Analyse du temps de réponse des systèmes temps réel

Pascal Richard

Laboratoire d'Informatique Scientifique et Industrielle, ENSMA
BP 40198 Téléport 2
F-86960 Futuroscope
pascal.richard@ensma.fr

RÉSUMÉ. *La validation des systèmes temps réel impose de vérifier que les tâches respectent leurs contraintes temporelles. Des techniques analytiques spécifiques aux systèmes temps réel ont été conçues pour répondre à cet objectif. Les trois principales approches sont fondées sur l'analyse du facteur d'utilisation du processeur, l'analyse de la demande processeur et l'analyse du temps de réponse. Cet article présente les concepts fondamentaux de cette dernière technique. Nous présenterons tout d'abord l'analyse des systèmes monoprocesseurs et des réseaux temps réel à travers l'exemple du réseau CAN. Nous présenterons ensuite l'analyse des systèmes distribués. Nous terminerons par les logiciels d'analyse du temps de réponse des systèmes temps réel.*

MOTS-CLÉS : *Ordonnancement, Analyse du temps de réponse, Analyse holistique, Tests d'ordonnançabilité. Outils*

1. Les techniques d'analyse des systèmes temps réel

Le fonctionnement des logiciels temps réel repose souvent sur des tâches périodiques. Celles-ci interagissent *en temps réel* avec l'environnement contrôlé. Les contraintes temporelles se traduisent par des échéances sur les tâches. Une tâche τ_i est définie par sa pire durée d'exécution C_i, son échéance D_i et sa période entre deux *réveils* successsifs T_i. La charge processeur associée à une tâche est définie par $u_i = \frac{C_i}{T_i}$. Une exécution d'une tâche définit une *instance*. L'exécution d'une tâche peut être interrompue par l'ordonnanceur. Cette opération est la *préemption* de la tâche. Soit r_{ik} la date de réveil de la k^{ieme} instance de τ_i et f_{ik} sa date de fin d'exécution, alors le temps de réponse de l'instance est défini par $R_{ik} = f_{ik} - r_{ik}$. Le pire temps de réponse de τ_i est le plus grand temps de réponse de toutes ses instances : $R_i = \max_{k=1,\ldots,\infty} R_{ik}$. Son meilleur temps de réponse est donné par : $E_i = \min_{k=1,\ldots,\infty} R_{ik}$. Une tâche τ_i est ordonnançable si, et seulement si, $R_i \leq D_i$. Une configuration de tâches est ordonnançable si toutes les tâches sont ordonnançables. Si la date de réveil de la première instance ne coïncide pas avec le démarrage de l'application alors la première activation de τ_i est notée r_i (*offset*). Dans la suite, n est le nombre de tâches dans le système.

Lorsque les tâches sont indépendantes, une tâche est activable dès qu'elle arrive dans le système, c'est-à-dire à son *réveil*. Ceci n'est plus vrai dès lors que les tâches sont dépendantes. Dans les modèles théoriques de base, les tâches ne comportent pas de synchronisation ou de communication dans leur corps. Les communications et synchronisations ont lieu au début et en fin d'exécution des tâches. Ainsi, une tâche recevant des messages sera *activable* après son réveil et lorsque tous les messages seront arrivés. Le décalage entre le réveil de l'instance et l'instant où elle est dans l'état activable est appelé la *gigue sur l'activation* (*release jitter*) et est noté J_i. Les émissions des messages sont effectuées à la fin des exécutions des tâches.

Les accès aux ressources logicielles ou physiques partagées par les tâches sont gérés par un protocole d'accès comme le protocole à priorité plafond. Une tâche τ_i peut être bloquée en attente qu'une tâche moins prioritaire libère une ou plusieurs ressources. Le pire temps de blocage d'une tâche τ_i est alors calculé en fonction du protocole de synchronisation utilisé. Ce facteur de blocage est généralement noté B_i dans la littérature. Une tâche peut aussi se suspendre durant son exécution en effectuant par exemple une opération d'entrée/sortie.

Les algorithmes d'ordonnancement implémentés dans les noyaux sont généralement des algorithmes en-ligne et impatient (ou conservatif) : l'ordonnanceur choisit une tâche parmi celles qui sont activables et elle est ordonnancée sans attente. Les algorithmes classiques dans la littérature de l'ordonnancement temps réel sont RM (*Rate Monotonic*), DM (*Deadline Monotonic*), EDF *Earliest Deadline First* et FIFO (*First In First Out*). Nous renvoyons à [1] concernant les principales caractéristiques des ces algorithmes d'attribution des priorités aux tâches.

La validation des systèmes temps réel revient à montrer que toutes les tâches respecteront leurs échéances. Trois techniques analytiques existent et reposent :

– sur l'utilisation du processeur (*Processor Utilization Analysis*) [2, 3]. Le facteur d'utilisation est la fraction de temps que le processeur passe à exécuter des tâches. Il se définit par $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$. Dans certains cas particuliers, le facteur d'utilisation du processeur permet de conclure si une configuration de tâches est ordonnançable ou non. Par exemple lorsque les tâches vérifient $r_i = 0$ et $D_i = T_i$ $(1 \leq i \leq n)$, un ordonnancement RM sera valide si $U \leq n(2^{\frac{1}{n}} - 1)$ et un ordonnancement EDF est faisable si, et seulement si : $U \leq 1$ [2]. Ce type de tests conduit généralement à des conditions suffisantes d'ordonnançabilité et sont difficiles à étendre pour prendre en compte des contraintes additionnelles.

– sur la demande processeur (*Processor Demand Analysis*). Cette approche repose sur le calcul de la demande cumulée des exécutions des tâches réveillées et terminées dans un intervalle de temps (*Demand Bound Function*). Cette approche générale permet de construire des tests d'ordonnançabilité [4, 5, 6] en limitant l'étude d'ordonnançabilité à quelques instants d'une période d'activité du processeur.

– sur l'analyse des temps de réponse (*Response Time Analysis*) [7, 8]. Cette analyse détermine les pires temps de réponse des tâches et des messages. Dans le cadre de l'ordonnancement à priorité fixe, cette méthode est désignée sous le nom d'analyse RMA (*Rate Monotonic Analysis*) [9]. L'analyse du temps de réponse permet de valider des configurations de tâches en vérifiant que $R_i \leq D_i$ pour toute tâche. Elle peut être aussi utilisée dans des algorithmes plus complexes pour affecter des priorités ou bien contrôler la qualité de l'ordonnancement en accord avec des critères de performances (temps de réponse moyen, encadrement de la gigue de sortie...).

Les deux dernières approches reposent sur les mêmes concepts et sont parfois désignées par le terme analyse de la demande de temps (*Time-Demand Analysis*) [10]. Le point commun de ces approches est une analyse précise des périodes d'activité du processeur (*busy periods*), c'est-à-dire quand le processeur n'est pas oisif.

De nombreuses méthodes de calcul du pire temps de réponse ne conduisent pas à sa valeur exacte, mais à une borne supérieure. Beaucoup d'articles ne font pas clairement cette distinction. Dans un contexte de validation des tâches, comparer le pire temps de réponse à l'échéance de la tâche ($R_i \leq D_i$) conduit alors à :

– un algorithme de décision si R_i est le pire temps de réponse. C'est-à-dire qu'une tâche est ordonnançable si, et seulement si, $R_i \leq D_i$.

– un algorithme de semi-décision si R_i est une borne supérieure du pire temps de réponse. Si $R_i \leq D_i$, la tâche est ordonnançable, sinon on ne peut pas conclure.

Les modèles de tâches considérés dans les techniques énoncées ci-dessus sont généralement simplistes au regard des applications industrielles. La simulation est alors souvent utilisées dans l'industrie pour contourner de telles difficultés. Les résultats obtenus, même s'ils aident à comprendre la dynamique du système, ne garantissent pas alors la validation des applications temps réel. En effet, les techniques de simulation n'intègrent pas le fait que les pires temps de réponse des tâches s'obtiennent sur des scénarios d'arrivée des tâches souvent difficiles à caractériser, ainsi que lorsque les tâches ne s'exécutent pas nécessairement avec leurs pires durées d'exécution. De plus, les méthodes de simulation modélisent généralement les lois d'arrivée des événements par des lois probabilistes qui sont détuites d'analyses statistiques du système. Cette approche est valide si, et seulement si, ces lois fondées sur l'analyse du passé représenteront toutes les évolutions possibles du système. Cette hypothèse n'est pas vérifiée pour de nombreux systèmes temps réel qui traitent en-ligne des événements.

Dans la suite nous présentons les principes de base de l'analyse du temps de réponse. Le paragraphe 2 présente l'analyse des systèmes monoprocesseurs. Le paragraphe 3 présente l'analyse du temps de réponse des tâches dans le contexte des systèmes distribués, désignée dans la littérature par le terme d'*analyse holistique*. Le paragraphe 4 présente les logiciels d'analyse du temps de réponse ; nous montrerons comment calculer simplement le temps de réponse à l'aide d'un tableur sans aucun effort de programmation. Cet article ne prétend pas couvrir l'ensemble des résultats du domaine. Son objectif est de présenter les analyses de base, ainsi que quelques extensions intégrant des facteurs pratiques.

2. Analyse des systèmes monoprocesseurs

2.1. *Période d'activité du processeur*

Les tâches périodiques sont exécutées indéfiniment. Bien que l'ordonnancement soit infini, la périodicité des tâches permet de limiter la recherche des fautes temporelles dans un intervalle de temps borné, appelé intervalle de faisabilité (*feasibility interval*). La périodicité de l'ordonnancement est égale au ppcm des périodes des tâches et est désignée sous le nom d'*hyperpériode*. De façon plus générale, il est inutile de rechercher une faute temporelle lorque le processeur est inutilisé. Ceci nous amène à définir la notion de période d'activité du processeur. Les intervalles de temps où le processeur est occupé ou libre sont les mêmes pour tous les algorithmes d'ordonnacement conservatifs. Nous considérons dans la suite de ce paragraphe des tâches indépendantes qui sont réveillées simultanément au début de l'application ($r_i = 0, 1 \leq i \leq n$).

Définition 1 *Une période d'activité du processeur est un intervalle de temps dans lequel le processeur est pleinement utilisé.*

La plus longue période d'activité du processeur survient lorsque toutes les tâches sont activées simultanément, un tel événevement est appelé un *instant critique* [2]. Considérons une période d'activité, où toutes les tâches sont réveillées simultanément au début de la période sauf une, disons τ_k. Sans perte de généralité, on peut fixer $r_j = 0, j \neq k$ et $r_k > 0$. Soit L la longueur de la période d'activité ainsi obtenue, alors diminuer r_k ne peut faire augmenter L, puisque la durée cumulée d'exécution de τ_k ne peut être qu'augmentée. Ceci nous permet d'affirmer que la plus longue période d'activité est celle qui est initiée par un instant critique.

Pour déterminer la longueur de cette période d'activité il faut calculer la durée cumulée des exécutions des tâches sur l'intervalle de temps $[0, t[$. Dans cet intervalle, le nombre de réveils de τ_j est défini par $\left\lceil \frac{t}{T_j} \right\rceil$. Notons que ces instances ne sont pas nécessairement terminées à la date t. La durée cumulée de travail processeur associée aux réveils de τ_j sur l'intervalle de temps $[0, t[$ est donc $\left\lceil \frac{t}{T_j} \right\rceil C_j$. La fonction de travail $W(t)$ définit la durée cumulée des tâches réveillées dans l'intervalle $[0, t[$:

$$W(t) = \sum_{j=1}^{n} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

La fonction de travail du processeur $W(t)$ est une fonction en escalier. La droite affine $f(t) = t$ définit la capacité maximum de traitement du processeur. Lorque $f(t) = W(t)$, alors à cette date le processeur a terminé toutes les instances réveillées avant t. Cette égalité correspond à un point fixe de l'équation $W(t) = t$. Considérons un exemple avec 5 tâches dont les paramètres sont donnés dans le tableau 1. Son facteur de charge est $0,92$ et un instant critique se produit toutes les 300 unités de temps. La fonction de charge correspondant à cette configuration est donnée figure 1. On constate que sur une hyperpériode des tâches, il existe de nombreux points fixes pour l'équation $W(t) = t$. Nous allons voir que le premier coïncide avec la longueur de la période d'activité initiée par un instant critique.

τ_i	C_i	T_i
1	7	20
2	5	20
3	8	30
4	3	100
5	2	100

Tableau 1. *Configurations de tâches*

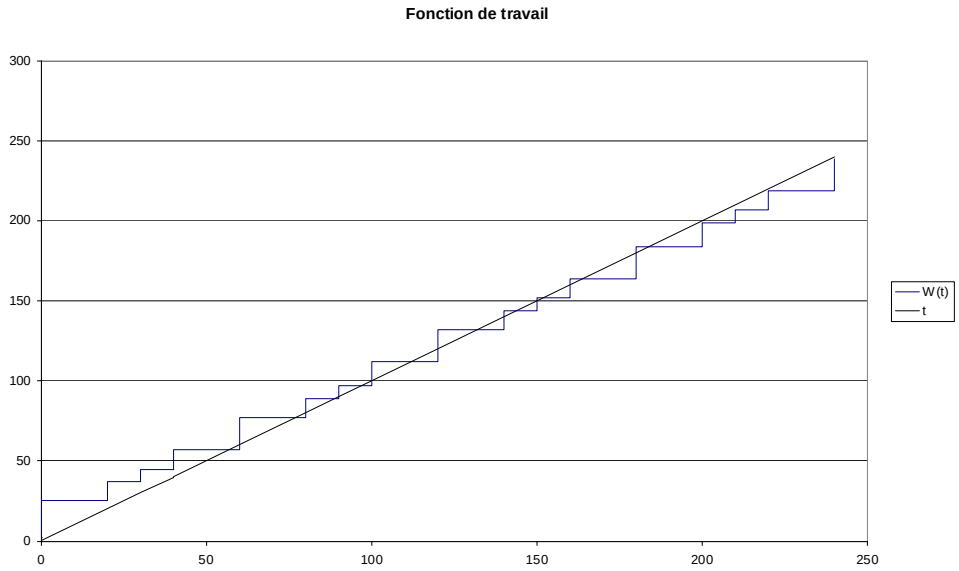

Figure 1. *Fonction de travail associée à la configuration de tâches du tableau 1*

La fin de la période d'activité initiée par un instant critique survient lorsque toutes les tâches réveillées dans l'intervalle de temps $[0,L[$ sont terminées. En conséquence, on vérifie : $W(L) = L$. L'inconnue de cette équation est la variable L. Elle va pouvoir être calculée de proche en proche en considérant d'abord les tâches réveillées à l'instant 0, générant une durée cumulée d'exécution de tâche égale à $t^{(0)} = \sum_{i=1}^{n} C_i \leq L$. De nouvelles instances de tâches peuvent être réveillées dans l'intervalle $[0,W(t^{(0)})[$. Posons $t^{(1)} = W(t^{(0)})$, alors la durée cumulée des tâches dans l'intervalle $[0,t^{(1)}[$ est $W(t^{(1)})$. Une nouvelle fois cette durée vérifie $W(t^{(1)}) \leq L$. En conséquence, déterminer L revient à calculer le plus petit point fixe de l'équation $W(t) = t, t > 0$. La résolution de cette équation s'obtient en calculant la suite suivante :

$$\begin{aligned} t^{(0)} &= \sum_{i=1}^{n} C_i \\ t^{(k+1)} &= W\left(t^{(k)}\right) \end{aligned} \qquad [1]$$

La longueur L est alors définie par le plus petit entier k tel que $L = t^{(k+1)} = t^{(k)}$. Si $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$ alors la convergence de la suite est assurée puisque pour tout t, $0 \leq t < L$, la propriété suivante est vérifiée : $W(t) > t$. Par contre la suite diverge dès lors que $\sum_{i=1}^{n} \frac{C_i}{T_i} \geq 1$. Ceci correspond au cas où le processeur ne

peut pas traiter entièrement la charge de travail. Nous illustrons ces calculs sur la configuration de tâches du tableau 1 ; la suite est initialisée avec $\sum_{i=1}^{n} C_i = 25$ et le plus petit point fixe est atteint après 4 itérations.

$$\begin{aligned}
W(25) &= \left\lceil \frac{25}{20} \right\rceil 5 + \left\lceil \frac{25}{20} \right\rceil 7 + \left\lceil \frac{25}{30} \right\rceil 8 + \left\lceil \frac{25}{100} \right\rceil 3 + \left\lceil \frac{25}{100} \right\rceil 2 = 37 \\
W(37) &= \left\lceil \frac{37}{20} \right\rceil 5 + \left\lceil \frac{37}{20} \right\rceil 7 + \left\lceil \frac{37}{30} \right\rceil 8 + \left\lceil \frac{37}{100} \right\rceil 3 + \left\lceil \frac{37}{100} \right\rceil 2 = 45 \\
W(45) &= \left\lceil \frac{45}{20} \right\rceil 5 + \left\lceil \frac{45}{20} \right\rceil 7 + \left\lceil \frac{45}{30} \right\rceil 8 + \left\lceil \frac{45}{100} \right\rceil 3 + \left\lceil \frac{45}{100} \right\rceil 2 = 57 \\
W(57) &= \left\lceil \frac{57}{20} \right\rceil 5 + \left\lceil \frac{57}{20} \right\rceil 7 + \left\lceil \frac{57}{30} \right\rceil 8 + \left\lceil \frac{57}{100} \right\rceil 3 + \left\lceil \frac{57}{100} \right\rceil 2 = 57
\end{aligned}$$

Nous faisons remarquer que la première période d'activité du processeur, correspondant au plus petit point fixe de la suite précédente, ne précède pas forcément un temps creux dans l'ordonnancement. La longueur L indique que toutes les tâches réveillées dans l'intervalle $[0,L[$ sont terminées. Par contre, de nouvelles instances peuvent se réveiller à la date L.

Nous montrons maintenant que l'algorithme de détermination de L est pseudo-polynomial. Le calcul de la fonction de travail $W(t)$ a clairement une complexité algorithmique en $O(n)$. Le nombre d'itérations de la suite est borné par la longueur L de la période d'activité. Il suffit donc d'établir une borne supérieure de L pour déterminer une borne supérieure du nombre d'itérations. Nous supposons que $u = \sum_{i=1}^{n} \frac{C_i}{T_i}$ est constant et strictement inférieur à 1. En conséquence, $L < H$, où H est la longueur de l'hyperpériode des tâches.

$$\begin{aligned}
L &= \sum_{i=1}^{n} \left\lceil \frac{L}{T_i} \right\rceil C_i \leq \sum_{i=1}^{n} \left(1 + \frac{L}{T_i}\right) C_i \\
L &\leq \frac{\sum_{i=1}^{n} C_i}{1-u}
\end{aligned}$$

La complexité algorithmique du calcul de L est donc $O(n \frac{\sum_{i=1}^{n} C_i}{1-u})$. Sous l'hypothèse u constant, ceci conduit à une complexité algorithmique pseudo-polynomiale : $O(n \sum_{i=1}^{n} C_i)$. L'existence d'un algorithme fortement polynomial est un problème ouvert.

2.2. *Systèmes à priorité fixe*

Lorsque les tâches sont périodiques et à démarrage simultané, la complexité du problème d'ordonnançabilité des tâches est ouverte. Les méthodes détaillées ci-après conduisent à des algorithmes pseudo-polynomiaux. Nous supposons que les indices de tâches indiquent leurs niveaux de priorité. Ainsi, la tâche τ_1 a le niveau de priorité le plus élévé et τ_n a le plus faible. Nous supposons ainsi que les tâches sont affectées à des niveaux de priorités différents.

2.2.1. *Le pire temps de réponse*

Nous considérons les configurations de tâches telles que $r_i = 0, D_i \leq T_i, 1 \leq i \leq n$. Ces configurations sont ordonnancées de façon fiable par l'algorithme Deadline Monotonic. Le calcul pratique du temps de réponse nécessite toujours de :

– caractériser le(s) scénario(s) d'arrivée des tâches conduisant au pire temps de réponse de la tâche étudiée,

– déterminer une fonction d'analyse de la durée cumulée des tâches correspondant au(x) scénario(s).

Dans un système à priorité fixe, une tâche de priorité i s'exécute que s'il n'existe pas de tâche activable plus prioritaire. Le concept de période d'activité va être affiné pour tenir compte des tâches ayant un niveau de priorité supérieur à i.

Définition 2 *[8] Une période d'activité du processeur de niveau i est un intervalle de temps où le processeur n'exécute que des tâches ayant une priorité supérieure ou égale à i (i-level busy period).*

Le calcul pratique du temps de réponse d'une tâche τ_i repose alors sur celui du calcul de la longueur d'une période d'activité, mais en se limitant aux tâches de priorité supérieure ou égale à i. On définit alors la durée cumulée $W_i(t)$ des tâches de priorité supérieure ou égale à i :

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

Le pire temps de réponse de τ_i, noté R_i, est le plus petit point fixe de $W_i(R_i) = R_i$. Pour la configuration du tableau 1, les temps de réponse obtenus sont donnés dans le tableau 2. On constate que la tâche la moins prioritaire a toujours un temps de réponse égal à la plus longue période d'activité du processeur. Nous avons déjà vu que ce calcul est pseudo-polynomial. Notons enfin que le calcul du pire temps de réponse de chaque instance d'une tâche est étudié dans [11, 12, 13, 14].

2.2.2. *Le meilleur temps de réponse*

La détermination du meilleur temps de réponse repose sur des principes très proches du calcul du pire temps de réponse. En effet, dans [11, 15] il est montré que le meilleur temps de réponse d'une tâche τ_i soumise à une gigue sur activation survient lorsque la fin de la tâche étudiée coïncide avec le réveil de toutes les tâches plus prioritaires. Le scénario d'activations des tâches conduisant au meilleur temps de réponse E_i de τ_i est en quelque sorte le scénario dual de celui conduisant au pire temps de réponse. La présence des gigues permet de prendre en compte tous les déphasages possibles sur l'activation des tâches. Sous cette hypothèse, les auteurs proposent un calcul exact du meilleur temps de réponse. En absence de gigue sur activation, la méthode conduit à une borne inférieure du meilleur temps de réponse. Nous détaillons ci-après ce dernier cas. La borne inférieure du meilleur temps de réponse repose sur la suppression de tous les temps creux pour que la fin de la tâche τ_i coïncide avec le réveil de toutes les tâches plus prioritaires. Ceci revient à calculer le plus petit point fixe de l'équation suivante [15] :

$$t = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t - T_j}{T_j} \right\rceil C_j \qquad [2]$$

La suite doit être initialisée par une borne supérieure du meilleur temps de réponse, notée E_i ci-après. Nous proposons une borne supérieure du pire temps de réponse puisque les auteurs de [15] n'en donnent pas. Depuis l'équation précédente, nous pouvons écrire en relachant la contrainte d'intégrité sur le nombre d'activations des tâches :

$$\begin{aligned} E_i &\leq C_i + \sum_{j=1}^{i-1} \left(1 + \frac{E_i - T_j}{T_j}\right) C_j \\ E_i &\leq \frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{T_j}} \end{aligned}$$

Notons toutefois que la qualité de ces bornes en moyenne et dans le pire cas n'est pas connue. Sur l'exemple du tableau 1 les bornes supérieures des meilleurs temps de réponse et les résultats des meilleurs

τ_i	R_i	$UB(E_i)$	E_i
1	5	5	5
2	12	9,33	7
3	20	20	8
4	55	22,5	3
5	57	19,35	2

Tableau 2. *Pires temps de réponse R_i, bornes supérieures du meilleur temps de réponse UB_i et meilleurs temps de réponse E_i des tâches de la configuration du tableau 1*

temps de réponse sont donnés dans le tableau 2. La recherche du point fixe avec l'équation 2 converge pour chaque tâche en deux itérations. Pour chaque tâche le meilleur temps de réponse est égal à sa pire durée d'exécution.

2.2.3. *Intégration de facteurs pratiques*

L'analyse RMA (*Rate Monotonic Analysis*) est de loin la plus développée puisqu'une monographie est entièrement dédiée à la considération de facteurs pratiques [9]. Deux types de facteurs pratiques sont usuellement considérés dans la littérature :

– extensions du modèle de tâche. Les tâches considérées dans les paragraphes précédents sont indépendantes et strictement périodiques. La technique d'analyse du temps de réponse doit être étendue pour être exploitable sur des problèmes industriels.

– prise en compte du comportement intrinsèque du noyau temps réel et du matériel. L'analyse doit intégrer, par exemple, l'activation périodique de la tâche horloge et les durées associées à la manipulation de la queue d'exploitation.

Ces extensions conduisent généralement à redéfinir la fonction de travail des tâches afin de tenir compte des contraintes additionnelles. En effet, les extensions du modèle des tâches induisent généralement des *anomalies d'ordonnancement* lorsqu'un ordonnanceur à priorité fixe est utilisé. Une anomalie se caractérise par le fait que la relaxation d'une contrainte du problème peut rendre la configuration de tâches non ordonnançable [16]. Par exemple, diminuer la durée d'une tâche peut rendre la configuration non ordonnançable, alors que celle-ci est validée lorsque toutes les tâches s'exécutent avec leurs pires durées d'exécution.

D'un point de vue complexité [1], le problème de décision associé au calcul du pire temps de réponse d'une tâche τ_i est de vérifier si $R_i \leq K$, où K est une constante quelconque. On remarque ainsi, que ce problème de décision généralise celui de l'ordonnançabilité de τ_i qui consiste à comparer le temps de réponse avec la constante D_i : $R_i \leq D_i$. En conséquence, si le problème de l'ordonnançabilité d'une tâche est $\mathcal{NP}$-Complet alors celui du calcul du temps de réponse l'est aussi. Le tableau 3 met en évidence l'évolution de la complexité du calcul du pire temps de réponse sur quelques extensions de base. On constate qu'à chaque fois que le problème du calcul du pire temps de réponse est difficile, aucune valeur exacte du pire temps de réponse n'est en général calculable efficacement.

Nous considérons des facteurs pratiques dans les paragraphes suivants, séparément les uns des autres.

2.2.3.1. Communications

Les communications entrantes sont opérées au début de l'exécution d'une instance de tâche et l'émission de messages ont lieu à la fin de son exécution. Aucune opération de synchronisation n'a lieu dans le corps d'une tâche. De plus, les tâches s'exécutant sur le même processeur sont indépendantes. Le début d'une tâche peut être différé en attente des communications entrantes. Ce décalage est modélisé par la gigue sur activation J_i. Ce paramètre n'est pas a priori connu puisqu'il dépend de la dynamique des autres composants

1. Le lecteur pourra se reporter à [17] pour une introduction pédagogique à la théorie de la complexité.

Extension	Complexité	Pire temps de réponse
-	ouverte	exact
dates de réveil	co-$\mathcal{NP}$-Complet au sens fort [18]	simulation
ressources	$\mathcal{NP}$-Complet au sens fort [19]	borne supérieure
non-préemption	$\mathcal{NP}$-Complet au sens fort [20]	borne supérieure
suspension	$\mathcal{NP}$-Complet au sens fort [21]	borne supérieure
contraintes de précédence	ouverte	borne supérieure
communications	ouverte	exact [22]

Tableau 3. *Extensions des tâches périodiques et complexité du calcul du temps de réponse*

du systèmes, c'est-à-dire des tâches émettrices s'exécutant sur un autre processeur et du réseau transmettant les messages. L'analyse d'une tâche peut être réalisée en laissant J_i comme une variable. Le fait que J_i soit non instanciée autorise tous les décalages possibles entre le réveil et l'activation de la tâche. Le pire scénario pour une tâche τ_i survient donc lorsqu'elle est *activée* simultanément avec toutes les tâches plus prioritaires. L'instant critique se produit alors sur l'activation et non sur le réveil comme dans le cas de tâches non soumises à la gigue sur activation. Pour les instances suivantes, les activations coïncident avec les réveils des tâches. Ce scénario est présenté figure 2, en supposant sans perte de généralité que les premières activations des tâches dans la période d'activité surviennent à la date 0. Les flèches en pointillés symbolisent les activations des tâches (i.e. les messages ont été délivrés) alors que les flèches pleines représentent les réveils périodiques des tâches.

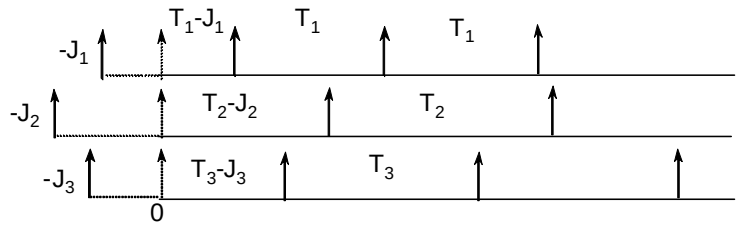

Figure 2. *Pire scénario d'activation de tâches soumises à la gigue sur activation*

La durée cumulée d'exécution des tâches de priorité supérieure ou égale à i dans l'intervalle de temps $[0,t[$ est donc donnée par :

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t + J_j}{T_j} \right\rceil C_j$$

Le pire temps de réponse est la durée de la période d'activité de niveau i avec la fonction de la demande processeur présentée ci-dessus. La durée L_i de cette période d'activité est calculée comme le plus petit point fixe de l'équation $W_i(t) = t$. Le pire temps de réponse de τ_i est donc égal à $R_i = J_i + L_i$. Notons que ce calcul est exact puisque le pire scénario peut toujours être artificiellement défini en instanciant de façon adéquate les gigues J_i. Mais en pratique, rien n'affirme que la dynamique du système distribué engendrera le pire scénario associé à la figure 2. Les équations de calcul des pires temps de réponse ne conduisent dans le contexte distribué qu'à des bornes supérieures des pires temps de réponse.

2.2.3.2. Ressources

La présence de sections critiques dans les tâches rend le problème d'ordonnancement $\mathcal{NP}$-Complet au sens fort. L'accès aux ressources est contrôlé par un protocole d'accès. Les protocoles garantissent généralement

qu'une tâche ne se trouve durablement bloquée par une tâche moins prioritaire (*le phénomène d'inversion de priorité* est borné) et l'absence d'interblocage. La pire durée de blocage d'une tâche τ_i (aussi désignée sous le nom de facteur de blocage), associée à un protocole donné, est définie par B_i. Nous renvoyons à [23] pour la détermination pratique de B_i pour les protocoles d'accès aux ressources les plus utilisés. Ainsi, la tâche de priorité i ne peut voir son activation différée par une tâche moins prioritaire d'une durée de plus de B_i unités de temps. Une borne supérieure du pire temps de réponse est donc définie par le plus petit point point fixe de l'équation suivante :

$$W_i(t) = C_i + B_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

2.2.3.3. Non préemption

L'absence de préemption rend le problème du calcul du pire temps de réponse $\mathcal{NP}$-Complet au sens fort. Comme dans le cas avec ressources, une tâche de priorité i peut se voir différée par une tâche d'un niveau de priorité inférieur qui a débutée quelques instants avant l'activation de τ_i. Lorque la préemption n'est pas autorisée, le tâche active s'exécute jusqu'à sa terminaison (nous supposons que l'ordonnanceur ne peut pas tuer des tâches). L'effet de blocage d'une tâche τ_i est donc égal à la plus longue tâche parmi les moins prioritaires : $\max_{j=i+1,\ldots,n}(C_j)$. Une borne supérieure du pire temps de réponse s'obtient donc comme le plus petit point fixe de l'équation suivante :

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + \max_{j=i+1,\ldots,n}(C_j)$$

2.2.3.4. Suspensions

Les suspensions étendent le modèle de tâche avec gigue sur activation. Le problème d'ordonnancement devient alors $\mathcal{NP}$-Complet au sens fort. En effet la gigue peut être aussi vue comme la suspension d'une tâche au début de son exécution. Jusqu'à maintenant, nous avons supposé que les tâches ne réalisaient aucune opération de synchronisation dans leur corps. En pratique une tâche peut se suspendre dans l'attente de la fin d'une opération d'entrée-sortie. C'est notamment le cas si elle réalise un appel de procédure à distance (*Remote Procedure Call*). La durée de la tâche C_i est obtenue lorsque la tâche ne se suspend pas. De plus la durée de suspension d'une tâche peut varier d'une instance à une autre.

Une tâche de priorité i ne subira aucune interférence de tâches de priorité inférieure. Par contre, une tâche de priorité i pourra se voir différée par la reprise après suspension d'une tâche plus prioritaire. Un moyen d'établir une borne supérieure du pire temps de réponse de τ_i est de considérer ces suspensions comme des facteurs de blocage.

Nous considérons que les tâches se suspendent au plus une fois ; soit X_i la pire durée de suspension de τ_i, alors la suspension va induire un facteur de blocage défini par :

$$X_i + \sum_{j=1}^{i-1} \min(C_j, X_j)$$

Une borne supérieure sur le pire temps de réponse se calcule comme $W_i(t) = t$, où cette fois la fonction de travail est définie par :

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + X_i + \sum_{j=1}^{i-1} \min(C_j, X_j)$$

2.2.3.5. Précédences

De façon surprenante, lorsque les tâches en précédence ne disposent pas des mêmes priorités, le problème de l'ordonnançabilité devient plus complexe. Par exemple, un système de tâches peut devenir non ordonnançable en diminuant la durée d'une tâche. Imaginons que $\tau_i \prec \tau_k, i > k$ alors la tâche τ_k est moins prioritaire que τ_i. Pour respecter la contrainte de précédence τ_i ne sera pas activée avant la terminaison de τ_k. Il résulte alors directement que des tâches de faible priorité engendrent un effet de blocage (lié aux contraintes de précédence) sur des tâches plus prioritaires.

L'analyse du temps de réponse de chaînes non réentrantes de tâches a été étudiée dans [24] (l'instance k de la première tâche de la chaîne ne peut s'exécuter avant la fin de l'instance $k-1$ de la dernière tâche de la chaîne). Dans [25], nous avons étudié le cas de graphes réentrants de tâches. Dans ces deux études, les instances sont non réentrantes (i.e., l'instance k d'une tâche ne peut s'exécuter avant la fin de son instance précédente). Nous renvoyons à ces références pour plus de détails.

Nous verrons que cette difficulté ne se rencontre pas lorsque l'algorithme EDF est considéré.

2.2.3.6. Serveurs de tâches apériodiques et sporadiques

L'ordonnancement des tâches apériodiques a été présenté dans [1]. Usuellement, le traitement des tâches apériodiques est effectué par un serveur periodique. Il convient alors de déterminer l'interférence du serveur sur les tâches moins prioritaires. Nous présentons rapidement l'analyse du serveur différé (*Deferrable Server*) et celle du serveur de tâches sporadiques (*Sporadic Server*). Les résultats présentés ci-après sont tirés de [26].

Le serveur est une tâche supplémentaire définie par sa durée d'exécution C_s et sa période entre deux réveils T_s. Dans le pire cas, le serveur différé se comporte comme une tâche périodique soumise à une gigue sur activation égale à $J_s = T_s - C_s$ (cette présentation élégante est due à [26]). En fixant cette hypothèse, le pire temps de réponse d'une tâche $\tau_i, i > s$ est donné par :

$$\begin{aligned} W_i(t) &= C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t+J_j}{T_j} \right\rceil C_j \\ J_s &= T_s - C_s \end{aligned}$$

Soit L_i le plus petit point fixe de l'équation $W_i(t) = t$, alors une borne supérieure du pire temps de réponse de τ_i est $R_i = J_i + L_i$.

Dans le cas du serveur de tâche sporadique, l'occurence multiple du serveur ne se pose pas, en conséquence l'interférence sur les tâches moins prioritaires dans l'intervalle $[0,t[$ est :

$$\left\lceil \frac{t}{T_s} \right\rceil C_s$$

Le pire temps de réponse d'une tâche s'obtient alors comme le plus petit point fixe de l'équation $W_i(t) = t$, où la fonction $W_i(t)$ est définie comme :

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + \left\lceil \frac{t}{T_s} \right\rceil C_s$$

La convergence des équations est assurée si la charge des tâches plus celle du serveur est inférieure ou égale à 1. Lorsque le serveur utilise des ressources, le phénomène de double blocage peut se produire si le *budget* assigné au serveur est terminé alors qu'il est en section critique [27]. L'analyse doit alors tenir compte de ce phénomène. Nous renvoyons à [26] pour l'extension de l'analyse du temps de réponse.

τ_i	C_i	D_i	T_i
1	2	4	4
2	3	7	7

Tableau 4. *Configurations de tâches*

2.2.3.7. Prise en compte des routines du noyau et du matériel

Les modèles de tâches présentés jusqu'à maintenant ignorent le comportement du noyau. Les changements de contexte d'exécution et les manipulations de la queue d'exploitation ne sont pas en pratique négligeables. La routine de traitement de l'interruption horloge temps réel dépend de sa période T_{CLK} et possède une durée C_{CLK}. La durée de manipulation de la queue d'exploitation pour une instance d'une tâche est donnée par C_{PER}. Le calcul d'une borne supérieure du pire temps de réponse repose donc sur le calcul du plus petit point fixe de l'équation suivante [28] :

$$t = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + \left\lceil \frac{t}{T_{CLK}} \right\rceil C_{CLK} + \sum_{j=1}^{n} \left\lceil \frac{t}{T_j} \right\rceil C_{PER}$$

D'autre part, l'architecture machine prend aussi son importance dans le calcul des pires temps de réponse des tâches. La majorité des processeurs actuels disposent d'une mémoire cache. Ce problème en présence de tâche temps réel à priorité fixe a été étudié dans [29]. On pourra se reporter à [30, 31] pour l'étude de systèmes tolérants aux fautes.

2.3. *Ordonnancement EDF*

EDF est un algorithme d'ordonnancement optimal : si une configuration de tâches est ordonnançable alors elle le sera nécessairement par EDF. Dans le cas des tâches à priorité fixe telles que $r_i = 0$,$D_i \leq T_i$,$1 \leq i \leq n$, alors le calcul du pire temps de réponse d'une tâche ne nécessite pas de considérer la première période d'activité du processeur. La configuration de tâche du tableau 4, tiré de [11], montre que le pire temps de réponse d'une tâche ne survient pas nécessairement dans la première période d'activité du processeur lorsque EDF est utilisé. Le première période d'activité a une longueur L égale à 7 :

$$W(5) = = \left\lceil \frac{5}{4} \right\rceil 2 + \left\lceil \frac{5}{7} \right\rceil 3 = 7$$

$$W(7) = = \left\lceil \frac{7}{4} \right\rceil 2 + \left\lceil \frac{7}{7} \right\rceil 3 = 7$$

La première instance de τ_2 a un temps de réponse égal à 5. La quatrième instance de τ_2 est réveillée à la date 21 (hors de la première période d'activité du processeur) et se termine à la date 27, conduisant ainsi à un temps de réponse de 6. L'ordonnancement est donné figure 3

2.3.1. *Pire temps de réponse*

Le calcul du pire temps de réponse d'une tâche va donc nécessiter une analyse plus fine que dans le cas de tâches à priorité fixe. Toutefois, l'analyse du temps de réponse d'une tâche τ_i impose d'étudier un ensemble de scénarios [32] :

– toutes les tâches autres que τ_i sont réveillées au début de la période d'activité.
– une instance de τ_i d'échéance d est étudiée dans la période d'activité.

Le pire temps de réponse de τ_i sera dans une période d'activité du processeur où toutes les tâches d'échéances inférieures ou égales à τ_i s'exécutent avant elle. La détermination du pire temps de réponse

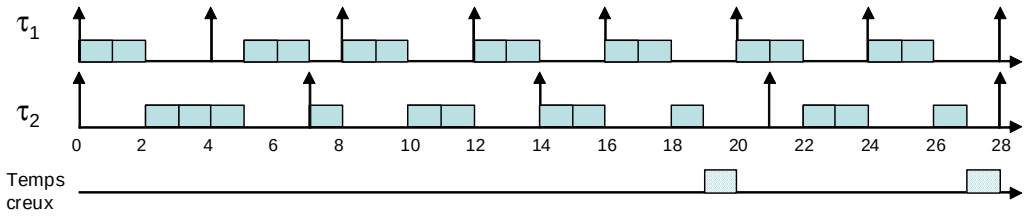

Figure 3. *Ordonnancement EDF des tâches du tableau 4. Le pire temps de réponse de τ_2 ne survient pas dans la première période d'activité du processeur.*

va encore une fois se réduire à calculer la longueur d'une période d'activité en se limitant aux tâches de priorité supérieure ou égale à celle de τ_i.

Considérons le scénario où une instance de τ_i a une échéance d, alors celle-ci est réveillée à la date $a = d - D_i$. Le premier réveil de τ_i dans la période d'activité survient donc à la date :

$$s_i(a) = a - \left\lfloor \frac{a}{T_i} \right\rfloor T_i$$

Les tâches autres que τ_i sont réveillées simultanément au début de la période d'activité. Sans perte de généralité, nous supposons que ces tâches sont réveillées à la date 0. Considérons une tâche $\tau_j, j \neq i$, alors τ_j s'exécutera avant τ_i dans la période d'activité si, et seulement si, $D_j \leq a + D_i$. Précisément pour une date t de la période d'activité, $\left\lceil \frac{t}{T_j} \right\rceil$ instances de τ_j ont été réveillées, mais au plus $1 + \left\lfloor \frac{a+D_i-D_j}{T_j} \right\rfloor$ ont une échéance inférieure ou égale à d. La durée cumulée d'exécution des tâches plus prioritaires que τ_i est donnée par :

$$K_i(a,t) = \sum_{D_j \leq a+D_i} \min\left(\left\lceil \frac{t}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right) C_j$$

Dans ce même scénario, la durée cumulée d'exécution des instances τ_i ayant une échéance inférieure ou égale à d sont :

– si $t < s_i(a)$, alors τ_i n'engendre aucune charge.
– si $s_i(a) \leq t < a$, alors la durée d'exécution des instances de τ_i est :

$$\left\lceil \frac{t - s_i(a)}{T_i} \right\rceil C_i$$

– si $t \geq a$, alors on ne considère que les instances d'échéance inférieure ou égale à d, soit une durée cumulée d'exécution égale à :

$$\left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i$$

En conséquence, la durée cumulée d'exécution des instances de τ_i est :

$$I_i(a,t) = \min\left(\left\lceil \frac{t - s_i(a)}{T_i} \right\rceil, \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) \right) C_i \quad \text{si} \quad t > s_i(a) \quad 0 \text{ sinon}$$

Le demande processeur conduisant au pire temps de réponse d'une instance de τ_i réveillée à la date a est :

$$W_i(a,t) = K_i(a,t) + I_i(a,t)$$

a	$L_1(a)$	$r_1(a)$	a	$L_2(a)$	$r_2(a)$	a	$L_3(a)$	$r_3(a)$	a	$L_4(a)$	$r_4(a)$	a	$L_5(a)$	$r_5(a)$
0	12	**12**	0	12	**12**	0	20	**20**	0	57	**57**	0	57	**57**
10	20	10	10	20	10	10	20	10						
20	20	5	20	20	7									
40	20	5	40	20	7									

Tableau 5. *Scénarios et pires temps de réponse des tâches du tableau 1. Les valeurs en gras indiquent les pires temps de réponse pour chaque tâche.*

La fonction de travail va être utilisée pour déterminer la longueur de la période d'activité, noté $L_i(a)$, correspondant au scénario où τ_i débute à la date a. Le calcul pratique va une nouvelle fois reposer sur une suite récurrente. L'initialisation de la suite nécessite d'établir une borne inférieure de $L_i(a)$. Il suffit pour cela de considérer une instance des tâches $\tau_j, j \neq i$ telles que $D_j \leq a + D_i$ et une instance de τ_i si elle est activée au début de la période d'activité (i.e $s_i(a) = 0$). Soit b un booléen tel que $b = 1$ si $s_i(a) = 0$, $b = 0$ sinon. $L_i(a)$ est le plus petit point fixe de la suite suivante :

$$\begin{aligned} L_i^{(0)}(a) &= b \times C_i + \sum_{j \neq i, D_j \leq a+D} C_j \\ L_i^{(k+1)}(a) &= W\left(a, L_i^{(k)}(a)\right) \end{aligned}$$

Soit $r_i(a)$ le pire temps de réponse correspondant au scénario où une instance de τ_i arrive à la date a. Si $L_i(a) < a$, alors l'instance de τ_i considérée dans le scénario (d'échéance $a + D_i$) n'appartient pas à cette période d'activité. On prend alors la convention $r_i(a) = C_i$. Si $a < L_i(a) \leq a + D_i$ alors le temps de réponse de τ_i est $L_i(a) - a$ sinon τ_i manque son échéance et on peut prendre pour convention $r_i(a) = \infty$. Le pire temps de réponse de τ_i est donc donné par :

$$R_i = \max_{L - C_i \geq a \geq 0} (r_i(a))$$

Le pire temps de réponse revient donc à considérer tous les scénarios tels que $L - C_i \geq a \geq 0$. La fonction de la demande processeur étant une fonction en escalier, sa variation survient lorsque l'échéance d coïncide avec l'échéance d'une autre tâche, ou bien lorsque τ_i est réveillée à l'instant 0 (i.e. instant critique). Soit L la longueur de la période d'activité initiée par un instant critique (c.f. équation 1), puisque L est la plus longue des périodes d'activité il est suffisant de se limiter à celle-ci pour déterminer tous les scénarios possibles de réveil des instances de τ_i. Soit A l'ensemble des dates de réveils possibles telles que l'échéance de τ_i coïncide avec l'échéance d'une tâche dans la période d'activité, cet ensemble de dates est défini par :

$$A = \left\{ kT_j + D_j - D_i \qquad j = 1, \ldots, n, \quad k = 1, 2, \ldots, \left\lfloor \frac{L - C_i + D_j}{T_j} \right\rfloor \right\}$$

Le pire temps de réponse est le maximum des temps de réponse des scénarios :

$$R_i = \max_{a \in A}(r_i(a))$$

Nous donnons dans le tableau 5 les résultats de la méthode présentée en considérant les tâches du tableau 1. Les temps de réponse indiqués en gras sont les pires temps de réponse. Pour cet exemple, les pires temps de réponse des tâches surviennent lorsque toutes les tâches sont réveillées simultanément. On constate de plus qu'avec EDF, lorsque les tâches ont les mêmes paramètres alors elles ont des pires temps de réponse identiques.

Notons enfin, qu'à notre connaissance, la détermination du meilleur temps de réponse est un problème ouvert. Le calcul du temps de réponse d'une instance numéro k est présenté dans [33], toutefois la complexité algorithmique de ce calcul est équivalent à la construction de la séquence par simulation. De plus, ces deux d'approches supposent que les tâches ont des durées fixes, a priori connues.

2.3.2. *Intégration de facteurs pratiques*

Il existe beaucoup moins de développements fondés sur EDF pour intégrer des facteurs pratiques. Il n'existe pas l'équivalent d'une méthode RMA pour l'ordonnancement à priorité dynamique selon EDF. Comme pour le cas de l'analyse à priorité fixe, nous présentons deux directions sur les extensions du modèle de tâches et la prise en compte du fonctionnement du noyau temps réel gérant les tâches.

2.3.2.1. Communications et ressources

Nous renvoyons à [32] pour la prise en compte des ressources et des communications. L'étude de tâche soumises à des dates de réveils non identiques (offset) est présentée dans [34].

2.3.2.2. Précédences

Notons que la considération de tâches dépendantes dans une analyse EDF peut toujours se ramener à une l'analyse d'une configuration de tâches indépendantes dotées d'échéances modifiées. Soit $\prec$ l'ordre partiel sur les tâches. Nous supposons que toutes les tâches en précédence ont une même période d'activation, car sinon tôt ou tard, une tâche manquera une échéance ou un signal de synchronisation sera perdu dans une mémoire tampon du noyau. Sous l'hypothèse de tâche à départ simultané ($r_i = 0, 1 \leq i \leq n$), alors les contraintes de précédence seront respectées si, et seulement si,

$$\tau_i \prec \tau_j \quad \Rightarrow \quad D_i < D_j$$

Le jeu des priorités induit par EDF induira le respect des contraintes de précédence (i.e., sans recourir explicitement à des primitives de synchronisation du noyau). La configuration de tâches avec les échéances modifiées est ordonnançable si, et seulement si, la configuration originelle est ordonnançable. L'analyse du temps de réponse peut donc être étudiée sans crainte avec la configuration modifiée.

2.3.2.3. Suspensions

A notre connaissance, la suspension des tâches n'a jamais été traitée avec un ordonnancement EDF.

2.3.2.4. Prise en compte des routines du noyau et du matériel

En ce qui concerne la prise en compte des contraintes du noyau temps réel, notons que les traitements liés aux routines d'interruption sont considérés dans [5]. A notre connaissance, peu d'études existent pour prendre en compte de tels facteurs pratiques dans un ordonnancement EDF. Ceci est principalement lié au fait que la majorité des noyaux temps réel commerciaux n'implémentent pas EDF.

2.4. *Autres politiques d'ordonnancement*

2.4.1. *Ordonnancement FIFO*

L'ordonnancement FIFO est très utilisé en pratique pour traiter des tâches apériodiques ou pour transmettre des messages sur un réseau. Avec des tâches périodiques, FIFO s'apparente à un algorithme à priorité dynamique : les priorités des tâches sont triées selon leurs dates de réveils. Cette politique d'ordonnancement conduit à une séquence sans aucune préemption des tâches. Le calcul du pire temps de réponse des tâches ordonnancées selon FIFO est présentée dans [35]. Nous proposons ici une preuve immédiate d'un de leurs résultats en montrant que le calcul du pire temps de réponse des tâches ordonnancées selon FIFO peut s'obtenir depuis les résultats connus sur EDF.

Nous montrons tout d'abord que tout ordonnancement FIFO d'une configuration de tâches s'obtient en ordonnançant les tâches selon EDF en imposant $D_i = 0$ pour toute tâche. En fixant $D_i = 0, 1 \leq i \leq n$, on obtient que les dates de réveils des tâches sont égales à leurs échéances. Ainsi l'ordonnancement EDF choisira à chaque instant la tâches d'échéance absolue la plus petite, et en conséquence la tâche ayant la plus

petite date de réveil. L'ordonnancement EDF résultant est équivalent à l'ordonnancement FIFO. Le corollaire immédiat de ce résultat est de montrer simplement que FIFO minimise le plus grand temps de réponse des tâches. Ceci est simple à montrer puisqu'EDF minimise le plus grand retard [36] et que $D_i = 0$ pour toute tâche, alors le retard d'une instance est égal à son temps de réponse.

Ces deux résultats nous montrent que le calcul du pire temps de réponse de tâches ordonnancées selon la politique FIFO peut être fait en calculant les pires temps de réponse de tâches ordonnancées selon EDF.

2.4.2. *Ordonnancement avec files multi-niveaux*

Dans les modèles précédents, chaque tâche à priorité fixe dispose de son niveau de priorité. En pratique, le nombre de niveaux différents de priorité est limité (souvent à 256). Les tâches affectées à la même priorité sont alors ordonnancées selon FIFO ou l'algorithme du tourniquet (round-robin). Supposons que $ep(i)$ contiennent les tâches autre que τ_i affectées au même niveau de priorité et $hp(i)$ celles qui sont affectées à des niveaux supérieurs de priorité. Dans le pire cas, τ_i subira l'interférence naturelle des tâches plus prioritaires et au plus une instance de chaque tâche de $ep(i)$ s'exécutera complètement avant que τ_i soit activée. Une borne supérieure du temps de réponse s'obtient donc en utilisant la fonction de travail suivante :

$$W_i(t) = C_i + \sum_{j \in hp(i)} \left\lceil \frac{t}{T_j} \right\rceil C_j + \sum_{j \in ep(i)} C_j$$

Notons qu'une analyse de calcul de temps de réponse de l'ordonnancement spécifié dans la norme POSIX 1003.b a été menée dans [37]. De plus, une étude statistique menée dans [38] sur une technique de répartition des tâches dans les différents niveaux de priorités montre que 256 niveaux de priorités permettent d'ordonnancer selon RM des systèmes contenant plusieurs milliers de tâches.

2.5. *Ordonnancement dans les réseaux*

Nous illustrons le calcul du pire temps de réponse d'un message à travers l'exemple du réseau CAN (Controller Area Network). L'analyse est très proche de celles à priorités fixes dans un contexte non préemptif. Nous indiquerons ensuite les principales références bibliographiques sur le calcul du pire temps de réponse des messages dans les réseaux les plus courants dans les systèmes temps réel distribués.

2.5.1. *Analyse du réseau CAN*

Le réseau CAN est un bus à priorité. Chaque message possède une priorité fixe et à un instant donné le message le plus prioritaire obtient le canal de communication. Le temps de propagation d'une trame de b octets dans le réseau CAN, où τ_{bit} est le temps de propagation d'un bit dans le réseau, est donné par la formule suivante [22]:

$$C_m = \left(\left\lfloor \frac{34 + 8b}{5} \right\rfloor + 47 + 8b \right) \tau_{bit}$$

Le réseau CAN est une ressource non partageable et chaque message possède une priorité fixe, arbitrant l'accès au canal de communication. En conséquence, l'ordonnancement du réseau s'apparente alors à l'ordonnancement monoprocesseur non préemptif de tâches. Lorsqu'un message de priorité i doit être transmis sur le réseau, son émission peut être au plus bloquée par l'émission du plus long message de priorité inférieure à i. La fonction de travail d'un message de priorité i, soumis à une gigue sur activation, est donc :

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t + J_j}{T_j} \right\rceil C_j + \max_{j=i+1,\ldots,n} (C_j)$$

Le pire temps de réponse du message m_i est comme dans les cas précédents défini par le plus petit point fixe L_i de $W_i(t) = t$. Le pire temps de réponse du message est alors $R_i = J_i + L_i$.

En fonction des applications, l'analyse de base doit être étendue de façon à tenir par exemple compte :

– des phénomènes de gigues [39] pour des applications de contrôle/commande,
– des problèmes d'accessibilité [40] lorsque des défaillances ou des erreurs de transmission peuvent survenir[41].
– du délai après reprise d'erreur [42].

2.5.2. *Autres réseaux*

Nous donnons ci-après uniquement quelques références concernant des réseaux pouvant être utilisés dans des applications temps réel. Dans [22] sont étudiés deux protocoles de communication : le jeton temporisé et le bus à priorité. Dans [43] le réseau FDDI est analysé. Le réseau FIP est étudié dans [44] avec un trafic de messages sporadiques. L'analyse du réseau ATM est présentée dans [45], et le réseau ARINC 629 (domaine de l'aéronautique) est étudié dans [46].

3. Analyse des systèmes distribués

Nous allons voir que l'analyse des systèmes distribués est une extension directe de l'analyse monoprocesseur avec des gigues sur les activations des tâches.

3.1. *Anomalie d'ordonnancement*

En monoprocesseur, lorsque les tâches contiennent des sections critiques, une tâche prioritaire peut être bloquée par une tâche moins prioritaire détenant une ressource. La transmission des messages sur un réseau, en environnement distribué, va provoquer le même phénomène d'inversion de priorité puisque la transmission d'un message quelconque ne peut 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 le message d'une tâche prioritaire. Ce problème est illustré dans les figures 4 et 5, où la tâche 1 émet un message m_1 pour la tâche 3 et 2 émet un message m_2 pour la tâche 4. Dans le cas de la figure 4, les tâches s'exécutent avec leurs pires temps d'exécution, et chaque tâche respecte son échéance. Dans le cas de la figure 5, la tâche 2 s'exécute plus vite entraînant l'émission de m_2 avant m_1 sur le réseau. La tâche 3 manque alors son échéance (représentée par une flèche descendante).

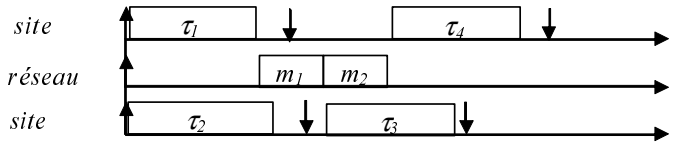

Figure 4. *Ordonnancement avec les pires durées d'exécution*

3.2. *Analyse holistique*

3.2.1. *L'analyse*

Le terme d'analyse holistique a été introduit par Ken Tindell [47], et symbolise la prise en compte de la dépendance entre l'ordonnancement des tâches et des messages dans les systèmes temps réel distribués.

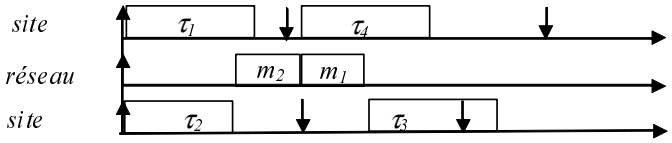

Figure 5. *Ordonnancement avec la tâche τ_2 plus rapide que dans la figure 4*

La méthode analyse les temps de réponse des tâches sur chaque processeur séparément. 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. 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. Les fonctions $TempsReponse$ 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, ont été présentées dans les paragraphes précédents. Soit n le nombre de tâches et de messages et $pred(i)$ l'ensemble des messages entrant (resp. tâches émettrices) de τ_i (resp. du message m_i), le système d'équations à résoudre est :

$$1 \leq i \leq n \begin{cases} J_i^{(0)} & = & 0 \\ R_i^{(k)} & = & TempsReponse\left(i, J_i^{(k-1)}\right) \\ J_i^{(k)} & = & \max\limits_{j \in pred(i)} \left(R_j^{(k)}\right) \end{cases} \qquad [3]$$

Les pires temps de réponse des tâches et des messages sont associés au plus petit point fixe du système :

$$R_i = R_i^{(k)} = R_i^{(k-1)} \qquad 1 \leq i \leq n$$

Sous l'hypothèse de convergence de chaque fonction $TempsReponse$, l'analyse holistique converge si ces fonctions sont non-décroissantes en fonction des gigues des tâches et des messages. Les fonctions calculant les temps de réponse des tâches ou des messages pour chaque processeur et pour chaque réseau ne sont pas dépendantes les unes des autres. Par exemple, un processeur peut être ordonnancé avec RM tandis qu'un autre peut l'être avec EDF. Nous décrivons un tel exemple dans [48].

3.2.2. *Performance des calculs*

Dans [49] sont proposées des heuristiques permettant d'accélérer la convergence de l'algorithme. Ces optimisations reposent sur l'ordre de résolution des équations récurrentes et sur l'amélioration de la borne inférieure qui initialisent les recherches des plus petits points fixes. Notons toutefois que le temps de résolution dépend principalement des données (i.e. des paramètres des tâches et des messages) et que l'influence des heuristiques sur l'ordre de résolution des équations est souvent négligeable. En pratique le nombre d'itérations de l'analyse holistique reste faible. Le temps de calcul est principalement lié au calcul des temps de réponse pour chaque tâche sur chaque processeur.

3.2.3. *Pessimisme de la méthode*

La méthode peut s'avérer pessimiste comme le met en évidence l'article [50]. Toutefois, nous avons mené des expérimentations avec une méthode d'affectation des priorités dans les systèmes distribués à priorité fixe conduisant à des charges de 70 pourcents sur chaque processeur [51]. Ceci illustre que la méthode peut être appliquée pour analyser des systèmes industriels.

4. Outils pour l'analyse du temps de réponse

Il existe peu d'outils commerciaux et académiques d'aide au calcul du pire temps de réponse dans les systèmes temps réel. D'un point de vue théorique, l'analyse RMA est certainement la plus développée. Nous donnons ci-après quelques exemples de logiciels :

– Rapid RMA de Tri-Pacific [52] fournit une boîte à outils fondée sur PERTS [53]. Comme son nom l'indique, cette boîte à outils repose sur l'analyse RMA. Il autorise les concepteurs de tester, simuler et exécuter des modèles du logiciel sur de nombreux scénarios.

– Time Wiz de TimeSys [54] contient le même type d'outils que Rapid RMA et est fondé lui aussi sur l'analyse RMA.

– gRMA (*a Graphical tool for Rate Monotonic Analysis*) est un logiciel libre, dont les sources sont disponibles afin d'étendre les analyses [55].

– ProtEx [56] étend le contexte d'étude afin de supporter les analyse de bout-en-bout. Cette boîte à outils permet d'analyser et prototyper des systèmes distribués temps réel.

– TkRTS [57] est un outil permettant d'analyser les systèmes monoprocesseurs avec les algorithmes classiques, ainsi que les files multi-niveaux.

L'usage industriel de tels outils impose une validation des logiciels utilisés. A notre connaissance, aucune certification spécifique de ces logiciels ne permet de garantir leur fiabilité. Ainsi, le concepteur d'un système temps réel se voit obliger de valider au sein de l'équipe de développement ou bien d'en développer un intégralement.

Nous montrons ci-après que l'analyse du temps de réponse pour les systèmes monoprocesseurs et distribués à priorité fixe peut être faite sans aucun nouveau développement. Nous proposons une mise en œuvre du calcul du pire temps de réponse avec un tableur comme Excel. Aucune programmation n'est nécessaire, rendant l'étape de validation de l'analyse quasi-immédiate (il faut tout de même s'assurer que les formules Excel ne contiennent pas d'erreurs !). Ce type d'implémentation permet à un enseignant de préparer rapidement des sujets de travaux dirigés, mais nous montrons aussi que cette approche peut être utilisée pour résoudre des problèmes pour des systèmes distribués.

Nous décrivons tout d'abord le calcul du pire temps de réponse de tâches à priorité fixe s'exécutant sur un processeur. Une implémentation naïve avec Excel du calcul du plus petit point fixe de l'équation $W_i(t) = t$ reposerait sur un ensemble de cellules Excel mémorisant les résultats de chaque itération de la suite. Une telle implémentation est donc pseudo-polynomial en espace (i.e. en nombre de cellules) et en temps. Une résolution plus simple consiste pour chaque tâche deux cellules représentant respectivement t et $W_i(t)$, $1 \leq i \leq n$. La formule dans la cellule représentant t pointe vers le résultat de celle contenant $W_i(t)$ et celle de $W_i(t)$ utilise la valeur de la cellule t. En mode de fonctionnement normal d'Excel, une référence circulaire est détectée induisant un cycle dans l'ordre des calculs à faire. Ce problème peut être évité un paramétrant le moteur de calcul d'Excel pour résoudre les références circulaires par itérations successives comme indiqué dans la boîte de dialogue des options présentée figure 6. La complexité en espace est en $O(n)$.

La feuille de calcul correspondant au calcul du pire temps de réponse des tâches à priorité fixe sur un processeur est donnée figure 7. La formule Excel dans la cellule $C12$, mémorisant $W_1(t)$, est la suivante :

$$\begin{aligned} = B2 \quad &+ \quad (A12 > \$A\$2) * ARRONDI.SUP(B12/\$C\$2; 0) * \$B\$2 \\ &+ \quad (A12 > \$A\$3) * ARRONDI.SUP(B12/\$C\$3; 0) * \$B\$3 \\ &+ \quad (A12 > \$A\$4) * ARRONDI.SUP(B12/\$C\$4; 0) * \$B\$4 \\ &+ \quad (A12 > \$A\$5) * ARRONDI.SUP(B12/\$C\$5; 0) * \$B\$5 \end{aligned}$$

Les conditions de la forme $(A12 > \$A\$2)$ permettent de prendre en compte seulement les tâches plus prioritaires que celle considérée, puisque la condition FAUX correspond à la valeur 0 alors que la valeur logique VRAI correspond à l'entier 1. Une condition logique peut ainsi être utilisée dans une expression arithmétique

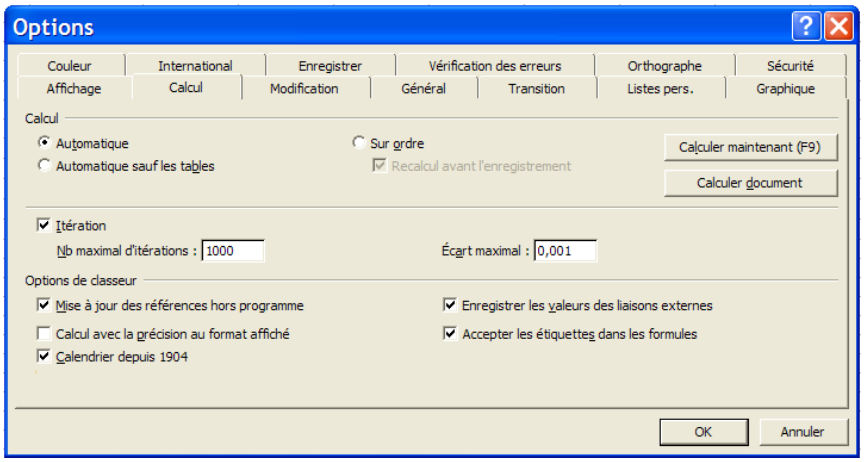

Figure 6. *Paramétres de résolution des références circulaires par itérations successives*

(comme dans le langage C). Cette formule est étendue vers le bas afin de définir les formules du calcul des pires temps de réponse des autres tâches. Ainsi quel que soit le nombre de tâches à étudier, deux formules sont nécessaires aux calculs de tous les temps de réponse une pour t_1 et une pour $W_1(t)$.

B8 {=SOMME(B2:B6/C2:C6)}

	A	B	C	D	E
1	Tasks	C_i	T_i		Formula
2	1	5	20		
3	2	7	20		
4	3	8	30		
5	4	3	100		
6	5	2	100		
7					
8	Utilization factor	0,92			
9					
10					
11	Tasks	t	W(t)		
12	1	5,00	5,00		
13	2	12,00	12,00		
14	3	20,00	20,00		
15	4	55,00	55,00		
16	5	57,00	57,00		
17					

Figure 7. *Calcul des pires temps de réponse des tâches de la table 1 sur un processeur*

Une fois le mode de calcul itératif selectionné, les cellules convergent vers le plus petit point fixe de chaque période d'activité de niveau i. Par contre, la modification des paramètres des tâches engendrera un recalcul correct tant que les suites sont initialisées dans la colonne t par des bornes inférieures des pires temps de réponse. Le plus simple est de redéfinir la formule de cette colonne avant tout recalcul. Ceci peut être facilement mis en œuvre à l'aide d'une macro-commande.

En utilisant le même principe pour chaque processeur et en tenant compte des gigues, il est aussi simple de faire une analyse holistique. Le nombre de formules dépend alors du nombre de sites et de réseaux. La complexité de la *construction* d'une telle feuille de calcul est donc en $O(m)$, où m est le nombre de sites et de réseaux considérés. Par contre la longueur des formules pour chaque site dépend du nombre de tâches (ou de message) à ordonnancer. L'intérêt est de ne pas coder une seule ligne de programme. De plus les formules du tableur sont les mêmes que les formules de fonction de travail de l'analyse du temps de réponse et la formulation de l'équation à résoudre est la même : $W(t) = t$.

5. Conclusion

Nous avons présenté l'analyse du temps de réponse dans les systèmes temps réel monoprocesseurs et distribués. L'intégration de facteur pratique complexifie le problème de la validation d'une configuration de tâches ainsi que le calcul des pires temps de réponse des tâches et des messages. Très souvent, uniquement une borne supérieure du pire temps de réponse peut être calculée. Les concepteurs d'application temps réel doivent alors surdimensionner le système temps réel à défaut de pouvoir le valider. La qualité des approximations réalisées dans les calculs de borne supérieure des pires temps de réponse n'est généralement pas étudiée dans la littérature. Le caractère pessimiste de cette analyse du temps de réponse est donc mal connu.

6. Remerciements

Je tiens à remercier Stéphane Pailler, doctorant au LISI, pour sa lecture attentive et ses nombreux commentaires sur cet article.

Références

[1] A. Geniet, Ordonnancement temps réel, proc Ecole d'été Temps réel (Toulouse) (2003) 14p.

[2] C. Liu, J. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM 20 (1) (1973) 40–61.

[3] C. Han, H. Tyan, A better polynomial time scheduliability test for real-time fixed-priority scheduling algorithms, Real-Time Systems Symposium (1997) 36–45.

[4] J. Lehoczky, L. Sha, Y. Ding, The rate monotonic scheduling algorithm: Extact characterization and average case behavior, Real-Time Systems Symposium (1989) 166–171.

[5] K. Jeffay, D. Stone, Accounting for interrupt handling costs in dynamic priority task systems, Real-Time Systems Symposium (1993) 212–221.

[6] S. Baruah, Dynamic- and static-priority scheduling of recurring real-time tasks, Journal of Real-Time Systems 24 (1) (2003) 93–128.

[7] M. Joseph, P. Pandya, Finding response times in a real-time system, BCS Computer Journal 29 (5) (1986) 390–395.

[8] J. Lehoczky, Fixed-priority scheduling of periodic task sets with arbitrary deadlines, Real-Time Systems Symposium (1990) .

[9] M. Klein, T. Ralya, B. Pollak, R. Obenza, M. Gonzales-Harbour, A practitioner's handbook for real-time system analysis, Kluwer Academic Publishers, 1993.

[10] J. Liu, Real-Time Systems, Prentice hall, 2000.

[11] J. Goossens, Scheduling of hard-real time periodic systems with various kinds of deadline and offset constraints, PhD Thesis, Université Libre de Bruxelles, 1999.

[12] O. Redell, M. Törgren, Calculating exact worst-case response times for static priority scheduled tasks with offsets and jitter, proc. Real-Time Systems and Applications Symposium (RTAS'02) (2002) .

[13] G. Bernat, Response time analysis of asynchronous real-time systems, Journal of Real-Time Systems, to appear (2003) .

[14] P. Hladik, A. Deplanche, Ordonnancement préemptif à priorités fixes : comparaison de méthodes analytiques de calcul de pire temps de réponse, proc. Embedded Systems (RTS'03), Paris (2003) 308–333.

[15] O. Redell, M. Sanfridson, Exact best-case response time analysis of fixed priority scheduled tasks, Euromicro Conf. on Real-Time Systems (ECRTS'02) (2002) .

[16] R. Graham, Bounds on multiprocessing timing anomalies, SIAM Journal of Applied Mathematics 17 (2) (1969) 416–429.

[17] A. Tovey, Tutorial on computational complexity, Interface 32 (3) (2002) 30–61.

[18] S. Baruah, L. Rosier, R. Howell, Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor, Journal of Real-Time Systems 1 (1990) 301–324.

[19] A. Mok, Fundamental design problems of distributed systems for the hard real-time environment, PhD Thesis, Massachusetts Institute of Technology, 1983.

[20] K. Jeffay, D. Stanat, C. Martel, On non-preemptive scheduling of periodic and sporadic tasks, Real-Time Systems Symposium (1991) .

[21] P. Richard, On the complexity of scheduling real-time tasks with self-suspensions on one processor, IEEE Euromicro Conf. on Real-Time Systems (ECRTS'03) (2003) 8p.

[22] K. Tindell, A. Burns, A. Wellings, Analysis of hard real-time communications, Journal of Real-Time Systems 9 (2) (1995) 287–300.

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

[24] M. GonzalezHarbour, M. Klein, J. Lehoczky, Timing analysis for fixed-prioriy scheduling of hard real-time systems, IEEE Transactions on Software Engineering 20 (1) (1994) 13–28.

[25] M. Richard, P. Richard, E. Grolleau, F. Cottet, Contraintes de précédences et ordonnancement monoprocesseur, proc. Embedded Systems (RTS'02), Paris (1) (2002) 121–138.

[26] G. Bernat, A. Burns, New results on fixed priority aperiodic servers, proc. Real-Time System Symposium (1999) .

[27] T. Ghazalie, T. Baker, Aperiodic servers in a deadline scheduling environment, Journal of Real-Time Systems (1995) 31–67.

[28] A. Burns, Preemptive priority-based scheduling: an appropriate engineering approach, in: Advances in Real-Time Systems (ed Sang H. Son), Prentice Hall (1994) 225–248.

[29] C. Lee, et al, Analysis of cache related preemption delay in fixed priority preemptive scheduling, IEEE Transactions on Computers 47 (6) (1998) 700–713.

[30] P. Chevauchot, I. Puaut, Holistic schedulability analysis of a fault tolerant real-time distributed run-time support, proc. Int Conf on Real-Time Computer Systems and Applications (RTCSA'00) (2000) .

[31] G. Lima, A. Burns, An effective schedulability analysis for fault-tolerant hard real-time systems, proc. Int Euromicro Conf on Real-Time Systems (ECRTS'01) (2001) .

[32] M. Spuri, Analysis of deadline scheduled real-time systems, INRIA Reseach Report 2772 (1996) 34p.

[33] R. Devillers, J. Goossens, General response time computation for the deadline driven scheduling algorithm of periodic tasks, Fundamenta Informaticae 40 (2-3) (1999) 199–219.

[34] J. Palencia, M. GonzalesHarbour, Offset-based response time analysis of distributed systems scheduled under edf, proc IEEE Euromicro Conf. on Real-Time Systems (ECRTS'03) (2003) .

[35] L. George, P. Minet, A fifo worst-case analysis for a hard real-time distributed problem with consistency constraints, International Conference on Distributed Computing Systems (ICDCS'97) (1997) .

[36] P. Brucker, Scheduling Algorithms, (Third edition) Springer Verlag, 2001.

[37] J. Migge, A. Jean-Marie, N. Navet, Timing analysis of compound scheduling policies : Application to posix1003.b, Journal of Scheduling, to appear (2003) .

[38] J. Lehoczky, L. Sha, Performance of real-time bus schedulng algorithms, ACM Performance Evaluation Review 14 (1986) .

[39] T. Nolte, H. Hansson, C. Norström, Minimizing can response-time jitter by message manipulation, proc. IEEE Real-Time and Embedded technology and Applications (RTAS'02) (2002) .

[40] L. Pinho, F. vasques, E. Tovar, Integrating inacessibility in response time analysis of can network, proc. IEEE Workshop on Factory Communication Systems (2000) .

[41] N. Navet, Y. Song, F. Simonot, Worst-case deadline failure probability in real-time applications distributed over controller area network, Journal of Systems Architecture 46 (2000) 607–617.

[42] K. Tindell, A. Burns, A. Wellings, Calculating controller area network (can) message response times, Control Engineering Practice 3 (8) (1995) 1163–1169.

[43] S. Zhang, A. Burns, Guaranteeing synchronous message sets in fddi networks, IFAC Workshop on Distributed Computer Control (1995) 107–112.

[44] P. Pedro, A. Burns, Worst-case reponse time analysis of real-time sporadic traffic in fip networks, Euromicro Workshop on Real-Time Systems (1997) .

[45] Ermedahl, H. Hansson, S. M, Response time guarantees in atm networks, proc. Real-Time Systems Symposium (1998) .

[46] N. Audsley, A. Grigg, Timing analysis of the arinc 629 databuses for real-time applications, proc. ERA Avionics Conference (1996) .

[47] K. Tindell, J. Clark, Holistic schedulability analysis for distributed real-time systems, Microprocessors and Microprogramming (1994) .

[48] P. Richard, M. Richard, F. Cottet, Analyse holistique des systèmes temps réel distribués: principes et algorithmes, in: ordonnancement des systèmes distribués, Hermès (2003) .

[49] M. Sjödin, H. Hansson, Improved response time analysis calculations, Real-Time Systems Symposium (1998) .

[50] I. Bate, A. Burns, Investigation of the pessimism in distributed systems timing analysis, Proceedings of the 10th Euromicro Workshop on Real Time Systems (1998) 107–114.

[51] M. Richard, Contribution à la validation des systèmes temps réel distribués : ordonnancement à priorités fixes et placement, Thèse de Doctorat, Université de Poitiers, 2002.

[52] T.-P. S. Inc., Real-time scheduling solutions, http://www.tricpac.com (2003) .

[53] J. Liu, et al, Perts: A prototyping environment for real-time systems, proc. Real-Time Systems Symposium (1993) .

[54] TimeSys, Real-time real solutions, http://www.timesys.com (2003) .

[55] S. Tregar, A graphical tool for rate monotonic analysis, http://www.tregar.com/gRMA/ (2003) .

[56] Y. Meylan, A. Bajpai, R. Bettati, Protex: A toolkit for the analysis of distributed real-time systems, proc. Real-Times Computer Systems and Applications (RTCSA'00) (2000) .

[57] J. Migge, Tkrts: A tool for computing response time bounds, http://www.migge.net/jorn/rts/ (1999) .