[k]}) \tag{11}$$

La longueur de la période d'activité se définit comme le point fixe de l'équation 13 :

$$t = t^{(k+1)} = t^{(k)} \tag{12}$$

$$t^{(k+1)} = (q+1)C_{[i]} + \sum_{j=1}^{i-1} \left(\left\lfloor \frac{LB(J_{[j]}) + t^{(k)}}{T_{[j]}} \right\rfloor + 1 \right) C_{[j]} + \max_{i<k\leq n} (C_{[k]}) \tag{13}$$

Remarquons enfin que lorsqu'une branche complète est considérée (i.e. toutes les tâches possèdent une priorité), le système (6) est totalement équivalent au système (2).

3.2.4 Estimation pour une tâche ou un message ne possédant pas de priorité

Les tâches et messages concernés par ce cas sont ceux qui n'ont pas encore été traités par la procédure de recherche arborescente. Ne possédant pas de priorité, la méthode d'estimation présentée ci-dessus ne peut donc pas être appliquée. Or la résolution du système (6) nécessite des bornes inférieures du temps de réponse pour chaque tâche. Dans ce calcul, seul le pire temps d'exécution de τ_i peut être considéré pour modéliser l'action de l'ordonnancement sur le temps de réponse de τ_i. Précisément, la priorité de τ_i n'étant pas connue, cette tâche pourrait éventuellement obtenir la plus forte priorité par la suite de la procédure d'affectation. L'unique borne inférieure envisageable est donc obtenue dans le cas où la priorité est la plus forte, c'est-à-dire lorsque l'interférence due aux tâches plus prioritaires est nulle.

Dans le cas où τ_i ne possède pas de priorité, la fonction *Evaluate* évalue une borne inférieure du pire temps de réponse de τ_i en se basant sur l'idée présentée ci-dessus. De manière plus précise, deux cas doivent à nouveau être distingués : ordonnancement préemptif ou non-préemptif.

Ordonnancement préemptif :(Cas d'un processeur) afin de prendre en compte la borne inférieure de la gigue sur l'activation de τ_i la plus récente, le calcul effectué dans ce cas utilise la valeur $LB(J_i^{(k-1)})$:

$$LB(Tr_i^{(k)}) = LB(J_i^{(k-1)}) + C_i \tag{14}$$

Ordonnancement non préemptif :(Cas d'un réseau) avec une politique d'ordonnancement non préemptive, les tâches (messages) ne peuvent voir leur exécution interrompue. L'instant critique a lieu lorsque la tâche la plus longue parmi les moins prioritaires débute son exécution une unité de temps avant toutes les autres. Or la tâche τ_i ne possède pas de priorité et l'estimation d'une borne inférieure de son pire temps de réponse a lieu dans un contexte où tout se passe comme si elle était la plus prioritaire. L'estimation de $LB(Tr_i)$ doit donc prendre en compte la plus longue tâche parmi celles affectées sur le même site que τ_i. Ainsi, on obtient la borne par l'équation 15.

$$LB(Tr_i^{(k)}) = LB(J_i^{(k-1)}) + \max_{\substack{j=1..n \\ j\neq i}}(C_j) + C_i \tag{15}$$

Dans les deux cas, il est trivial de montrer que les bornes inférieures des pires temps de réponse ($LB(Tr_i)$) sont non décroissantes en fonction des gigues sur l'activation des tâches.

3.3 Arbre de recherche : Test et coupe

Les bornes inférieures de temps de réponse estimées par la méthode présentée dans la section précédente sont utilisées pour tester la validité de l'ensemble des

priorités en cours. Pour ce faire, le test suivant est réalisé pour toutes les tâches τ_i ou tous les messages m_i :

$$LB(Tr_i) > D_i \tag{16}$$

La règle ci-dessous est basée sur le test précédent :

Règle 1 : *Si il existe $i \in 1..n$ tel que $LB(Tr_i) > D_i$ alors l'ensemble des priorités en cours ne peut mener à une affectation valide. La branche associée au nœud est coupée, et un retour arrière (Backtracking) est effectué.*

Preuve :
Soit τ_i la tâche de priorité $Prio_i$ telle que $LB(Tr_i) > D_i$. Le théorème 1 montre qu'il ne peut exister de fonction d'affectation ϕ_x, telle que $\phi_x(\tau_i) = Prio_i$, aboutissant à un ensemble de priorité permettant de valider l'ordonnançabilité de l'application. Le nœud en cours peut donc être supprimé sans perte de solutions vis-à-vis de l'analyse holistique.
□

La méthode s'arrête dès qu'une branche complète de l'arbre conduit à une affectation des priorités permettant de valider l'application par une analyse holistique.

3.4 Expérimentation

Afin d'illustrer le type d'application à valider, nous décrivons figure 6 l'architecture informatique embarquée pour la gestion d'une automobile [CSSLC00]. Le système regroupe un ensemble d'ECU – Electronic Component Units – qui assurent les différentes fonctions. Ces unités ECU sont liées par deux réseaux : le réseau CAN, dédié aux fonctions critiques de l'automobile, telle que l'ABS et le contrôle moteur et le réseau VAN utilisé pour relier les ECU assurant des fonctions non-critiques (possédant des contraintes temporelles plus lâches) comme l'affichage sur la console et la gestion de la climatisation. L'ECU 6 de ce réseau est une passerelle entre le réseau CAN et le réseau VAN.

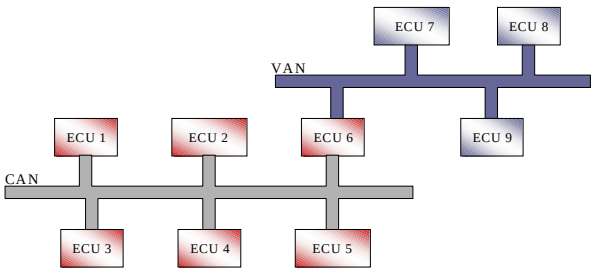

FIG. 6 – Structure d'un système embarqué dans l'automobile

Cette application est composée de 44 tâches s'exécutant sur les différents ECU et 19 messages circulant sur les deux réseaux. Les différentes charges et la répartition des

tâches et messages sont présentés dans le tableau ci-dessous. Le choix des priorités des tâches et des messages est prépondérant pour ne pas surdimensionner le système.

Site	Charge	Nb tâches	Site	Charge	Nb tâches
Site 1	0,67	7	Site 2	0,36	4
Site 3	0,5	6	Site 4	0,48	2
Site 5	0,67	5	Site 6	0,47	7
Site 7	0,07	4	Site 8	0,31	5
Site 9	0,08	4	Site 10(CAN)	0,2	12
Site 11(VAN)	0,06	7			

Pour ce jeu de valeurs, notre méthode n'a trouvé aucune affectation des priorités aux tâches et aux messages permettant de valider l'application. Le temps de calcul sur une machine de type personnel (Pentium III 500) est de 121,7 secondes et le nombre de *coupes* réalisées durant la procédure de recherche est de 128134.

Des tests plus approfondis de la méthode présentée dans cet article sont en cours de réalisation. Le test principal concerne la taille des problèmes pouvant être efficacement résolus avec notre méthode. Les principaux indicateurs sont le temps de calcul, le nombre de nœuds visités et le nombre de retours en arrières effectués. Le temps de calcul permet de déterminer les limites de notre approche. Le nombre de nœuds visités et de retours en arrières permettent d'évaluer la taille de l'espace de recherche explicitement parcouru pour énumérer implicitement l'espace des solutions.
Nous souhaitons évaluer le comportement de notre méthode sur différentes typologies de problème en fonction de la charge du réseau et des sites. De plus, les différentes améliorations possibles présentées au cours de cet article sont également mises en place afin de tester leurs influences sur les performances de la méthode :

- ordre des sites selon la charge,
- pré-classement des tâches selon Deadline Monotonic.

3.5 Conclusion

Nous avons étudié le problème de l'affectation des priorités fixes pour les tâches à échéances strictes d'un système temps réel distribué. Dans ce type d'application, les priorités sont fixées à la conception du système. En monoprocesseur, des règles simples permettent d'attribuer efficacement les priorités aux tâches, comme Rate Monotonic ou Deadline Monotonic. De telles règles n'existent pas pour les systèmes distribués et un choix arbitraire des priorités conduit généralement à une mauvaise utilisation des ressources du système (processeurs et réseaux). Nous avons présenté une méthode optimale de détermination des priorités vis-à-vis de l'analyse holistique, qui est la meilleure méthode de validation des systèmes distribués dans le contexte temps réel dur.

La méthode présentée est actuellement en cours d'expérimentation afin de permettre d'affiner ses performances et d'en fixer ses limites. Les perspectives de ce travail sont d'étendre l'approche à l'affectation des priorités simultanément avec le placement des tâches sur les sites en vue d'une exécution en-ligne de celles-ci sur chaque site.

L'objectif sera alors de minimiser les communications sur le réseau en affectant les tâches communicantes sur le même site. Une contrainte de communication sera alors modélisée par une simple contrainte de précédence (i.e. communication de durée nulle lorsqu'une mémoire partagée est utilisée pour échanger des données).

Références

[Aud91] N.C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS-164, University of York, Nov 1991.

[But97] G.C. Buttazzo. *Hard real-time computing systems : predictable scheduling algorithms and applications*. Kluwer Academic Publishers, 1997.

[CSSLC00] P. Castelpietra, Y-Q. Song, F. Simonot-Lion, and O. Cayrol. Performance evaluation of a multiple networked in-vehicle embedded architecture. In *Procedings of the 3rd IEEE International Workshop on Factory Communication Systems*, Porto, september 2000.

[GH95] J.J. Gutiérrez Garcia and M. Gonzales Harbour. Optimized priority assignement for tasks and messages in distributed hard real-time systems. In *Procedings of the 3d Workshop on Parallel and Distributed Hard Real-Time Systems*, pages 124–132, Santa Barbara, April 1995.

[GM95] M. Gondran and M. Minoux. *Graphes et algorithmes*. Eyrolles, 1995.

[GRS96] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Technical Report 2966, Rapport INRIA, 1996.

[Leh90] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *11th Real-Time Systems Symposium*, pages 201–209, Lake Buena Vista, Florida, 1990.

[LL73] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in real-time environment. 20(1) :46–61, 1973.

[RRC00] P. Richard, M. Richard, and F. Cottet. Analyse holistique des systèmes temps réel distribués : Principes et algorithmes. *Calculateurs Parallèles, Réseaux et Systèmes Répartis – N° spécial : Les problèmes d'ordonnancement dans les machines parallèles et architecture distribuées*, à paraître, 2000.

[TBW92] K.W. Tindell, A. Burns, and A.J. Wellings. Allocating hard real-time tasks : an NP-hard problem made easy. *Real-Time System Journal*, 4(12) :145–165, 1992.

[TC94] K. Tindell and J. Clarck. Holistic schedulability analysis for distributed hard real-time systems. In *Microprocessors and Microprogramming*, volume 40, 1994.

[Tin94] K.W. Tindell. *Fixed priority scheduling of hard-real-time systems*. PhD thesis, University of York, 1994.