Rapport Interne No: 2001–001

Méthode d'affectation optimale des priorités des tâches et des messages dans les systèmes distribués temps-réel basée sur l'analyse holistique

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

Laboratoire d'Informatique Scientifique et Industrielle
École Nationale Supérieure de Mécanique et d'Aérotechnique
Futuroscope

24 avril 2001

Table des matières

Chapitre 1

Ordonnancement des systèmes temps-réel distribués

1.1 Problématique

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 1.1, 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 un premier temps, nous rappelons brièvement les principes d'une méthode de validation *pire cas* considérant les priorités des tâches comme des paramètres : l'analyse holistique. Puis, dans le second chapitre, nous présentons une méthode d'affectation des priorités aux tâches et aux messages qui est optimale vis-à-vis de l'analyse holistique. Cette méthode est étendue afin de prendre en compte le partage de ressources critiques entre tâches d'un même site. Nous évaluons ensuite les performances de notre méthode sur une application issue du milieu industriel automobile, ainsi que sur un ensemble d'applications générées aléatoirement. Enfin, dans une troisième partie, nous proposons les bases d'une extension de cette méthode d'affectation permettant de placer les tâches sur les différents processeurs et d'affecter les priorités de ces dernières.

Notations :

τ_i identificateur de tâche

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

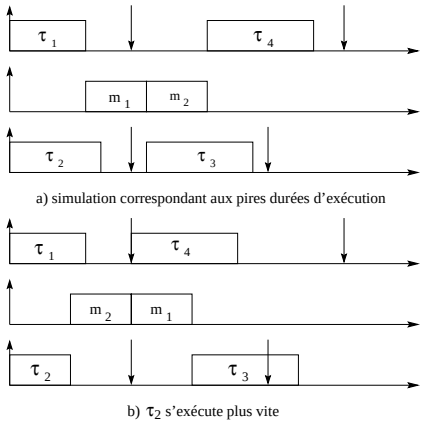

FIG. 1.1 – Simulation et validation d'une configuration de tâches

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

1.2 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 1.2 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.

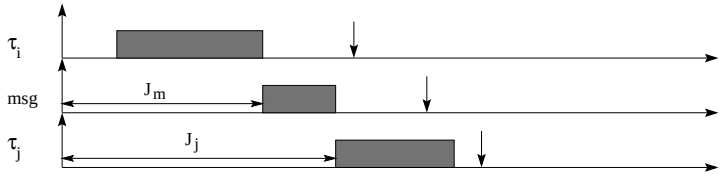

FIG. 1.2 – La gigue modélise l'attente due à la fin de la tâche précédente.

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

$$\begin{array}{ll} \text{Pour les tâches :} & J_i = \max_{j \in \Gamma_i^{-1}} (Tr_j) \\ \text{Pour les messages :} & J_m = \max_{j \in \Gamma_m^{-1}} (Tr_j) \end{array} \quad (1.1)$$

Toute modification des gigues va entraîner des modifications des temps de réponse des tâches dépendantes ou 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 :

$$1 \leq i \leq n \qquad \left\{ \begin{array}{l} \left\{ \begin{array}{lcl} Tr_i^{(0)} & = & C_i \\ J_i^{(0)} & = & 0 \end{array} \right. \\ \left\{ \begin{array}{lcl} Tr_i^{(k)} & = & ResponseTime\left(J_i^{(k-1)}\right) \\ J_i^{(k)} & = & \max\limits_{j \in \Gamma_i^{-1}} \left(Tr_j^{(k)}\right) \end{array} \right. \end{array} \right. \tag{1.2}$$

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

$$Tr_i = Tr_i^{(k)} = Tr_i^{(k-1)}, 1 \leq \ i \leq n \tag{1.3}$$

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 [RRC00b, RRC00a].

1.3 objectif

L'analyse holistique permet donc 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, Da01, OF00], qui ne parcourent pas toutes les solutions potentielles, mais seulement un sous-ensemble (i.e. méthodes heuristiques).

Chapitre 2

Méthode d'affectation des priorités des tâches et des messages

Notre méthode d'affectation des priorités aux tâches et aux messages est basée sur le principe des *méthodes par séparation et évaluation*. Ces procédures de recherche s'appuient sur une structure arborescente permettant la mémorisation des solutions potentielles.

2.1 Structure de l'arbre de recherche

Dans cette section, nous présentons la structure de l'arbre de recherche. Ce dernier contient l'ensemble des permutations possibles des priorités aux tâches. Chaque nœud de l'arbre représente l'affectation d'une priorité à une tâche.

Une énumération simple de toutes les affectations possibles des priorités produit un arbre possédant des branches redondantes. La figure 2.1 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 2.1 énumère un espace de 12 solutions.

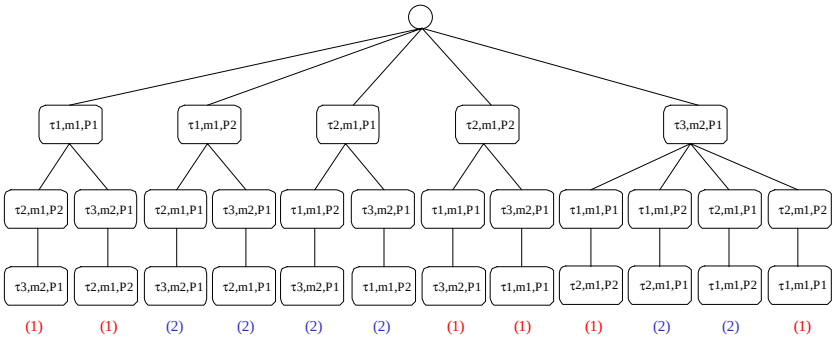

FIG. 2.1 – Phénomène de redondance lors d'une énumération simple

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 2.2 présente notre technique d'énumération sur le même exemple que précédemment (i.e. figure 2.1). 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.

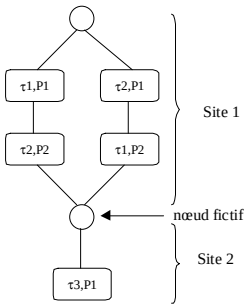

FIG. 2.2 – Structure générale de l'arbre de recherche

2.2 Principes généraux de la procédure de recherche

Nous présentons ci-dessous l'algorithme de la *procédure par séparation et évaluation*. Les nœuds non traités sont placés dans un ensemble de nœuds Actifs A. La racine d'un sous-arbre est initialisée avec l'ensemble des tâches ordonnançable au niveau de priorité le plus faible, sur le site correspondant.

```
Procédure de Recherche des Priorités
  Initialisation de l'ensemble Actif A avec les noeuds {1..n} correspondant aux
  tâches ordonnançables à la plus faible priorité sur le premier sous-arbre.
  TantQue A ≠ ∅ Faire
     Selection d'un noeud dans l'ensemble A suivant la règle de construction,
     Génération d'un ensemble B de noeuds fils en fonction de la règle de
     branchement,
     Calcul de la borne inférieure pour chaque nœ ud ∈ B selon le
     principe d'évaluation,
     Suppression des noeuds de B selon la règle de coupe,
     Si il existe une feuille valide dans B
        Stop
     Sinon
        Copie de tous les éléments de B dans A.
     FinSi
  FinFaire
```

La règle de sélection permet de choisir le prochain nœud à explorer dans la liste. Dans la littérature, il existe différentes stratégies : FIFO et LIFO. La stratégie LIFO correspond à un parcours en profondeur, alors que FIFO correspond à un parcours en largeur. Nous avons implémenté un parcours en profondeur.

La règle de branchement génère les nœuds fils du nœud exploré et les stocke dans l'ensemble B. Deux cas sont à considérer :

- Il reste des tâches ne possédant pas de priorité dans le sous-arbre courant : un nœud est généré pour toutes ces tâches avec le niveau de priorité suivant,
- Toutes les tâches possèdent une priorité : un nœud est créé pour chaque tâches du sous-arbre suivant en se voyant affecter le niveau de priorité le plus faible.
  Si le sous-arbre courant est le dernier, aucun nœud n'est généré.

L'ensemble B est pré-trié selon la politique d'ordonnancement DM. Ainsi en utilisant une stratégie de sélection en profondeur, la première feuille obtenue est celle qui assigne les priorités aux tâches selon DM sur chaque processeurs et réseaux.

La règle d'évaluation est utilisée afin de calculer les bornes inférieures du pire temps de réponse de chaque tâche. Précisément, pour chaque nœud traité, une borne inférieure est calculée pour l'ensemble des tâches et messages possédant ou non une priorité.

La règle de coupe est appliquée à la suite du calcul des bornes inférieures des pires temps de réponse. Un nœud fils est considéré comme valide *si et seulement si* $\forall\ 1 \leq i \leq n,\ Tr_i \leq D_i$. Si il existe une tâche ou un message possédant une borne inférieure de son pire temps de réponse supérieure à son échéance, le nœud (i.e. la solution en cours) peut-être abandonnée. (i.e. le nœud n'est pas inséré dans l'ensemble B).

2.3 Structure arborescente : principe de séparation

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).

2.3.1 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.

2.3.2 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 mono-processeur, 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 :

$$\sum_{j=1}^{i-1} \left\lceil \frac{J_j + t}{T_j} \right\rceil C_j \tag{2.1}$$

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 mono-processeur 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 mono-processeur 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 (2.1). 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.

2.3.3 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.

2.4 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 (1.2). Précisément, la structure générale de l'analyse holistique est reprise, et la principale différence réside dans le fait que toutes les priorités ne sont pas encore 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.

2.4.1 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 2.3 : 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.

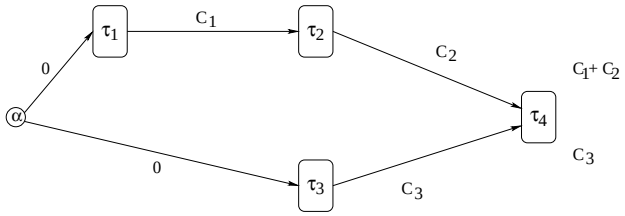

FIG. 2.3 – Graphe de calcul des dates d'activations au plus tôt

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 (2.2) s'effectue selon l'ordre topologique du graphe de précédence. Les valeurs obtenues par les deux équation ci-dessous (2.2) vont initialiser le système (2.3).

$$\begin{array}{rcl} LB\left(J_i^{(0)}\right) & = & \max\limits_{j\in\Gamma_i^{-1}}\left(LB^{(0)}\left(J_j\right)+C_j\right) \\ LB\left(Tr_i^{(0)}\right) & = & LB\left(J_i^{(0)}\right)+C_i \end{array} \tag{2.2}$$

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 :

$$1\leq i\leq n \quad \left\{ \begin{array}{rcl} LB\left(Tr_i^{(k)}\right) & = & Evaluate\left(LB\left(J_i^{(k-1)}\right)\right) \\ LB\left(J_i^{(k)}\right) & = & \max\limits_{j\in\Gamma^{-1}}\left(LB\left(Tr_j^{(k)}\right)\right) \end{array} \right. \tag{2.3}$$

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 $k\in\mathbb{N}$:

$$LB\left(Tr_i\right)=LB\left(Tr_i^{(k-1)}\right)=LB\left(Tr_i^{(k)}\right),\quad 1\leq i\leq n \tag{2.4}$$

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 (2.3) 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 2.4.2 présente les calculs effectués dans le cas où la tâche analysée possède une priorité, tandis que la section 2.4.3 décrit l'analyse effectuée dans le cas d'une tâche non encore affectée.

2.4.2 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 :

$$LB\left(Tr_{[i]}\right) = \max_{q=1,2,\ldots,Q} \left(t + LB\left(J_{[i]}\right) - qT_{[i]}\right) \tag{2.5}$$

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 (2.7). 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 (2.6). Le point fixe est obtenu lorsque $t = t^{(k+1)} = t^{(k)}$.

$$\sum_{j=1}^{i} \left\lceil \frac{LB\left(J_{[j]}\right) + t}{T_{[j]}} \right\rceil C_{[j]} \tag{2.6}$$

$$t^{(k+1)} = (q+1)C_{[i]} + \sum_{j=1}^{i-1} \left\lceil \frac{LB\left(J_{[j]}\right) + t^{(k)}}{T_{[j]}} \right\rceil C_{[j]} \tag{2.7}$$

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 (2.8) :

$$\sum_{j=1}^{i-1} \left(\left\lfloor \frac{LB(J_{[j]}) + t}{T_{[j]}} \right\rfloor + 1 \right) C_{[j]} + \max_{i<k\leq n} \left(C_{[k]}\right) \tag{2.8}$$

La longueur de la période d'activité se définit comme le point fixe de l'équation 2.10 :

$$t = t^{(k+1)} = t^{(k)} \tag{2.9}$$

$$t^{(k+1)} = (q+1)C_{[i]} + \sum_{j=1}^{i-1} \left(\left\lfloor \frac{LB(J_{[j]}) + t^{(k)}}{T_{[j]}} \right\rfloor + 1 \right) C_{[j]} + \max_{i<k\leq n} (C_{[k]}) \tag{2.10}$$

Remarquons enfin que lorsqu'une branche complète est considérée (i.e. toutes les tâches possèdent une priorité), le système (2.3) est totalement équivalent au système (1.2).

2.4.3 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 (2.3) 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 dans 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 $LB(J_i^{(k-1)})$:

$$LB(Tr_i^{(k)}) = LB(J_i^{(k-1)}) + C_i \tag{2.11}$$

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 2.12.

$$LB(Tr_i^{(k)}) = LB(J_i^{(k-1)}) + \max_{\substack{j=1\ldots n \\ j\neq i}} (C_j) + C_i \tag{2.12}$$

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.

2.4.4 Optimalité vis-à-vis de l'analyse holistique

Le théorème 3 ci-dessous montre que la méthode d'affectation des priorités est bien optimale vis-à-vis de l'analyse holistique.

Théorème 3 : *Soit $A = (V, E)$ l'arbre de recherche construit par la procédure de recherche, avec V l'ensemble des nœuds, et E l'ensemble des arcs. Ainsi $\forall\ a,\ b \in V^2$ tel que $a \prec b$ dans l'arbre de recherche, alors $R_i^{(a)} \leq R_i^{(b)},\ 1 \leq i \leq n$, où $R_i^{(a)}$ (resp. $R_i^{(a)}$) est une borne inférieure du pire temps de réponse de la tâche τ_i au niveau du nœud a (resp. b).*

Preuve:
Soient une branche quelconque de l'arbre de recherche, et a, b deux nœuds appartenant à cette branche. Nous faisons l'hypothèse que $a \prec b$. Deux cas sont à considérés selon que a et b appartiennent au même sous-arbre ou non.

Si a et b se situent sur deux sous-arbres différents, le temps de réponse $R_i^{(b)}$ ne dépend que de la gigue sur l'activation des tâches. Le système (2.3) est résolu et ce dernier implique que les gigues sur l'activation des tâches sont non-décroissantes pour toutes les tâches. On obtient bien ainsi $R_i^{(a)} \leq R_i^{(b)},\ 1 \leq i \leq n$.
Si les nœuds a et b appartiennent au même sous-arbre, ils représentent alors l'affectation des priorités aux tâches sur le même processeur. Considérons une tâche τ_i quelconque s'exécutant sur le processeur modélisé par le sous-arbre contenant les nœuds a et b. Si τ_i possède une priorité, au niveau du nœud a ainsi qu'au niveau du nœud b, son temps de réponse ne peut évoluer qu'en fonction de la gigue sur l'activation de τ_i (i.e. temps de réponse de ces prédécesseurs immédiats). Or, la fonction *Evaluate* étant non-décroissante en fonction des gigues sur l'activation des tâches, nous montrons de nouveau que $R_i^{(a)} \leq R_i^{(b)},\ 1 \leq i \leq n$.
Si τ_i possède une priorité dans le nœud b, alors qu'elle n'était pas encore affectée dans le nœud a, la borne inférieure du pire temps de réponse de τ_i n'est pas calculée de la même manière en a qu'en b. Précisément, dans le cas préemptif, la borne obtenue par l'équation (2.11) est nécessairement inférieure à celle obtenue par la résolution de l'équation (2.7). De même, dans un contexte non-préemptif, si la tâche τ_k entraînant le facteur de blocage dans l'équation (2.12) n'est pas la même que dans l'équation (2.10), alors τ_k ne peut être ordonnancée qu'à un niveau de priorité supérieur à celui de τ_i. Ainsi la borne calculée par l'équation (2.12) (i.e. au niveau du nœud a) est nécessairement inférieure ou égale à celle obtenue par (2.10) (i.e. au niveau du nœud b). Nous montrons bien que $R_i^{(a)} \leq R_i^{(b)},\ 1 \leq i \leq n$ et ainsi, l'optimalité de la procédure de recherche vis-à-vis de l'analyse holistique.
□

2.5 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 :

$$LB(Tr_i) > D_i \tag{2.13}$$

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.

2.6 Extension au protocole d'accès aux ressources critiques

Dans la majorité des applications industrielles, les tâches s'exécutant sur un même processeur partagent des ressources critiques. Dans cette section nous faisons l'hypothèse que deux tâches s'exécutant sur des processeurs différents ne peuvent pas partager de ressources critiques. De plus, sur chaque processeur, nous supposons que l'accès aux ressources est régi par un protocole d'accès. Les noyaux temps-réel de type OSEK/VDX [OSE97], qui deviennent de plus en plus répandus dans le milieu automobile, proposent le protocole à priorité plafond (PPP, Priority Ceiling Protocol) [SRL90, SRL87, RSL88]. La validation d'un ensemble de tâches s'exécutant sur un même processeur et partageant des ressources critiques est généralement basée sur l'analyse des pires temps de réponse des tâches. L'analyse prend alors en considération le pire temps de blocage induit par le partage de ressources critiques (noté B_i). Nous étendons les formules permettant de calculer une borne inférieure de ce pire temps de blocage (i.e. facteur de blocage) dans le cas où toutes les priorités ne sont pas connues lors de la procédure de recherche.

Le protocole à priorité plafond permet de borner le phénomène d'inversion de priorité et d'éviter la formation d'interblocage et de chaîne de blocage. Le principe de ce protocole est d'utiliser une règle gérant la demande d'obtention d'un sémaphore binaire libre. Précisément, une tâche ne peut accéder à une ressource s'il existe des sémaphores vérouillés pouvant la bloquer dans la suite de son exécution. Cette règle implique qu'une tâche τ_i ne pourra jamais être bloquée par une tâche de priorité inférieure lorsque τ_i peut entrer dans sa première section critique. La technique utilisée est d'affecter, à chaque sémaphore binaire protégeant une ressource, une *priorité plafond* égale à la priorité de la tâche de niveau de priorité le plus élevé pouvant accéder à ce sémaphore durant son exécution. Ainsi une tâche τ_i est autorisée à entrer en section critique si, et seulement si, sa priorité est supérieure aux priorités plafonds de tous les sémaphores vérouillés par une tâche autre que τ_i au moment où τ_i effectue sa demande. En annexe, nous présentons plus en détail le protocole à priorité plafond ainsi que le calcul du facteur de blocage, lorsque toutes les priorités des tâches sont connues. Le lecteur peut consulter [SRL90] pour une description complète du protocole à priorité plafond.

Afin de définir une borne inférieure du pire temps de blocage induit par le partage de ressources, nous introduisons les notations suivantes :

- Chaque sémaphore S_k de l'application se voit assigner une priorité plafond $C(S_k)$ égale à la priorité de la tâche de niveau de priorité le plus élevé pouvant demander le verrouillage de S_k. Les priorités n'étant pas connues, le calcul des priorités plafonds doit être effectué au cours de la procédure de recherche.
- $L_{k,p}^{(\nu)}$ est l'ensemble des tâches pouvant verrouiller le sémaphore S_k et possédant une priorité au niveau du nœud ν sur le processeur p. Cet ensemble est mis à jour tant qu'il reste des priorités non-affectées aux tâches sur le processeur p.
- π_i représente le niveau de priorité d'une tâche τ_i au niveau d'un nœud de la structure arborescente. Le niveau de priorité le plus fort est le niveau 1.
- l_p est le nombre de niveau de priorité déjà affecté sur le processeur p au moment de l'exploration d'un nœud.
- $D_{i,k}$ est la durée de la plus longue section critique exécutée par la tâche τ_i parmi celles gardées par le sémaphore binaire S_k. Ces valeurs permettent de modéliser la prise multiple (non-entrelacée) d'une même ressource au cours de l'exécution d'une instance de τ_i. Elles peuvent être calculées pour chaque sémaphore et chaque tâche avant le début de la procédure de recherche.
- Z_i est l'ensemble des sémaphores binaires protégeant les ressources critiques nécessaires à l'exécution de τ_i.
- B_i est le pire temps de blocage induit par le partage de ressources critiques.

Dans la suite nous proposons deux extensions du calcul du facteur de blocage dans le cas où toutes les priorités ne sont pas affectées dans la branche en cours. La première traite le cas dans lequel aucune priorité n'est affectée sur un processeur p en proposant une borne inférieure triviale ; $B_i = 0$. Dans la deuxième extension, nous présentons une méthode améliorant la borne inférieure de la pire durée de blocage dans ce cas.

2.6.1 Première extension

Plusieurs cas doivent être considérés en fonction du processeur p sur lequel s'exécute la tâche τ_i et en fonction du nombre de priorités affectées au moment de l'estimation de la borne inférieure du pire temps de réponse de la tâche τ_i.

1. si $l_p = n_p$, avec n_p le nombre de tâches s'exécutant sur le processeur p, alors toutes les tâches du site p possèdent une priorité. Ainsi, les priorités plafonds $C(S_k)$ de tous les sémaphores binaires S_k protégeant l'accès aux ressources critiques de ce site peuvent être calculées. Le facteur B_i peut alors être calculé selon l'équation (2.14) conformément à la formule issue de [SRL90, But97] (Cf. en annexes).

$$B_i = \max_{j,k} \{D_{j,k} | \pi_j > \pi_i,\ C(S_k) \leq \pi_i\} \tag{2.14}$$

   Dans ce cas, le facteur de blocage n'évoluera pas dans la suite du traitement des nœuds fils. En effet, les priorités sont déjà toutes assignées sur le processeur et ne seront donc plus modifiées. Les priorités plafonds restent alors identiques, ainsi que le facteur B_i.

2. si $1 < l_p < n_p$ alors le processeur p est en cours de traitement dans la procédure de recherche. Les priorités plafonds des sémaphores ne peuvent être calculées de manière exacte puisque au moins une tâche s'exécutant sur ce site ne possède pas de priorité. Soit τ_i une tâche appartenant au site p : deux cas doivent être considérés :

(a) si τ_i possède une priorité, alors tous les niveaux de priorités inférieurs au sien sont affectés à des tâches. Dans ce cas, toutes les priorités plafonds des sémaphores pouvant bloquer τ_i peuvent être calculées. Ainsi, une borne inférieure du pire temps de blocage B_i est obtenue par l'équation (2.14). Notons que comme précédemment, le facteur B_i n'évoluera pas dans l'exploration des solutions réalisant les mêmes affectations de priorités, pour le processeur p, que dans la branche en cours.

(b) si τ_i ne possède pas encore de priorité, elle peut être affectée au niveau de priorité le plus fort durant la suite de la procédure de recherche. Nous calculons alors une borne inférieure de B_i en considérant uniquement les tâches de plus faible priorité que τ_i, c'est-à-dire les tâches déjà affectées. Par exemple, si $n_p = 4$ et $l_p = 1$, une tâche possède le niveau de priorité le plus faible (i.e. 4) et les autres ne sont pas encore affectés. Ainsi, dans le calcul de la borne inférieure du pire temps de blocage B_i, nous ne considérons uniquement les tâches τ_j telles que :

$$\pi_j > n_p - l_p \Leftrightarrow \pi_j \geq n_p - l_p + 1 = 4$$

En considérant les tâches affectées dans l'intervalle de priorité $[n_p - l_p + 1 \cdots n_p]$, l'équation (2.15) permet de calculer une borne inférieure du pire temps de blocage B_i.

$$B_i = \max_{j,k} \{D_{j,k} | \pi_j > n_p - l_p, \pi_j > \pi_i,\ C(S_k) \leq \pi_i\} \tag{2.15}$$

Dans ce contexte, le facteur de blocage B_i est mis-à-jour tant que τ_i ne possède pas de priorité au niveau du nœud courant. En pratique, lors de l'affectation d'une priorité à une tâche τ_j, $\tau_j \neq \tau_i$, la plus longue section critique de τ_j peut entraîner la mise-à-jour de la pire durée de blocage B_i. B_i est non-décroissant dans la suite de l'exploration des nœuds fils du nœud courant.

3. si $lp = 0$, le processeur p n'a pas encore été exploré. Les priorités plafonds des sémaphores ne peuvent pas être évaluées puisque aucune tâche ne possède de priorité sur ce processeur. Ainsi, toute tâche est potentiellement candidate pour être ordonnancée au niveau de priorité le plus faible. La seule borne inférieure obtenue dans ce cas est donnée par la formule (2.16).

$$B_i = 0 \tag{2.16}$$

Les formules obtenues dans les différents cas sont regroupées dans le système (2.17).

$$\begin{cases} C(S_k) & = & \min\limits_{i \in L_{k,p}^{(\nu)}} (\pi_i) \\ B_i & = & \max \left(0;\ \max\limits_{j \in L_{k,p}^{(\nu)}} \{D_{j,k} | \pi_j > \pi_i,\ C(S_k) \leq \pi_i\} \right) \end{cases} \tag{2.17}$$

Lorsque toutes les tâches d'un même processeur p ne possèdent pas de priorité, nous avons vu précédemment qu'une borne inférieure du facteur de blocage B_i pouvait être obtenue par (2.16). Dans la section suivante, présentant la deuxième extension, nous proposons une amélioration de la borne inférieure dans ce cas.

2.6.2 Deuxième extension

Pour les cas $l_p = n_p$ et $1 < l_p < n_p$, la borne inférieure du pire temps de blocage induit par le partage de ressources critiques est obtenue comme précédemment. Dans la suite nous proposons une amélioration de la borne inférieure estimée dans le cas $l_p = 0$.

Si $l_p = 0$, aucune des tâches s'exécutant sur le processeur p ne possèdent de priorité. Nous faisons alors l'hypothèse que la tâche τ_i est ordonnancée au niveau de priorité le plus élevé. Cette hypothèse est la même que celle faite dans le même contexte, mais sans partage de ressources. Deux cas se présentent :

1. la tâche τ_i ne partage pas de ressources critiques avec les autres tâches qui s'exécutent sur p : dans ce cas le calcul de la borne inférieure du pire temps de blocage B_i est le même, i.e. $B_i = 0$.
2. la tâche τ_i partage au moins une ressource critique avec au moins une autre tâche s'exécutant sur p : soit Z_i l'ensemble des sémaphores binaires protégeant les ressources critiques nécessaires à τ_i. D'après l'hypothèse faite, tout se passe comme si τ_i possédait la plus forte priorité. Ainsi, tous les sémaphores appartenant à Z_i possèdent une priorité plafond égale à la priorité de τ_i. Le facteur B_i correspond donc

à la plus grande section critique exécutée par une tâche τ_l avec $l \neq i$ et protégée par un sémaphore $S_{i,k}$ appartenant à Z_i. B_i est ainsi obtenu par (2.18).

$$B_i = \max_{\forall j \neq i, k \in Z_i} \{D_{j,k}\} \tag{2.18}$$

Nous intégrons maintenant les bornes inférieures du facteur de blocage B_i dans l'analyse du pire temps de réponse de τ_i. B_i représente l'interférence des tâches de niveau de priorité inférieur à celui de τ_i dans la periode d'activité de niveau i. Ainsi, il ne doit être considéré qu'une seule fois dans celle-ci. Les équations obtenues dans le contexte sans ressources paratgées sont modifiées en ajoutant le facteur B_i afin de prendre en compte le blocage induit par le partage de ressource dans la période d'activitée de niveau i.

Si au moins une tâche possède une priorité sur le processeur p, les bornes inférieures des pires temps de réponse sont non-décroissantes en fonction des gigues et des durées de blocage B_i dans chaque branche de l'arbre de recherche. Le théorème 3 reste valide et l'optimalité de notre méthode est conservée. Le lemme 2 étend le théorème 3 aux processeurs non explorés dans l'arbre de recherche.

Lemme 2 : *Soit p un processeur sur lequel s'exécutent n_p tâches partageant des ressources critiques et ne possédant pas de priorité. Soit τ_i une tâche s'exécutant sur p. Une borne inférieure du pire temps de réponse est obtenue par le système suivant :*

$$LB(Tr_i^{(k)}) \quad = \quad LB(J_i^{(k-1)}) + C_i + \max_{\forall j \neq i, k \in Z_i} \{D_{j,k}\} \tag{2.19}$$

$LB(Tr_i)$ est non-décroissant dans chaque branche de l'arbre.

Preuve:
Soient a, b deux nœuds de l'arbre de recherche affectant des priorités à deux tâches du processeurs p. Sans perte de généralité, nous considérons $a \prec b$ dans la branche de l'arbre de recherche en cours d'exploration. Nous supposons que la tâche τ_i ne possède pas de priorité dans le nœud a et en possède une dans le nœud b.
L'evaluation du temps de réponse de τ_i dans le nœud a suppose que τ_i possède la plus forte priorité. Le calcul du facteur de blocage conduit à considérer la plus longue section critique parmi les autres tâches (qui sont moins prioritaires). Soit $D_{l,k} = \min_{\forall j \neq i,\ k \in Z_i} \{D_{j,k}\}$ cette section critique appartenant à la tâche τ_l.
τ_i possède une priorité pour le nœud b. Nous devons considérer deux cas, suivant que τ_l est plus prioritaire que τ_i ou non.

1. $\pi_l > \pi_i$: τ_l est moins prioritaire que τ_i. En conséquence, $D_{j,k}$ est toujours la plus longue section critique pouvant bloquer τ_i. On vérifie donc que $LB^{(b)}(Tr_i) \geq LB^{(a)}(Tr_i)$.
2. $\pi_l < \pi_i$: τ_l est plus prioritaire que τ_i. Elle sera donc prise en compte dans l'interférence de τ_i, et non plus dans le facteur de blocage. Comme $C_l \geq D_{l,k}$, on vérifie que $LB^{(b)}(Tr_i) \geq LB^{(a)}(Tr_i)$.

□

2.7 Expérimentation

2.7.1 Cas d'étude

Dans cette section nous présentons le fonctionnement de notre méthode sur un exemple simple composé de trois sites,sur lesquels sont répartis 14 tâches et d'un réseau sur lequel circulent 2 messages. Les tâches et messages ainsi que leur répartition sur les différents sites sont présentés dans la table 2.1. La structure des communications est donnée sur la figure 2.4.

Les sites sont triés dans l'ordre des charges décroissantes. Précisément, le premier sous-arbre de la structure arborescente est le site P_2, le second est le site P_1 et le dernier correspond au réseau. De plus, les tâches sont pré-triées selon la politique d'ordonnancement DM.

La figure 2.5 détaille l'exécution de la méthode de recherche des priorités. Sept coupes ont été effectuées durant la procédure de recherche ; elles sont représentées par les nœuds rayés sur la figure 2.5. Notons que toutes les sous-branches situées à gauche de la solution contiennent un nœud rayé. Précisément, les solutions contenues dans cette partie de l'arborescence ont été analysé sans succès. Les sous-branches se situant à droite de la solution ne contiennent pas de nœuds rayés, à l'exception d'un fils de τ_4 (lorsque l'on se trouve sur un nœud, tous les fils

Nom	Processeur	C_i	D_i	T_i
τ_0	P_1	52	160	100
τ_1	P_1	52	180	160
τ_2	P_2	10	60	40
τ_3	P_2	20	80	60
τ_4	P_2	20	124	160
τ_5	P_2	20	188	100
m_0	CAN	1	100	100
m_1	CAN	1	160	160

TAB. 2.1 – Caractéristiques des tâches

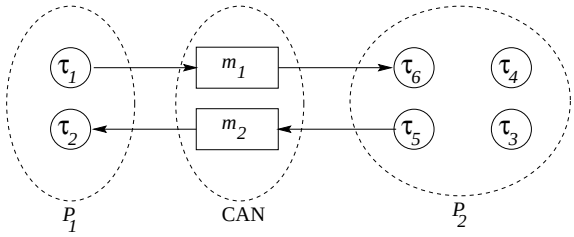

FIG. 2.4 – Graphe de précédence des tâches et des messages

de ce nœud sont testés et éventuellement insérés). Elles représentent l'ensemble les solutions potentielles non parcourues durant la procédure de recherche.

Notre méthode nécessite 0.1 seconde de calcul sur un ordinateur type PC pour valider cette application. Les temps de réponse obtenus, ainsi que l'ensemble des priorités affectées correspondantes sont présentés dans la table 2.2. L'ensemble des priorités permettant de valider l'application par une analyse holistique ne suit pas l'ordre DM, ce qui montre l'intérêt de la méthode.

2.7.2 Cas industriel

Afin d'illustrer le type d'application à valider, nous décrivons figure 2.6 l'architecture physique d'un système 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 [CAN94], dédié aux fonctions critiques de l'automobile, telles que l'ABS et le contrôle moteur ; le réseau VAN [VAN94] 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.

L'architecture logicielle de cette application est composée de 44 tâches s'exécutant sur les différents ECU et 19 messages circulant sur les deux réseaux. La figure 2.7 présente la structure des communications entre les différents ECU.

Tâche	Site	Priorité	Temps de réponse
τ_0	P_1	0	52.0
τ_1	P_1	1	178.0
τ_2	P_2	1	30.0
τ_3	P_2	2	60.0
τ_4	P_2	0	20.0
τ_5	P_2	3	164.0
m_0	*Res*	0	54.0
m_1	*Res*	1	22.0

TAB. 2.2 – Temps de réponse et ensemble de priorité

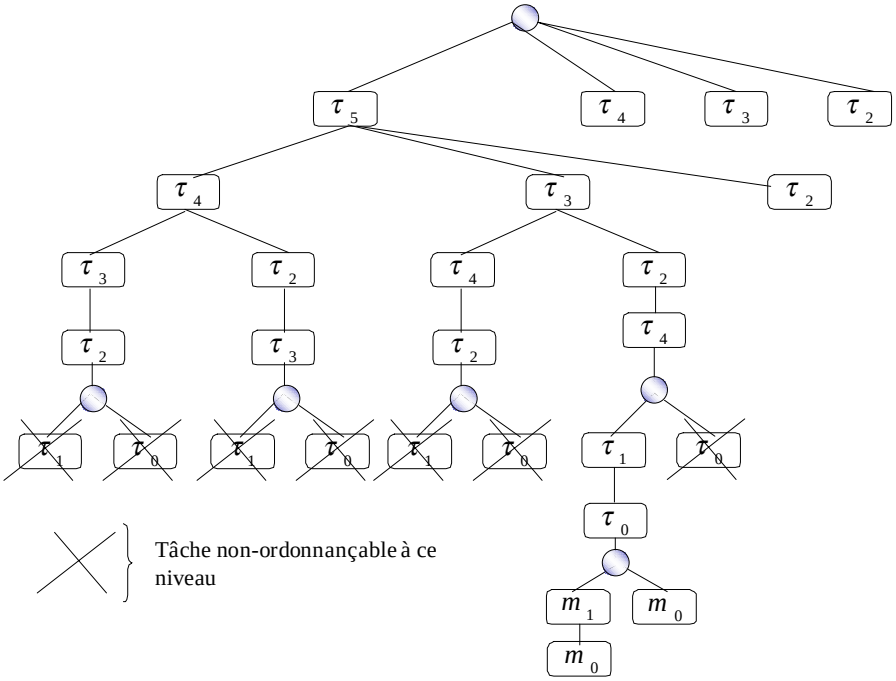

FIG. 2.5 – Arbre de l'exécution de la méthode de recherche des priorités

Nous générons aléatoirement les paramètres temporels des tâches afin que la charge sur chaque site – hors réseau – soit comprise dans un intervalle de charge prédéfinie. Pour que les résultats soient représentatifs du fonctionnement de notre méthode d'affectation, nous générons 30 configurations différentes pour chaque intervalles de charge. Ces derniers sont tous de longueur 0,1 compris entre 0,1 et 0,9 (i.e. $0,1 \leq U \leq 0,2$; $0,2 \leq U \leq 0,3$; ...; $0,8 \leq U \leq 0,9$).

La figure 2.8 présente le temps de calcul moyen nécessaire pour valider ou non une configuration en fonction de la charge des sites la composant. Le temps moyen de résolution des 30 configurations est calculé pour chaque intervalle de charge. La méthode est arrêtée après une heure de calcul ; la configuration correspondante est considérée non-résolue.

Comme le montre la figure 2.8, les pires temps de résolution sont obtenus pour des charges intermédiaires (i.e. comprises entre 0,4 et 0,6). Les temps de résolution sont au contraire très courts pour des configurations de charges faibles ou importantes. Pour un coefficient de charge U tel que $0,1 \leq U \leq 0,3$, la méthode produit rapidement une réponse. Dans le cas $0.7 \leq U \leq 0.9$, le principe d'évaluation permet de couper rapidement les branches infaisables de l'arbre de recherche.

La figure 2.9 présente le nombre de configurations validées, non validées, et non résolues en une heure. Dans ce dernier cas le nombre maximum est obtenu pour une charge comprise entre 0.5 et 0.6 et représente 10% des configurations testées.

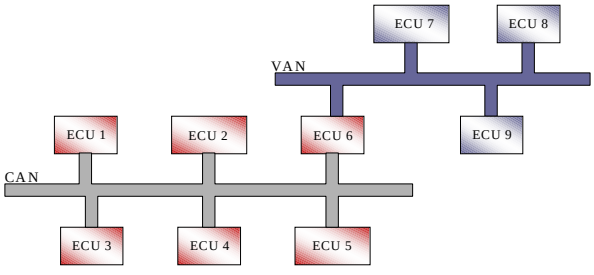

FIG. 2.6 – Structure d'un système embarqué dans l'automobile

La figure 2.10 présente le nombre moyen de coupes nécessaires (sans considérer les configurations non-résolues en une heure). Pour une charge moyenne, le nombre de coupes nécessaires à l'obtention d'un résultat est important. Notons que le nombre moyen de coupes est plus important pour trouver une solution que pour prouver l'absence de solution. Ceci montre l'efficacité des évaluations effectuées sur chaque sommets de l'arbre de recherche.

Les tests présentés dans cette section montrent les performances de notre méthode pour différentes configurations de charge produites aléatoirement et mappées sur une structure d'application physique fixe. La section suivante présente des tests ne reposant pas sur une architecture d'application prédéfinie.

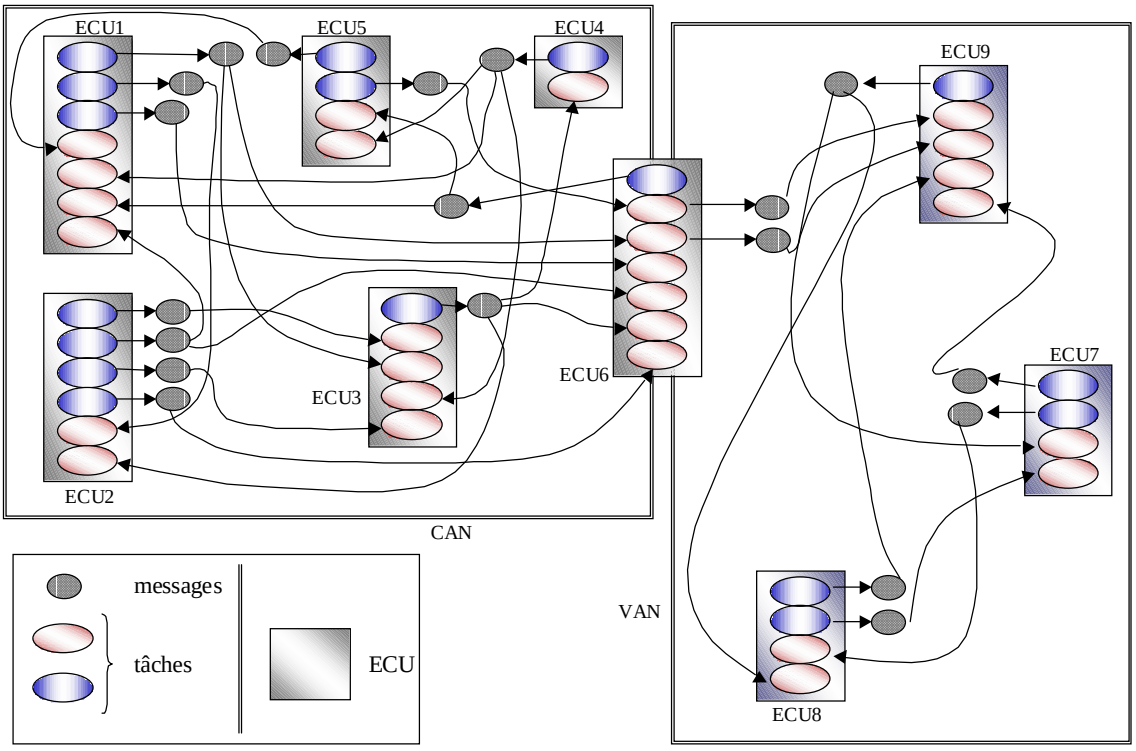

FIG. 2.7 – Structure des communications entre les différents ECU

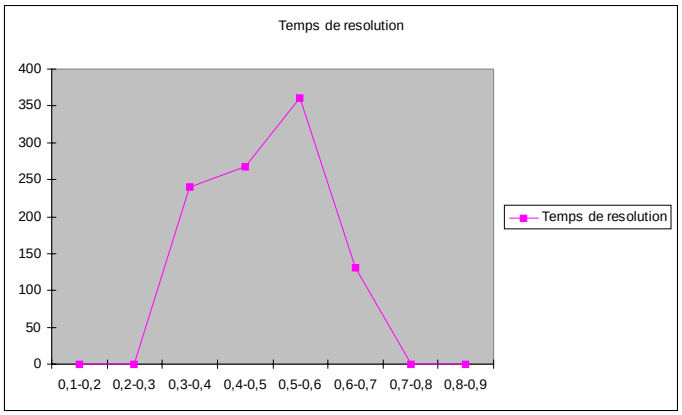

FIG. 2.8 – Temps de calcul

2.7.3 Tests approfondis

Génération aléatoire de configurations temps réel

Afin de tester de manière plus approfondie notre méthode, nous avons créé un générateur aléatoire d'applications temps réel. Le nombre de sites, le nombre de réseaux, un intervalle de charge (i.e. U_{min} et U_{max}), et le nombre de messages, c'est-à-dire le nombre de communications, sont des paramètres réglables du générateur.

Le nombre de tâches sur chaque site est complètement dépendant de la charge désirée. Dans ce but, les paramètres temporels des tâches sont générés aléatoirement, et sont à échéance sur requête. Chaque tirage aléatoire suit une loi normale à une dimension, ou gaussienne (Cf. figure 2.11).
Les paramètres choisis sont les suivants :

- Pire temps d'exécution des tâches : $(m = 1, \sigma = 8)$.
- Périodes des tâches :$(m = 12, \sigma = 70)$.

La génération des communications est réalisée par un tirage aléatoire de la tâche émettrice et de la tâche réceptrice, sous les hypothèses suivantes :

- La taille du message (i.e. nombre d'octet) est tirée aléatoirement dans un intervalle [1..8] afin que la charge du réseau respecte l'intervalle choisie. Dans le cas du réseau, la charge n'excédera pas 0.3.
- Les communications internes à un site sont interdites,
- Les cycles sont détectés, et une nouvelle communication est générée,
- Les paramètres temporels des tâches (i.e. période et échéance) sont ajustées en fonction de la tâche émettrice. Précisément, si τ_i communique avec τ_j par l'intermédiaire du message m_m, alors : $T_m = T_j = T_i$. Il en est de même pour les échéances puisque nous générons des tâches à échéance sur requête.

Toujours dans le but d'obtenir des résultats objectifs, nous produisons aléatoirement 30 configurations différentes pour chaque type de problème.

Performance de la méthode

Temps de calcul moyen : La figure 2.12 regroupe les temps de calcul moyen nécessaires à notre méthode pour fournir un résultat. Les temps de calcul obtenus correspondent à la moyenne des temps de calcul nécessaire à la résolution de l'ensemble des problèmes d'un intervalle de charge. La courbe obtenue est très proche de celle présentée dans le cas de l'application réaliste (cf. figure 2.8). Ceci montre une certaine indépendance de la méthode vis-à-vis de la structure physique de l'application. D'une manière générale, le problème de l'affectation des priorités d'un système distribué temps réel est facile dans les cas de charge très faible ou très élevée. À l'inverse, dans le cas d'une charge intermédiaire, le temps de calcul nécessaire pour obtenir un résultat est beaucoup plus long, ceci montrant l'intérêt de notre méthode.

Influence des communications : Précédemment, nous avons mis en exergue une relative indépendance des performances de la méthode vis-à-vis de la structure physique de l'application étudiée. Or, comme le montre la figure 2.13, les conséquences de l'augmentation du nombre de communications sont similaires quelque soit l'intervalle de charge considéré. Plus précisément, pour un nombre de sites fixes, et pour une charge donnée,

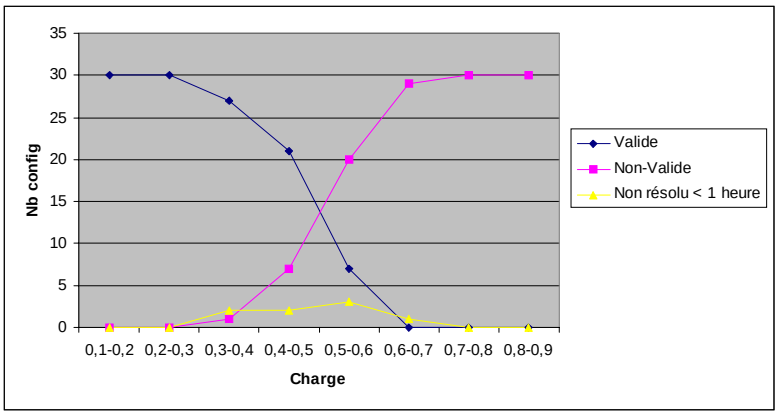

FIG. 2.9 – Configurations valides, non-valides ou non-résolues

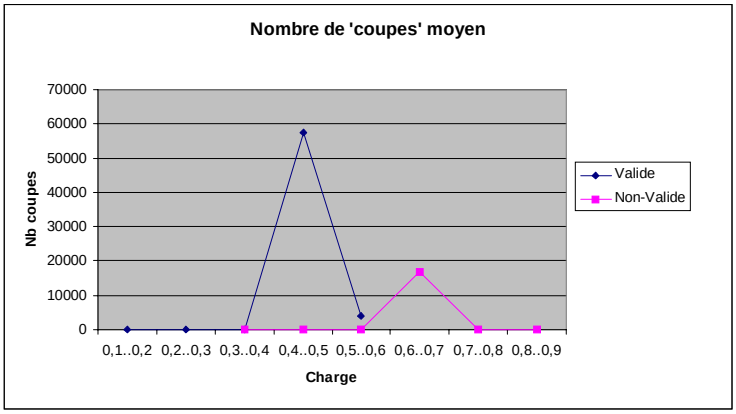

FIG. 2.10 – Nombre de coupes moyen nécessaire pour obtenir un résultat.

l'augmentation du nombre de communications fait croître le temps de calcul nécessaire à la résolution du problème. Ce phénomène peut expliquer cette indépendance.

Pour un nombre de sites fixés, la figure 2.14 montre l'évolution du temps de calcul en fonction du rapport entre le nombre de sites (i.e. tâches) et le nombre de communications (i.e. messages). Cette courbe montre que jusqu'à un facteur de charge de 0.6, l'augmentation du nombre de communications fait croître le temps de résolution du problème. Pour $U \geq 0.7$, augmenter le nombre de messages diminue le temps de calcul nécessaire à l'obtention d'un résultat. Ceci s'explique par le fait que l'augmentation du nombre de communications revient à contraindre de plus en plus le système. Ainsi, le nombre de solutions valides diminue, et notre méthode de recherche détecte les mauvais assignements de plus en plus haut dans l'arbre de recherche (i.e. de plus en plus rapidement), les coupes étant alors plus bénéfiques.

Nombre de configurations validées : La figure 2.15 présente l'influence de la taille de l'application et du nombre de communications sur le nombre de configurations validées.

Pour un coefficient d'utilisation inférieur à 0.6, le pourcentage de configuration validée par notre méthode est, dans la majorité des cas, supérieur à 50%, ce qui montre la qualité de la condition suffisante utilisée. De plus, pour chaque courbe, l'influence du nombre de messages sur ce pourcentage est visible et est de plus en plus important pour les charges élevées.

Nous finirons cette section sur les performances de la méthode en présentant la figure 2.16 montrant le nombre de configurations non résolues en une heure de calcul. Dans le pire cas, le pourcentage de telles configurations n'a pas dépassé 23%, faisant ressortir la qualité de la procédure de recherche.

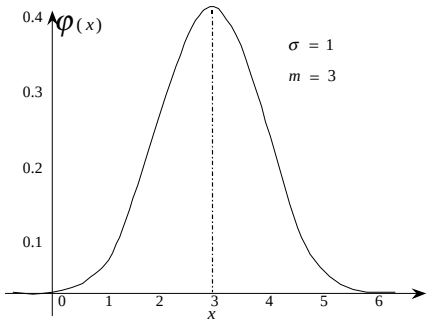

FIG. 2.11 – Loi normale à une dimension

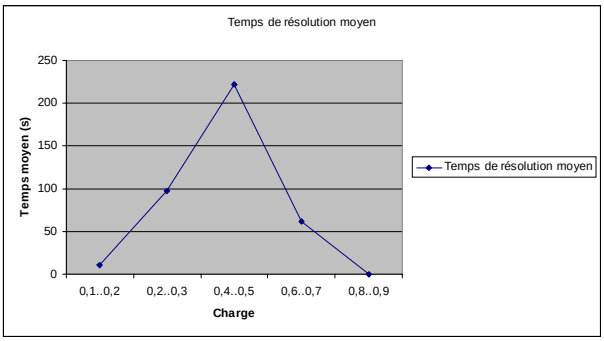

FIG. 2.12 – Temps de calcul moyen

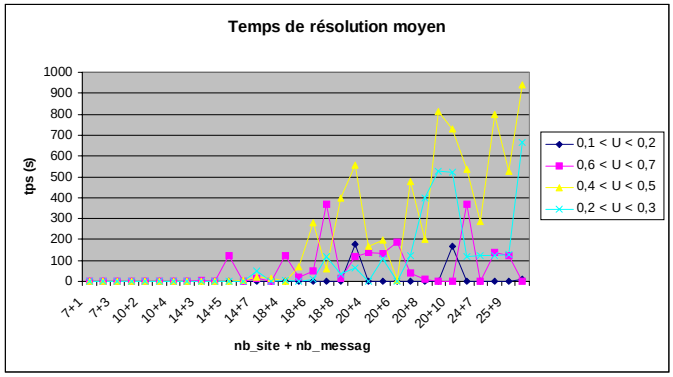

FIG. 2.13 – Temps de calcul moyen

Influence des heuristiques

Pré-ordre des tâches Les tests présentés ci-dessus sont réalisés en pré-classant les tâches sur chaque site selon la politique *Deadline Monotonic*. Nous avons réalisé des tests en utilisant la politique *Inverse Deadline Monotonic*. D'une manière générale, les résultats en terme de rapidité sont moins bons, mais dépend tout de même fortement des données. L'utilisation de la politique *IDM* revenant à parcourir l'arbre de la branche droite à la branche gauche, deux cas se présentent :

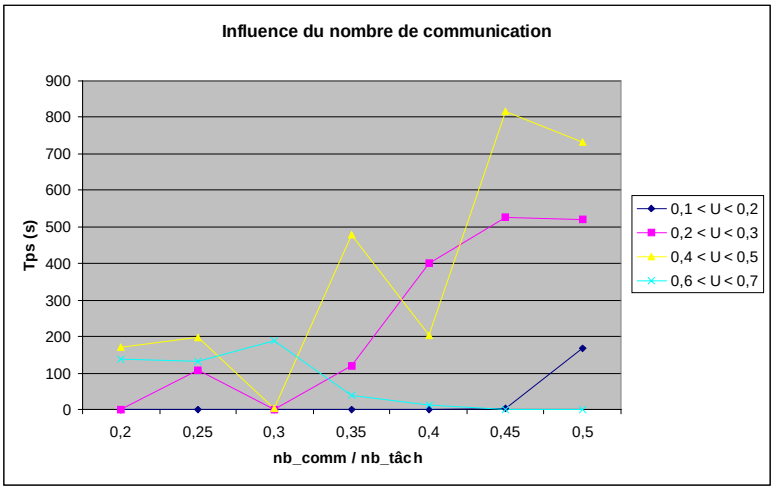

FIG. 2.14 – Influence du nombre de communications

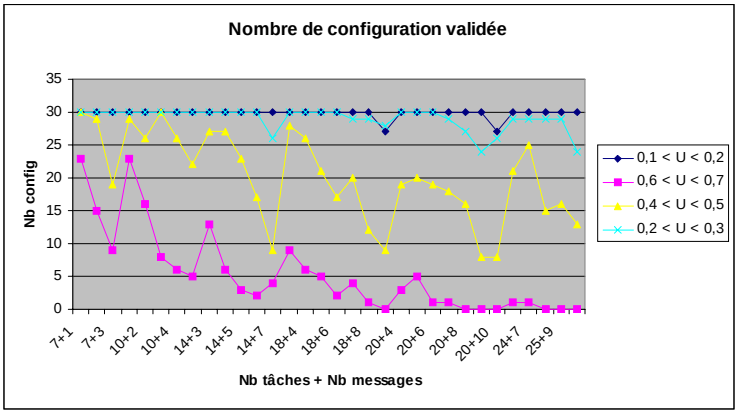

FIG. 2.15 – Influence de la charge sur le nombre de configurations validées

- Pour une charge faible ou forte, les résultats sont très proches.
- Pour une charge intermédiaire, l'utilisation de la politique *IDM* pour pré-classer les tâches sur chaque site augmente le temps de calcul dans la majorité des cas.

Des tests comparatifs sur un pré-classement des tâches sur chaque site selon l'ordre topologique global sont en cours.

Classement des sous-arbres dans la structure arborescente Les résultats ci-dessus sont obtenus dans le cas ou les sous-arbres (i.e. les sites et réseaux) sont classés selon la charge décroissante. En triant les sous-arbres selon la charge croissante, les résultats obtenus sur les mêmes jeux de tests sont moins bons. Dans le premier cas (i.e. charge croissante) les coupes réalisées sont très bénéfiques puisqu'elles interviennent beaucoup plus haut que dans le second cas.

Dans tous les tests effectués, l'ordre des sites dans la structure arborescente est fixé dès le début de la procédure de recherche et reste statique tout au long de celle-ci. Nous pensons améliorer les performances de la méthode en disposant d'une structure dynamique. Précisément, à chaque nœud fictif, le prochain site à traiter sera choisi parmi les sites restant. Le site choisi est celui qui est le plus difficile à ordonnancer, en quelque sorte le site *goulet*. Ce critère est évalué en fonction des estimations des temps de réponse dépendant de l'affectation en cours de traitement. Cette heuristique est en cours d'implémentation.

En conclusion, les points forts de notre méthode, mis en évidence par les tests ci-dessus, sont les suivants :

- la méthode supporte des applications industrielles (grand nombre de sites, tâches, messages),

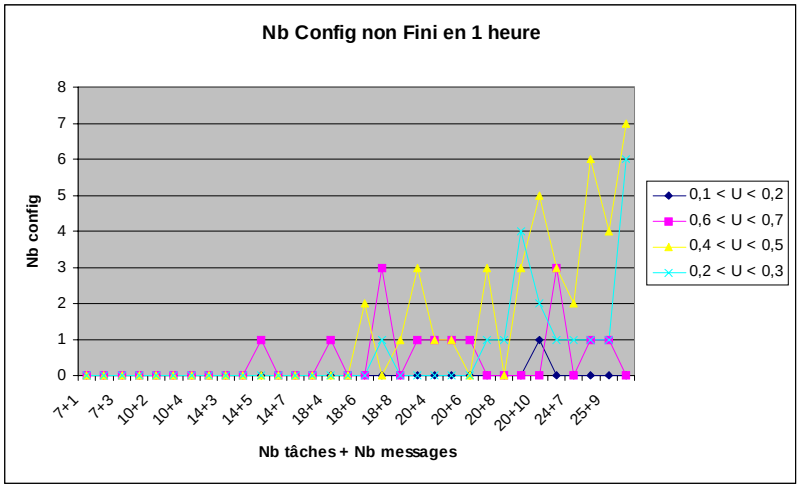

FIG. 2.16 – Configuration non-résolue en une heure de calcul

- la méthode supporte plusieurs protocoles réseau différents,
- le temps d'obtention d'un résultat est très satisfaisant au vu de la taille des applications testées (qualité du principe d'évaluation et bonne propagation des contraintes),
- la méthode est capable de validée des applications dans des cas de charge importante – $U = 0,7$ – (qualité de la borne supérieure, c'est-à-dire de l'évaluation du pire scénario),
- la méthode est très évolutive, c'est-à-dire que la prise en compte de nouvelles contraintes – ressources, communications internes à un site, type de communications différents ... – est facile et ne nécessite pas de profondes modifications (qualité de la structure arborescente et du principe d'évaluation).

Chapitre 3

Méthode d'affectation et de placement des tâches et des messages

Nous proposons dans ce chapitre une évolution de la méthode précédente permettant de placer les tâches sur les différents sites et les messages sur les différents réseaux de l'application. L'affectation des priorités est alors réalisée en fonction du placement. Cette évolution nécessite une modification de la structure de recherche arborescente. Cette dernière est présentée dans la section suivante.

3.1 Structure de l'arbre de recherche

Cette nouvelle structure arborescente est basée sur les travaux présentés dans [BFR75] et [Vig97]. Après avoir présenté les principes de ces deux méthodes, nous détaillons la structure et le fonctionnement de l'arbre de recherche utilisé dans notre procédure de placement-affectation.

3.1.1 Structure arborescente [BFR75] [Vig97]

Dans [BFR75], les auteurs présentent une façon d'énumérer, sans répétition tous les arrangements possibles de n tâches sur m machines.

Type de nœud
Cette énumération est réalisée en générant un arbre de recherche contenant deux types de nœuds : les *nœuds ronds* et *les nœuds carrés.*
• *Nœuds ronds :* si le chemin passe par un nœud rond, cela signifie que la tâche est ordonnancée sur la machine courante.
• *Nœuds carrés :* en revanche, si le chemin passe par un nœud carré, alors la tâche correspondante est ordonnancée sur un nouveau site, ce dernier devenant alors le site courant.

Règles de construction
La structure arborescente présentée ci-dessus est construite en respectant l'ensemble de règles suivantes :

1. Le niveau 0 de la structure arborescente contient seulement un nœud fictif.
2. le niveau 1 est formé de n nœuds carrés (i.e. énumération de toutes les tâches au niveau de priorité le plus bas sur la première machine courante).
3. Un chemin partant du niveau 0 et se terminant au niveau i, $1 \leq i < n$, peut être agrandit au niveau $i+1$ par n'importe quel nœud rond parmi n ou n'importe quel nœud carré parmi n, si les trois règles suivantes sont respectées.
4. ni un nœud rond k, ni un nœud carré k ne peut être utilisé pour agrandir un chemin si ce dernier contient déjà un nœud rond k ou un nœud carré k.
5. Un nœud carré k ne peut pas être utiliser pour agrandir un chemin contenant déjà un nœud carré l tel que $l > k$.
6. Aucun chemin ne peut être étendu par un nœud carré quelconque si ce chemin contient déjà m nœuds carrés.

La règle 4 assure qu'une tâche ne sera placé qu'une seule fois, la règle 5 permet d'éviter les redondances (i.e. chemin différents, mais affectation et placement identique), et enfin la règle 6 assure le respect du nombre de site. La figure 3.1 montre la structure arborescente obtenue par les règles ci-dessus pour un problème de placement

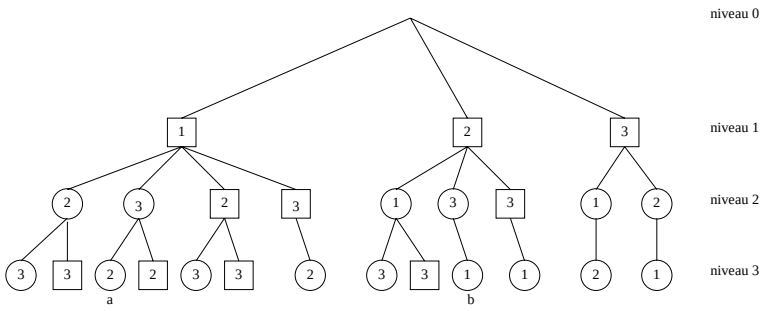

FIG. 3.1 – Énumération et placement de 3 tâches sur 3 sites

de 3 tâches sur 3 sites. Le chemin (a) correspond au placement dans lequel la tâche 1 est la moins prioritaire sur le site 1, la tâche 3 est la plus prioritaire sur cette même première machine, et enfin la tâche 2 et la seule tâche sur le site 2 et est forcément la plus prioritaire. Le site 3 n'est pas utilisé dans ce placement.
Le chemin (b) correspond au placement des trois tâches sur le site 1 uniquement, dans l'ordre inverse des priorités.

Dans [Vig97], l'auteur propose le même type de structure arborescente (i.e. deux types de nœuds). Cependant, il impose qu'il y ait au moins une tâche sur chaque site. La règle suivante est ajoutée aux six précédentes afin de prendre en compte cette hypothèse.

7. Aucun chemin ne se termine s'il contient moins de m nœuds carrés, sauf si $n < m$.

L'application de cette règle sur la figure 3.1 élimine les branches 1, 3, 8, 9, 10 et 11.

3.1.2 Structure et fonctionnement de l'arbre de recherche

Structure de l'arbre de recherche
Le type d'application temps réel que nous nous proposons d'étudier est composé de S sites et de Re réseaux. Parmi les S sites, il peut y avoir S_1 sites identiques, S_2 sites identiques mais différents des S_1 précédents et S_3 sites dédiés (i.e. parmi les n tâches de l'application, il existe n_3 tâches devant être placées sur les sites S_3). De plus, conformément à l'analyse holistique, les réseaux sont vus comme des processeurs (i.e. sites). Ainsi, il constitue un type de site différent puisqu'il sont considéré comme dédié. Précisément, si $Re > 1$, il faut alors considérés Re types de sites différents.

La structure de l'arbre de recherche présenté dans [Vig97] doit être modifiée puisque dans notre cas nous ne disposons pas de S sites identiques. Pour résoudre ce problème, l'ensemble des sites est partagé en sous-ensemble de sites identiques. Au minimum une application distribuée est donc constituée d'un ensemble de S sites identiques et d'un site S_r modélisant le réseau.

Le placement des tâches est réalisé sur chaque sous-ensemble de sites identiques. La structure arborescente utilisée pour un sous-ensemble est celle présentée dans [Vig97]. La figure 3.2 présente la structure générale (i.e. pour l'application entière) de l'arbre de recherche de notre méthode. L'arbre de recherche est composé de plusieurs sous-arbres. Ces derniers correspondent aux différents sous-ensembles de sites identiques. À chaque sous-arbre est associé un ensemble de tâches. Le sous-arbre considéré représente l'ensemble des placements et des affectations de priorités possibles pour les q tâches associées à ces sites. Notons enfin qu'un réseau constitue un sous-arbre à part entière, sur lequel est réalisé l'affectation des priorités aux différents messages lui étant associés (i.e. le seul niveau de ce sous arbre contenant des *nœuds carrés* est le premier). Les sous-arbres sont reliés entre eux par des nœuds fictifs.

Parcours de la structure
Une branche de l'arbre représente un placement de toutes les tâches et de tous les messages de l'application ainsi qu'une affectation des priorités pour ces tâches et messages. Le parcours de l'arbre de recherche est le même que celui de la méthode sans placement, c'est-à-dire un parcours en *profondeur d'abord*.

Optimisation de la procédure de placement
Au début de la procédure de placement-affectation, le graphe de précédence est connu. Or si deux tâches τ_i et τ_j communiquent par l'intermédiaire d'un message m_i, deux politiques de placement peuvent être tenues.
• τ_i et τ_j sont placées sur des sites différents et le message m_i sur le réseau utilisé par l'application, auquel cas le pire temps de réponse de τ_j dépend de celui de m_i, qui lui-même dépend de celui de τ_i.
• τ_i et τ_j sont placées sur le même site. Dans ce cas, le message m_i est inutile. En effet, la communication entre deux tâches sur un même site est considérée comme de durée nulle. La seule contrainte est donc que la tâche émettrice est finie son exécution avant le début de l'exécution de la tâche réceptrice (i.e. contrainte de précédence). Cette solution économise l'ordonnancement du message m_i sur le réseau, et ce dernier n'occasionne plus de retard pour la tâche τ_j. La gigue sur l'activation des tâches permet de modéliser ces contraintes de précédence. De plus le fonctionnement de l'analyse holistique n'est pas perturbé dans ce cas ; le message m_i est supprimé de l'application et la relation de précédence $\tau_i \rightarrow m_i \rightarrow \tau_j$ est transformée en $\tau_i \rightarrow \tau_j$.

Dans un souci d'optimisation des ressources du système, la dernière politique de placement présentée ci-dessus est plus performante que la première. Mais, celle-ci peut également être difficilement réalisable dans le cas de communication entre deux tâches dédiées (i.e. capteur, commande). De plus la surcharge d'un site particulier peut entraîner l'augmentation des pires temps de réponse d'autres tâches du même site et éventuellement d'autres sites.

Optimisation de la procédure d'affectation
Les optimisations de la procédure d'affectation des priorités de cette méthode sont les mêmes que celle de la méthode d'affectation simple. Précisément, l'optimisation sur les tâches est conservée. En ce qui concerne les sites, l'idée est la même. En effet chaque ensemble de site se voit affecté un coefficient de charge général notée U_t. Le classement des ensembles de sites est alors réalisé de manière croissante ou décroissante selon U_t.

La section suivante présente les différents tests et coupes effectués lors de la procédure de placement-affectation.

3.2 Tests et coupes

L'idée générale de cette méthode est de réaliser conjointement le placement et l'affectation de priorités des tâches et messages. Les principes des méthodes d'évaluation de la méthode précédente sont repris et modifiés afin de prendre en compte les différents cas de placement.

Le détail de ces méthodes d'évaluation feront l'objet d'un autre rapport.

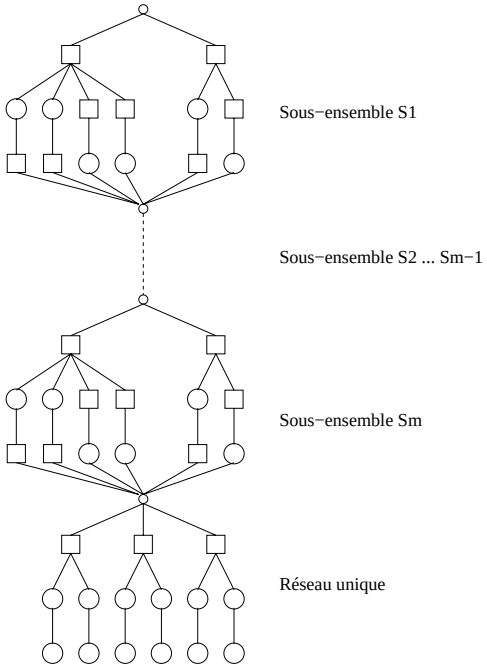

FIG. 3.2 – Structure générale de l'arbre de recherche

Chapitre 4

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 mono-processeur, 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.

Les tests menés montrent de très bonnes performances en temps de calcul. De plus le nombre de configurations validées font ressortir le grand interêt de cette méthode comme outils d'aide à la conception *et* à la validation de systèmes temps réel distribués. Enfin, en gérant le partage de ressources entre tâches, nous avons mis en exergue les qualités d'évolutivité de la structure arborescente ainsi que du principe d'évaluation des bornes inférieures des pires temps de réponse.

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).

Bibliographie

[Aud91] N.C. Audsley. Optimal priority and feasability of static priority tasks with arbitrary start times. *Real-Time research group, Dept of Computer Science, University of York*, November 1991. Rapport Interne N° Y01 5DD.

[BFR75] P. Bratley, M. Florian, and P. Robillard. Scheduling with earliest start and due date constraints on multiple machines. *Naval Research Logistic Quaterly*, 22(1) :165–173, 1975. University of Montreal, Canada.

[But97] G. C. Buttazzo. *Hard Real-Time Computing Systems : Predicatable Scheduling Algorithms and Applications*. Kluwer Academic Publisher, 1997.

[CAN94] Road vehicles – low-speed serial data communication – part 2 : Low-speed controller area network (can). ISO 11519-2, 1994.

[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 3^{rd} IEEE International Workshop on Factory Communication Systems*, Porto, september 2000.

[Da01] L. C. Dipippo and all. Scheduling and priority mapping for static real-time middleware. *Real-Time System Journal*, 20 :155–182, 2001.

[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 3^{d} 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, Septembre 1996.

[Leh90] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Procedings of the 11^{th} IEEE Real-Time Systems Symposium*, December 90.

[LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1) :46–61, 1973.

[OF00] R. Silva De Oliviera and J. Fraga. Fixed priority scheduling of tasks with arbitrary precedence constraints in distributed hard real-time systems. *Journal of System Architecture*, 46 :991–1004, 2000.

[OSE97] Osek/vdx operating system vers2.0r1. http ://www-iiit.etec.univ-karlsruhe.de/ osek/, 1997.

[RRC00a] M. Richard, P. Richard, and F. Cottet. Analyse holistique des systèmes temps réels distribués. Technical Report 2000-006, LISI – ENSMA, www.lisi.ensma.fr, aout 2000.

[RRC00b] 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.

[RSL88] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of thr IEEE Real-Time Systems Symposium*, pages 259–269, December 1988.

[SRL87] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols. Technical Report CMU-CS-87-181, Departments of CS, ECE and Statistics – Carnegie Mellon University, December 1987.

[SRL90] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols : An approach to real-time synchronization. *IEEE Trans. Computers*, 39 :175–185, Sept 1990.

[TBW92] K. W. Tindell, A. Burns, and A.J. Wellings. Allocating real-time tasks, an np-hard problem made easy. *Real-Time System Journal*, 4(2), 1992.

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

[Tin94] K. W. Tindell. *Fixed Priority Scheduling of Hard Real-Time Systems.* PhD thesis, Univeristy of York, 1994.

[VAN94] Vehicle area network, serial data communication – road vehicles, serial data communication for automotive application. ISO 11519-3, 1994.

[Vig97] A. Vignier. *Contribution à la résolution des problèmes d'ordonnancement de type monogamme, multimachine ("Flow-shop hybride").* PhD thesis, Ecole d'Ingénieurs en Informatique pour l'Industrie, 1997.

Chapitre 5

Annexes

5.1 Protocole à priorité plafond

Le protocole à priorité plafond – PCP Priority Ceiling Protocol –, (issu du protocole à héritage de priorité) a été présenté par [SRL90, SRL87]. Il permet de prévenir les interblocages, les chaînes de blocages et les inversions de priorités non bornées. [SRL90] ont montré que cette borne correspondait, au plus, à la durée d'une section critique.

hypothèses L'utilisation du protocole à priorité plafond est soumis aux hypothèses suivantes :
- chaque ressource du système est protégée par un sémaphore binaire,
- une tâche ne peut demander l'accès à une ressource qu'elle possède déjà afin d'éviter l'auto interblocage,
- toutes les instances d'une tâche doivent libérer les ressources acquises au cours de leur exécution avant la fin de celle-ci.
- une tâche peut posséder plusieurs sections critiques si et seulement si elles ne s'entrelacent pas. Ainsi, le schéma de prise de ressources présenté en (5.1) est accepté, tandis que le schéma (5.2) est interdit.

$$P(S_1) \ldots\ldots P(S_2) \ldots\ldots V(S_2) \ldots\ldots V(S1) \tag{5.1}$$

$$P(S_1) \ldots\ldots P(S_2) \ldots\ldots V(S_1) \ldots\ldots V(S2) \tag{5.2}$$

Après avoir rappelé le fonctionnement et les propriétés nous intéressant, nous présentons la méthode de calcul du pire temps de blocage induit par le partage de ressources critiques.

5.1.1 Principe du protocole à priorité plafond

Lorsqu'une tâche τ_i préempte une section critique d'une autre tâche τ_j et exécute sa propre section critique z, la priorité à laquelle s'exécute τ_i durant sa section critique z doit être strictement supérieure aux priorités de toutes les autres sections critiques. Si cette condition ne peut-être respectée, l'accès de τ_i à la ressource est refusée. De plus, la tâche bloquant τ_i (i.e. τ_j) hérite de sa priorité.

La définition précise du protocole à priorité plafond est donnée ci-dessous [But97] :

1. Chaque sémaphore S_k se voit assigner une priorité plafond, notée $C(S_k)$. Cette priorité correspond à la plus forte priorité des tâches pouvant demander l'accès au sémaphore S_k. Ainsi :

$$C(S_k) = \max_j \{C(\tau_j) | \tau_j \rightarrow S_k\} \tag{5.3}$$

   avec $\tau_j \rightarrow S_k$ signifiant que τ_j demandera au moins une fois l'accès au sémaphore S_k durant son exécution.
2. Soit τ_i la tâche de plus forte priorité parmi les tâches prêtes : τ_i se voit assigner le processeur
3. S^* représente le sémaphore possédant la plus forte priorité plafond parmi tous les sémaphores bloqués par toutes les autres tâches à un instant t lors de l'exécution de l'application. Sa priorité plafond est notée $C(S^*)$.

4. Pour entrer dans une section critique gardée par un sémaphore S_k, une tâche τ_i doit posséder une priorité strictement supérieure à $C(S^*)$. Si $P_i \leq C(S^*)$, τ_i est bloquée.
5. Lorsqu'une tâche τ_i est bloquée sur un sémaphore, elle transmet sa priorité à la tâche τ_k détenant ce sémaphore. Ainsi, τ_k termine l'exécution de sa section critique à la priorité héritée P_i. On dit que τ_k hérite de la priorité de τ_i.
6. Quand τ_k sort de sa section critique, elle relâche le sémaphore protégeant cette dernière. La tâche de plus forte priorité bloquée par ce sémaphore, si il y en a, est réveillée et passe dans l'état prêt. De plus, la priorité de la tâche τ_k est mise à jour comme suit :
   - Si d'autre tâches sont bloquées par une section critique de τ_k[1], elle est alors ordonnancée avec la plus forte priorité des tâches qu'elle bloque à cet instant.
   - Sinon, elle est ordonnancée avec sa priorité initiale, c'est-à-dire la priorité qu'elle s'est vue affectée au début de l'application.
7. L'héritage de priorité est transitif. Précisément, si τ_3 bloque τ_2 et τ_2 bloque également τ_1, alors τ_3 hérite de la priorité de τ_1 via τ_2 et est ordonnancée à ce niveau de priorité.

Exemple de calcul des priorités plafonds associées aux sémaphores [But97] : Soit trois tâches τ_0, τ_1 et τ_3 de priorité décroissante. La tâche τ_0 de plus forte priorité contient deux sections critiques successives protégées par les sémaphores S_0 et S_1. La tâche τ_2 contient une section critique protégée par le sémaphore S_2 et la tâche τ_3 accède à deux sections critiques parfaitement imbriquées protégées par les sémaphores S_2 et S_1. Les priorités plafonds obtenues pour chaque sémaphores sont présentées dans la table 5.1.

Tâche	Sections critiques	Sémaphores	Priorité Plafond
τ_0	... S_{0_d} .. S_{0_f} ... S_{1_d} .. S_{1_f} ...	S_0	P_0
τ_1	... S_{2_d} .. S_{2_f} ...	S_1	P_0
τ_0	... S_{2_d} .. S_{1_f} ... S_{1_d} .. S_{2_f} ...	S_2	P_1

TAB. 5.1 – Exemple

5.1.2 Propriété du protocole à priorité plafond

Les deux théorèmes ci-dessous, présentés dans [SRL90, SRL87] permettent de borner le pire temps de blocage d'une tâche τ_i dû au partage de ressources critiques. Le théorème 4 permet de garantir qu'il n'existera pas d'attente infinie due à l'un des trois phénomènes induit par le partage de ressources critiques. Le théorème 5 utilise le principe de fonctionnement du protocole à priorité plafond pour montrer qu'en utilisant ce dernier, le pire temps de blocage d'une tâche induit par l'exécution d'une tâche en section critique est borner à la durée d'une unique section critique.

Théorème 4 : *Le protocole à priorité plafond prévient les interblocages, les chaînes de blocage et l'inversion de priorité non-bornée.*

Théorème 5 : *En utilisant le protocole à priorité plafond, pour une configuration de n tâches périodiques dans un contexte mono-processeur, une tâche τ_i peut être bloquée pour au plus la durée d'une section critique $z_{i,k}$ pouvant bloquer τ_i.*

5.1.3 Calcul du pire temps de blocage B_i

Afin de calculer le pire temps de blocage B_i d'une tâche τ_i, nous considérons Z, l'ensemble des sections critiques pouvant bloquer τ_i. Précisément, chaque éléments de Z est une section critique accéder par une tâche moins prioritaire et gardée par un sémaphore dont la priorité plafond est supérieure ou égale à la priorité de τ_i. Ainsi, par le théorème 5, la tâche τ_i peut être bloquée pour au plus la durée d'un unique élément de Z. Le pire temps de blocage B_i correspond donc à la durée du plus grand élément de Z.

[1] Cas des sections critiques parfaitement imbriquées.

Avec le protocole à priorité plafond, une section critique $z_{i,k}$ peut bloquer une tâche τ_i si et seulement si $P_j < P_i$ et $P(S_k) \geq P_i$. En notant $D_{j,k}$ la plus longue section critique de τ_j parmi celles gardées par S_k[2], la pire durée de blocage B_i pour la tâche τ_i est donnée par l'équation 5.4.

$$B_i = \max_{j,k} \{D_{j,k} | P_j > P_i, P(S_k) \leq P_i\} \tag{5.4}$$

[2]Une tâche peut demander plusieurs fois l'accès à une ressource critique

5.2 Algorithmes de la procédure d'affectation des priorités

```
ALGORITHME Affectation_Priorité
Entrée: L: liste noeud
Var: node : noeud; test : booléen
     tab_s, tab_t : tableau d'entier
Début
  tab_s = classe_site;  /*classement des sites selon la charge*/
  tab_t = classe_tache(Tab_s[0]); /*classement des tâches selon inverse DM*/
  Pour i = 1..nb_tache(tab_s[0]) faire
    f_i=creer_fils(tab_t[i]);
    insert_fils(L,f_i);
  Fin Pour
  Tant que Liste_vide(L)≠ Vrai Faire
    node=extraire_noeud_tete(L);
    creer_fils(node);
    Pour tous les fils f_i créés
      test = analyse_fils(f_i);
      Si (test = Vrai)    /* si l'affectation partielle
                             du noeud est ordonnançable*/
         insert_fils(L,f_i); /*la branche sera continuée*/
      Fin Si
    Fin Pour
  Fin Tant que
Fin
```

```
ALGORITHME Analyse_fils
Entrée: fils :noeud
        matrice mat:1..n,1..n; /*contrainte de précédence*/
Sortie: Booléen;
Var: LB(Tr_i) : Réel
Début
  Faire
    Pour i = 1..n Faire
      Fini = Vrai;
      LB(Tr_i^(k-1)) = LB(Tr_i^(k));
      LB(Tr_i^(k)) = Evaluate(i);
      Si LB(Tr_i^(k-1)) ≠ LB(Tr_i^(k))
        Fini=Faux;
      Fin Si
    Fin Pour
    Pour i = 1..n Faire
      Pour j = 1..n Faire
        Si mat[i,j]=1  /*j est un successeur de i*/
          J_j = max(J_j, LB(Tr_i^(k)));
        Fin Si
      Fin Pour
    Fin Pour
  Tant Que NON Fini  /*point fixe atteint*/
  Si (LB(Tr_i^(k)) > D_i)
    retourne Faux;
  Sinon
    retourne Vrai;
  Fin Si
Fin
```

```
ALGORITHME Evaluate
Entrée: C_i, T_i, LB(J_i);
Sortie: LB(Tr_i^(k));
Var: r,t,t', C_max;
Début
  LB(Tr_i^(k)) = 0.0; q = 0;
  Si (τ_i) possède une priorité
```

```
    Si ordonnancement préemptif
      Faire
        Faire
          t = t' ;
          t = (q + 1) * C_i ;
          Pour j = 1..i − 1 faire
            t = t + ⌈ (LB(J_j) + t') / T_j ⌉ * C_j ;
          Fin Pour
        Jusqu'à t = t'
        r = t + LB(J_i) − qT_i ;
        LB(Tr_i^(k)) = max(LB(Tr_i^(k)); r);
        q = q + 1;
      Jusqu'à (t > (q + 1)T_i)
      return LB(Tr_i^(k));
    Sinon Si ordonnancement non-préemptif
      Faire
        Faire
          t = t' ;
          t = (q + 1) * C_i ;
          Pour j = 1..i − 1 faire
            t = t + ⌈ (LB(J_j) + t') / T_j ⌉ * C_j ;
          Fin Pour
        Jusqu'à t = t'
        Pour j = i + 1..n
          C_max = max(C_max; C_j);
        Fin Pour
        r = t + LB(J_i) − qT_i + C_max ;
        LB(Tr_i^(k)) = max(LB(Tr_i^(k)); r);
        q = q + 1;
      Jusqu'à (t > (q + 1)T_i)
      return LB(Tr_i^(k));
    Fin Si
  Sinon Si (τ_i) ne possède pas de priorité
    Si ordonnancement préemptif
      LB(Tr_i^(k)) = C_i + LB(J_j);
      return LB(Tr_i^(k));
    Sinon Si ordonnancement non-préemptif
      Pour j = i + 1..n
        C_max = max(C_max; C_j);
      Fin Pour
      LB(Tr_i^(k)) = C_i + LB(J_j) + C_max ;
      return LB(Tr_i^(k));
    Fin Si
  Fin Si
Fin
```