Laboratoire d'Informatique Scientifique et Industrielle
EA 1232
ENSMA et Université de Poitiers

Ordonnancement temps réel en-ligne : contraintes, conception et analyses

Habilitation à diriger des recherches
Synthèse des travaux

Pascal Richard
Maître de Conférences à l'IUT de Poitiers

16 juin 2004

Composition du jury :

Pr. Claude Kaiser	Rapporteur	Cedric/CNAM, Paris
Pr. Yvon Trinquet	Rapporteur	IRCCyN, Nantes
Pr. Zoubir Mammeri	Rapporteur	IRIT, Toulouse
Pr. Christian Proust	Examinateur	LI, Tours
Pr. Jacques Carlier	Examinateur	Heudyasic, Compiègne
Pr. Francis Cottet	Directeur	LISI, Poitiers

à mes petites fées, Clara et Lisa

Remerciements

Je tiens à remercier les membres du jury pour l'intérêt qu'ils ont porté à mes travaux de recherche. Les échanges et collaborations que j'ai pu avoir avec eux m'ont fortement incité à appliquer les résultats théoriques sur des systèmes *réalistes*.

Je tiens à remercier Michaël Richard, qui a travaillé durant sa thèse avec moi, pour l'ensemble de son travail. Une partie importante des contributions présentées dans ce mémoire proviennent de sa thèse. Cette première expérience d'encadrement de thèse a été un moteur dans mon travail et une joie quotidienne. Mille merci ! L'expérience se répète à nouveau avec Frédéric Ridouard, actuellement en thèse. La qualité de son travail durant son stage de DEA et en début de thèse m'apporte quotidiennement les mêmes sources de motivation.

Je tiens à remercier Francis Cottet pour le temps qu'il m'a consacré à mon arrivée dans son équipe de recherche. Son soutien et sa disponibilité permanente ont été une aide précieuse. Il m'a immédiatement guidé dans ma reconversion thématique et en a facilité la réalisation. Sur ce point, je tiens aussi à remercier Emmanuel Grolleau, mon collègue de bureau au LISI, pour nos nombreuses et fructueuses discussions sur l'ordonnancement temps réel. Je remercie aussi l'ensemble des membres du LISI pour la convivialité des relations de travail.

Je tiens à remercier aussi mes anciens collègues du Laboratoire d'Informatique de Tours. La formation rigoureuse par la recherche que j'y ai reçue fonde aujourd'hui, et certainement pour toute ma carrière, le ciment de mon travail actuel et à venir.

Je tiens à remercier mes collègues de l'IUT pour l'ambiance exceptionnelle de travail. Le bon fonctionnement de notre département Gestion des Entreprises et Administrations accueillant plus de 350 étudiants est supporté par une équipe pédagogique soudée de personnels techniques et administratifs, d'enseignants et de professionnels. Sans cette implication collective au service de l'enseignement, je n'aurais pas pu mener de front mes activités d'enseignement et de recherche. Pour terminer, je remercie mes étudiants qui me rappellent à chaque instant que la rigueur pédagogique et la rigueur scientifique reposent sur les mêmes exigences et constituent l'essence du métier d'enseignant-chercheur.

Table des matières

Introduction

Ce document présente la synthèse des travaux de recherche depuis ma nomination aux fonctions de maître de conférences en informatique (section CNU 27) en 1998 à l'IUT de Poitiers, département Gestion des Entreprises et Administrations. Dès mon recrutement, j'ai été intégré à l'équipe de recherche en *informatique temps réel* du Laboratoire d'Informatique Scientifique et Industrielle (LISI), localisé dans l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique (ENSMA).

L'équipe *temps réel* était alors composée de trois chercheurs permanents, qui développaient deux axes de recherche : d'une part, l'ordonnancement en-ligne (Francis Cottet), et d'autre part, l'ordonnancement hors-ligne reposant sur des modèles formels (Annie et Dominique Geniet). J'ai été recruté pour renforcer le thème ordonnancement en-ligne. Ma thèse, passée à l'Université de Tours en 1997, traitait des réseaux de Petri et de planification de système de production. Mon insertion au sein de l'équipe temps réel a donc nécessité une reconversion thématique. Dans les années suivantes, j'ai eu l'occasion d'encadrer une thèse (Michaël Richard, maintenant maître de conférences à l'ENSMA), deux stages de DEA (Frédéric Ridouard et Stéphane Jeannenot) et de débuter l'encadrement d'une nouvelle thèse en septembre 2003 (Frédéric Ridouard). A travers ces activités en informatique temps réel, j'ai été amené à collaborer avec des chercheurs d'autres laboratoires : Claude Kaiser (Cedric/CNAM) et Joël Goossens (Université Libre de Bruxelles). Ces travaux seront présentés dans la première partie.

J'ai aussi eu l'occasion de travailler sur d'autres problèmes de recherche liés aux systèmes de production (contrat de recherche sur l'ordonnancement des activités des laboratoires pharmaceutiques - projet iW4L) et sur les réseaux de Petri. L'ensemble de ces travaux m'a amené à collaborer avec des membres extérieurs de mon laboratoire : C. Haro (Laboratoire d'Informatique, Tours) sur les réseaux de Petri, Patrick Martineau sur l'ordonnancement en-ligne de système de production (Laboratoire d'Informatique, Tours). Ces travaux seront présentés dans la seconde partie.

La troisième partie présente les perspectives de recherche en informatique temps réel.

Les références bibliographiques en italique dans le texte indiquent les diffusions des résultats effectuées sur les travaux présentées dans ce mémoire.

Itinéraire de recherche

Ce mémoire présente des travaux sur l'ordonnancement temps réel, d'une part, et sur l'ordonnancement dans les laboratoires pharmaceutiques et sur les réseaux de Petri, d'autre part. La première partie est directement liée à l'activité de l'équipe de recherche à laquelle j'appartiens, tandis que la seconde s'est principalement déroulée sur le contrat de recherche iW4L. Nous synthétisons les résultats obtenus dans ces deux parties, qui seront présentés dans le reste de ce mémoire.

Ordonnancement temps réel

Nos contributions portent sur l'ordonnancement en-ligne de systèmes temps réel avec des contraintes temporelles strictes. Elles s'organisent selon trois axes :

- l'ordonnancement de tâches dépendantes,
- l'ordonnancement dans les systèmes distribués,
- l'aide à la conception et au paramétrage d'application temps réel.

Ordonnancement de tâches dépendantes

L'analyse des logiciels temps réel conduit généralement à décomposer les tâches en sous-tâches dépendantes. Chacune de ces sous-tâches constitue une portion de code sans synchronisation. Une sous-tâche n'utilise les primitives de synchronisation du noyau temps réel qu'à son début et sa fin d'exécution. Une sous-tâche ne peut pas se suspendre durant une opération d'entrée/sortie par exemple. Une telle décomposition facilite l'analyse de l'ordonnançabilité du système de tâches. Nous avons présenté trois contributions sur ce thème.

Nous avons tout d'abord étudié l'ordonnancement de tâches dépendantes qui s'exécutent avec des fréquences d'activations différentes. Les relations de précédence sont alors dites généralisées. Nous avons montré qu'un tel problème pouvait se résoudre en dépliant le graphe de précédence généralisée en un graphe de précédence simple, où toutes les tâches dépendantes s'exécutent avec la même période d'activation. L'algorithme correspondant est exponentiel, mais nous avons montré qu'il devenait pseudo-polynomial en considérant que le ppcm des périodes des tâches originelles est une entrée de l'algorithme de transformation du graphe de précédence. La méthode correspondante est indépendante de la plateforme d'exécution (systèmes monoprocesseurs, multiprocesseurs ou distribués), des algorithmes d'ordonnancement (en-ligne ou hors-ligne) et des techniques de validation qui leurs correspondent. Du point de vue de l'ordonnançabilité du système, la configuration de tâches résultant du processus de dépliage est équivalente à la configuration de tâches en entrée de la méthode. Nous avons appliqué cette méthode avec succès

à un cas industriel (suivi temps réel d'un laminoir d'aluminium), ainsi qu'à un système distribué. L'unique problème ouvert est la complexité du problème d'ordonnançabilité en présence de contraintes de précédence généralisée.

Nous avons étudié ensuite l'ordonnancement de tâches à priorité fixe s'exécutant sur un processeur. Les tâches en dépendance peuvent avoir des priorités quelconques. Dans ce contexte, nous avons montré que des anomalies d'ordonnancement peuvent survenir lorsqu'un algorithme d'ordonnancement en-ligne à priorité fixe est utilisé. Nous avons présenté un test d'ordonnançabilité qui repose sur le calcul des pires temps de réponse des tâches. La complexité algorithmique de ce test est pseudo-polynomial, mais la complexité du problème d'ordonnançabilité n'est pas connue.

Nous avons enfin étudié le problème d'ordonnancement de tâches qui peuvent se suspendre durant leur exécution. Nous avons montré que le problème d'ordonnancement devenait alors $\mathcal{NP}$-Difficile au sens fort dans de nombreux cas. La conséquence directe est que les algorithmes d'ordonnancement utilisés, tels que l'ordonnancement par priorité fixe ou selon l'échéance la plus proche sont totalement inefficaces pour affronter ce type de contrainte.

Ordonnancement à priorité fixe dans les systèmes distribués

Ces travaux font l'objet de la thèse de Mr Michaël Richard, soutenue en décembre 2002. Nous avons étudié le problème d'ordonnancement de systèmes temps réel distribué à priorité fixe des tâches et des messages. Le contexte industriel de cette étude était les systèmes informatiques embarqués dans une automobile, dans le cadre du 12ème contrat de plan Etat-Région. Nous avons effectué principalement deux contributions.

Tout d'abord, nous avons étudié le problème d'affectation des priorités aux tâches en connaissant initialement le placement des tâches sur les différents processeurs du système. Nous avons proposé une méthode arborescente, reposant sur les principes d'une procédure par séparation et évaluation. La méthode a été expérimentée sur des configurations de tâches analogues à celles embarquées dans une automobile.

Dans un deuxième temps, nous avons étendu la problématique au placement des tâches sur des ensembles de processeurs identiques. L'architecture matérielle peut donc être constituée de groupes de processeurs identiques, appelés pools. Chaque tâche est pré-affectée à un pool, mais pas à un processeur du pool. L'affectation de la priorité de la tâche et son placement sur un processeur de son pool est conduit conjointement dans la méthode proposée. A notre connaissance, notre méthode est la seule considérant simultanément le placement et l'affectation des priorités aux tâches. L'intérêt de coupler ces deux problèmes est d'augmenter le taux d'utilisation des ressources (processeurs et réseaux). Les expérimentations numériques qui ont été menées montrent que la méthode peut être appliquée avec succès sur des applications industrielles.

Aide à la conception et au paramétrage des applications temps réel

Le choix des paramètres des tâches est le problème central en ordonnancement temps réel. Nous avons conduit trois études différentes dans le contexte de systèmes monoprocesseurs :

- une méthode graphique pour définir les paramètres temporels d'une tâche tout en garan-

tissant l'ordonnançabilité du système,
- une méthode de calcul des priorités des tâches pour optimiser un critère de Qualité de Service,
- des méthodes de détermination de la fréquence de travail d'un processeur à vitesse variable afin de minimiser l'énergie consommée dans un système embarqué autonome.

La méthode graphique proposée pour choisir les paramètres temporels d'une tâche repose sur l'énumération des valeurs C_i,D_i,T_i possibles pour garantir l'ordonnançabilité d'une tâche dans un système. Bien sûr, l'énumération complète est fortement combinatoire puisqu'un test d'ordonnançabilité est nécessaire pour chaque solution possible. Nous avons proposé des règles simples pour propager l'analyse d'un point aux solutions adjacentes afin de limiter l'explosion combinatoire.

Nous avons ensuite présenté une méthode d'affectation des priorités fixes aux tâches afin d'optimiser des critères de qualité de service avec un ordonnanceur en-ligne. La méthode repose sur une procédure par séparation et évaluation possédant des parties génériques et des parties spécifiques devant être spécialisées pour chaque nouveau critère considéré. Des expérimentations numériques montrent que la méthode permet d'atteindre les performances optimales pour des configurations de tâches avec une vingtaine de tâches.

La dernière étude porte sur l'ordonnancement de processeur à vitesse variable. Ces processeurs permettent de réduire dynamiquement leur vitesse de fonctionnement pour minimiser la consommation d'énergie. Nous avons considéré les processeurs possédant un nombre discret de paliers de vitesse. Nous avons proposé deux heuristiques simples et une plus complexe : un recuit simulé. Ces trois algorithmes ont été comparés dans le cadre d'une expérimentation numérique.

Ordonnancement dans les laboratoires pharmaceutiques

Dans le cadre du projet iW4L (Innovative Workload for Laboratory), nous avons étudié l'ordonnancement en ligne des analyses d'échantillons. L'objectif est de concevoir un logiciel de planification des analyses. Nous avons conduit une étude théorique sur l'ordonnancement en-ligne des machines à traitement par lot et nous avons proposé une méthode d'aide à la décision pour planifier les activités du laboratoire.

Nous avons étudié l'ordonnancement en-ligne des machines à traitement par lot en parallèle. Plusieurs échantillons sont placés dans un même mot lot et la durée de traitement du lot est la plus longue analyse d'un échantillon du lot (traitements en parallèle des analyses des échantillons du lot). Nous avons proposé un ensemble d'algorithmes en-ligne garantissant les meilleurs performances possibles vis à vis de l'analyse de compétitivité (avec un algorithme hors-ligne optimal). L'intérêt de ces études théoriques est de montrer qu'un délai d'attente doit être introduit dans le planning des analyses.

Nous avons proposé une méthode d'aide à la décision pour planifier les analyses sur les différentes ressources du laboratoire. La méthode repose sur un algorithme par séparation et évaluation avec un branchement chronologique des activités au sein de l'arbre des solutions. Ce module du logiciel est appelé le moteur de planification. Le décideur envoie les unes après les autres les

tâches complexes à planifier automatiquement par le moteur de planification. Le décideur choisit aussi les délais d'attente avant le lancement d'un lot dans une machine à traitement par lot, si celui-ci n'est pas complet.

Les réseaux de Petri

Nous avons publié deux articles de synthèse sur la modélisation et l'analyse de systèmes par réseaux de Petri. Le premier est lié à la modélisation d'un programme informatique, le second à la modélisation et l'analyse de systèmes industriels.

Nous avons aussi étudié le problème de détermination de la plus courte séquence de tirs pour atteindre un marquage. Nous avons obtenu des résultats pour les machines à état et les graphes d'événements vivants. La méthode se décompose en deux étapes : calcul du vecteur caractéristique optimal (par des algorithmes efficaces reposant sur la théorie des graphes), et la seconde étape consiste à construire le graphe des marquages en se limitant aux solutions respectant le vecteur caractéristique calculé dans la première étape et des conditions de dominances. Dans le cas des graphes d'événements vivants, nous avons de plus introduit une nouvelle condition de dominance qui repose sur la répartition équilibrée des tirs des transitions au sein d'une séquence de tirs. Ce type de séquence permet d'atteindre tout marquage accessible dans un graphe d'événements vivant.

Première partie

Contributions à l'ordonnancement temps réel

Chapitre 1

Problématique

1.1 Résumé

Ce chapitre présente la problématique de l'ordonnancement temps réel. L'objectif est ici de présenter uniquement les notions de base nécessaires à la compréhension des chapitres suivants. Nous définissons les caractéristiques générales des systèmes temps réel, les mécanismes d'attribution des priorités aux tâches, ainsi que les principales techniques de validation. Pour terminer, nous détaillons le plan de la partie *Contributions à l'ordonnancement temps réel.*

1.2 Introduction

L'utilisation d'ordinateur pour contrôler des systèmes aux fonctions critiques sont en constante augmentation. Le fonctionnement du système ne dépend pas uniquement de la correction des résultats mais aussi des dates où ils sont produits. Cette double exigence caractérise les *systèmes temps réel.* Le système informatique de commande doit être *réactif* vis-à-vis du procédé contrôlé. Le logiciel de commande est généralement décomposé en tâches s'exécutant en parallèle sur des processeurs avec la *préemption* autorisée. Avant la mise en service de ce logiciel, les concepteurs doivent s'assurer que les contraintes temporelles des tâches seront respectées durant toute la vie de l'application. Lorsqu'une tâche ne respecte pas une contrainte de temps, il y a une *faute temporelle.* Montrer qu'il n'existera jamais de telles fautes durant toute la vie de l'application est appelée la *validation* [56].

L'exécution des tâches est supportée par un *noyau temps réel.* Le noyau fournit les primitives de base de gestion des processus associés aux tâches, ainsi que les primitives de synchronisation. Il met aussi en œuvre les routines d'ordonnancement des tâches. Chaque appel du noyau possède un temps de réponse connu. Chaque tâche s'exécute ainsi sur une machine virtuelle ayant un comportement *prédictible* [14, 99]. De la même façon, les durées d'exécution des tâches doivent aussi être prédictibles. Une analyse dépendant simultanément de la plateforme d'exécution et du logiciel doit être menée afin de calculer les pires durées d'exécution des tâches (WCET : Worst-Case Execution Time) [70].

Les tâches critiques sont généralement des *tâches périodiques.* Ces tâches réalisent périodiquement des acquisitions de données via des capteurs et envoient leurs consignes au procédé contrôlé via des actionneurs. Le fonctionnement périodique permet de définir des *états* du système et

évite de propager des *événements*, dont la perte pourrait être fatale pour le système. Toutefois, certaines tâches sont par nature totalement *apériodiques*, comme par exemple des alarmes ou *sporadiques*, lorsque les périodes d'activation de la tâche sont soumises à des variations.

L'ordonnancement des tâches fixe l'ordre de leurs exécutions sur chaque processeur. Lorsque le système est distribué, il convient aussi de définir l'ordre de transmission des messages en accord avec le protocole d'accès au médium de communication. Dans un système totalement prédictible, cet ordre d'attribution des ressources peut être calculé *hors-ligne*. Une table de séquencement des tâches mémorise alors les intervalles d'activation des tâches. L'ordonnancement est dit *dirigé par le temps ou hors-ligne*. Historiquement, cette approche a été la première utilisée pour définir l'ordonnancement des tâches dans les systèmes critiques, et reste encore très utilisée dans certains milieux industriels comme dans l'industrie aéronautique [103]. Les noyaux temps réel fournissent des ordonnanceurs fondés sur des priorités. Chaque tâche se voit attribuée une priorité en-ligne, fixe ou pouvant varier dynamiquement en fonction de la criticité de la tâche dans le temps. L'ordonnancement est alors *dirigé par les priorités ou en-ligne*. La validation de l'application nécessite donc de tenir compte du comportement de l'ordonnanceur.

Dans un ordonnancement dirigé par les priorités, à chaque instant, la tâche la plus prioritaire s'exécute. L'ordonnancement est dit *au plus tôt*, car s'il existe une tâche prête à s'exécuter alors celle-ci obtient le processeur sans attente. L'ordonnancement en-ligne soulève des problèmes d'*instabilité* d'une part, et l'*inversion de priorité* d'autre part. Les durées d'exécution des tâches ne sont pas connues a priori, car le concepteur ne peut déterminer que leurs pires durées. Les durées d'exécution d'une tâche périodique varie donc d'une exécution à une autre. Graham a montré, dans un environnement multiprocesseur, que réduire la durée d'une tâche peut augmenter le temps de réponse d'une autre tâche [32]. Réduire la durée d'une tâche peut donc conduire à bouleverser l'ordonnancement de façon suffisamment importante pour conduire une autre tâche à faire une faute temporelle. L'inversion de priorité peut survenir lorsque les tâches partagent des ressources en exclusion mutuelle. L'inversion de priorité survient dans le scénario suivant : une tâche τ_a de faible priorité détient une ressource, bloquant l'accès d'une tâche τ_b plus prioritaire que τ_a ; une tâche τ_c se réveille alors avec une priorité comprise entre celles de τ_a et τ_b ; par le jeu des priorités, τ_a est préemptée et τ_c s'exécute jusqu'à son terme et cela malgré l'existence d'une tâche τ_b qui a une plus forte priorité dans le système (mais bloquée par une tâche de faible priorité détenant une ressource en exclusion mutuelle). Les noyaux temps réel fournissent des protocoles d'accès aux ressources afin de supprimer ce phénomène et en garantissant souvent de plus l'absence d'interblocage.

Dans la suite, nous nous intéresserons qu'à l'ordonnancement en-ligne de tâches périodiques. Nous définirons précisément les tâches et leurs contraintes temporelles, puis les principes d'attribution des priorités. Nous présenterons ensuite les techniques d'analyse permettant de valider le logiciel temps réel. Nous terminerons par la présentation des contributions qui seront présentées dans les chapitres qui suivent.

1.3 Ordonnancement en-ligne

Nous définissons les systèmes de tâches qui seront considérés dans la suite, puis les principales règles d'atttribution des priorités aux tâches.

1.3.1 Systèmes de tâches

Une *tâche périodique*, τ_i est caractérisée par une pire durée d'exécution C_i, et une période T_i entre deux *réveils* successifs [55]. Les réveils suivent donc la suite $0,T_i,2T_i,...$ Chaque exécution de τ_i est appelée une *instance*. La contrainte temporelle associée à une telle tâche est que chaque instance doit avoir terminée son exécution avant que la suivante n'arrive dans le système (i.e, avant le prochain réveil de la même tâche). L'échéance d'une instance coïncide avec la date de réveil de l'instance suivante. Pour cette raison, cette tâche est dite à *échéance sur requête*.

Lorsque les échéances et les réveils d'une tâche ne coïncident pas, l'échéance d'une instance est définie par rapport à sa date de réveil par un intervalle de temps D_i. L'instance réveillée à la date $kT_i, \quad k \geq 0$, doit impérativement se terminée avant la date $kT_i + D_i$. L'échéance D_i est appelée une *échéance relative* au réveil de la tâche (ou *délai critique*). Deux cas doivent être distingués suivant que l'échéance peut être ou non plus grande que la période de la tâche. Si $D_i \leq T_i$, la tâche est à *échéance contrainte*. Sinon l'échéance peut être quelconque par rapport à la période, et la tâche est alors à *échéance non reliée* à la période [5].

Dans tout ce qui précède, les tâches sont supposées être réveillées simultanément, au début de l'application. Elles sont *à démarrage simultané*. Si les tâches ne sont pas réveillées la première fois au même instant, elles sont dites à *départ différé*. Chaque tâche τ_i possède alors une date d'arrivée dans le système, définie par le paramètre r_i.

Jusqu'à maintenant, les tâches sont indépendantes les unes des autres et ne sont pas autorisées à se suspendre (autrement que par le mécanisme de préemption par une autre tâche plus prioritaire). En pratique, les tâches peuvent être soumises à diverses contraintes. Ces contraintes sont souvent désignées sous le terme de *facteurs pratiques*:

- contraintes de précédence : deux tâches τ_i et τ_j se synchronisent de façon à ce que chaque instance de τ_j soit précédée par une instance de τ_i. Nous noterons la précédence entre ces tâches de la façon suivante : $\tau_i \prec \tau_j$.
- partage de ressource : des tâches entrent en compétition pour accéder à des ressources. Les protocoles d'accès aux ressources permettent de définir la durée maximum de bloquage dans l'attente d'une ressource. Cette durée, notée usuellement B_i dans la littérature, est aussi appelée un *facteur de blocage*.
- suspension des tâches : une tâche peut libérer le processeur durant son exécution pendant un intervalle de temps durant une opération d'entrée/sortie.
- non préemption : une tâche peut interdire temporairement la préemption pour réaliser une opération critique sans recourir à un protocole d'accès aux ressources.

1.3.2 Attribution des priorités

Nous distinguons ci-après les tâches à priorité fixe des tâches à priorité variable. Dans le premier cas, le concepteur de l'application est confronté à un problème d'ordonnancement en choisissant une priorité pour chaque tâche. Par contre, dans le second cas, l'unique choix du concepteur est le choix de la méthode d'ordonnancement (i.e., l'attribution des priorités est donnée par la règle elle-même).

Priorités fixes

Les noyaux temps réel fournissent en standard un ordonnanceur à *priorité fixe*. Chaque tâche se voit attribuer une priorité qui sera fixe durant toute la vie de l'application. Chaque tâche possède une priorité distincte. Le concepteur de l'application se voit confronter à un problème d'ordonnancement puisqu'il doit choisir une priorité pour chaque tâche de telle façon que durant son exécution ses échéances seront respectées. Ce problème est combinatoire puisqu'en monoprocesseur si le système comporte n tâches, alors il existe $n!$ façons différentes d'attribuer les priorités aux tâches.

Contrairement au cas multiprocesseur ou distribué, ce problème combinatoire admet des solutions simples en monoprocesseur. Les méthodes présentées ci-après sont *optimales* dans la classe des ordonnancements à priorité fixe, dans le sens où s'il existe une affectation de priorités conduisant à respecter les échéances, alors elles le seront par ces méthodes. Lorsque les tâches sont à démarrage simultané, les méthodes suivantes permettent d'attribuer les priorités :

- tâches à échéance sur requête : les priorités des tâches seront inversement proportionnelles aux périodes. Cet algorithme de définition des priorités est appelé *Rate Monotonic* [55].
- tâches à échéances contraintes : les priorités des tâches seront inversement proportionnelles aux échéances relatives. Cet algorithme de définition des priorités est appelé *Deadline Monotonic* [54]. Ce résultat tient aussi si les tâches sont sporadiques.

Pour les tâches à départ différé, la règle Deadline Monotonic n'est pas optimale [54], mais il a été démontré dans [2] qu'il est nécessaire de considérer au plus $(n^2 + n)/2$ affectations de priorités. Toutefois, l'affectation de chaque priorité est assujettie à un test d'ordonnançabilité. Nous verrons plus loin que ce problème est complexe dans le cas de tâches à départ différé.

Le paragraphe suivant décrit deux algorithmes d'ordonnancement à priorité dynamique qui choisissent sans l'aide du concepteur la tâche la plus prioritaire à chaque instant.

Priorités variables

Afin de respecter les échéances des tâches, une méthode optimale consiste à donner la plus forte priorité à l'instance (d'une tâche) ayant l'échéance la plus proche. Cet algorithme est connu sous le nom d'EDF (Earliest Deadline First). Si plusieurs échéances de tâches coïncident, l'une d'entre elle est choisie arbitrairement. Ainsi, la priorité d'une tâche varie d'une instance à l'autre. Mais, la priorité d'une instance de tâche ne varie pas durant son exécution.

Une autre façon d'affecter optimalement les priorités aux instances de tâches, consiste à donner le processeur à la tâche ayant la laxité la plus petite. Soit $l_i(t)$ la durée restant à exécuter à l'instance courante de τ_i à la date t devant se terminer avant la date d_i, la *laxité* de cette instance de τ_i à l'instant t est $d_i - l_i(t)$. Cet algorithme est connu sous le nom de LLF (Least Laxity First). Contrairement à EDF, une instance de tâche peut voir sa priorité changer durant son exécution. Cette méthode d'affectation des priorités n'est généralement pas utilisée car elle engendre beaucoup plus de préemptions qu'EDF.

Les priorités étant maintenant définies, il est nécessaire de valider l'application temps réel.

Problèmes	Complexité	Références
$r_i = 0, D_i = T_i$	$\mathcal{P}$	[55]
$r_i = 0, D_i \neq T_i$	ouverte	
$r_i \neq 0$	co-$\mathcal{NP}$-Complet au sens fort	[8]
ressources	$\mathcal{NP}$-Complet au sens fort	[60]
non-préemption	$\mathcal{NP}$-Complet au sens fort	[43]
suspension	$\mathcal{NP}$-Complet au sens fort	*[84]*

TAB. 1.1 – *Complexité du problème d'ordonnançabilité.*

1.4 Techniques d'analyse d'ordonnançabilité

La validation des systèmes temps réel revient à montrer que toutes les tâches respecteront leurs échéances. Très souvent les méthodes et algorithmes sont désignés sous le terme de *test d'ordonnançabilité ou de faisabilité*. Ce problème de décision est souvent très complexe à résoudre comme l'illustre le tableau 1.1 qui donne les complexités des problèmes de base.

Trois techniques d'analyse existent et reposent sur :

- la construction de l'ordonnancement sur une intervalle de temps fini [53].
- l'analyse du facteur d'utilisation du processeur (*Processor Utilization Analysis*) [55, 35].
- l'analyse de la demande processeur (*Processor Demand Analysis*) [51, 44, 4].

1.4.1 Construction de l'ordonnancement

Cette première technique de validation tire partie de la périodicité de l'ordonnancement. Les tâches étant périodiques et toujours exécutées au plus tôt par l'ordonnanceur, l'ordonnancement devient périodique au bout d'un temps fini. Dans le cas d'un système monoprocesseur et de tâches à départ différé, l'ordonnancement devient au plus tard périodique à la date $\max_{i=1..n}(r_i) + 2 \times ppcm_{i=1..n}(T_i)$ [53]. L'entrée en régime périodique de l'ordonnancement a été plus finement étudiée dans [30]. Ces résultats permettent de définir un intervalle de temps, appelé *intervalle de faisabilité*, où l'on peut se limiter à rechercher les fautes temporelles. La construction de l'ordonnancement, par simulation, sur l'intervalle de faisabilité permet de vérifier si toutes les échéances sont respectées.

Pour de nombreuses configurations de tâches, des tests d'ordonnançabilité ont été proposés conduisant à des temps calculs d'une complexité plus faible. L'extension de ces résultats à des systèmes multiprocesseurs ou distribués ne semble pas être un problème trivial.

1.4.2 Analyse du facteur d'utilisation

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 Rate Monotonic sera valide si $U \leq n(2^{\frac{1}{n}} - 1)$ et un ordonnancement EDF est faisable si, et seulement si : $U \leq 1$ [55]. Ce type de tests conduit généralement à des conditions suffisantes d'ordonnançabilité.

Le facteur d'utilisation est très utile pour les tâches à échéance sur requête. Toutefois, son intérêt devient très limité lorsque les échéances sont différentes des périodes. Dans ce contexte, des tests polynomiaux tenant compte de nombreux facteurs pratiques ont été proposés [25].

1.4.3 Analyse de la demande processeur

L'analyse de la demande processeur 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 [51, 44, 4] en limitant l'étude d'ordonnançabilité à quelques instants d'une période d'activité du processeur.

Cette approche permet de calculer aussi les *pires temps de réponse* des tâches, notés dans la suite R_i [45, 52]. Cette analyse détermine les plus grands temps de réponse de chaque tâche. Dans le cadre de l'ordonnancement à priorité fixe avec la méthode Rate Monotonic, l'analyse est désignée sous le nom d'analyse RMA (*Rate Monotonic Analysis*) [48]. 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 performance (temps de réponse moyen, encadrement de la gigue de sortie...).

Lorsque les tâches sont à démarrage simultané, la demande processeur se définit comme la somme cumulée des durées d'exécution des tâches réveillées depuis l'instant initial. Afin de déterminer le temps de réponse d'une tâche τ_i, à priorité fixe ($D_i \leq T_i$), il n'est nécessaire de tenir compte que des tâches plus prioritaires que τ_i, ainsi qu'une exécution de τ_i. La plus longue *période d'activité* du processeur où celui-ci n'exécute que des tâches plus prioritaires que τ_i va définir la plus grande *interférence* (attente liée à l'exécution de tâches plus prioritaires) que subira τ_i. Cette longueur sera maximale lorsque toutes ces tâches seront réveillées simultanément [55]. Nous supposons que les tâches sont triées dans l'ordre de leurs priorités. Ainsi, la tâche τ_1 est la plus prioritaire et τ_n est la moins prioritaire. La fonction de la demande processeur des tâches de priorité supérieure ou égale à i, dans l'intervalle de temps [0,t[est donnée par [45, 52]:

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \tag{1.1}$$

Lorsque $W_i(t) = t$ alors toutes les tâches réveillées dans l'intervalle [0,t[sont terminées. La plus petite valeur de t telle que $W_i(t) = t$ est la longueur de la période d'activité de niveau i (i.e., composée de tâches de priorité supérieure ou égale à i). Cela définit en conséquence le pire temps de réponse de la tâche τ_i. D'un point de vue pratique, calculer le plus petit point fixe de l'équation (1.1) peut se faire de façon itérative en calculant la suite qui converge vers le plus petit point fixe. La suite est initialisée par une borne inférieure du temps de réponse de τ_i : chaque tâche s'exécutera au moins une fois dans la période d'activité de niveau i :

$$\begin{aligned} W_i(t^{(0)}) &= \sum_{j=1}^{i} C_j \\ W_i(t^{(k+1)}) &= C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t^{(k)}}{T_j} \right\rceil C_j \end{aligned}$$

Le pire temps de réponse de τ_i est calculé pour la plus petite valeur entière k telle que :

$$R_i = W_i(t^{(k)}) = W_i(t^{(k+1)})$$

L'algorithme de calcul correspondant à la recherche du plus petit point fixe de l'équation $W(t) = t$ s'effectue en temps pseudo-polynomial : $O(n \sum C_i)$. A notre connaissance, la complexité du problème est toujours ouverte.
Lorsque des facteurs pratiques sont intégrés à l'analyse du temps de réponse, de nombreuses méthodes de calcul du pire temps de réponse ne conduisent pas à sa valeur exacte, mais à une borne supérieure. 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, $R_i \leq D_i$. Sinon cette comparaison conduit 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.

L'analyse du temps de réponse des systèmes distribués peut se ramener à l'analyse de systèmes monoprocesseurs indépendants. La dépendance entre le réveil d'une tâche et l'instant où elle peut s'activer, ses messages en entrée étant délivrés, est appelé la *gigue sur activation*. L'analyse du temps de réponse reposant sur ce principe est *l'analyse holistique* [97].

La figure 1.1 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.

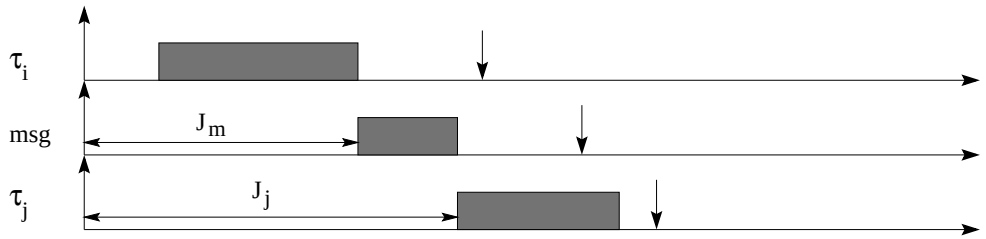


FIG. 1.1 – *La gigue modélise l'attente due à la fin de la tâche précédente.*

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. Soit Γ_i^- la liste des prédécesseurs de la tâche ou du message numéro i, nous obtenons ainsi les équations suivantes :

$$\begin{array}{ll} \text{Pour les tâches :} & J_i = \max\limits_{j \in \Gamma_i^-} (R_j) \\ \text{Pour les messages:} & J_m = \max\limits_{j \in \Gamma_i^-} (R_j) \end{array} \tag{1.2}$$

Toute modification des gigues va entraîner des modifications des temps de réponse des tâches dépendantes ou ceux des tâches ordonnancées sur le même processeur. Le principe de l'analyse

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

$$1 \leq i \leq n \qquad \left\{ \begin{array}{l} \left\{ \begin{array}{lcl} R_i^{(0)} & = & C_i \\ J_i^{(0)} & = & 0 \\ R_i^{(k)} & = & ResponseTime\left(J_i^{(k-1)}\right) \\ J_i^{(k)} & = & \max\limits_{j \in \Gamma_i^-}\left(R_j^{(k)}\right) \end{array} \right. \end{array} \right. \tag{1.3}$$

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

$$R_i = R_i^{(k)} = R_i^{(k-1)}, 1 \leq \ i \leq n \tag{1.4}$$

Les modèles de tâches considérés dans les techniques énoncées ci-dessus sont généralement simplistes au regard des applications industrielles. La simulation est alors souvent utilisée 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, de même 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 déduites d'analyses statistiques du système. Cette approche est valide si, et seulement si, ces lois fondées sur l'analyse du passé représentent correctement toutes les évolutions possibles du système. Cette hypothèse n'est pas vérifiée pour de nombreux systèmes temps réel qui traitent en-ligne des événements.

Nous avons présenté deux articles de synthèse sur ce sujet : une synthèse sur le calcul du temps de réponse dans les systèmes temps réel *[83]* et une sur l'analyse du temps de réponse dans les systèmes distribués *[89]*.

1.5 Contributions

Dans les chapitres suivants, nous présentons nos contributions à l'ordonnancement temps réel :

Le chapitre 2 présente nos résultats sur les tâches dépendantes. Nous étudions les tâches en précédence fonctionnant à des périodes différentes. Ce type de précédence généralise les contraintes de précédence, et sont appelées *précédences généralisées*. Puis, nous présentons l'analyse du temps de réponse pour les tâches à priorité fixe, sur un processeur. Et, enfin nous présentons les résultats de complexité sur l'ordonnancement de tâches avec suspensions.

Le chapitre 3 présente nos résultats sur l'affectation des priorités dans les systèmes distribués à priorité fixe. Nous étudions dans un premier temps l'affectation des priorités seules, puis

l'affectation des priorités conjointement avec le placement des tâches. Nous présentons ensuite une application dans le cas des systèmes embarqués dans l'automobile.

Le chapitre 4 étend la problématique de l'ordonnançabilité à l'aide à la conception d'applications temps réel. Nous présentons une méthode graphique définissant l'ordonnançabilité des tâches en fonction de leurs paramètres temporels. Dans un deuxième temps, nous présentons une méthode d'affectation des priorités fixes aux tâches en vue d'optimiser des critères de Qualité de Service (QdS). Enfin, nous traitons de l'ordonnancement de tâches avec contrainte d'énergie sur des systèmes monoprocesseurs à vitesse variable. Dans ces systèmes, le concepteur doit définir l'ordre de passage et la vitesse de chaque tâche à chaque instant.

Chapitre 2

Tâches dépendantes

2.1 Résumé

2.1.1 Thématique

Ce chapitre présente trois contributions à l'ordonnancement de tâches temps réel dépendantes. Nous donnons pour chacune de ces contributions les collaborations extérieures et les communications et publications associées dans le paragraphe diffusion des résultats. Dans un souci de concision, nous ne donnons aucune des preuves des différents résultats. Elles pourront être consultées dans les articles associés. La suite du chapitre présente les contributions suivantes :

- ordonnancement sous contraintes de précédence généralisée,
- ordonnancement à priorité fixe sous contraintes de précédence,
- ordonnancement avec suspension autorisée des tâches.

Ce chapitre se termine sur les collaborations, la diffusion des résultats et les perspectives de recherche associées à ces contributions.

2.1.2 Démarche et outils

Le traitement des contraintes de précédence généralisée repose principalement sur une technique de transformation de graphe. Le graphe de précédence généralisée est déplié en un graphe de précédence simple. Cette approche est souvent utilisée dans les problèmes d'ordonnancement cyclique ou bien dans la théorie des réseaux de Petri. Nous utilisons alors le graphe après dépliage pour utiliser les résultats connus sur l'ordonnancement temps réel des systèmes temps réel soumis à des contraintes de précédence simple. L'algorithme complet est exponentiel.

Nous proposons une méthode de calcul des temps de réponse des tâches à priorité fixe soumises à des contraintes de précédence simple. Nous considérons les chaînes réentrantes de tâches périodiques non-réentrantes. A travers des exemples, nous mettons en évidence la nature des anomalies d'ordonnancement pouvant survenir à l'exécution lorsqu'un ordonnanceur à priorité fixe est utilisé pour ordonnancer les tâches à l'exécution. Puis à travers une approche classique de l'ordonnancement temps réel, nous développons une méthode de calcul des temps de réponse fondée sur la recherche des scénarios engendrant les pires temps de réponse et calculant les durées d'activité du processeur pour chacun de ces scénarios (*busy period*). L'algorithme complet

détermine des bornes supérieures des pires temps de réponse des tâches et sa complexité est pseudo-polynomiale.

Enfin, nous abordons le problème d'ordonnancement des tâches avec suspensions sous l'angle de la complexité du problème de faisabilité (i.e., d'existence d'un ordonnancement faisable). Nous présentons plusieurs résultats négatifs de $\mathcal{NP}$-Complétude. Nous montrons qu'il n'existe pas d'algorithme polynomial ou pseudo-polynomial pour décider si un système de tâches avec suspensions est ordonnançable ou non.

2.2 Tâches soumises à des précédences généralisées

2.2.1 Précédences généralisées

Deux tâches périodiques τ_i et τ_j sont soumises à une contrainte de précédence simple si toute exécution de τ_j est précédée par une exécution de τ_i. Les relations entre les tâches définissent un ordre partiel sur l'ensemble des tâches. Cet ordre partiel est souvent représenté par un graphe de précédence, où les sommets représentent les tâches et les arcs représentent les contraintes de précédence. Le graphe de précédence est nécessairement sans circuit puisque sinon le début d'exécution d'une tâche dépend de sa propre terminaison. Ceci revient à dire que les tâches appartenant à un circuit voient leurs dates de début d'exécution différées indéfiniment.

Dans certaines applications, les tâches en précédence ne s'exécutent pas en suivant la même période. Cette relation de précédence sera alors dite généralisée dans la suite. Ces relations de précédence sont simples à mettre en œuvre en utilisant des signalisations sur des événements (i.e., ou de façon moins immédiates des sémaphores). Il suffit de considérer une tâche périodique qui se synchronise avec une autre toutes les n activations. Ceci peut être simplement mis en œuvre en plaçant une primitive de synchronisation dans une branche *if then else* de l'algorithme de la tâche. Nous formalisons maintenant les contraintes de précédence généralisée. Notons que cette formalisation est indépendante des dates de réveils des tâches et des périodes des tâches.

Définition 1 *Soit $B_t(i)$ (resp. $E_t(i)$) le nombre de débuts (resp. de fins) d'exécution de τ_i à la date t. Une contrainte de précédence généralisée entre les tâches τ_i et τ_j, de périodes respectives T_i et T_j , est définie par :*

$$\forall t \in \mathbb{N} \qquad E_t(i) \times T_i \geq B_t(j) \times T_j \tag{2.1}$$

Nous noterons dans la suite que τ_i précède τ_j par : $\tau_i \prec \tau_j$.

Le poids d'une précédence généralisée est le rapport des périodes des tâches en relation de précédence. Chaque arc d'un graphe de précédence généralisée sera étiqueté par son poids (les poids unitaires seront omis).

Définition 2 *Soit un arc d'un graphe de précédence, $p = (\tau_i,\tau_j)$ le poids de l'arc p, noté $H(p)$, est : $H(p) = \frac{T_i}{T_j}$. Par extension, le poids d'un chemin ρ dans un graphe de précédence généralisée G est défini par le produit des poids des arcs qui le compose :*

$$H(\rho) = \prod_{p \in \rho} H(p) \tag{2.2}$$

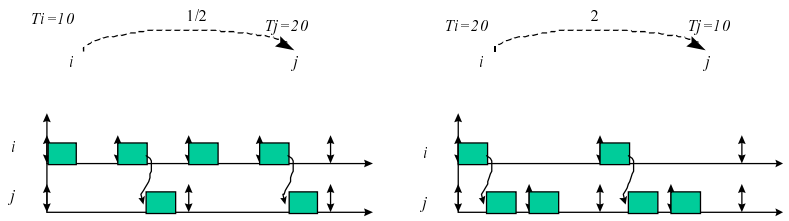


FIG. 2.1 – *Diagrammes de Gantt de tâches soumises à la précédence généralisée :* $\tau_i \prec \tau_j$

La relation de précédence classique ($T_i = T_j$) simplifie la relation de la définition 1 qui s'écrit alors $E_t(i) \geq B_t(j)$ (i.e. précédence généralisée de poids 1). En conséquence chaque requête de la tâche τ_j est précédée par une requête de la tâche τ_i. La figure 2.1 donne des exemples de tâches soumises à des contraintes de précédence généralisée. Une double flèche symbolise l'échéance d'une exécution et l'arrivée d'une nouvelle instance au même instant (tâche à échéance sur requête).

2.2.2 Dépliage du graphe de précédence

Nous proposons une méthode de validation permettant depuis une configuration de tâches soumises à des contraintes de précédence, de construire une configuration équivalente de tâches indépendantes, en deux étapes. La première étape consiste en le dépliage du graphe de précédence généralisée en un graphe de précédence simple, la seconde en la validation du système de tâches issu de la première étape.

Les contraintes de précédence généralisée peuvent être modélisées par des familles de contraintes de précédence simple. Pour cela une tâche τ_i est modélisée par n_i duplicata. Le $k^{ième}$ duplicata de la tâche τ_i sera noté τ_i^k, $k \in \{1 \dots n_i\}$. La $n^{ième}$ requête du duplicata τ_i^k représente la $((n-1)n_i + k)^{ième}$ requête de la tâche originelle τ_i.

Les méthodes de dépliage sont utilisées dans de nombreux domaines [61, 62, 63]. Le travail qui suit s'inspire de ces méthodes, mais le graphe considéré a une structure très différente de ceux considérés dans la littérature (acyclique versus fortement connexe). Cette différence dans la structure du graphe engendre des résultats analytiques différents.

Nous illustrons tout d'abord le principe sur un exemple simple. Considérons une contrainte de précédence généralisée entre une tâche τ_i de période 30 ms, et une tâche τ_j d'une période de 40 ms. Le diagramme de Gantt de l'exécution de ces deux tâches est donné figure 2.2. Trois exécutions de τ_i sur quatre sont en précédence avec la tâche τ_j. Il est possible de modéliser cette contrainte de précédence généralisée en dupliquant quatre fois τ_i et trois fois τ_j, et en créant des précédences simples entre les duplicata des tâches. Il est clair que l'exécution donnée figure 2.3 est alors équivalente à celle présentée figure 2.2. Dans la figure 2.3, tous les duplicata ont une période de 120 unités de temps. Cet ordonnancement partiel peut donc être répété indéfiniment.

La première étape consiste à calculer le nombre nécessaire de duplicata (théorème 1). Nous ne considérons à ce stade qu'une contrainte de précédence généralisée. Le cas d'un graphe quel-

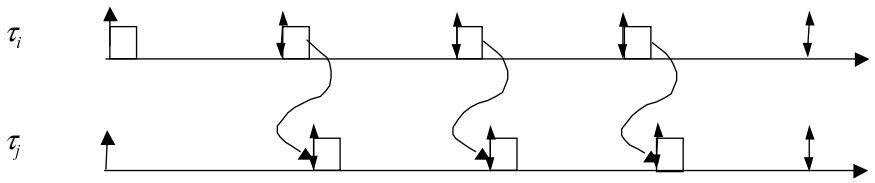


FIG. 2.2 – *Diagrammes de Gantt de tâches soumises à la précédence généralisée : $\tau_i \prec \tau_j$. Cet ordonnancement est répété à l'infini*

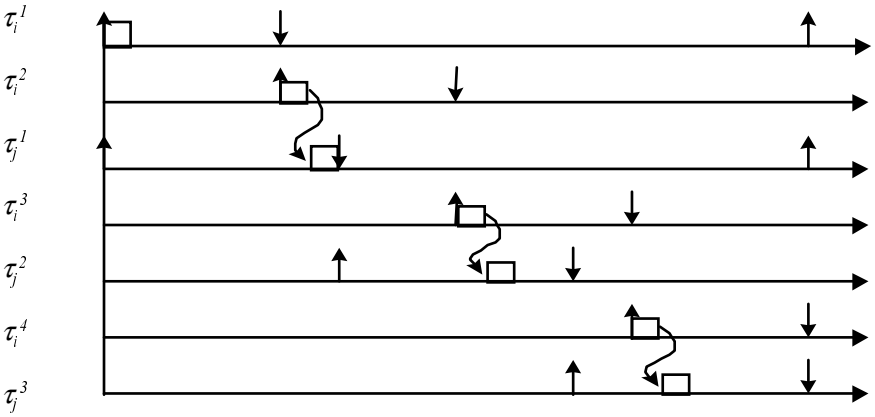


FIG. 2.3 – *Configuration après duplication et dépliage du graphe de précédence généralisée. Cet ordonnancement est répété à l'infini. (flèche vers le bas : échéance d'une tâche ; flèche vers le haut : réveil d'une tâche)*

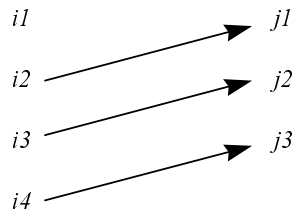


FIG. 2.4 – *Graphe déplié d'une contrainte de précédence généralisée*

conque est ensuite présenté.

Théorème 1 *Soit n_i et n_j le nombre de duplicata de τ_i et τ_j, la contrainte de précédence généralisée $\tau_i \prec \tau_j$ peut être modélisée par un ensemble fini de contraintes de précédence simple entre les duplicata si, et seulement si :*

$$n_i \times T_i - n_j \times T_j = 0 \qquad n_i \in \mathbb{N}^*, n_j \in \mathbb{N}^* \tag{2.3}$$

Le théorème 2 calcule les relations de précédence entre les duplicata.

Théorème 2 *Soit $\tau_i \prec \tau_j$ une contrainte de précédence généralisée, n_i et n_j le nombre de duplicata des tâches τ_i et τ_j. Alors elle sera représentée de façon équivalente par :*

- *n_i contraintes de précédence simple si $T_i > T_j$:*

$$\forall k \in \{1 \dots n_i\} \qquad \tau_i^k \prec \tau_j^{a_k}, a_k = \left\lfloor \frac{(k-1)T_i}{T_j} \right\rfloor + 1 \tag{2.4}$$

- *n_j contraintes de précédence simple sinon :*

$$\forall k \in \{1 \dots n_j\} \qquad \tau_i^{b_k} \prec \tau_j^k, b_k = \left\lceil \frac{kT_i}{T_j} \right\rceil \tag{2.5}$$

Sur l'exemple de la figure 2.2 : la solution minimale de l'équation (1) est $n_i = 4$ et $n_j = 3$. On remarque que $T_i \leq T_j$, la seconde partie du théorème 2 est donc utilisée pour déplier la contrainte de précédence généralisée. Puisque $k \in \{1 \dots n_j\}$, les calculs effectués sont :

$$k = 1, b_1 = \left\lceil \frac{1 \times 40}{30} \right\rceil = 2 \quad \Rightarrow \quad \tau_i^2 \prec \tau_j^1$$
$$k = 2, b_2 = \left\lceil \frac{2 \times 40}{30} \right\rceil = 3 \quad \Rightarrow \quad \tau_i^3 \prec \tau_j^2$$
$$k = 3, b_3 = \left\lceil \frac{3 \times 40}{30} \right\rceil = 4 \quad \Rightarrow \quad \tau_i^4 \prec \tau_j^3$$

Le graphe de précédence déplié correspondant est donné figure 2.4.

Pour calculer le nombre de duplicata, la relation (2.3) doit être vérifiée pour tous les arcs du graphe de précédence généralisée. Le système d'équations à résoudre est :

$$\Sigma(G) : \qquad n_i T_i = n_j T_j \qquad \tau_i \prec \tau_j, n_i \in \mathbb{N}^*, n_j \in \mathbb{N}^*$$

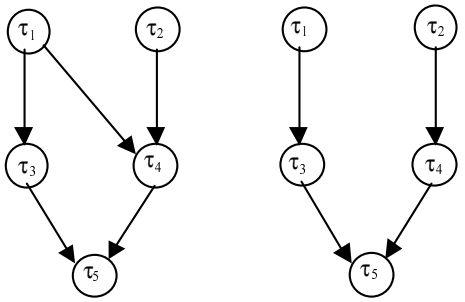


FIG. 2.5 – *Graphe G et arbre maximal inclus dans G*

Le théorème 3 va montrer que tout graphe sans circuit de précédence généralisée peut être déplié en un graphe de précédence simple. Il fournit de plus le nombre minimal de duplicata pour chaque tâche. Sa preuve est basée sur le fait que le nombre de duplicata peut être calculé en ne considérant qu'un arbre maximal inclus dans le graphe de précédence généralisée G.

Nous illustrons ce fait sur le graphe G de la figure 2.5. Dans G, il existe deux chemins reliant τ_1 à τ_5. Il est simple de montrer que l'arc (τ_1,τ_4), correspondant à l'équation suivante de $\Sigma(G)$: $n_1T_1 = n_4T_4$, est une combinaison linéaire des autres équations :

$$\begin{aligned} n_1T_1 &= n_3T_3 \\ n_2T_2 &= n_4T_4 \\ n_3T_3 &= n_5T_5 \\ n_4T_4 &= n_5T_5 \end{aligned}$$

Cette équation est donc inutile pour résoudre le système $\Sigma(G)$. En répétant ce principe de suppression des arcs inutiles dans le graphe de précédence G, nous obtenons un arbre maximal inclus dans G. Sur l'exemple de la figure 2.5 après suppression l'arc (τ_1,τ_4) dans G, le graphe résultant est un arbre maximal. L'ensemble des équations correspondantes permet donc de calculer le nombre minimum de duplicata.

Nous montrons tout d'abord une propriété sur les chemins entre deux sommets dans le graphe de précédence généralisée. La notion de poids va permettre de caractériser une propriété importante des chemins entre deux sommets d'un graphe de précédence généralisée.

Lemme 1 *Soit τ_i et τ_j deux sommets d'un graphe de précédence généralisée G sans circuit. Tout chemin ρ de τ_i à τ_j vérifie : $H(\rho) = \frac{T_i}{T_j}$ (i.e. sont de même poids).*

Théorème 3 *Tout graphe de précédence généralisée sans circuit peut être déplié en un graphe de précédence simple et la solution minimum de $\Sigma(G)$ est donnée par :*

$$n_i = \frac{ppcm(T_1,\ldots,T_n)}{T_i} \qquad 1 \leq i \leq n \tag{2.6}$$

De façon à terminer la transformation, nous devons définir les paramètres temporels des tâches créées par le dépliage (i.e. les duplicata). Clairement les duplicata ont les mêmes durées d'exécution et échéances que la tâche dont ils sont issus. Seules les dates d'activations et les périodes

des duplicata sont à calculer de façon à respecter l'équivalence entre la configuration de tâches originelles et leurs duplicata.

Théorème 4 *Soit une configuration de tâches* $S = \{\tau_i(r_i,C_i,D_i,T_i),1 \leq i \leq n\}$, $\prec$ *un ordre partiel sur* S *définissant les précédences généralisées,* n_i *le nombre de duplicata des tâches* τ_i*, alors la configuration de tâches :*

$$S^* = \{\tau_i^k(r_i^k,C_i^k,D_i^k,T_i^k),1 \leq i \leq n,1 \leq k \leq n_i\} \tag{2.7}$$

est construite telle que :

$$\forall k \in \{1 \dots n_i\} \begin{cases} r_i^k = r_i + (k-1)T_i \\ T_i^k = n_i T_i \\ D_i^k = D_i \\ C_i^k = C_i \end{cases} \tag{2.8}$$

alors S *est ordonnançable si, et seulement si,* S^* *est ordonnançable.*

Ce résultat découle directement des résultats précédents. La transformation montre que la configuration construite est équivalente à la configuration initiale. Remarquons que le nombre de duplicata peut être très grand (exponentiel) puisque leur nombre est fonction du ppcm des périodes des tâches originelles.

La complexité de l'algorithme de dépliage d'un graphe G est :

$$O\left(|Q|\sum_{i\in T} n_i + |P|\sum_{(i,j)\in P} \min(n_i,n_j)\right) \tag{2.9}$$

où Q est l'ensemble des sommets (i.e. des tâches) et P l'ensemble des arcs (i.e. des précédences généralisées) du graphe de précédence. Ainsi le dépliage conduit à un traitement pseudo-polynomial en supposant que les valeurs n_i sont des entrées du problème, c'est-à-dire qu'elles ont été préalablement calculées. Notons toutefois que la $\mathcal{NP}$-Complétude du problème d'ordonnancement de tâches périodiques soumises à des contraintes de précédence généralisée est un problème ouvert.

2.2.3 Validation de la configuration obtenue

Dans le paragraphe précédent nous réduisons la validation d'une configuration de tâches soumises à des précédences généralisées, à une autre ne contenant que des précédences simples. Ce problème a été résolu dans la littérature [14, 56].

Pour conclure sur ce thème, des travaux publiés après les notres [34] traitent un problème similaire mais se limitant aux poids N et $1/N$ comme rapports de périodes. Une technique spécifique d'analyse d'ordonnançabilité est aussi proposée pour intégrer ces contraintes de précédence. Notre approche est plus générale puisqu'elle ne se limite pas à des rapports de périodes particuliers. De plus elle ne nécessite pas de développer de nouvelle technique d'analyse d'ordonnançabilité.

2.3 Tâches à priorité fixe et contraintes de précédence

Dans ce paragraphe, nous montrons que les résultats classiques de l'ordonnancement des tâches à priorité fixe sur un monoprocesseur ne peuvent pas être appliqués lorsque les tâches sont synchronisées. Notre modèle de l'application temps réel incorpore explicitement un mécanisme de synchronisation (e.g. les sémaphores). Nous supposons que les opérations de verrouillage (*Wait* sur un sémaphore) ne peuvent intervenir qu'au début d'une tâche et les opérations de libération (*Signal* sur un sémaphore) en fin de tâche. Nous supposons de plus que les durées de ces opérations sont nulles et n'engendrent pas de délais liés à des changements de contexte de tâche (absence *d'overhead*). Nous mettons en évidence ci-dessous des anomalies d'ordonnancement qui sont connues dans les systèmes multiprocesseurs. Le graphe des tâches est réentrant, c'est-à-dire que deux instances de deux tâches appartenant à une même composante connexe peuvent entrer en concurrence pour l'obtention du processeur. Toutefois, deux instances d'une même tâche sont exécutées selon la politique FIFO (i.e. une tâche est non réentrante). Puis nous présentons, dans un contexte général, un algorithme de calcul d'une borne supérieure du pire temps de réponse des tâches. Cette borne permet d'établir une condition suffisante d'ordonnançabilité d'une configuration de tâches périodiques, et soumises à des contraintes de précédence. Cette condition suffisante d'ordonnançabilité se base sur l'approche de [36] qui traite l'ordonnancement de chaînes non réentrantes de tâches. Nous présentons enfin un exemple d'application de cette méthode de validation dans un troisième paragraphe.

2.3.1 Anomalies d'ordonnancement

La prise en compte des contraintes de précédence en ordonnancement à priorité fixe sur un processeur engendre des anomalies analogues à celles constatées en ordonnancement multiprocesseur. Ces anomalies ont été mises en évidence par Graham [32]. Des exemples de ces anomalies sont reportés dans [96, 14].

Nous montrons que ces anomalies peuvent se présenter en ordonnancement temps réel de tâches périodiques à priorité fixe en monoprocesseur. *"Si une affectation des priorités est faisable pour une configuration de tâches périodiques à priorité fixe avec un protocole de synchronisation, s'exécutant avec préemption autorisée sur un processeur, avec des contraintes de précédence, alors diminuer la durée d'une tâche ou relâcher une contrainte de précédence peut rendre l'affectation non faisable"*. Ce résultat est illustré au travers des deux exemples ci-dessous.

Réduction des durées d'exécution

L'exemple suivant provient de [16] et illustre que la réduction d'une durée d'une tâche peut en rendre une autre non ordonnançable. Les paramètres temporels des tâches sont donnés dans le tableau 2.1. Les tâches sont à départ simultané et il existe une contrainte de précédence : $\tau_3 \prec \tau_4$.

La configuration de tâches, avec les paramètres indiqués dans le tableau 2.1, est ordonnançable (figure 2.6.*a)*). Par contre si l'on considère que la durée de la tâche τ_1 est réduite de 5 à 3 unités de temps, la configuration n'est plus ordonnançable (figure 2.6.*b)*).

Suppression d'une contrainte de précédence

Nous considérons maintenant une configuration de tâches dont les paramètres sont donnés dans le tableau 2.2, avec une contrainte de précédence : $\tau_3 \prec \tau_1$. La configuration est ordonnançable (figure 2.7.*a)*), par contre si l'on supprime la contrainte de précédence, elle ne l'est plus (figure 2.7.*b)*). Il suffit de considérer pour cela l'ordonnancement par *Deadline Monotonic*, puisqu'il est optimal en monoprocesseur lorsque les tâches sont indépendantes, à départ simultané et que les échéances sont inférieures ou égales aux périodes. Il n'existe donc pas d'ordonnancement faisable lorsque ces tâches sont indépendantes.

Le point commun de ces deux types d'anomalies est qu'elles surviennent exclusivement lorsqu'une tâche est bloquée dans l'attente d'une synchronisation avec ses prédécesseurs. Il devient nécessaire, comme lorsque les tâches partagent des ressources, de valider l'application en tenant compte des modifications de l'ordonnancement en-ligne des durées d'exécution des tâches.

Notations

τ_i **:** chaîne de tâches reliées par des contraintes de précédence.

$\tau_{i,j}$ **:** $j^{ième}$ tâche de la chaîne τ_i.

$C_{i,j}$ **:** pire durée d'exécution de la $j^{ième}$ tâche de la chaîne τ_i. Cette durée est mesurée lorsque la tâche s'exécute su un unique processeur sans aucune interférence.

D_i **:** échéance de la chaîne τ_i, c'est-à-dire échéance de bout-en-bout.

$D_{i,j}$ **:** échéance de la $j^{ième}$ tâche de la chaîne τ_i.

T_i **:** période de la chaîne τ_i et de l'ensemble des tâches la composant.

$\pi_{i,j}$ **:** priorité de la $j^{ième}$ tâche de la chaîne τ_i. La priorité 0 sera la plus forte.

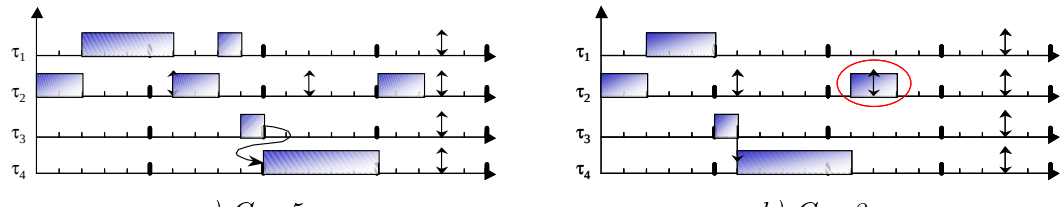
$\pi_i(j)$ **:** priorité de la chaîne τ_i lorsque le temps de réponse de la tâche $\tau_{i,j}$ est étudiée. Elle est définie comme la plus faible des priorités des tâches $\tau_{i,1}, \ldots, \tau_{i,j}$.

E_i **:** pire durée d'exécution d'une sous-chaîne de tâches (segment).

$\rightarrow$ **:** Synchronisation assurer par un mécanisme de type sémaphore.

Tâches τ_i	C_i	$D_i = T_i$	π_i
τ_1	5	18	3
τ_2	2	6	2
τ_3	1	18	4
τ_4	5	18	1

TAB. 2.1 – *Configuration de tâches périodiques à échéance sur requête*



a) C_1=5 *b)* C_1=3

FIG. 2.6 – *Effet de la réduction des durées d'exécution sur l'ordonnancement*

Tâches τ_i	C_i	D_i	T_i	π_i
τ_1	1	4	6	1
τ_2	2	3	3	2
τ_3	1	4	6	3

TAB. 2.2 – *Configuration de tâches périodiques, $D_i \leq T_i$*

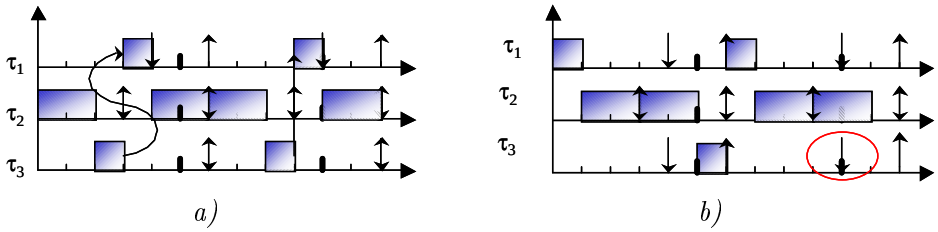


FIG. 2.7 – *Ordonnancement avec contrainte de précédence*

2.3.2 Les étapes du test

La complexité du problème de faisabilité d'une tâche est ouverte et aucune condition nécessaire et suffisante n'est connue à ce jour. Nous détaillons rapidement les différentes étapes du test :

1. *Transformation du graphe de précédence en chaînes :* nous montrons que tout graphe de précédence connexe peut être transformé en une chaîne. La configuration de tâches obtenue est équivalente à la configuration initiale. Ceci permet d'utiliser la démarche de test de Harbour et al. [36].
2. *Mise sous forme canonique des priorités de la chaîne étudiée :* cette transformation permettra de calculer précisément les dates de fin des prédécesseurs de la tâche étudiée.
3. *Regroupement des tâches par rapport à la plus petite priorité ($\pi_i(j)$) parmi les prédécesseurs de la tâche étudiée :* les tâches ayant une priorité supérieure à $\pi_i(j)$ sont sans influence sur le temps de réponse de la tâche étudiée. Les autres peuvent interférer soit en blocage en début de période d'activité, soit en préemption suivant qu'elles sont ou non précédées par des tâches de priorité supérieure à $\pi_i(j)$.
4. *Calcul de la période d'activité induisant le pire temps de réponse :* la détermination de la plus grande période d'activité caractérise le pire scénario d'arrivées des tâches que doit affronter la tâche étudiée. Ceci permet de déterminer le nombre d'instances de cette tâche dans la période d'activité.
5. *Calcul de la date de fin des instances des tâches dans l'ordre de la chaîne :* les dates de fin permettent de calculer le pire temps de réponse de la tâche étudiée.

Il est très important de noter que les étapes 2 à 5 sont dépendantes de la tâche considérée. Elles doivent être complètement refaites pour chaque tâche à valider.

2.3.3 Transformation d'un graphe en chaînes

Nous montrons ci-après que pour une affectation donnée des priorités, le graphe de précédence peut être transformé en chaînes de tâches. La règle suivante transforme chaque composante connexe du graphe de précédence en une chaîne. Nous rappelons qu'une composante connexe est une partie du graphe pour laquelle pour tout couple de sommets il existe un chemin non orienté

reliant ces deux sommets. La règle de transformation est la suivante :

> *Pour chaque composante connexe du graphe de précédence une chaîne est construite en plaçant à chaque étape la tâche la plus prioritaire parmi les tâches sans prédécesseurs ou dont tous les prédécesseurs ont déjà été placés.* (2.10)

Le résultat suivant montre que cette transformation laisse le problème d'ordonnancement inchangé.

Lemme 2 *Soit τ un ensemble de tâches, π l'affectation des priorités aux tâches (toutes les tâches possèdent une priorité différente), $\prec$ un ordre partiel quelconque sur τ et $\prec'$ les chaînes résultant de la transformation de $\prec$ par la règle (2.10) alors :*
$(\tau,\pi,\prec)$ est ordonnançable si, et seulement si, $(\tau,\pi,\prec')$ l'est.

Ce résultat montre donc que tout problème d'ordonnancement à priorités fixes de tâches périodiques à départ simultané soumises à des contraintes de précédence peut se réduire à un problème équivalent et ne comportant que des précédences de type chaîne. Ce dernier problème a été étudié dans [36, 48, 56] dans le cas de chaînes non réentrantes (i.e. l'instance k de la première tâche d'une chaîne ne peut pas s'exécuter avant la fin de la $(k-1)^{ième}$ instance de la dernière tâche de cette chaîne). Nous relaxons dans la suite cette contrainte.

Exemple de transformation : La figure 2.8 montre la transformation du graphe de précédence (figure 2.8(a)) en chaînes de précédence (figure 2.8(b)). Les numéros situés au-dessous des ronds correspondent au numéro des tâches, et ceux situés au-dessus représentent la priorité des tâches.

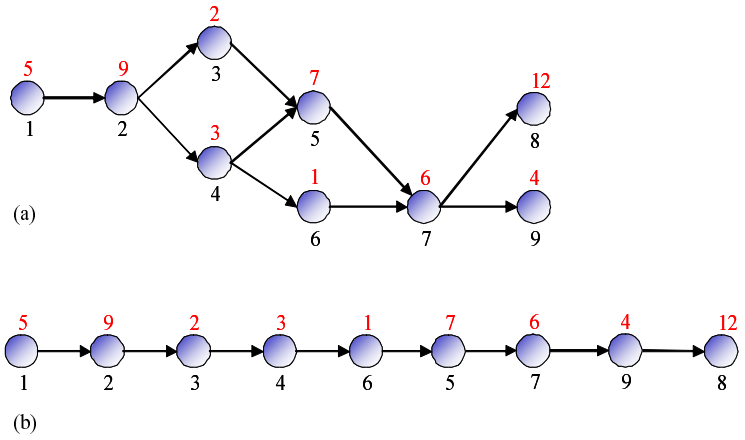


FIG. 2.8 – *Transformation d'un graphe de précédence en chaîne de précédence*

Plusieurs chaînes différentes peuvent être déterminées par cette transformation, mais nous allons voir qu'une seule forme canonique correspondra à ces différentes chaînes.

2.3.4 Forme canonique des priorités et temps de réponse

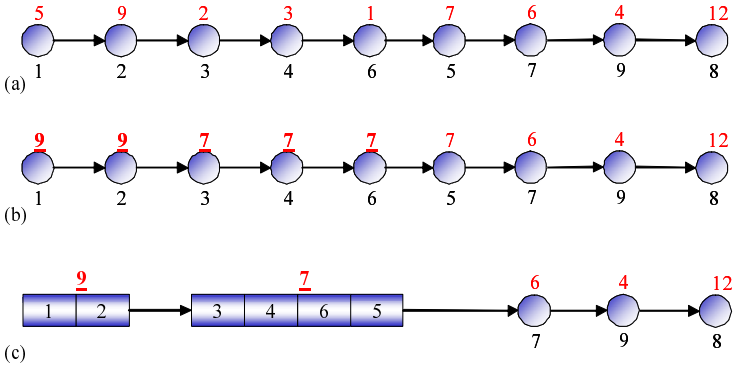
Nous détaillons dans la suite le calcul du pire temps de réponse de la $j^{ième}$ tâche de la chaîne de tâches numéro i, désignée par τ_i. Dans la suite τ_{ij} désigne la $j^{ième}$ tâche de τ_i et $n(i)$ désigne le nombre de tâches de cette chaîne. La durée de $\tau_{i,j}$ est $C_{i,j}$ et sa priorité est $\pi_{i,j}$. $\pi_i(j)$ désigne la tâche de plus faible priorité du début de la chaîne jusqu'à $\tau_{i,j}$:

$$\pi_i(j) = \max_{k=1...j}(\pi_{i,k}) \tag{2.11}$$

L'analyse du pire temps de réponse d'une tâche repose sur la mise sous forme canonique des priorités des tâches prédécesseurs de celle étudiée [36]. Cette modification consiste à propager la priorité la plus faible en partant de la tâche étudiée vers la première tâche de la chaîne. La conséquence de cette modification est d'obtenir des priorités non décroissantes du début de la chaîne jusqu'à la tâche étudiée. Ceci facilitera le calcul du pire temps de réponse.

De plus, si lors de la mise sous forme canonique, la priorité de deux sous-tâches adjacentes est la même, elles sont rassemblées en une seule (dont la durée d'exécution est la somme des durées des deux sous-tâches et l'échéance est celle de la dernière tâche regroupée). Ce regroupement est effectué même si les tâches n'ont pas une même échéance, puisqu'elles ne seront considérées que pour calculer leur interférence sur la tâche étudiée. Le nombre de sous-tâches formant la chaîne est décrémenté (i.e. $n(i)$).

Exemple de mise sous forme canonique : La figure 2.9 présente la mise sous forme canonique de la chaîne de tâches obtenue par transformation sur la figure 2.8 lorsque la tâche 7 est étudiée. Les nouvelles priorités des tâches sont données figure 2.9(b)(numéro souligné) après mise sous forme canonique. Sur la figure 2.9(c), les tâches adjacentes de même priorité ont été regroupées.



(a) Chaîne de départ (b) Mise sous forme canonique de la chaîne
(c) Regroupement des tâches de priorité égale

FIG. 2.9 – *Mise sous forme canonique de la chaîne obtenue sur la figure 2.8(b)*

Regroupement des tâches en segment par rapport à $\pi_i(j)$

Le pire temps de réponse de $\tau_{i,j}$ surviendra dans une période d'activité où toutes les priorités des tâches sont inférieures ou égales à $\pi_i(j)$. Cette période d'activité est délimitée par des temps

creux ou bien des tâches de priorités strictement supérieures à $\pi_i(j)$. On parle alors de période d'activité de niveau (de priorité) $\pi_i(j)$ [52, 36].

Dans la suite une sous-chaîne de τ_i est désignée sous le nom de segment. L'algorithme va distinguer deux types de segments :

1. les segments H (High priority) qui ne contiennent que des tâches de plus forte priorité que $\tau_{i,1}$ (i.e. inférieure ou égale à $\pi_i(j)$).
2. les segments L (Low priority) qui ne contiennent que des tâches de plus faible priorité que $\tau_{i,1}$ (i.e. supérieure à $\pi_i(j)$).

Puisque nous autorisons la réentrance des chaînes de tâches, l'interférence des segments les uns sur les autres sera différente du cas non réentrant considéré dans [36].

Interférence des chaînes τ_j, $j \neq i$ sur τ_i : Nous considérons tout d'abord l'interférence des chaînes τ_j, $j \neq i$. Un segment L ne peut pas s'exécuter dans une période d'activité de niveau $\pi_i(j)$ et n'a en conséquence aucune influence sur le pire temps de réponse de $\tau_{i,j}$. La figure 2.10 présente tous les cas intéressants d'enchaînement des segments L et H. Un segment H, n'appartenant pas à la chaîne étudiée peut, quant à lui, provoquer deux effets différents sur τ_{ij} suivant qu'un segment H est précédé ou précède un segment L :

- *effet de blocage :* lorsqu'un segment H est précédé par un segment L, le segment H ne peut intervenir dans la période d'activité que si le segment L s'est terminé juste avant le début de celle-ci. Le lemme 3 montre que parmi ces segments H, un seul pourra interférer dans la période d'activité. Ces segments H sont notés $\mathbf{H}_{sb}$ (i.e. singly blocking).
- *effet de préemption :* Un segment H non précédé par un segment L (mais qui peut en précéder un) peut s'activer plusieurs fois puisque les chaînes sont réentrantes. On parle alors de segment à préemption multiple, noté $\mathbf{H}_{mp}$ (multiply preemptive segment).

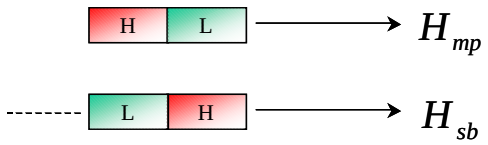


FIG. 2.10 – *Enchaînement des segments H et L dans une chaîne τ_j, $j \neq i$*

Lemme 3 *Parmi les segments H d'une chaîne τ_j, $j \neq i$, précédés par un segment L (i.e. segments H_{sb}), un seul pourra contribuer à un effet de blocage.*

Interférence des segments de τ_i sur elle-même : Nous examinons maintenant le regroupement de la chaîne τ_i (i.e. de la tâche étudiée) et examinons l'interférence des différents segments de τ_i sur elle-même. La figure 2.11 précise l'enchaînement des différents segments de la chaîne τ_i. Nous décrivons ci-dessous les interférences induites par ces différents segments sur l'exécution de τ_i. La chaîne τ_i peut subir trois types d'effet :

- *effet de préemption :* Les tâches appartenant au premier segment (i.e. de $\tau_{i,1}$ à $\tau_{i,j}$) ont des priorités non décroissantes puisqu'elles sont sous la forme canonique. Elles peuvent intervenir plusieurs fois dans la période d'activité de niveau $\pi_i(j)$. En conséquence, lors de l'estimation de cette dernière, l'enchaînement des tâches $\tau_{i,1}, \ldots, \tau_{i,j}$ est considéré comme

un segment H_{mp}. De plus, s'il existe un segment H adjacent à $\tau_{i,j}$, ce dernier sera inclus dans le segment H_{mp}.

- *effet de blocage multiple :* Le lemme 4 précise qu'un segment H suivant immédiatement $\tau_{i,j}$ ne peut pas introduire d'effet de préemption sur les tâches $\tau_{i,1} \dots \tau_{i,j}$, mais peut s'exécuter plusieurs fois dans la période d'activité. On parle alors de segment à blocage multiple, nommé ensuite $\mathbf{H}_{mb}$. Ce blocage est dû au fait que la chaîne τ_i est réentrante. Ainsi, si k instances de τ_i se sont exécutées dans la période d'activité, il peut se produire $(k-1)$ blocages induits par ce segment H_{mb}.
- *effet de blocage :* Un segment H de τ_i précédé par un segment L peut provoquer *une fois* un effet de blocage et sera en compétition avec les segments H des autres chaînes. Nous parlons alors de segments H à blocage simple ou segment H_{sb} (singly blocking).

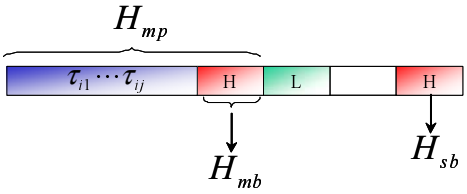


FIG. 2.11 – *Enchaînement des segments de τ_i*

Lemme 4 *Les activations successives du premier segment H de τ_i ne peuvent pas s'entrelacer.*

2.3.5 Période d'activité et temps de réponse

Une tâche est faisable si son pire temps de réponse n'est pas supérieur à son échéance. Lorsque les tâches sont indépendantes et que les échéances ne sont pas supérieures aux périodes, le pire temps de réponse d'une tâche survient lorsque la tâche est réveillée en même temps que toutes les tâches plus prioritaires. (i.e., l'instant critique) [55, 52]. La paragraphe 2.3.1 à montré que la notion d'instant critique ne permet plus de déterminer le pire temps de réponse d'une tâche lorsque les tâches sont soumises à des contraintes de synchronisation.

Afin de déterminer le pire scénario d'arrivée des tâches, il faut déterminer quel est le plus grand segment H_{sb} pouvant interférer dans une période d'activité de niveau $\pi_i(j)$. Le segment H_{mb} ne peut pas débuter une période occupée de niveau $\pi_i(j)$, car ce segment est plus prioritaire que le niveau de priorité $\pi_i(j)$. Ceci contredit le fait que le segment H_{mb} débute une période occupée de niveau $\pi_i(j)$.

Théorème 5 *L'interférence maximale que peut subir $\tau_{i,j}$ dans une période d'activité de niveau $\pi_i(j)$ débutant à la date t_0, survient dans le scénario suivant :*

- *(a) τ_i est activée à la date t_0.*
- *(b) une instance de chaque tâche débutant par un segment H_{mp} arrive à la date t_0.*
- *(c) une instance du plus long segment H_{sb}, choisie parmi toutes les chaînes, arrive à la date t_0.*

L'étude de la tâche $\tau_{i,j}$, c'est-à-dire le calcul de son pire temps de réponse se décompose en trois étapes :

1. le calcul du nombre d'activations de τ_i dans la période d'activité de niveau $\pi_i(j)$ qui permet la prise en compte de l'interférence de l'éventuel segment H_{mb}.

2. le calcul de la date de fin de la première tâche de la chaîne (i.e. $\tau_{i,1}$).
3. le calcul des dates de fin des tâches $\tau_{i,2}, \ldots, \tau_{i,j}$. Ce calcul prend en compte les dates de fin du prédécesseur immédiat.

Nombre d'activations de τ_i dans la période d'activité de niveau $\pi_i(j)$: Nous déterminons maintenant le nombre d'activations de τ_i dans la période d'activité de niveau $\pi_i(j)$ correspondant au pire scénario d'arrivée des tâches.

La durée d'un segment h_i de type H est notée E_i et est égale à la somme des durées d'exécution des tâches composant ce segment. La période d'un tel segment est héritée de la chaîne le contenant et est notée T_i. Dans ce calcul, deux ensembles sont à considérer :

MP (*multiple preemption*) : c'est l'ensemble des segments de type H_{mp}. Plus précisément, il regroupe les segments de type H_{mp} de toutes les chaînes τ_j, $j \neq i$, ainsi que le segment H_{mp} de la chaîne τ_i. Ce dernier est composé, au minimum des tâches $\tau_{i,1}, \ldots, \tau_{i,j}$ auxquelles peut se rajouter une suite de tâches $\tau_{i,j+1}, \ldots, \tau_{i,l}$ formant le segment H_{mb}, s'il existe.

SB (*Singly Blocking*) : Tous les segments H précédés par un segment L (i.e. H_{sb}) sont regroupés dans l'ensemble SB.

Une chaîne peut bien sûr avoir des segments simultanément dans les ensembles MP et SB. La charge des tâches de priorité supérieure ou égale à $\pi_i(j)$ dans l'intervalle $[0,t)$ est donnée par :

$$W_i(t) = \max_{h_i \in SB} (E_i) + \sum_{h_i \in MP} \left\lceil \frac{t}{T_i} \right\rceil E_i \tag{2.12}$$

Le premier terme désigne le plus grand segment H_{sb}, tandis que le second calcule la charge cumulée des segments H_{mp}. La période d'activité de niveau $\pi_i(j)$ est calculée comme le plus petit point fixe L_i du système suivant :

$$\begin{cases} L_i^{(0)} & = & \sum_{k=1}^{j} C_{i,k} \\ L_i^{(n+1)} & = & W\left(L_i^{(n)}\right) \\ L_i = \min\left(n \geq 0,\, L_i^{(n+1)} = L_i^{(n)}\right) \end{cases} \tag{2.13}$$

Le nombre d'activations N_i de τ_i dans cette période d'activité de niveau $\pi_i(j)$ est :

$$N_i = \left\lceil \frac{L_i}{T_i} \right\rceil \tag{2.14}$$

2.3.6 Calcul du pire temps de réponse de $\tau_{i,j}$

Le nombre d'activations de $\tau_{i,j}$ dans la période d'activité est maintenant connu. La dernière opération consiste à calculer toutes les dates de fin des instances de $\tau_{i,j}$ dans cette période d'activité. Ceci permet de calculer le temps de réponse de chaque instance et déterminer ainsi le pire temps de réponse de $\tau_{i,j}$. Les dates de fin de $\tau_{i,j}$ dépendent directement des dates de fin de son prédécesseur immédiat dans la chaîne τ_i. L'algorithme va donc calculer les dates de fin des tâches dans l'ordre topologique de la chaîne τ_i (i.e., $\tau_{i,1} \ldots \tau_{i,j}$). Le calcul des dates de fin de $\tau_{i,1}$ va se différencier des autres puisqu'il doit tenir compte (et lui seulement) du segment H_{mp} de la chaîne étudiée.

Dans la suite $F_{i,j}(k)$ désigne la date de fin de la $k^{ième}$ instance de $\tau_{i,j}$ dans la période d'activité.

Calcul des dates de fin de $\tau_{i,1}(k)$: La date de fin de $\tau_{i,1}(k)$ dépend :
- du blocage lié au plus long segment de l'ensemble SB,
- de l'ensemble MP_1 contenant les segments H_{mp} (formés des tâches plus prioritaires que $\tau_{i,1}$), n'appartenant pas à τ_i.
- du premier segment H de la chaîne τ_i (segments H_{mp} et H_{mb} s'il existe (Cf. figure 2.11)). La durée de ce segment est notée E_i,1.

La charge induite par τ_i jusqu'à l'instance k de $\tau_{i,1}$ est donnée par :

$$(k-1)E_{i,1} + C_{i,1} \tag{2.15}$$

La charge des tâches de priorité supérieure ou égale à $W_{i,1}(k)$ dans l'intervalle [0,t) est :

$$W_{i,1,k}(t) = \max_{h_i \in SB}(E_i) + \sum_{h \in MP_1} \left\lceil \frac{t}{T_h} \right\rceil E_h + (k-1)E_{i,1} + C_{i,1} \tag{2.16}$$

Les dates de fin de $\tau_{i,1}(k), k = 1 \ldots N_i$, vont se définir comme le plus petit point fixe de l'équation suivante :

$$\begin{array}{l} \left\{ \begin{array}{lcl} L_i^{(0)} & = & C_{i,1} \\ L_i^{(n+1)} & = & W_{i,1,k}\left(L_i^{(n)}\right) \end{array} \right. \\ F_{i,1}(k) = \min\left(n \geq 0,\ L_i^{(n+1)} = L_i^{(n)}\right) \end{array} \tag{2.17}$$

Calcul des dates de fin de $\tau_{i,j}(k)$: Les calculs des dates de fin de $\tau_{i,2}(k)$ à $\tau_{i,j}(k)$ vont reposer sur les mêmes principes. Examinons l'exemple de $\tau_{i,2}$ qui va dépendre directement des dates de fin de $\tau_{i,1}$. Les segments H_{sb} ont déjà été pris en compte dans le calcul des dates de fin de $\tau_{i,1}$, et en conséquence ne doivent pas être comptabilisés à nouveau dans les calculs sur $\tau_{i,2}$. Parmi les tâches des segments de l'ensemble MP_1, il ne faut pas tenir compte pour les calculs sur $\tau_{i,2}$ des tâches qui sont moins prioritaires qu'elle. Il est donc nécessaire de déterminer un ensemble MP_2 qui va contenir les segments H_{mp} vis-à-vis de $\pi_{i,2}$. Puisque τ_i est sous forme canonique, il suffit d'appliquer l'algorithme de regroupement sur les segments de MP_1 pour déterminer les nouveaux segments de MP_2. De façon générale, on obtient donc MP_j en appliquant l'algorithme de regroupement sur les segments de MP_{j-1} vis-à-vis de $\pi_{i,j}$. Les tâches écartées par l'algorithme de regroupement ne peuvent pas intervenir sur les dates de fin de $\tau_{i,j}$ car elles ont été prises en compte dans le calcul des dates de fin de $\tau_{i,j-1}$.

La charge des tâches plus prioritaires que $\tau_{i,j}(k)$ sur l'intervalle [0,t) est donnée par la date de fin de $\tau_{i,j-1}(k)$, plus toutes les tâches de priorité inférieure ou égale à $\pi_{i,j}$ arrivées depuis la fin de $\tau_{i,j-1}(k)$ et enfin la charge de $\tau_{i,j}$ elle-même :

$$W_{i,j,k}(t) = F_{i,j-1}(k) + \sum_{h \in MP_j} \left(\left\lceil \frac{t}{T_h} \right\rceil - \left\lceil \frac{F_{i,j-1}(k)}{T_h} \right\rceil \right) E_h + C_{i,j} \tag{2.18}$$

Le calcul des dates de fin de $\tau_{i,j}(k)$ est :

$$\begin{array}{l} \left\{ \begin{array}{lcl} L_i^{(0)} & = & F_{i,j-1}(k) + C_{i,j} \\ L_i^{(n+1)} & = & W_{i,j,k}\left(L_i^{(n)}\right) \end{array} \right. \\ F_{i,j}(k) = \min\left(n \geq 0,\ L_i^{(n+1)} = L_i^{(n)}\right) \end{array} \tag{2.19}$$

Le test d'ordonnançabilité de $\tau_{i,j}$ découle directement des résultats précédents.

Théorème 6 *Une tâche $\tau_{i,j}$ est ordonnançable si :*

$$\max\left((k-1)T + D_{i,j} - F_{i,j}(k)\right) \geq 0 \qquad 1 \leq k \leq N_i \tag{2.20}$$

Ce test est une condition suffisante puisque les dates de fin calculées sont des bornes supérieures du pire temps de réponse. Aucune condition nécessaire et suffisante n'est connue pour ce problème.

La vérification de toutes les échéances des tâches revient à appliquer la méthode complète sur chaque tâche. L'algorithme correspondant est pseudo-polynomial. La complexité du problème est ouverte. Nous avons présenté un exemple complet de la méthode dans *[78]*.

2.4 Suspension des tâches

Les tâches possédant plusieurs flots de contrôle, ou *thread*, sont mis en œuvre dans la majorité des systèmes temps réel. Ces tâches sont décomposées en ensemble de threads qui sont exécutés de façon concurrente par le processeur. Les threads partage le même espace d'adressage, mais possèdent chacun une pile d'exécution qui leur est propre afin d'autoriser leurs exécutions concurrentes. Ainsi, une tâche peut posséder plusieurs flots de contrôle. Nous modélisons les tâches multi-threads par un graphe acyclique, où les sommets sont des portions de code et les arcs représentent les délais de suspension entre elles. Les *suspensions* des tâches permettent de représenter les opérations exécutées sur un autre processeur ou les opérations d'Entrée/Sortie. Les synchronisations des opérations d'entrée-sortie correspondante se font soit au début d'un thread, soit à la fin d'un thread. Sous cette hypothèse, un thread n'invoque des opérations d'entrée-sortie qu'à son début ou à sa fin. En conséquence, un thread n'est pas autorisé à se suspendre lui-même dans le corps de son exécution. Tous les threads des tâches s'exécutent sur le même processeur. Nous définissons plus formellement les tâches comme un graphe acyclique.

Définition 3 *Une tâche multi-thread τ_i est définie par un graphe acyclique (V_i,E_i):*

- *V_i est l'ensemble des sommets qui modélisent les threads:*
  $V_i = \{\tau_{ij},\ 1 \leq j \leq n_i\}$. Aucun thread ne peut se suspendre lui-même. Un thread τ_{ij} de τ_i est défini par sa pire durée d'exécution C_{ij}, et une échéance D_{ij} relative à son arrivée dans le système.
- *E_i est l'ensemble des arcs. E_i définit une relation d'ordre irréflexive sur V_i. Soit $e = (\tau_{ij},\tau_{ik}) \in E_i$ un arc connectant τ_{ij} à τ_{ik}, alors $l(e)$ est un entier non négatif définissant le délai minimum de suspension entre la fin τ_{ij} et le début de τ_{ik}.*

Soit f_{ij}^m la date de fin de la m^{th} exécution de τ_{ij} et s_{ik}^m la date de début de m^{th} exécution de τ_{ik}, alors:

$$\begin{gathered}\forall e = (\tau_{ij},\tau_{ik}) \in E_i, 1 \leq i \leq n \\ s_{ik}^m - f_{ij}^m \geq l(e) \quad \forall m \in N\end{gathered}$$

Dans la définition 3, les délais de suspension sont définis comme des entiers non-négatifs. Ceci ne pourrait pas être le cas dans des applications réelles. De plus, des bornes inférieures et supérieures des délais de suspensions peuvent être établies en utilisant des analyses meilleurs et pires cas. Alors, nous nous limitons au cas particulier où la borne inférieure est toujours égale à la borne

supérieure. En conséquence les deux bornes sont égales à un entier non-négatif. Cette hypothèse permet de simplifier les preuves de complexité.

Notons que le modèle de tâche proposé dans la définition 3 est différent de celui des tâches récurrentes présenté dans [9]. Dans ce dernier modèle, les réveils des instances de tâches dépendantes sont séparés d'un délai minimum. Dans notre modèle, les délais sont entre la fin d'une instance de thread et le début des instances d'autres threads (i.e., les successeurs dans le graphe). De plus, tous les threads (correspondant aux sommets du graphe) sont ordonnancés dans notre modèle ; alors que seuls les threads appartenant à un chemin du graphe d'une tâche seront ordonnancés dans le modèle de tâches récurrentes.

Nous définissons maintenant que les tâches séquentielles sont un cas particulier des tâches multi-threads.

Définition 4 *Une tâche multi-thread est séquentielle si son graphe est une chaîne.*

Les *threads d'entrée* (respectivement les *threads de sortie*) d'une tâche τ_i, sont ceux n'ayant pas de prédécesseur (respectivement successeur) dans le graphe associé à τ_i. Sans perte de généralité, si une tâche a plusieurs threads d'entrée, nous supposons qu'ils sont réveillés simultanément à la date r_i (date de réveil de τ_i). L'échéance de bout-en-bout d'une tâche est définie par la plus grande échéance relative des threads qui appartiennent à la tâche. Une tâche est *préemptive* si tous les threads sont préemptifs, et ainsi peuvent être préemptés n'importe quand pour être repris plus tard.

La suspension d'une tâche est différente de la préemption puisque le statut d'une tâche suspendue change (elle n'est pas éligible tant que la durée de suspension n'est pas écoulée). Les tâches non-préemptive peuvent se suspendre pour réaliser des opérations d'entrée/sortie, comme le font les tâches préemptives. De plus, les tâches avec suspension étendent le modèle classique de tâches périodiques soumises à des contraintes de précédences (il suffit de considérer des délais de suspensions nuls). Si une tâche débute son exécution pour une suspension, alors le problème de faisabilité correspondant est connu sous le nom de *release jitter problem*. En conséquence, ordonnancer des tâches avec suspension étend l'ordonnancement avec gigue sur activation.

Si tous les délais de suspension sont nuls, alors le graphe associé à une tâche est un graphe de précédence. Dans ce cas particulier, il est bien connu qu'EDF permet de respecter les contraintes de précédence si les dates de réveil et les échéances respectent l'ordre partiel induit par le graphe de précédence [95]. Mais, comme nous allons le voir, cette technique n'est pas applicable lorsque les tâches se suspendent avec des délais non nuls.

2.4.1 Résultats de complexité pour les tâches séquentielles

Dans ce paragraphe, nous montrons qu'ordonnancer les tâches préemptives ou non préemptives est $\mathcal{NP}$-difficile au sens fort.

Théorème 7 *Le problème de faisabilité d'un ensemble de tâches à démarrage simultané, périodique, séquentielle et préemptive avec au plus une suspension par tâche est $\mathcal{NP}$-difficile au sens fort.*

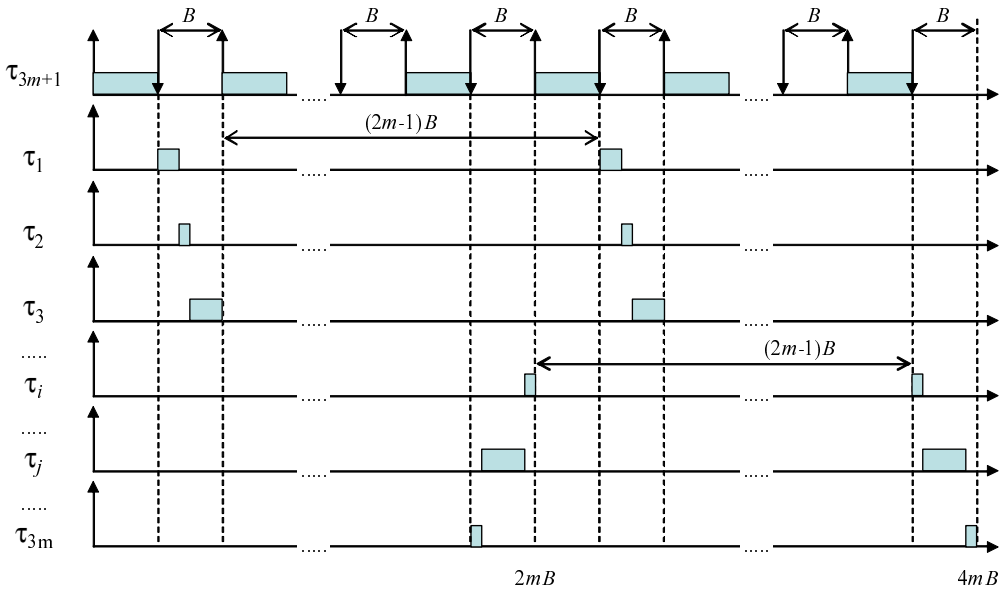


FIG. 2.12 – *Structure de tout ordonnancement faisable dans la réduction du théorème 7. La structure des intervalles* $[0,2mB)$ *et* $[2mB,4mB)$ *sont identiques et permettent d'ordonnancer les threads des tâches modélisant les éléments du problème de 3-Partition.*

Nous donnons le principe de la réduction, sans entrer dans les détails de la preuve. Nous réduisons depuis le problème de 3-Partition connu $\mathcal{NP}$-Complet au sens fort [28] :

Instance: Soit un ensemble A de $3m$ éléments, une borne $B \in N$, et une taille $s_j \in N$ pour $j = 1..3m$ telle que $B/4 < s_j < B/2$ et $\sum_{j=1..3m} s_j = mB$
Question: Peut-on partitionner A en m ensembles disjoints $A_1,A_2,...,A_m$ tels que, pour $1 \leq i \leq m, \sum_{j \in A_i} s_j = B$ (chaque A_i doit contenir exactement trois éléments de A) ?

Pour chaque instance de problème de 3-Partition, nous construisons une configuration de tâches avec suspension. Nous créons $3m + 1$ tâches :

- les tâches $\tau_1,...,\tau_{3m}$. Chacune de ces tâches est définie par une chaîne d'exactement deux threads : (τ_{i1},τ_{i2}), $1 \leq i \leq 3m$:

$$1 \leq i \leq 3m \qquad \begin{cases} C_{i1} = C_{i2} & = & s_i \\ l(\tau_{i1},\tau_{i2}) & = & (2m-1)B \\ D_i = T_i & = & 4mB \end{cases}$$

- Une tâche avec un seul thread τ_{3m+1} with $C_{3m+1} = B$, $D_{3m+1} = B$ et $T_{3m+1} = 2B$. Remarquons que cette tâche ne se suspend pas.

La figure 2.12 présente la structure de tout ordonnancement faisable. La tâche τ_{3m+1} n'a aucune laxité laissant des trous de longueur B dans l'ordonnancement. Ces trous accueilleront les tâches modélisant les éléments du problème de 3-Partition.

Le résultat suivant montre que si les tâches sont à échéance sur requête, mais peuvent comporter plus d'une suspension par tâche alors le problème de faisabilité reste $\mathcal{NP}$-difficile au sens fort. La preuve est très proche de la précédente et consiste à remplacer la tâche τ_{3m+1} par une tâche définie comme une chaîne de threads avec des suspensions de longueur B de telle façon à définir la structure d'ordonnancement présentée figure 2.12.

Théorème 8 *Le problème de faisabilité pour des tâches à démarrage simultané, périodique, séquentielle et préemptive avec des suspensions telles que $D_i = T_i$, pour $1 \leq i \leq n$, est $\mathcal{NP}$-difficile au sens fort.*

En conséquence, EDF n'est plus une règle optimale d'ordonnancement lorsque les tâches sont autorisées à se suspendre elles-même pour effectuer des opérations d'entrée/sortie. Dans un ordonnancement EDF si deux tâches ont une même échéance, alors l'une d'elles est choisie arbitrairement pour être ordonnancée. En présence de suspension, ceci ne tient plus.

Dans [43], il est démontré qu'ordonnancer des tâches non préemptives est $\mathcal{NP}$-difficile au sens fort, mais la preuve repose sur des tâches qui ne sont pas à démarrage simultané. Puisque les tâches non-préemptives peuvent être vues comme un cas particulier de tâches préemptives, nous pouvons conclure que le problème de faisabilité associé est $\mathcal{NP}$-difficile au sens fort aussi pour les tâches qui ne sont pas à démarrage simultané.

Afin de traiter les tâches avec suspensions, nous établissons le résultat suivant :

Théorème 9 *Le problème de faisabilité de tâches à démarrage simultané, périodique, séquentielle et non-préemptive avec au plus une suspension est $\mathcal{NP}$-difficile au sens fort.*

Théorème 10 *Le problème de faisabilité de tâches à démarrage simultané, périodique, séquentielle et non-préemptive avec des suspensions et $D_i = T_i$, pour $1 \leq i \leq n$, est $\mathcal{NP}$-difficile au sens fort.*

Voici une liste de problèmes de complexité ouverts pour les tâches séquentielles avec suspensions.

- périodiques, séquentielles avec des délais de suspension arbitraires et des threads de durées égales.
- périodiques, séquentielles avec des durées égales de suspension et des durées égales de threads.

Un cas particulier est défini par les tâches ayant des threads de durées égales à une unité de temps. Dans le paragraphe suivant, nous traitons le cas de tâches avec un graphe quelconque et des durées unitaires pour tous les threads.

2.4.2 Résultats de complexité pour les tâches multi-threads

Dans le paragraphe précédent, nous avons traité les tâches dont les graphes étaient définis par des chaînes. Nous considérons maintenant les tâches dont les graphes sont quelconques. Et nous établissons le résultat suivant :

Théorème 11 *Le problème de faisabilité de tâches multi-threads quelconques ayant des durées unitaires d'exécution et des délais de suspensions à valeurs entières est $\mathcal{NP}$-difficile au sens fort.*

Ce problème a été montré $\mathcal{NP}$-difficile au sens fort dans [27] (dans un cadre de tâches apériodiques). Mais ces auteurs proposent une réduction très complexe fondée sur le problème 3SAT, reprenant ainsi une idée générale présentée dans [100]. Nous proposons une preuve plus simple pour les tâches périodiques qui repose sur une réduction d'un problème d'ordonnancement multiprocesseur, appelé *precedence constrained scheduling* [28]. Ce problème est connu $\mathcal{NP}$-difficile au sens fort [100] :
Instance: Soit un ensemble de tâches S, ayant chacun une durée unitaire de traitement, $m \in N$ le nombre des processeurs, un ordre partiel $\prec$ sur S et une échéance de bout-en-bout $D \in N$ commune à toutes les tâches.
Question: Y a t'il un ordonnancement σ sur les m processeurs tel que l'échéance D et les contraintes de précédence soient respectées (i.e. tel que $t \prec t'$ implique $\sigma(t') \geq \sigma(t) + 1$)?

Nous transformons une instance de ce problème en une configuration de tâches avec suspensions. Nous définissons m tâches de la façon suivante :

- une tâche τ_1 avec un ensemble V_1 de n threads τ_{1j},$1 \leq j \leq n$,$n = |S|$ qui correspondent aux tâches de S, et un ensemble E_1 définissant le même ordre partiel de $\prec$ sur S, mais assujetti aux délais de suspensions de $m-1$ unités de temps :

$$1 \leq j \leq n \qquad \begin{cases} C_{1j} &= 1 \\ D_{1j} &= T_{1i} = (2m-1)D \end{cases}$$

$$l(e) \quad = \quad m-1, \quad \forall e = (\tau_{1i},\tau_{1k}) \in E_1$$

- $m-1$ tâches avec chacune un thread (i.e., sans suspension) τ_2,..,τ_m. Les paramètres de ces tâches sont :

$$2 \leq i \leq m \qquad \begin{cases} C_i &= 1 \\ D_i &= m-1 \\ T_i &= 2m-1 \end{cases}$$

La figure 2.13 présente la transformation d'une instance du problème multiprocesseur en notre instance du problème d'ordonnancement avec suspension. La deuxième famille de tâches s'ordonnance de la même façon dans tous les ordonnancements faisables. Cette structure va définir des trous de longueur m. Le premier de ces trous permettra d'ordonnancer les tâches exécutées à la première unité de temps dans le problème multiprocesseur, le second trou celles exécutées à le deuxième unité de temps, etc...

2.5 Collaborations et diffusion des résultats

La méthode présentée, fondée sur le dépliage du graphe de précédence généralisée en un graphe de précédence classique, a été appliquée à :

- un système industriel de suivi d'un laminoir industriel dans le processus de fabrication de boîtes en aluminium. Ce travail a été une collaboration avec le Prof. Claude Kaiser (Cedric/CNAM Paris). Une présentation complète de l'application est présentée dans

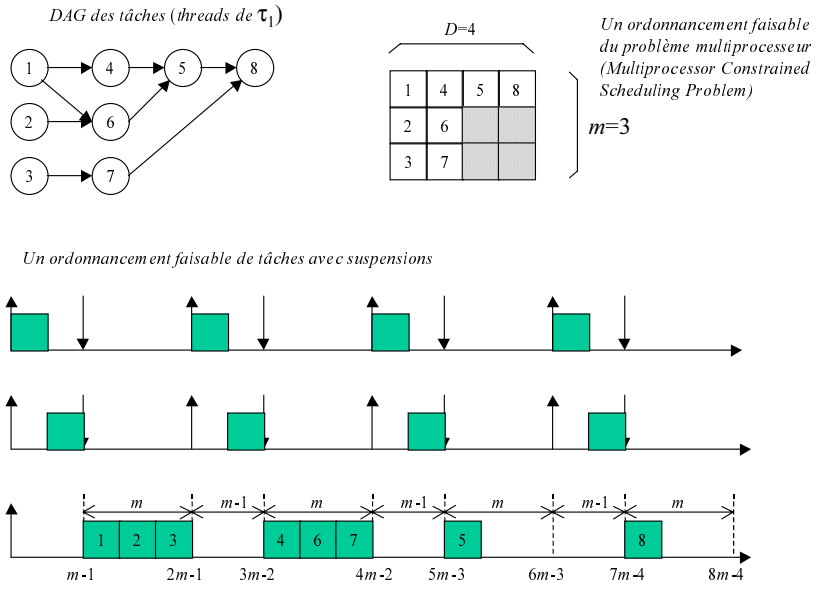


FIG. 2.13 – *Structure de tout ordonnancement faisable dans la réduction du théorème 11.*

[46]. Notre collaboration a donné lieu à une communication à la Conférence International Francophone sur l'Automatique (CIFA'00, éditions IEEE) *[85]* et une publication dans le Journal Européen des Systèmes Automatisés (JESA, éditions Hermès) *[87]*.

– un système informatique distribué. Ce travail a été présenté à la conférence 7^{th} int. Conf. on Complex Computer Systems (ICECCS'01, éditions IEEE Computer Press) *[86]*.

La contribution sur l'ordonnancement de tâches en précédence simple et à priorité fixe est une partie du travail de thèse de Mr Michaël Richard. Il a été présenté à la conférence Real-Time Systems, à Paris, en 2002 (RTS'02, éditions Teknéa) *[78]*.

Les résultats de complexité sur le problème d'ordonnancement de tâches avec suspension ont été présentés à la quinzième conférence Internationale Euromicro on Real-Time Systems, (ECRTS'03, éditions IEEE Computer Press) *[84]*.

Chapitre 3

Ordonnancement dans les systèmes distribués à priorité fixe

3.1 Résumé

3.1.1 Thématique

Le travail présenté dans ce chapitre est associé à la thèse de Mr Michaël RICHARD [72], dont j'étais co-encadrant (encadrant F. Cottet). Cette thèse s'est déroulée en trois ans et a été soutenue en décembre 2002. Ce travail s'intègre dans un programme de recherche pluridisciplinaire du douzième Contrat de Plan Etat-Région (CPER) : le programme Transport-Terrestre. Ce programme concerne l'étude des moteurs hybrides (combustion et électrique). Nous avons mené dans ce but des recherches prospectives sur l'informatique temps réel embarquée dans une automobile.

Nous avons développé deux méthodes d'ordonnancement pour les systèmes distribués à priorité fixe. La première méthode considère un placement donné des tâches, tandis que la seconde pourra conjointement à l'affectation des priorités, placer des tâches sur les ensembles de processeurs identiques. Nous illustrons enfin l'application au secteur automobile.

3.1.2 Démarche et outils

Les méthodes d'affectation des priorités et des tâches reposent sur des procédures par *séparation et évaluation* (i.e., *Branch and Bound*). Nous nous sommes appuyés sur des schémas de *séparation* utilisés classiquement dans la théorie de l'ordonnancement d'atelier ou de l'optimisation combinatoire, et sur des mécanismes d'*évaluation* propre à l'ordonnancement temps réel reposant sur le calcul des pires temps de réponse des tâches et des messages.

3.2 Affectation des priorités

3.2.1 Architecture du système distribué

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. Comme en multiprocesseur, les anomalies de Graham vont engendrer des anomalies d'ordonnancement [32]. Les techniques consistant à simuler un ordonnancement ne permettront donc pas de valider l'application. En conséquence, seule une analyse du *"pire cas"* peut être effectuée pour valider l'application (condition suffisante d'ordonnançabilité).

Nous utiliserons dans la suite les notations suivantes :

Notations :

τ_i identificateur de tâche,

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

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.

R_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.

Lorsque toutes les tâches disposent de priorité, la validation de l'application peut être effectuée par une analyse holistique [97]. Cette analyse permet de déterminer les pires temps de réponse des tâches et des messages. La validation consiste alors à vérifier que tous les pires temps de réponse des tâches et des messages ne dépassent pas leurs échéances.

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 [33, 98], qui ne parcourent pas toutes les solutions potentielles, mais seulement un sous-ensemble (i.e. méthodes heuristiques).

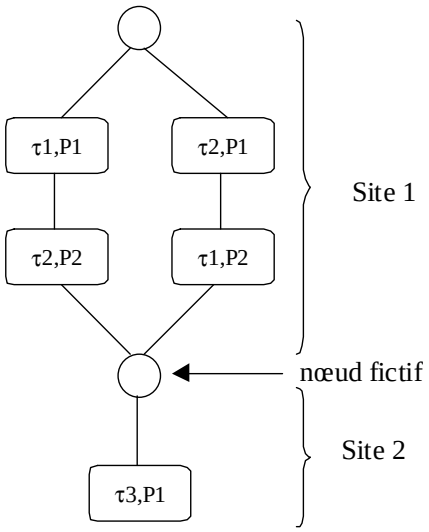


FIG. 3.1 – *Structure générale de l'arbre de recherche.*

Nous proposons dans la suite une méthode par séparation et évaluation (*Branch and Bound* pour affecter des priorités aux tâches. La méthode trouvera toujours une solution à ce problème d'affectation s'il existe une solution pouvant être validée par une analyse holistique.

3.2.2 Méthode d'affectation des priorités

Une procédure par séparation et évaluation (Branch and Bound) permet d'énumérer implicitement l'ensemble des affectations possibles des priorités aux tâches et aux messages. La procédure construit progressivement une solution en affectant à chaque étape une priorité à une tâche. Dès qu'une affectation des priorités conduit à un ordonnancement validé par l'analyse holistique, la procédure est arrêtée. Cette dernière repose sur deux fonctions: le principe de séparation et le principe d'évaluation.

Le principe de séparation indique pour un nœud donné la construction des nœuds fils (i.e. successeur immédiat dans l'arbre de recherche). Enfin le principe d'évaluation permet de déterminer si le nœud permettra de conduire à une solution réalisable. Si ce n'est pas le cas, le nœud est effacé de l'arbre de recherche et ses successeurs ne seront pas considérés dans la suite. Nous détaillons dans les parties suivantes les principes de séparation et d'évaluation.

Nous avons proposé 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 3.1 présente notre technique d'énumération. 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.

A chaque création d'un nœud, il faut pouvoir décider si la tâche correspondante est ordonnançable avec le niveau de priorité assigné. Nous décrivons dans la suite les évaluations réalisées

dans les différents cas rencontrés lors de la procédure de recherche.

Le principe d'évaluation repose sur une idée simple et originale. Nous utilisons l'analyse holistique en la faisant fonctionner sur les bornes inférieures des pires temps de réponse des tâches et messages seront évaluées ($LB(R_i)$). Les gigues sur l'activation des tâches et des messages qui en découlent sont donc elles-aussi des bornes inférieures ($LB(J_i)$). Cette évaluation est effectuée en temps pseudo-polynomial.
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 (3.1) s'effectue selon l'ordre topologique du graphe de précédence. Les valeurs obtenues par les deux équations ci-dessous (3.1) vont initialiser le système.

$$\begin{aligned} LB\left(J_i^{(0)}\right) &= \max_{j \in \Gamma_i^{-1}} \left(LB^{(0)}\left(J_j\right) + C_j\right) \\ LB\left(R_i^{(0)}\right) &= LB\left(J_i^{(0)}\right) + C_i \end{aligned} \tag{3.1}$$

Nous devons donc déterminer des bornes inférieures des pires temps de réponse des tâches et des messages. Deux cas se présentent durant l'évaluation d'un nœud quelconque de l'arbre de recherche :

- la tâche (ou le message) a déjà été affecté à un niveau de priorité.
- la tâche (ou le message) ne dispose pas de priorité.

Dans tous les cas, une borne inférieure doit être calculée. Nous renvoyons à [72] pour les détails sur les calculs des bornes. Remarquons enfin que lorsqu'une branche complète est considérée (i.e. toutes les tâches et les messages possèdent une priorité), le système résolu par notre mécanisme de borne inférieure est totalement équivalent à une analyse holistique.

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

3.3 Placement et affectation des priorités

3.3.1 Architectures matérielles et logicielles

Lors de la conception des systèmes distribués temps réel, la structure logicielle est affectée sur l'architecture matérielle disponible. L'étape de validation temporelle est alors totalement dépendante du résultat de l'allocation des différentes fonctions aux calculateurs. La validation temporelle a fait l'objet de nombreuses études dans un contexte monoprocesseur. Dans le cas d'un système distribué, elle ne dépend plus uniquement des caractéristiques des tâches, mais aussi de leur répartition sur les différents processeurs. Nous proposons une méthode de placement et d'ordonnancement des tâches telle que leurs spécifications temporelles soient respectées. L'originalité de notre méthode, vis-à-vis des travaux déjà effectués sur le sujet, est d'une part, d'effectuer le placement et l'ordonnancement simultanément ; et d'autre part, le cas échéant, d'établir l'inexistence d'une solution validable par une analyse holistique.

Nous présentons ci-dessous les architectures logicielles et matérielles étudiées ainsi que les hypothèses et les contraintes à considérer.

Architecture logicielle

Chaque site (i.e. processeur) présent dans l'application exécute un système d'exploitation temps réel ordonnançant les tâches à priorités fixes réalisant les fonctions de l'application. Chaque tâche τ_i est définie comme un quadruplet (C_i,D_i,T_i,π_i) où :

- C_i est le pire temps d'exécution,
- D_i est l'échéance relative au réveil de τ_i,
- T_i est la période des activations de τ_i,
- π_i est la priorité de τ_i, locale au processeur sur lequel elle s'exécute.

Si notre méthode supporte les tâches dont les échéances sont non reliées aux périodes, nous considérons dans la suite, afin de simplifier les formules, des tâches à échéances reliées aux périodes (i.e. $D_i < T_i$ ou $D_i = T_i$).

Les tâches sont ordonnancées en fonction de leurs priorités par un algorithme d'ordonnancement préemptif fourni en standard dans le système d'exploitation. À chaque instant, la tâche possédant le niveau de priorité le plus important obtient le processeur. Nous rappelons que la priorité π_i d'une tâche τ_i est fixe durant toute la vie de l'application. La valeur 0 désigne le plus fort niveau de priorité. Notons enfin que les tâches sont toutes périodiques et activées au départ de l'application.

Nous faisons l'hypothèse que les réceptions des messages s'effectuent au début de l'exécution d'une tâche et les émissions s'opèrent à la fin de l'exécution de celle-ci. Ceci implique qu'aucune communication ne peut avoir lieu dans le corps d'une tâche.

Les tâches peuvent communiquer et/ou se synchroniser au moyen de deux mécanismes. Si deux tâches s'exécutent sur le même processeur, la communication s'effectuent via la mémoire associée localement au processeur. Cette communication est alors modélisée par une contrainte de précédence. Dans le cas contraire, les deux tâches échangent des messages via le réseau. Notons qu'une tâche peut recevoir et envoyer plusieurs messages. De plus, un même message peut être émis vers plusieurs destinataires. Un message m_i est défini comme un triplet (n_i,D_i,T_i) où n_i est le nombre d'octets constituant le message, D_i l'échéance relative à la date d'envoi du message et T_i la période d'envoi du message. Nous utilisons les mêmes notations que dans le paragraphe précédent.

Architecture matérielle

Nous considérons des architectures multiprocesseurs sans mémoire partagée interconnectées par un ou plusieurs réseaux. Ce modèle d'architecture englobe le cas simple d'un unique processeur (i.e. contexte monoprocesseur). Chaque site dispose d'un processeur Pr_i et d'une mémoire locale autorisant les communications entre deux tâches s'exécutant sur le site. Deux tâches ordonnancées sur deux sites différents ne communiquent que par messages transitant sur le réseau.
Afin de pouvoir considérer les contraintes de placement, nous définissons ci-dessous la notion de *pool de processeurs*.

Définition 5 *Un pool de processeurs Pl_i est défini par un couple (PR_i,Θ_i) où :*

- *PR_i est un ensemble de m_i processeurs identiques,*

- Θ_i *est un ensemble de* n_i *tâches à placer sur les processeurs* $Pr_i \in PR_i$.

Dans le cas d'un pool de processeurs Pl_i, l'ensemble tâches Θ_i est fixe (i.e. le nombre de tâches le composant est fixe et connu).

3.3.2 Placement des tâches et affectation des priorités

Notre méthode de placement et d'affectation des priorités aux tâches et 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. Nous nous appuyons à nouveau sur une procédure par séparation et évaluation. Pour énumérer sans redondance le placement des tâches et l'affectation des priorités nous adaptons le principe d'énumération conçu dans [11]. Ces auteurs présentent une façon d'énumérer, sans répétition, tous les arrangements possibles de n tâches sur m processeurs identiques. Ils utilisent pour cela deux types de nœuds : les nœuds ronds et les nœuds carrés. Si le chemin passe par un nœud rond, la tâche est placée sur le processeur courant. Si, en revanche, le chemin passe par un nœud carré, alors la tâche correspondante est affectée au processeur suivant, ce dernier devenant alors le processeur courant. Les priorités sont affectées dans l'ordre décroissant des niveaux de priorités. Ainsi, les tâches modélisées par un nœud carré possèdent le niveau de priorité le plus fort sur le nouveau processeur. Notons que l'affectation des priorités est locale à un processeur. Ceci implique qu'à chaque changement de processeur (i.e. chaque nœud carré dans la branche) les priorités sont réinitialisées. Ainsi, une tâche modélisée par un nœud carré possède le niveau de priorité le plus fort, c'est-à-dire la priorité 0. La construction de l'arbre de recherche repose sur le respect de l'ensemble de règles. Ces règles permettent d'une part de parcourir l'ensemble des solutions potentielles, et d'autre part d'éviter les phénomènes de redondances.
La figure 3.2 présente la structure arborescente globale sur laquelle s'appuie notre méthode. Les différents sous arbres, modélisant les pools de processeurs et les réseaux, sont obtenus par l'application des règles présentées ci-dessus. Cette structure de sous arbre permet de modéliser un ensemble de m processeurs identiques. La modélisation de p ensembles de m_k processeurs identiques, $k \in \{1,\ldots,p\}$, différents entre eux est réalisée par la juxtaposition de p sous-arbres. Chaque sous-arbre modélisant un pool de processeurs est relié au suivant par un nœud fictif. Ceci sous-tend que toutes les feuilles d'un sous-arbre sont reliées au *même* sous-arbre inférieur. Nous renvoyons à la thèse de Michaël Richard pour les détails sur les règles de construction de l'arbre de recherche.

3.3.3 Évaluation

À chaque création d'un nœud, il faut pouvoir décider si la tâche est ordonnançable au niveau de priorité choisi sur le processeur courant. Or, il n'existe pas de condition nécessaire et suffisante d'ordonnançabilité pour les systèmes distribués à priorités fixes. Dans la suite, après avoir brièvement rappelé les principes de la méthode pire cas sur laquelle s'appuie notre méthode, nous présentons les règles d'évaluation de la solution en cours de construction dans les différents contextes pouvant se présenter lors de la procédure de recherche arborescente.

La technique d'évaluation est basée sur l'analyse holistique présentée dans [97]. Remarquons que celle-ci fait l'hypothèse que les tâches sont indépendantes sur chaque processeur. Nous levons

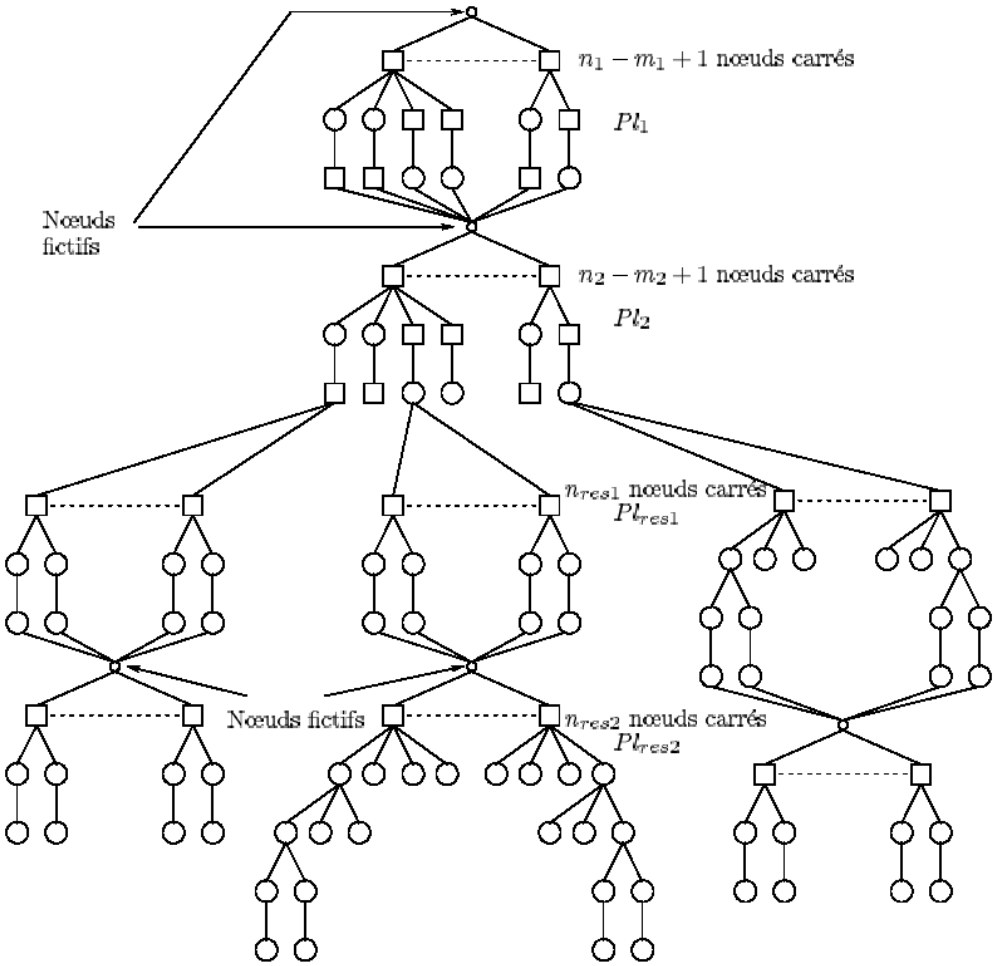


FIG. 3.2 – *Structure générale de l'arbre de recherche.* ○ *est l'affectation d'une priorité à une tâche sur le processeur courant du pool courant.* □ *est l'affectation d'une priorité à une tâche sur le processeur suivant dans le pool courant.* n_i *est nombre de tâches dans le pool* i. m_i *est nombre de processeurs dans le pool* i.

cette hypothèse en assurant une affectation des priorités aux tâches respectant l'ordre partiel. Le mécanisme des priorités assure le respect des contraintes de précédence.
Lorsqu'une tâche est placée à un niveau de priorité, un nœud est créé. Afin de déterminer si le placement est valide, nous évaluons une borne inférieure $LB(R_i)$ du pire temps de réponse R_i. Pour cela nous reprenons le système de recherche de point fixe de l'analyse holistique. La fonction d'évaluation des temps de réponse, *Evaluer*, est modifiée ainsi que la fonction de mise à jour des gigues sur l'activation, *Propager*.

$$1 \leq i \leq n \quad \begin{cases} LB\left(R_i^{(k)}\right) & = & Evaluer\left(LB\left(J_i^{(k-1)}\right)\right) \\ LB\left(J_i^{(k)}\right) & = & Propager\left(LB\left(R_j^{(k)}\right)\right) \end{cases} \tag{3.2}$$

Le point fixe est obtenu lorsque pour le plus petit $k \in \mathbb{N}$:

$$LB(R_i) = LB\left(R_i^{(k-1)}\right) = LB\left(R_i^{(k)}\right),\ 1 \leq i \leq n \tag{3.3}$$

L'évaluation des bornes inférieures des temps de réponse est appliquée sur un nœud donné, c'est-à-dire pour un placement des tâches et une affectation des priorités incomplets. Les calculs dépendent donc du contexte du nœud analysé, c'est-à-dire :

- du placement : le placement des différentes tâches sur les processeurs va influer sur le graphe des communications, et donc sur le calcul des bornes inférieures des gigues sur l'activation.
- du traitement : si le placement et l'affectation des priorités ont été effectués ou non.
- du type de pool : si le pool est régi par une politique d'ordonnancement préemptive (processeur), ou non préemptive (réseau).

Le principal intérêt du système (3.2) est de propager les résultats, c'est-à-dire les bornes inférieures, de la fonction *Evaluer* aux tâches et aux messages via la mise à jour des gigues sur l'activation, et ce, de manière immédiate, par l'intermédiaire de la fonction *Propager*. À chaque fois qu'un nœud est inséré dans l'arbre de recherche, le système (3.2) est résolu pour l'ensemble des tâches et messages de l'application. Or, le nombre de messages circulant sur le réseau dépend totalement du placement des tâches. La valeur n peut non seulement être différente entre deux branches de l'arbre, mais aussi au cours du parcours d'une même branche. Ainsi, au début de la procédure de recherche, la valeur n correspond au nombre de tâches puisqu'on ne connaît pas le nombre de messages sur le réseau.

Nous renvoyons à [72] pour les détails concernant les fonctions *Propager* et *Evaluer* dans les différents contextes rencontrés au cours de la construction d'une branche de l'arbre de recherche. La mise à jour est différente dans le cas d'une tâche ou d'un message.

3.3.4 Expérimentations numériques

Nous avons créé un générateur aléatoire d'applications temps réel. Le nombre de sites, le nombre de réseaux, la charge globale et le nombre de communications sont des paramètres réglables du générateur. Les tâches générées sont à échéance sur requête, et le nombre de pools de processeurs est fixé à 1. Afin de produire des configurations réalistes, la génération des durées d'exécution et des périodes suit deux lois normales différentes. La loi utilisée pour les durées d'exécution est paramétrée par : $m = 1$, et $\sigma = 12$. Concernant les périodes, les paramètres m et σ sont dépendant du facteur de charge globale désiré. Dans le contexte distribué, la génération des

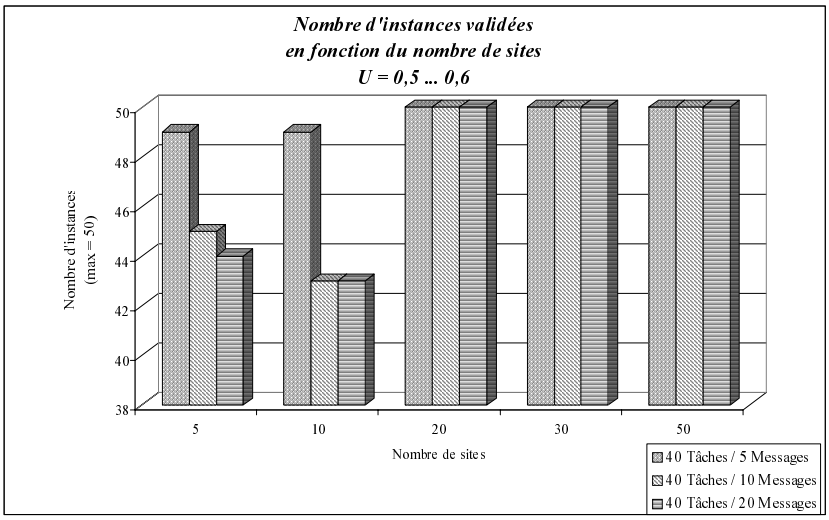


FIG. 3.3 – *Influence du nombre de sites et de messages sur le nombre d'instances validées*

communications est réalisée par un tirage aléatoire de l'émetteur, du récepteur et de la taille de la section de données du message. Période et échéance sont héritées des tâches émettrices et réceptrices. Enfin, dans le but d'obtenir des résultats objectifs, nous produisons aléatoirement 50 configurations différentes pour chaque taille de problème. Si le temps de calcul est supérieur à une heure, la recherche est stoppée.

Architecture distribuée

La figure 3.3 présente l'évolution du nombre de configurations validées en fonction du nombre de sites pour une charge fixe variant entre 0,5 et 0,6. Le nombre de tâches est fixe (40) et le nombre de messages varie entre 5 et 20. Quelle que soit la taille du problème, le nombre d'instances validées est supérieur ou égal à 43, soit 86%. Ce résultat est obtenu pour des problèmes composés de 5 ou 10 processeurs et d'au moins 10 messages. Lorsqu'au moins 20 sites sont disponibles, toutes les instances sont validées. Le pourcentage d'instances validées pour des problèmes de taille réaliste montre l'intérêt de notre méthode.

Les courbes de temps de calculs de la figure 3.4 correspondent aux traitements des instances de la figure 3.3. Ces valeurs sont des moyennes et, pour les trois premières courbes, intègrent les configurations non résolues en moins d'une heure. Le temps de recherche moyen, dans ce cas, est toujours inférieur à 1000 secondes quelle que soit la taille du problème traité. En réalité, le temps moyen (i.e. réel cf. figure 3.4) permettant de valider une configuration ne dépasse pas 35 secondes pour cette taille d'instance. La méthode supporte donc des applications de taille réaliste.

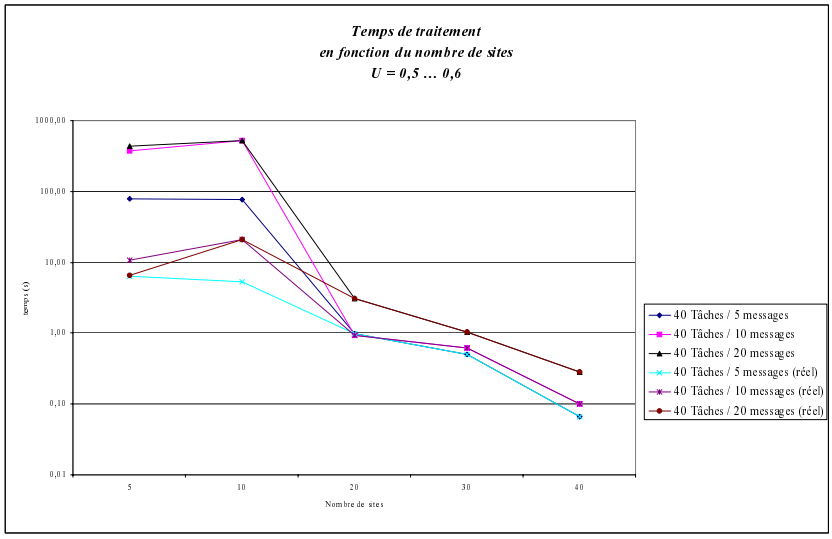


FIG. 3.4 – *Évolution du temps de traitement en fonction du nombre de sites et de messages*

Comparaison des différents parcours

À ce niveau, nous avons implémenté deux types de parcours différents : un parcours dit *"de remplissage"* et un parcours dit *"d'équilibrage"*.

- *Parcours de remplissage :* dans ce cas, les nœuds favorisés sont les nœuds ronds. Ainsi tant que la règle de charge est respectée, et que les tâches modélisées par les nœuds ronds sont ordonnançables, elles sont ajoutés sur le même processeur. Les solutions obtenues sont alors déséquilibrées en terme de répartition des tâches sur les processeurs du pool.
- *Parcours d'équilibrage :* afin d'estomper le phénomène précédent, nous avons implémenté un parcours visant à construire en premier lieu les solutions les plus équilibrées. Précisément, nous ajoutons un critère de construction permettant de créer le plus rapidement possible des solutions pour lesquelles le placement des tâches sur les processeurs du pool est équilibré. Ainsi, pour un pool Pl_i composé de n_i tâches, m_i processeurs et possédant une charge globale U_{pl_i} deux stratégies sont envisagées :
  1. *fonction du nombre de tâches :* les nœuds ronds sont favorisés (i.e. construit en premier) tant que le nombre de tâches placées sur le processeur en cours de traitement est inférieur à :

     $$\left\lceil \frac{n_i}{m_i} \right\rceil$$

     Lorsque cette borne est atteinte, les nœuds carrés sont construits en premier permettant ainsi de changer de processeur.
  2. *fonction de la charge :* de la même manière que précédemment, les nœuds ronds sont favorisés tant que la charge induite par les tâches déjà placées sur le processeur en cours de traitement est inférieure à :

     $$\frac{U_{Pl_i}}{m_i}$$

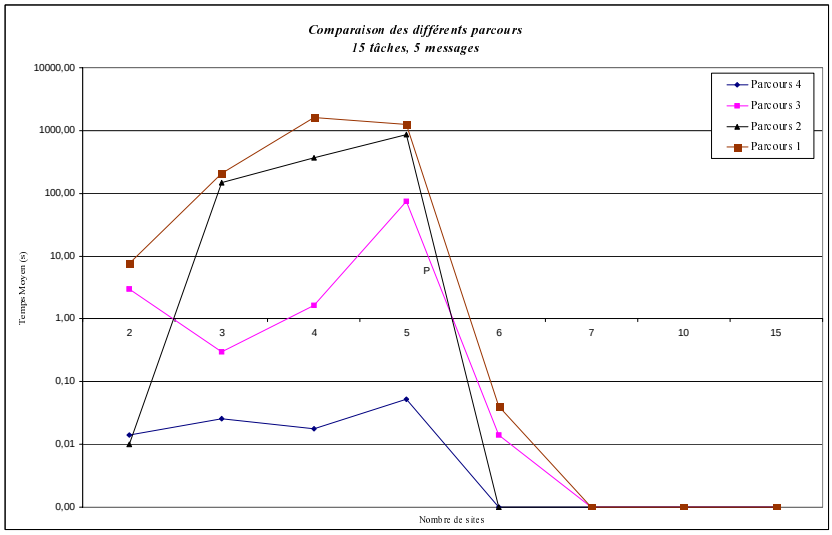


FIG. 3.5 – *Temps de calcul pour les différents types de parcours*

Nous proposons dans ce paragraphe une comparaison des performances des différents types de parcours présentés. Précisément, chaque parcours est l'association d'une stratégie de parcours au sein d'un pool de processeur et d'une stratégie de parcours global de l'arbre de recherche :

- *le parcours 1 :* est l'association d'un parcours de remplissage et d'un parcours en profondeur simple de la structure arborescente globale,
- *le parcours 2 :* est l'association d'un parcours d'équilibrage en fonction du nombre de tâches et d'un parcours en profondeur simple de la structure arborescente globale,
- *le parcours 3 :* est l'association d'un parcours de remplissage et d'un parcours en parallèle auto adaptatif de la structure arborescente globale. L'arbre contenant le plus de solutions en cours de contruction est privilégié,
- *le parcours 4 :* est l'association d'un parcours d'équilibrage en fonction du nombre de tâches et d'un parcours en parallèle auto adaptatif de la structure arborescente globale. L'arbre contenant le plus de solutions en cours de construction est privilégié.

La figure 3.5 présente les temps de traitement moyens en fonction du nombre de sites, pour 50 configurations traitées, de ces différents parcours. La configuration est composée de 15 tâches et 5 messages. Les parcours 3 et 4, qui effectuent un parcours en parallèle auto adaptatif, affichent des performances nettement supérieures aux parcours 1 et 2. La stratégie de parcours au sein d'un pool de processeurs améliore encore les temps de calculs (parcours 2 et 4) quel que soit le type de stratégie de parcours de la structure arborescente globale.

3.4 Application aux systèmes embarqués dans l'automobile

Afin d'illustrer le type d'application à valider, nous décrivons figure 3.6 l'architecture informatique embarquée pour la gestion d'une automobile [17]. 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 [40], dédié aux fonctions critiques de l'automobile, telle que l'ABS et le contrôle moteur et le réseau VAN [41] utilisé pour relier les ECU assurant des fonc-

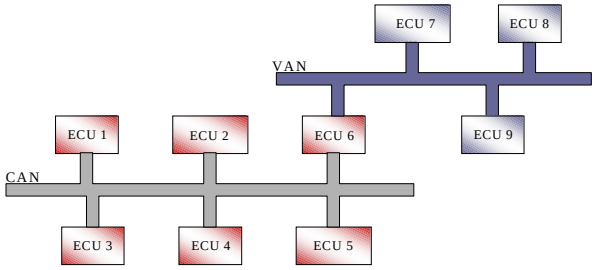


FIG. 3.6 – *Structure d'un système embarqué dans l'automobile*

tions 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.

Cette application est composée de 44 tâches s'exécutant sur les différents ECU et 19 messages circulant sur les deux réseaux. Les différentes charges et la répartition des tâches et messages sont présentés dans *[75]*. Le choix des priorités des tâches et des messages est prépondérant pour ne pas sur-dimensionner le système.

Nous avons tiré aléatoirement des configurations de tâches respectant l'architecture présentée (le nombre de processeurs, de réseaux de tâches, de messages ainsi que les contraintes de précédences entre les tâches et les messages sont définitivement fixés). Nous tirons aléatoirement les paramètres temporels des tâches en suivant une loi uniforme et en contrôlant le facteur d'utilisation de chaque processeur. Trente problèmes ont été générés pour les intervalles de charges de processeurs : [0,2-0,3] jusqu'à [0,8-0,9]). La limite d'exécution du programme a été fixée à une heure.
La figure 3.7, dans le graphique de gauche, donne le nombre de problèmes résolus dans la limite de temps. Ce graphique illustre que le principe d'évaluation n'est pas trop pessimiste puisque des solutions sont trouvées alors que les processeurs sont chargés à 60 pour-cent. Le graphique de droite de cette même figure donne la durée moyenne de résolution pour chaque taille de problème (i.e., intervalle de facteur de charge). Ceci montre que lorsque les processeurs sont faiblement chargés alors il est facile de trouver une affectation des priorités conduisant à une validation de l'application. De même, lorsque les processeurs sont fortement chargés, la méthode montre rapidement qu'il n'existe pas d'ordonnancement faisable.

3.5 Collaborations et diffusion des résultats

Ces travaux font l'objet de la thèse de Mr. Michaël Richard [72]. La partie affectation des priorités a été présentée à la conférence Real-Time Systems (RTS01, editions Teknéa), et la conférence internationale on Fieldbus systems and their applications *[75]* (FET01, éditions Elsevier). La partie avec placement a été présentée à la conférence Real-Time Systems *[77]* (RTS'03, éditions Teknéa), à la conférence internationale Emerging Technologies and Factory Automation *[76]* (ETFA'03, éditions IEEE) et dans la revue Technique et Science Informatiques *[74]* (TSI, 2003, éditions Hermès).

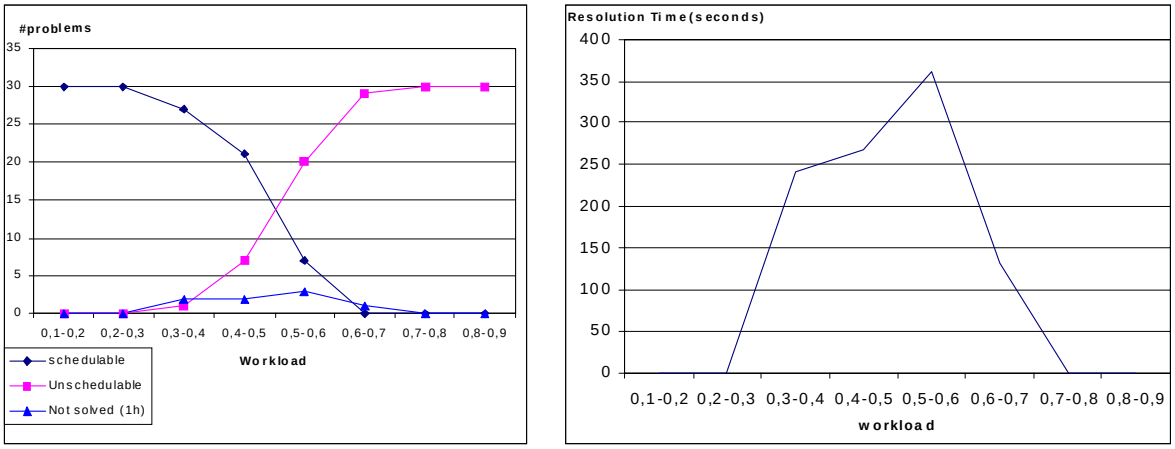


FIG. 3.7 – *Nombre d'instances résolues en 1 heure (a) et temps de résolution moyen (b)*

Chapitre 4

Aide à la conception des applications temps réel

4.1 Résumé

4.1.1 Thématique

Dans ce chapitre nous présentons des méthodes d'aide à la conception ou au paramétrage des applications temps réel. Nous présentons tout d'abord, une méthode graphique pour définir les valeurs possibles des paramètres des tâches. Cette méthode permet de définir les caractéristiques des tâches garantissant l'ordonnançabilité d'un système monoprocesseur. Dans un second temps nous présentons une méthode générique d'affectation des priorités fixes aux tâches afin d'optimiser un critère de Qualité de Service (QdS). Nous présentons ensuite la spécialisation de la méthode au critère de minimisation du temps de réponse moyen des tâches. Enfin, nous traitons de l'ordonnancement de tâches sur des processeurs à vitesse variable en vue de minimiser l'énergie consommée par les tâches.

4.1.2 Démarche et outils

L'approche graphique d'aide à la conception repose sur la représentation graphique des valeurs admissibles des paramètres des tâches en conservant l'ordonnançabilité de l'application temps réel. Cette approche permet d'analyser la sensibilité d'un paramètre ou deux paramètres sur l'ordonnançabilité de l'application. Des tests classiques d'ordonnançabilité sont utilisés comme heuristiques pour améliorer la détermination des valeurs admissibles des paramètres. Les points restants sont ensuite analysés par construction explicite de l'ordonnancement (i.e., simulation).

Le calcul des priorités afin d'optimiser un critère de performance repose sur des procédures par séparation et évaluation. Nous avons séparé la partie générique de la méthode, qui est indépendante du critère de performance optimisé, et la partie spécifique au critère à optimiser.

L'ordonnancement de tâches sur des processeurs à vitesse variable est traité à l'aide de la programmation non linéaire en nombres entiers (e.g., la fonction objectif est quadratique et les contraintes sont linéaires), des heuristiques dont un recuit simulé. L'étude se limite aux configurations de tâches pour lesquelles une condition nécessaire et suffisante d'ordonnançabilité est calculable en temps polynomial (i.e., tâches à départ simultané et à échéance sur requête).

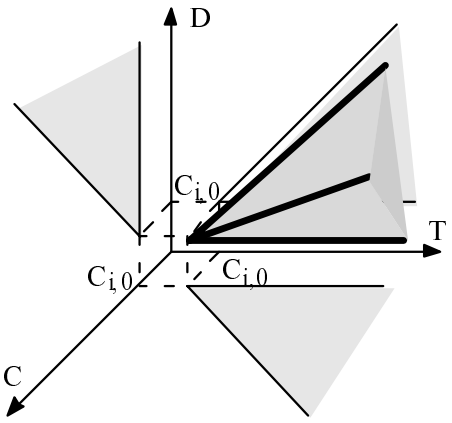


FIG. 4.1 – *Modèle graphique tridimensionnel de représentation d'une tâche*

4.2 Approche graphique d'aide à la conception des applications

4.2.1 Représentation graphique des paramètres des tâches

Nous présentons une méthode d'aide à la conception d'application temps réel. L'objectif est de déterminer l'ensemble des valeurs des possibles des paramètres conduisant à un système ordonnançable.

Chaque tâche τ_i peut être représentée par un graphique tridimensionnel selon les axes de sa durée C_i, son échéance relative D_i et sa période T_i. La représentation graphique de l'évolution conjointe de ces trois paramètres est donnée figure 4.1. L'analyse de l'évolution possible des paramètres est plus simple à appréhender en projetant les points sur les trois plans possibles (C_i,D_i), (D_i,T_i) et (C_i,T_i). Sur chacun de ces plans s'intègrent les contraintes usuelles d'ordonnancement : $C_i \leq D_i \leq T_i$. Dans de nombreux cas industriels, chaque paramètre de tâche sera défini par un intervalle de valeurs possibles. Par exemple, la période d'une tâche τ_i réalisant un échantillonnage est généralement définie par une période maximale T_{max}, et l'utilisation d'une période minimale T_{min}.

La méthode consiste à tracer le domaine d'ordonnançabilité d'une tâche en tenant compte de la ressource processeur utilisée par les autres tâches critiques du système. En pratique, ceci revient à vérifier pour chaque point du triplet (C_i,D_i,T_i) que la configuration de tâches est ou non ordonnançable. Si le point est non ordonnançable, alors il ne sera pas inclus dans la zone ordonnançable. Bien sûr, il sera intégré dans le cas contraire. Puisque la charge est nécessairement inférieure ou égale à 1, on vérifie :$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$. Par conséquent, les autres tâches de la configuration ayant des paramètres préalablement déterminés et fixés, une tâche τ_i de la configuration doit avoir sa période encadrée par la relation suivante :

$$T_i \geq \frac{C_i}{1 - \sum_{k=1,k\neq i}^{n} \frac{C_k}{T_k}} = T_u$$

Ceci est représenté par la figure 4.2.

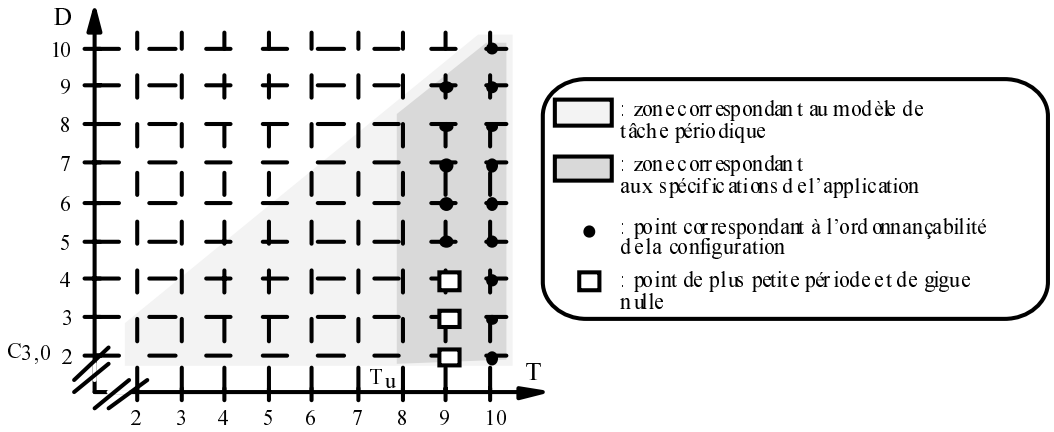


FIG. 4.2 – *Représentation des points d'ordonnançabilité des tâches en fonction des valeurs dans le plan* (D,T). T_u *est une borne inférieure de la période minimum de la tâche.*

4.2.2 Amélioration de l'analyse

Analyser chaque point séparément est très coûteux en temps puisqu'une analyse d'ordonnançabilité est nécessaire pour chacun d'entre eux. Nous avons proposé des tests permettant de propager l'ordonnançabilité d'un point à ses voisins. Par exemple, si une tâche est ordonnançable pour un point $D_i = k$, alors les points où D_i est égal à $k+1$,$k+2$,... seront eux aussi ordonnançables, puisqu'augmenter l'échéance d'une tâche ne peut rendre une autre tâche non ordonnançable. Nous avons donc proposé un ensemble de propriétés graphiques, permettant de propager localement des résultats sur un point donné. Nous renvoyons à *[73]* pour l'ensemble des détails. Ceci est illustré dans la figure 4.3.

4.3 Optimisation de la Qualité de Service

L'intégration de contraintes de Qualité de Service (QdS) dans les applications temps réel à contraintes temporelles strictes conduit les concepteurs d'application à proposer des algorithmes d'ordonnancement tentant d'optimiser un critère de performance tout en garantissant les échéances des tâches. Dans la suite nous nous intéressons à la minimisation des pires temps de réponse moyen dans les systèmes de tâches à priorité fixe. Cette première étude ayant été concluante, nous avons cherché à rendre cette méthode générique dans le sens où elle puisse être facilement adaptée à d'autres critères de performance.

Dans les systèmes monoprocesseurs, minimiser le temps de réponse moyen est équivalent à la minimisation de la moyenne des dates de fin des instances des tâches. L'intérêt pratique de minimiser le temps de réponse moyen est double, puisque cela revient simultanément à [3] :

- minimiser le nombre moyen d'instances de tâches non terminées dans le système,
- minimiser le nombre moyen d'instances des tâches en attente du processeur après leurs réveils.

En conséquence, le système sera plus réactif en garantissant un flux de traitement plus élevé des

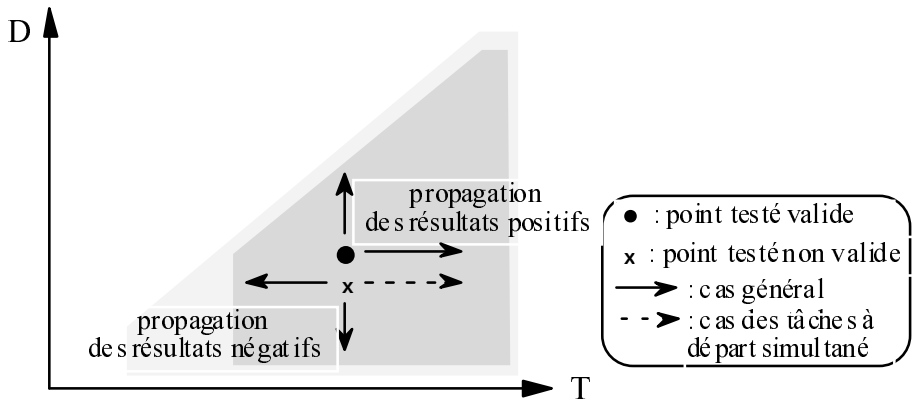


FIG. 4.3 – *Représentation des points d'ordonnançabilité des tâches en fonction des valeurs dans le plan* (D,T)

instances des tâches. Lorsque les tâches ne sont pas soumises à des échéances impératives, la minimisation du temps de réponse moyen est obtenu en choisissant à chaque instant la tâche dont la durée restante à exécuter est la plus petite (SRPT : Shortest Remaining Processing Time) [3]. Bien sûr, en présence d'échéance stricte, une telle politique d'ordonnancement ne garantira pas le respect des échéances. Nous proposerons dans la suite une méthode d'affectation des priorités fixes aux tâches afin de minimiser :

- la moyenne pondérée des pires temps de réponse des tâches,
- la moyenne pondérée des temps de réponse des tâches (tenant compte de toutes les instances).

La moyenne pondérée associe un poids w_i à chaque tâche. Ce paramètre numérique permet d'indiquer *l'importance* de la tâche vis-à-vis du critère optimisé. Par exemple, une tâche τ_i n'entrant pas en compte dans la qualité de service du système se verra attribuer le poids $w_i = 0$. Elle n'aura donc aucune incidence sur la fonction objectif optimisée.

4.3.1 Minimisation des pires temps de réponse moyen

Nous considérons des tâches périodiques, à démarrage simultané, telles que $C_i \leq D_i \leq T_i, i = 1, \ldots, n$. Le pire temps de réponse d'une tâche est alors facile à calculer à l'aide de la recherche du plus petit point fixe de la fonction demande processeur (c.f. chapitre 1) : $W_i(t) = t$. Ce calcul est fait en temps pseudo-polynomial pour chaque tâche.

L'objectif est de minimiser la moyenne pondérée des pires temps de réponse des tâches. Cette étude a été la première initiée dans ce domaine d'optimisation de performance. Le critère est donc :

$$Min \sum_{i=1}^{n} w_i R_i$$

Algorithme 1: Procédure par séparation et évaluation

```
Calculer une borne supérieure avec une heuristique ;
Initialiser l'ensemble des noeuds actifs A avec les sommets 1...n (premier niveau de
l'arbre de recherche);
while A ≠ ∅ do
    Sélectionner un sommet avec la règle de sélection ;
    if c'est une feuille et que le critère est amélioré then
        mettre à jour la meilleure solution;
    else if ce n'est pas une feuille then
        Générer l'ensemble B des noeuds fils selon la règle de branchement;
        Calculer la borne inférieure de chaque fils dans B;
        Eliminer de B les noeuds selon la règle d'élimination ;
        Trier les sommets de B dans l'ordre inverse de la règle de sélection;
        Transférer les noeuds de B dans l'ensemble A;
    Supprimer le noeud séparé de A;
end
```

Procédure par séparation et évaluation

Nous avons défini une méthode d'affectation des priorités fondée sur une procédure par séparation et évaluation (Branch and bound). Le principe de ces algorithmes est d'énumérer de façon intelligente toutes les solutions possibles. Les solutions sont stockées dans un arbre de recherche. Chaque nœud définit une solution partielle au problème. Dans notre cas, une solution partielle sera une assignation partielle des priorités aux tâches. Nous supposerons dans ce paragraphe que les niveaux de l'arbre constitue les niveaux de priorités croissants des tâches. Ainsi une tâche représentée dans le premier niveau de l'arbre de recherche sera en pratique assignée à la priorité la plus faible du système. Par contre, une tâche modélisée par une feuille de l'arbre sera affectée au niveau de priorité le plus élevé.

En pratique, l'arbre de recherche est stocké dans une liste où chaque élément mémorise une branche de l'arbre de la racine au nœud courant. Cette liste est appelée la liste des nœuds actifs, car ils n'ont pas encore été considérés par l'algorithme. La façon de gérer cette liste va définir l'ordre de parcours de l'arbre de recherche. Ce parcours sera donc défini par les règles suivantes :

- la règle de sélection : qui détermine le prochain nœud à séparer (i.e., le nœud courant).
- la règle de branchement : qui définit l'ordre des fils du nœud séparé dans l'arbre de recherche.
- la règle d'élimination : qui détermine les nœuds qui ne pourront améliorer la solution ou respecter les contraintes du problème. Pour chaque fils est calculée une borne inférieure du critère. Si cette borne est supérieure à la meilleure solution connue, alors le nœud ne pourra pas conduire à l'amélioration du critère. Ses fils ne seront pas construits.

L'algorithme 1 donne le pseudo-code d'une procédure par séparation et évaluation.

Algorithme 2: Heuristique d'initialisation - Borne Supérieure

Soit X l'ensemble des tâches et n le nombre de tâches;
Soit $z = 0$ la fonction objectif;
while $X \neq \emptyset$ **do**
  Soit X' les tâches pouvant être ordonnancées au niveau de priorité n parmi les tâches de X;
  Soit τ_i celle avec le plus petit poids w_i dans X';
  Affecter τ_i au niveau de priorité n;
  $n = n - 1$;
  $X = X \setminus \{a\}$;
end
Calculer la fonction objectif avec les priorités définies;

Borne supérieure

La borne supérieure permet de définir la valeur initiale de la procédure par séparation et évaluation. Dans le contexte de tâches périodiques, Deadline Monotonic est une règle d'affectation des priorités conduisant à un ordonnancement faisable. Toutefois, cette approche ne tient pas compte du critère optimisé et conduira dans la majorité des cas à une borne de très mauvaise qualité. Nous avons proposé une heuristique permettant de tenir compte du critère optimisé, tout en garantissant le respect des échéances. Son pseudo-code est présenté dans l'algorithme 2.

L'affectation des priorités se fait du niveau le plus faible jusqu'au plus élevé. Pour chacun des niveaux, nous déterminons parmi les tâches non assignées à une priorité celles qui seront ordonnançables au niveau de priorité courant. On choisit ensuite la tâche qui augmente le moins le critère optimisé.

Borne inférieure

Nous trions les tâches dans l'ordre non-décroissant des ratios C_i/w_i (règle connue sous le nom de Modified Smith's rule), puis nous relaxons trois contraintes du problème, ainsi :

- la préemption n'est plus autorisée,
- les tâches sont apériodiques,
- les tâches ne sont pas soumises à une échéance stricte.

Nous ordonnançons ainsi les tâches sans préemption, puis leurs dates de fin sont calculées. Enfin, la fonction objectif est calculée. L'algorithme 3 présente le pseudo-code de l'algorithme calculant cette borne inférieure. Dans cet algorithme, la variable $v.S$ est l'ensemble des tâches avec une priorité pour le sommet v. Ainsi, la contribution au critère des tâches avec une priorité est calculée, puis celle des tâches sans priorité en les ordonnançant selon l'ordre non décroissant des C_i/w_i.

Règles d'élimination

Il existe trois possibilités pour couper une branche dans l'arbre de recherche en fonction du nœud courant :

- la régle de sélection immédiate,
- une échéance n'est pas respectée,

Algorithme 3: Algorithme de détermination de la borne inférieure

Entrée: Sommet v ;
Sortie: Borne inférieure lb ;
Variable : Entier z, Ensemble des tâches U ;
$lb = 0$;
for *toute tâche i dans v.S* **do**
  ;
  $lb = lb + w_i \times R_i$;
end
$U = T - S$;
$z = 0$;
for *toute tâche* $j \in U$ *dans l'ordre non décroissant des* C_i/w_i **do**
  $z = z + C_j$;
  $lb = lb + z * w_j$;
end
Retourner lb;

– la borne inférieure est supérieure à la meilleure solution connue (i.e., la borne supérieure).

Nous détaillons uniquement ci-après la règle de sélection immédiate qui repose sur le résultat suivant :

Théorème 12 *Pour toute paire de tâche* (τ_i,τ_j)*, si:*

$$\left\lceil \frac{T_i}{T_j} \right\rceil \times C_j + C_i > D_i \tag{4.1}$$

alors la priorité de τ_i *doit être supérieure à celle de* τ_j*.*

Expérimentations numériques

Les expérimentations numériques ont été menées sur ce problème d'optimisation. Ce problème est $\mathcal{NP}$-Difficile. Nous avons généré aléatoirement des instances de problèmes et nous avons exécuté l'algorithme pour chacune d'entre elles. La méthode atteint systématiquement une solution optimale lorsque le nombre de tâches est inférieur à 20. Pour cette raison nous pensons que la méthode peut être appliquée à des problèmes industriels. Nous renvoyons à *[82]* pour les résultats détaillés des expérimentations numériques.

4.3.2 Vers une méthode générique

Les résultats précédents, nous ont conforté dans la possibilité d'ordonnancer des tâches à priorité fixe afin de respecter les contraintes temporelles et d'optimiser un critère de qualité de service. Nous avons donc cherché à généraliser la méthode précédente, qui ne traite que de la minimisation de la moyenne des pires temps de réponse à d'autres critères. La méthode repose toujours sur une procédure par séparation et évaluation, mais nous avons défini les parties pouvant être réutilisées pour tout critère de performance (*partie générique*), de celles devant être spécialisées pour chaque critère (*partie spécifique*). Notre méthode ne supporte que les tâches à départ simultané.

Dans ce projet, nous souhaitons définir un logiciel avec interface graphique permettant de définir :

– les parties spécifiques associées à un nouveau critère,

- le lien entre la partie générique et des parties spécifiques,
- des expérimentations numériques,
- des outils d'analyse des résultats.

Nous détaillons ci-après la partie générique, puis l'exemple des parties spécifiques par rapport à un nouveau critère de performance.

Partie générique

Le principe d'énumération de l'espace des solutions peut être appliqué quel que soit le critère de performance. De même, les règles d'élimination sont elles aussi génériques puisque liées aux contraintes temporelles ou à la nature de la procédure par séparation et évaluation.

Les règles de sélection, de branchement et d'élimination sont donc totalement génériques. L'ensemble du programme correpondant peut être appliqué à n'importe quel critère de performance. Il définit l'énumération des solutions, c'est-à-dire l'énumération des différentes façon d'attribuer les priorités aux tâches. Comme précédemment, les niveaux de l'arbre de recherche définisent les niveaux de priorité des tâches. Mais cette fois, l'attribution des priorités sera effectuée de façon décroissante. Ainsi, le premier niveau de l'arbre est associé au niveau de priorité le plus élevé, et les tâches associées aux feuilles de l'arbre seront affectées au niveau de priorité le plus faible. Les niveaux de l'arbre sont associés à des niveaux de priorités différents. Deux tâches ne pourront donc pas avoir la même priorité.

L'évaluation des nœuds est fortement liée au critère de performance optimisé. En conséquence, ces évaluations seront définies dans les parties spécifiques. Toutefois, pour chaque feuille nous pouvons construire l'ordonnancement correspondant à l'attribution des priorités définies dans la branche de la racine de l'arbre de recherche jusqu'à la feuille courante. Cet ordonnancement est construit par simulation sur l'intervalle de faisabilité $[0,ppcm_i T_i]$. Il est très simple connaissant un ordonnancement complet d'évaluer n'importe quel critère en temps polynomial en la longueur de l'ordonnancement (i.e., du ppcm des périodes des tâches).

Partie spécifique

Les deux parties devant être spécialisées sont : le calcul de la borne supérieure, nécessaire à l'initialisation de la procédure par séparation et évaluation, et le calcul de la borne inférieure pour évaluer la qualité du nœud courant ainsi que celles de ses fils.

Nous allons illustrer ce problème sur la minimisation du temps de réponse moyen (pondéré) des tâches. Ce critère est complexe puisqu'il nécessite de connaître tous les temps de réponse de toutes instances des tâches.
Soit $R_{i,k}$ le temps de réponse de la $k^{(\text{ième})}$ instance de τ_i, et $\bar{R}_i$ les temps de réponse moyen de toutes les requêtes de τ_i. Le critère à minimiser est donc :

$$\overline{R} = \lim_{f \to \infty} \frac{1}{f} \sum_{i=1}^{n} \sum_{j=1}^{f} w_i \cdot R_{i,j} \tag{4.2}$$

Nous allons tout d'abord simplifier l'équation 4.2. Puisque les tâches sont périodiques, l'ordonnancement sera lui-même périodique avec une période $H = ppcm\{T_i | i = 1, \ldots, n\}$. En consé-

quence, la somme des temps de réponse des tâches sur l'intervalle $[0,k \cdot H)$ $(k \geq 0)$ peut être définie par :

$$\sum_{j=1}^{k \cdot H/T_i} R_{i,j} = k \cdot \sum_{j=1}^{H/T_i} R_{i,j}$$

Ainsi, pour $\bar{R}_i$ (le temps de réponse moyen des instances de τ_i) nous obtenons :

$$\overline{R}_i = \lim_{k \to \infty} \frac{T_i}{k \cdot H} \sum_{j=1}^{k \cdot H/T_i} R_{i,j} = \frac{T_i}{H} \sum_{j=1}^{H/T_i} R_{i,j} \tag{4.3}$$

Donc, le critère optimisé est :

$$\bar{R} = \sum_{i=1}^{n} \left(\frac{w_i \cdot T_i}{H} \sum_{j=1}^{H/T_i} R_{i,j} \right) \tag{4.4}$$

Nous réutilisons la borne supérieure concernant la minimisation $\sum w_i R_i$, qui définit bien sûr une borne supérieure du nouveau critère. Nous détaillons ci-après la borne inférieure. Nous distinguons deux cas : les tâches ayant une priorité et celles qui n'en n'ont pas.

Tâches avec une priorité Considérons une tâche τ_i, qui a déjà été énumérée lorsque le nœud courant de l'arbre de recherche est séparé.
Soit dans l'intervalle $[0,H)$:

- $R_i^{\max}$ le plus grand temps de réponse de τ_i.
- $R_i^{\min}$ le plus petit temps de réponse de τ_i.

Nous considérons le cas pessimiste où $(\frac{H}{T_i} - 1)$ instances de τ_i ont un temps de réponse de $R_i^{\min}$ et une instance de τ_i à un temps de réponse égal à $R_i^{\max}$, en conséquence :

$$\bar{R}_i \geq \left((\frac{H}{T_i} - 1) \cdot R_i^{\min} + R_i^{\max} \right) \frac{T_i}{H}$$

Par le principe d'énumération des priorités aux tâches, nous connaissons toutes les tâches plus prioritaires que τ_i, $R_i^{\max}$ est la plus petite solution positive de l'équation suivante :

$$R_i^{\max} = C_i + \sum_{j \in \mathrm{hp}(i)} \left\lceil \frac{R_i^{\max}}{T_j} \right\rceil \cdot C_j \tag{4.5}$$

Le calcul de $R_i^{\min}$ est plus complexe, dans [71] une borne inférieure de R_i^{lb} a été présentée. Elle utilise le fait que le meilleur temps de réponse d'une tâche survient lorsque la tâche se termine par un instant critique (i.e., au moment où toutes les tâches plus prioritaires sont réveillées). Ceci donne le moyen de calculer une borne du meilleur temps de réponse par l'équation suivante [71]:

$$R_i^{\mathrm{lb}} = C_i + \sum_{j \in \mathrm{hp}(i)} \left\lceil \frac{\max\{R_i^{\mathrm{lb}} - T_j, 0\}}{T_j} \right\rceil \cdot C_j$$

Ceci peut être résolu de façon itérative en calculant la plus grande solution de la suite suivante :

$$\begin{aligned}\mathcal{R}_0 &= \text{init} \\ \mathcal{R}_k &= C_i + \sum_{j \in \text{hp}(i)} \left\lceil \frac{\max\{\mathcal{R}_{k-1} - T_j, 0\}}{T_j} \right\rceil \cdot C_j\end{aligned}$$

Dans [71] aucune valeur initiale n'est donnée, nous en proposons une. Premièrement, observons que la valeur initiale doit être plus grande que R_i^{lb}:

$$\begin{aligned}\text{init} &\geq C_i + \sum_{j \in \text{hp}(i)} \left\lceil \frac{\max\{R_i^{\text{lb}} - T_j, 0\}}{T_j} \right\rceil \cdot C_j \\ &\geq \frac{C_i}{1 - \sum_{j \in \text{hp}(i)} \frac{C_j}{T_j}}\end{aligned}$$

En conséquence, nous proposons le calcul suivant :

$$\begin{aligned}\mathcal{R}_0 &= \frac{C_i}{1 - \sum_{j \in \text{hp}(i)} \frac{C_j}{T_j}} \\ \mathcal{R}_k &= C_i + \sum_{j \in \text{hp}(i)} \left\lceil \frac{\max\{\mathcal{R}_{k-1} - T_j, 0\}}{T_j} \right\rceil \cdot C_j\end{aligned}$$

Tâches sans une priorité Considérons une tâche τ_i sans priorité (i.e., elle n'a pas encore été traitée par la procédure par séparation et évaluation). Le niveau de priorité de cette tâche sera choisi dans l'intervalle $[k+1,n]$ (où k est le nombre de tâches avec une priorité pour le nœud courant). Notons que baisser le niveau de priorité d'une tâche ne peut pas faire augmenter le temps de réponse d'une autre tâche. En conséquence nous affectons toutes les tâches sans priorité au niveau $k+1$. Nous noterons pour ces tâches leurs temps de réponse R_i^{lb}.
Maintenant, nous pouvons donner la borne inférieure du critère:

$$\sum_{i=1}^{n} \frac{w_i \cdot T_i}{H} \left((\frac{H}{T_i} - 1) \cdot R_i^{\text{lb}} + R_i^{\max} \right) \tag{4.6}$$

Expérimentations numériques

Nous avons mené des expérimentations numériques sur le critère minimisation de la moyenne pondérée des temps de réponse des tâches. La méthode conduit très souvent à l'optimum pour des instances de 15 tâches. Par contre, la barrière de 25 tâches semble difficile à franchir. Pour ce type de problème des améliorations devront être proposées. Les résultats détaillés sont présentés dans *[31]*.

4.4 Minimisation de l'énergie

Les systèmes embarqués intègrent de plus en plus de fonctionnalités nécessitant une puissance de calcul importante. Toutefois, le fonctionnement de tels systèmes repose sur des batteries. La

minimisation de l'énergie consommée par le système devient alors un critère très important. Les techniques proposées dans ce but reposent sur le matériel et son architecture comme l'utilisation des processeurs à vitesse variable, ou sur des techniques logicielles comme la mise en veille des composants non utilisés et enfin sur des techniques hybrides combinant les deux approches précédentes [29, 67].

Dans la suite nous considérons des processeurs permettant d'adapter dynamiquement leur fréquence de fonctionnement à la quantité de travail à traiter, comme l'Intel StrongArm et le Crusoe de Transmeta. Ces processeurs permettent d'ajuster conjointement la fréquence de l'horloge et la tension d'alimentation du processeur afin de réduire l'énergie consommée. Pour une fréquence donnée, la tension d'alimentation minimisant l'énergie consommée peut être déterminée [29]. La fonction de consommation d'énergie est usuellement quadratique en la vitesse du processeur (dans tous les cas elle est convexe). Nous supposerons que la vitesse de travail du processeur est bornée par deux constantes S_{min} et S_{max}.

L'ordonnancement de tâches sur des processeurs à vitesse variable génère de nombreux travaux dans la littérature. L'ordonnanceur ne se limite pas à définir l'ordre des tâches à exécuter par le processeur. Il doit en plus définir, à chaque instant, la vitesse du processeur. La prise en compte de cette nouvelle dimension généralise la problématique de l'ordonnancement temps réel. Deux paramètres émergent pour classifier les problèmes étudiés dans la littérature :

- les vitesses évoluent de façon continue ou discrète dans le temps. Dans le cas continu, nous supposons que les vitesses appartiennent à l'intervalle $[S_{min},S_{max}]$ et dans le cas discret, il existe m différents paliers de vitesse : $\{S_1,\ldots,S_m\}$. A notre connaissance, tous les processeurs existants utilisent un nombre fini de paliers de vitesse.
- les tâches ont des profils de consommation identiques ou variables. Lorsque le profil est identique, la consommation de chaque tâche est identique dans le temps. Ainsi quelle que soit la tâche exécutée sur un intervalle de temps donné, la même quantité d'énergie sera consommée. Le profil variable de consommation autorise les tâches à consommer une quantité d'énergie différente. Ceci permet de tenir compte plus précisément de l'activité des tâches utilisant des ressources matérielles différentes (comme l'utilisation d'un DSP par exemple).

Nous présentons trois heuristiques de résolution du problème d'ordonnancement de tâches périodiques, à départ simultané et à échéance sur requête ($D_i = T_i$). Pour ce type de tâches, l'algorithme d'ordonnancement en-ligne EDF (Earliest Deadline First) est optimal. Nous supposons que les vitesses du processeur appartiennent à l'intervalle compris entre deux valeurs $S_{min} = 300$ et $S_{max} = 833$, et que l'intervalle entre deux vitesses successives est donnée par S_{step} (par exemple 33MHz pour le processeur Crusoe de Transmeta). Les fonctions de puissance utilisées sont obtenues par régression quadratique à partir des points de fonctionnement du processor Crusoe de Transmetta (TM5500) à plus au moins 10% près [42]. Nous nous limitons à une vitesse par tâche, c'est-à-dire que toutes les instances d'une même tâche s'exécutueront à la même vitesse durant toute la vie de l'application temps réel.

4.4.1 Formulation du problème

Ce problème discret peut se modéliser comme une extension du problème de sac-à-dos. Cette formulation a été proposée dans [59, 58, 93]. Considérons un processeur avec m paliers de vitesse. Soit e_{ij} l'énergie consommée par la tâche τ_i lorsqu'elle s'exécute à la vitesse S_j du processeur

et $u_{ij} = \frac{C_i}{T_i S_j}$ est la charge processeur correspondante. On définit l'affectation d'une tâche τ_i à la vitesse j par la variable bivalente $x_{ij} = 1$, pour toutes les autres valeurs de $k \neq j$ on vérifie alors $x_{ik} = 0$. La formulation du problème est alors un programme mathématique linéaire avec variables bivalentes :

$$
\begin{aligned}
Min \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} e_{ij} x_{ij} \\
s.t. \quad & \\
& \sum_{j=1}^{m} x_{ij} = 1 \qquad 1 \leq i \leq n \\
& \sum_{i=1}^{n} \sum_{j=1}^{m} u_{ij} \leq 1 \\
& x_{ij} \in \{0,1\} \qquad 1 \leq i \leq n, 1 \leq j \leq m
\end{aligned}
$$

La première contrainte impose que chaque tâche est affectée à un palier de vitesse tandis que la seconde assure que le facteur d'utilisation du processeur est inférieur ou égal à 1 (test d'ordonnançabilité pour EDF). Bien que le programme mathématique soit linéaire, sa résolution est un problème $\mathcal{NP}$-Difficile au sens faible [93].

Il est important de remarquer que ce programme linéaire avec variables bivalentes n'est pas optimal puisqu'il suppose que chaque tâche s'exécutera toujours à la vitesse constante. A notre connaissance, il n'a pas été démontré que l'ensemble des ordonnancements allouant une vitesse constante par tâche contient nécessairement une solution minimisant l'énergie totale (i.e., ensemble dominant). Toutefois, dans la suite nous nous limiterons aux ordonnancements affectant une vitesse constante pour chaque tâche. Nous pensons que cette hypothèse est réaliste puisque, comme l'illustreront les expérimentations numériques, avec un nombre important de paliers de vitesse, la solution discrète est proche de la solution continue pour laquelle l'hypothèse permet de conduire à une solution optimale. Dans la suite, nous présentons trois heuristiques pour résoudre ce problème minimisation d'énergie.

4.4.2 L'algorithme du "palier constant"

Dans le cas continu (la vitesse peut varier de façon continue), il est possible de choisir la vitesse optimale identique pour toutes les tâches minimisant la consommation totale en énergie. Nous adaptons cette idée au cas discret en affectant toutes les tâches au palier le plus proche de la valeur idéale établie dans le cas continu.

Palier constant :
La vitesse du processeur est constante et égale à :

$$
\begin{aligned}
& \overline{S} = S_{min} + k * S_{step} \\
& \text{où } k = min \left\{ \left\lceil \frac{U - S_{min}}{S_{step}} \right\rceil ; \left\lfloor \frac{S_{max} - S_{min}}{S_{step}} \right\rfloor \right\} \text{ et } S_{step} = \text{constante}
\end{aligned}
$$

4.4.3 L'algorithme de la "descente en cascade"

Le principe de l'algorithme est de fixer en premier lieu toutes les tâches à la vitesse maximale du processeur. Ensuite les tâches sont classées par ordre décroissant de "saut d'énergie". Un "saut d'énergie" est égal à la différence d'énergie entre l'exécution de la tâche à la vitesse courante (par exemple pour le palier de vitesse i) et celle à la vitesse inférieure (pour le palier $i-1$). Si cette modification de la solution courante ne permet pas de respecter une charge processeur inférieure ou égale à 1, alors la transformation est rejetée et la tâche suivante est considérée. Le traitement s'arrête lorsque, à une itération donnée, les n tâches ont été parcourues sans avoir pu réduire leur vitesse.

4.4.4 L'algorithme du "recuit simulé"

Le recuit simulé est né d'une analogie entre l'optimisation combinatoire et la thermodynamique. En optimisation combinatoire, le but est de minimiser une fonction en évitant les minima locaux, alors que le nombre de solutions réalisables est souvent très élevé. Cet algorithme s'inspire des méthodes de descente, mais au lieu de rejeter une transformation entraînant une augmentation de la fonction objectif que l'on veut minimiser, on l'accepte avec une certaine probabilité [18]. La température est un paramètre de contrôle (par analogie avec le recuit métallurgique) qui rendra l'acceptation des transformations désavantageuses de moins en moins probables. Le principe de l'algorithme consiste à effectuer une transformation simple passant de la solution X_i à X_j ; on calcule la variation $\Delta f_{ij} = f(X_j) - f(X_i)$ de la fonction objectif f, et on accepte la transformation avec la probabilité $P(i,j)$ définie par :

$$\begin{aligned} P(i,j) &= 1 && \text{si } \Delta f_{ij} \leq 0 \\ P(i,j) &= e^{\frac{-\Delta f_{ij}}{t}} && \text{si } \Delta f_{ij} > 0 \end{aligned}$$

Dans cette expression, t est un paramètre de contrôle : quand la température t est élevée (initialement), un grand nombre de transformations est accepté ; quand t aura atteint une valeur basse, proche de zéro, seules les transformations avantageuses seront retenues, et les remontées n'auront plus lieu.

Pour résoudre le problème d'ordonnancement à vitesse variable, nous mémorisons dans une solution X_i les valeurs des vitesses des tâches (i.e. un vecteur de dimension n). Le recuit simulé fait intervenir six paramètres, que l'on doit choisir judicieusement de manière à obtenir une solution acceptable en un temps de calcul raisonnable [18]. Notre implémentation repose sur les paramètres suivants :

- La solution initiale est calculée par l'algorithme de "Descente en cascade".
- La température initiale est calculée à partir du maximum des Δf et du taux d'acceptation α. Δf est ici égal à la différence d'énergie lors du passage de toutes les tâches de la vitesse minimale à la vitesse maximale. On obtient ainsi une borne maximale de Δf. Étant donné que Δf peut prendre des valeurs élevées, on choisit, en contrepartie, un taux d'acceptation faible, fixé à $\alpha = 0{,}3$. En conséquence la température initiale est fixée à l'aide de la formule classique [18]: $t = -\frac{\Delta f}{\ln \alpha}$.
- La décroissance de la température est calculée par une suite géométrique de raison $\mu = 0{,}95$ permettant une décroissance rapide de la température (la température à l'itération $p+1$ est définie par celle à l'itération p comme $t_{p+1} = \mu * t_p$)

- Le nombre de changements de température au cours de l'algorithme est constant pour tout le plan d'expérimentations et vaut 60. Cette valeur est couramment utilisée dans les recuits simulés [18].
- Le nombre de transformations élémentaires à température fixée est choisi proportionnel au carré du nombre de tâches [18]. Par exemple, pour 5 tâches, ce nombre vaut 25.
- Le voisinage correspond à l'ensemble des solutions admissibles. Le choix d'une solution dans le voisinage fait ici appel à des tirages de nombres aléatoires dans une distribution uniforme. Dans le cas présent, le voisinage est engendré par la modification de la vitesse d'une seule tâche tirée au hasard parmi l'ensemble des tâches. Sa vitesse est alors modifiée après tirage d'un nombre choisi aléatoirement dans l'intervalle $[0;1]$: la vitesse est augmentée (palier supérieur de vitesse) si ce nombre est supérieur ou égal à 0,75 ou que la vitesse correspond au palier de vitesse minimale. Dans le cas contraire, la vitesse est réduite. On favorise donc la réduction de la vitesse des tâches, vu que le changement de vitesse n'est pas équiprobable. Une fois la solution du voisinage obtenue, on vérifie qu'elle est admissible, c'est à dire que la charge processeur totale est inférieure ou égale à 1. Si ce n'est pas le cas, on retire une solution dans le voisinage.

4.4.5 Expérimentations numériques

Nous avons mené des expérimentations numériques des trois heuristiques proposées dans [42]. Nous avons utiliser les caractéristques énergétiques du processor Crosœde Transmeta afin de définir des configurations de tâches avec des profils de variables de consommation d'énergie. Les configurations sont dotées de 3, 5, 10 ou 15 tâches et entre 3 et 15 paliers de vitesse pour le processeur, avec un pas de 1. 25 instances ont été générées pour chaque taille de problème (i.e., pour un nombre de tâches et un nombre de paliers de vitesse fixés). Nous donnons uniquement dans la suite les conclusions sur le comparatif des trois méthodes entre-elles.

- Heuristique du "Palier Constant" : quel que soit le nombre de tâches, nous observons qu'elle est de plus en plus performante plus le nombre de paliers augmente. Ceci est normal, car plus l'intervalle des vitesses du processeur est discrétisé, plus le modèle continu est approché. La déviation par rapport à la meilleure heuristique reste inférieure à 10%. Ainsi, pour les systèmes qui ne sont pas soumis à des contraintes énergétiques trop importantes, un processeur à vitesse constante pourra être utilisé.
- Heuristique de la "Descente en Cascade" : cette heuristique permet de calculer rapidement une solution discrète proche de l'optimale. En effet, la déviation par rapport à la meilleure heuristique est toujours inférieure à 1%.
- Heuristique du "Recuit Simulé" : le recuit n'améliore ainsi la solution initiale que dans 5 à 15 % des cas traités. Ce faible taux de réussite est certainement dû, d'une part aux choix des paramètres du recuit, et d'autre part la solution initiale est déjà très proche de la solution recherchée (voire de l'optimum). Le recuit apporte alors une amélioration lorsqu'il est nécessaire de faire remonter l'énergie consommée (i.e., la fonction objectif) au cours de la recherche de la solution. Or, la recherche d'une solution se fait par le choix aléatoire dans un voisinage, et parfois la meilleure solution n'est pas atteinte dans le temps imparti. Nous pensons que lorsque le recuit améliore la solution initiale, il trouve en général la solution proche de l'optimum global. Pour approfondir l'étude de la qualité du recuit simulé, il faudrait calculer la solution optimale systématiquement pour chaque expérience à partir des données enregistrées au cours du plan d'expérimentations.

4.5 Collaborations et diffusion des résultats

La méthode graphique pour l'aide à la conception d'applications temps réel ordonnançables à fait l'objet d'une publication dans la revue Technique et Science Informatiques (TSI, édition Hermès) *[73]*.

L'optimisation de critère de qualité de service a été tout d'abord mené sur la minimisation de la moyenne pondérée des pires temps de réponse des tâches. Ce travail a été présenté à la conférence internationale Techniques and Tools for Performance Computer Evaluation (TOOLS'02, éditions Springer Verlag) *[82]*. En collaboration avec Joël Goossens (Université Libre de Bruxelles), nous avons ensuite cherché à étendre la méthode précédente pour la rendre générique. Cette étude sera présentée à la conférence Real-Time Systems (RTS'04, éditions Teknéa).

L'ordonnancement de processeur à vitesse variable est issu du stage de DEA de Stéphane Jeannenot (étudiant de l'ENSMA), que j'ai encadré durant l'année universitaire 2002-2003. Cette étude sera présentée à la conférence Real-Time Systems (RTS'04, éditions Teknéa).

Deuxième partie

Autres travaux

Chapitre 5

Ordonnancement en-ligne : projet iW4L

5.1 Résumé

5.1.1 Thématique

Nous présentons dans ce chapitre le projet iW4L (Innovative Workload For Labs), qui traite de l'ordonnancement des analyses médicales dans les laboratoires pharmaceutiques. J'ai été amené à travailler sur ce projet en raison de mon expérience dans le domaine, antérieure à mon arrivée au LISI (la planification de la production dans l'industrie du verre). Ce projet a été porté par l'équipe Interface Homme Machine (IHM) du LISI. Avec le support de l'Anvar Poitou Charentes et de l'Incubateur régional (Université de Poitiers, ENSMA et CNRS), un projet de création d'entreprise est en cours (société ENGINN Software) qui est chargée de terminer le développement et de commercialiser le produit.

D'un point de vue scientifique, ce projet m'a amené à étudier l'ordonnancement en-ligne de machines à traitement par lot et à étudier ce problème avec l'analyse de compétitivité des algorithmes (competitive analysis). Sur ce thème, j'ai collaboré avec Patrick Martineau (Laboratoire d'Informatique, Tours) et encadré le stage de DEA de Frédéric Ridouard.

Nous présenterons tout d'abord le projet, puis l'ordonnancement en-ligne de machines à traitement par lot et nous terminerons par le développement du moteur de planification du logiciel iW4L.

5.1.2 Démarche et outils

Notre principale contribution porte sur l'ordonnancement en-ligne des machines à traitement par lot. Nous avons utilisé l'analyse de compétitivité (i.e., competitive analysis) pour comparer des méthodes d'ordonnancement ne connaissant les travaux à traiter que lorsqu'ils arrivent dans le système avec un algorithme clairvoyant qui lui connait le problème à résoudre dès le départ.

Le logiciel développé dans le cadre du contrat de recherche repose sur une méthode arborescente de recherche d'un ordonnancement respectant toutes les contraintes. Les opérations s'effectuant sur des machines simples sont démarrées au plus tôt, tandis que le démarrage des machines à traitement par lot est paramétrable par l'utilisateur du logiciel.

L'analyse des flux d'activité sera développée dans le cadre d'un prochain contrat. Elle reposera, a priori, sur des techniques de simulation.

5.2 Présentation du projet

5.2.1 Problématique

Les laboratoires pharmaceutiques analysent des échantillons pour des clients. La planification des activités permet de satisfaire les contraintes opérationnelles des analyses, de satisfaire les clients au plus tôt et améliorer la rentabilité des laboratoires. Bien que la gestion des échantillons soit informatisée, aucun logiciel ne propose une solution d'aide à la planification des activités afin d'améliorer la réactivité et les performances des laboratoires. Le projet iW4L ambitionne de combler ce vide sur le marché. L'objectif est de permettre au responsable du laboratoire de planifier les ressources (hommes, instruments et consommables) afin de satisfaire les contraintes propres associées à ces analyses.

La planification des activités est faite sur un horizon dont la granularité dépend directement de la section du laboratoire. L'horizon peut varier de quelques jours à plusieurs mois. La disponibilité des ressources est connue a priori, mais est sujette à des modifications (pannes d'instruments, absences de personnels, rupture de stock de consommable ...). A chaque demande d' *analyse* d'un client est associé un ensemble d' *échantillons*. Sur chacun d'entre eux seront réalisés des ensembles d'analyses. Chaque demande d'un client définit une *tâche*. Chaque analyse liée à une tâche définit une *opération*. L'enchaînement des analyses liées à une demande d'un client définit un ordre partiel sur l'ensemble des opérations qui la compose. La durée d'exécution d'une opération dépend si l'échantillon est intégré ou non à un lot d'analyses. Les opérations associées à un même ensemble d'analyses peuvent être réalisées simultanément (instrument traitant parallèlement plusieurs analyses). Ceci donne lieu à la notion de sous-échantillon.

Les *sous-échantillons* conduisent à des opérations différentes qui partagent des ressources. Des contraintes d'exclusion associées à des critères de qualité conduisent souvent les laboratoires à analyser des échantillons par des techniciens différents afin de recouper les résultats. Ceci introduit des contraintes d'exclusion sur les ressources (i.e., ces tâches ne doivent pas utiliser les mêmes ressources humaines). Les opérations utilisent simultanément des ressources, avec des durées différentes.

Les demandes des clients arrivent sporadiquement ou périodiquement. A chaque demande peut être associée une échéance impérative (durée de vie de l'échantillon) ou une date due (négociée avec le client). De plus, des tâches associées à des analyses rejetées (problème de qualité) sont réinjectées parmi celles à planifier. La charge de travail n'est donc pas prévisible et est sujette à de forte variation dans le temps.

La planification iW4L consiste à planifier en temps réel l'ensemble des tâches connues dans le système d'information. L'intégration de nouvelles tâches nécessitera de réordonnancer les analyses en cours afin de satisfaire les contraintes opérationnelles et les exigences des clients en terme de délai et de qualité des résultats d'analyse.

5.2.2 Organisation des laboratoires

Les laboratoires d'analyse s'organisent en différentes *sections*. Chaque section est dédiée à un type d'analyse, mais les ressources utilisées, notamment humaines, peuvent être partagées entre différentes sections. Une section peut consister en un certain nombre d'instruments, dont certains peuvent paralléliser des analyses sur plusieurs échantillons. Un échantillon, en fonction des besoins en terme d'analyses, peut nécessiter des analyses dans plusieurs sections. Une demande d'un client suit donc le processus suivant :

- réception des demandes des clients, enregistrement informatique ;
- préparation des échantillons et des éventuels sous-échantillons ;
- analyse dans les sections (après constitution des lots), enregistrement des résultats ;
- validation et édition du rapport complet d'analyse.

L'enchaînement des opérations à traiter sur une demande d'un client est défini lorsque celui-ci arrive dans le laboratoire.

5.2.3 Le problème d'ordonnancement

Le problème d'ordonnancement qui va être présenté relève de *l'ordonnancement d'atelier*. Nous renvoyons aux principaux ouvrages du domaine pour une présentation des problèmes classiques d'ordonnancement, ainsi que les notations usuelles [22, 3, 15, 49, 12].

Hiérarchisation des problèmes

La gestion du flux des activités dans le laboratoire repose sur trois niveaux hiérarchiques :

- Les *demandes* arrivent dynamiquement dans le temps.
- Toute demande se décompose ensuite en *échantillons* qui seront analysés séparément mais dont les résultats seront finalement regroupés au sein d'un même rapport. L'analyse d'un échantillon est une *tâche complexe*.
- Une *tâche complexe* est une séquence de *tâches élémentaires* ou simplement *tâches*.

Une demande est définie par un graphe décrivant l'enchaînement des tâches à réaliser. Au plus bas niveau une tâche est représentée par un sommet et un arc représente une relation de précédence entre deux activités. La décomposition hiérarchique est illustrée figure 5.1.

L'ordonnancement est effectué au niveau des tâches élémentaires, mais il semble prudent de prévoir des outils de présentation de la séquence des résultats sur chaque niveau précédemment présenté (tâches complexes, demandes clients). Dans la suite nous nous intéressons à la planification des tâches au niveau le plus bas correspondant aux analyses effectuées dans les différentes sections des laboratoires.

Horizon de planification

La planification est faite sur un horizon (i.e. un intervalle de temps) paramétrable. Le glissement de l'horizon est à définir. Le glissement en temps continu semble complexe à mettre en œuvre. Un glissement en temps discret semble plus simple à réaliser et plus réaliste dans l'organisation

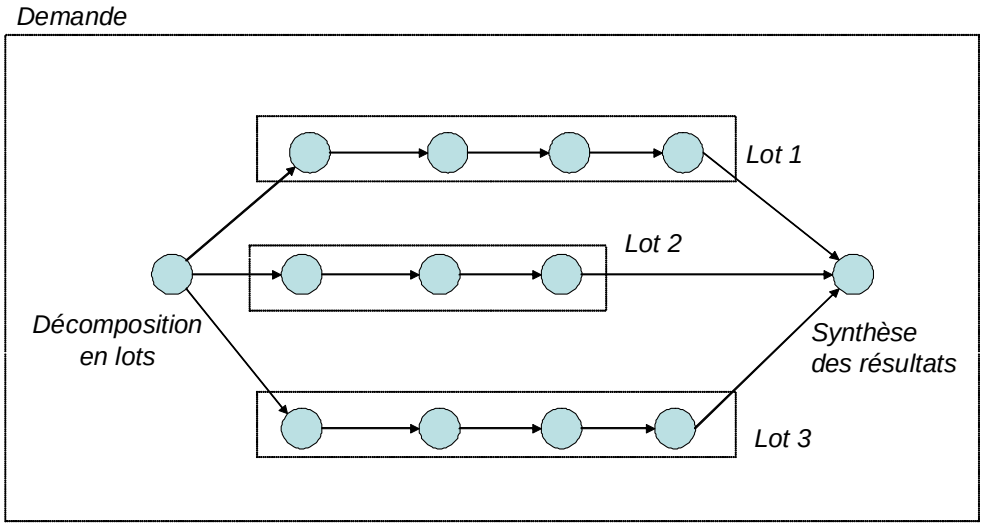


FIG. 5.1 – *Organisation hiérarchique des tâches. Les tâches encadrées sont des tâches complexes et les cercles représentes les tâches élémentaires.*

du travail d'un manager de laboratoire. Par exemple, un glissement d'horizon dont l'unité est le mois peut être fondé sur la semaine ; ou bien, si longueur de l'horizon est une semaine alors le glissement peut être fait à la fin de chaque journée... la définition de l'horizon de planification nécessite de définir :

- la durée de l'horizon,
- la périodicité du glissement (en temps discret).

5.3 Caractéristiques du problème d'ordonnancement

Caractéristiques des ressources

Deux types de ressource sont considérés :

- les ressources consommables. La ressource s'épuise lorsqu'on l'utilise. Elle doit être réapprovisionnée à chaque début d'horizon de planification.
- les ressources renouvelables (hommes, instruments). La ressource est restituée après chaque utilisation et conserve la même efficacité dans le temps.

Un *calendrier de fonctionnement* est défini pour chaque ressource sur l'horizon de planification. La disponibilité de la ressource repose donc sur une liste d'intervalles disjoints dans le temps. Chaque intervalle où la ressource est disponible sera appelé un *bloc*. Les caractéristiques d'un bloc sont :

- une date de début,
- une durée.

Certains instruments permettent de faire plusieurs analyses simultanément. Ceci correspond à la notion de traitement par lot (ou batch) dans la littérature.

Les dates de fin des analyses contenues dans un lot sont égales à la date de fin du lot. La durée d'une série correspond à la plus grande durée des tâches contenues dans la série. Ainsi, toutes les analyses se terminent simultanément. Lorsque la série est lancée, aucune analyse ne peut être ajoutée ou supprimée.

Caractéristiques des tâches

Les tâches complexes définissent le déroulement des analyses. Les demandes des clients définiront des *occurrences* (ou *instances*) de tâches complexes, et par là des occurrences de tâches élémentaires. Une demande peut être soumise à une échéance (*due date*) négociée avec le client. Cette échéance sera héritée par toutes les tâches complexes associées à la demande.

Tâches complexes : Les instances d'une tâche complexe possèdent les mêmes caractéristiques (enchaînement des tâches élémentaires, caractéristiques des tâches élémentaires, etc...).

Tâches élémentaires : Les instances de tâches complexes génèrent des instances de tâches élémentaires. Chaque tâche élémentaire peut utiliser des ressources (instruments) consommables et renouvelables. Les caractéristiques des tâches sont les suivantes :

- La durée d'une tâche dépend si elle est dans un traitement en série ou non. Deux durées possibles d'exécution sont donc prévues : $C_i^{(1)}$ en traitement par lot et $C_i^{(2)}$ sinon. On vérifie : $C_i^{(1)} \leq C_i^{(2)}$.
- Une tâche ne peut pas être interrompue (pas de préemption des analyses).
- Une tâche prend les ressources nécessaires à sa réalisation à son démarrage et les restitue (cas de ressources renouvelables) à la fin de son exécution.

La durée d'exécution d'une série s d'analyses est la somme des durées $\max_{i \in s} C_i^{(1)}$. Si l'instrument ne traite qu'une analyse (i.e., pas une série) alors la durée de traitement est $C_i^{(2)}$.

Caractéristiques des précédences : Aux relations de précédence entre les tâches complexes peuvent être associées des contraintes temporelles. Soient τ_i et τ_j deux tâches en précédence, la contrainte temporelle est définie par un intervalle $[a,b]$, où a représente la durée minimum entre la fin de τ_i et le début de τ_j, et b est la durée maximum entre la fin de τ_i et le début de τ_j. Nous supposons que a et b sont des nombres réels positifs vérifiant : $a \leq b$ (i.e., on ne pourra donc pas définir des périodes de recouvrement entre deux tâches en dépendance). Un intervalle va permettre de définir un ensemble de contraintes temporelles différentes :

- $[0,\infty]$: précédence simple. Le début de τ_j ne peut avoir lieu avant la fin de τ_i.
- $[a,\infty]$: durée d'attente minimum entre la fin de τ_i et le début de τ_j.
- $[0,b]$: durée de transfert limitée ou période de réaction avec durée limitée.
- $[a,b]$: attente entre les deux tâches définies par une fenêtre temporelle.
- $[a,a]$: attente de durée fixe.
- $[0,0]$: enchaînement sans attente des deux tâches ; c'est un cas particulier du précédent.

Chaque arc du graphe présenté figure 5.1 sera donc toujours étiqueté par un intervalle, dont la sémantique a été définie ci-dessus.

Critères de performance

La qualité de l'ordonnancement est associée à un (ou plusieurs) critères de performance. La présence de date due sur les demandes des clients nous invite à éviter au maximum les retards des tâches. L'ordonnancement de chaque tâche J_j est caractérisé par les indicateurs suivants :

- *Completion time* C_j : sa date de fin d'exécution.
- *Flow time* $F_j = C_j - r_j$: le temps de traversé (réponse) d'une tâche.
- *Lateness* $L_j = C_j - d_j$: l'écart avec son échéance.
- *Tardiness* $T_j = max(0; C_j - d_j)$ le retard de la tâche.
- *Unit penalty* $U_j = 0$ si $C_j \leq d_j$, $U_j = 1$ sinon. U_j indique que la tâche respecte son échéance.

Les critères d'optimalité les plus couramment utilisés sont :

- la longueur de l'ordonnancement (*makespan*) $C_{max} = \max_{j=1,\ldots,n} C_j$,
- le plus grand retard $L_{max} = max_{j=1,\ldots,n} L_j$,
- *Date moyenne de fin* $\sum C_j$ ou $\sum w_j C_j$,
- *Temps de réponse moyen* $\sum F_j$ ou $\sum w_j F_j$,
- *Retard moyen* $\sum T_j$ ou $\sum w_j T_j$,
- *Nombre de tâches en retard* $\sum U_j$ ou $\sum w_j U_j$.

5.3.1 Ordonnancement en-ligne

Les demandes définissent des instances de tâches complexes. Les taux d'arrivée des instances de tâches complexes ne sont pas a priori connus et peuvent être soumis à de fortes variations (causes épidémiologiques par exemple). Résoudre le problème d'ordonnancement va consister à affecter des dates de début aux instances de tâches élémentaires qui composent toute tâche complexe afin de respecter les contraintes temporelles et les disponibilités des ressources. Toutefois, une optimisation brutale d'un tel problème va conduire à répartir illégalement les analyses d'une même famille dans le temps. L'algorithme devra donc regrouper autant que possible les analyses du même type.

5.4 Etat de l'art

Avant de nous lancer dans le développement d'une méthode, nous avons cherché d'éventuels travaux de recherche dans le domaine. Nous nous sommes intéressés bien sûr aux travaux traitant directement de la problématique industrielle, puis au travaux sur l'ordonnancement en-ligne de machines en général.

5.4.1 Ordonnancement dans les laboratoires

Nous n'avons trouvé qu'un groupe finlandais travaillant sur l'ordonnancement d'une machine à traitement par lot (AutoDELFIA) [65, 66]. Les auteurs construisent un ordonnancement des

activités au sein de la machine avec une limite de temps de 2 minutes. L'algorithme proposé repose sur des règles d'ordonnancement (SJF - Shortest job First et LFJ - Longest Job First). Notons que ce travail a été notamment présenté à une conférence sur les systèmes temps réel.

5.4.2 Ordonnancement en-ligne

La validation d'une méthode d'ordonnancement en-ligne se fait pour :

- connaître la meilleure méthode d'ordonnancement à utiliser,
- mesurer la qualité de la solution vis-à-vis de l'optimum.

Les comparaisons peuvent porter sur les performances relatives des méthodes entre elles ou sur le comportement pire cas vis-à-vis d'une solution optimale. Les deux premiers types de comparaison sont faits en générant aléatoirement des problèmes et en appliquant les méthodes d'ordonnancement à comparer. Cette approche est fondée sur la *simulation*. La dernière comparaison est une approche analytique visant à déterminer le pire comportement d'une méthode d'ordonnancement vis-à-vis d'un *algorithme clairvoyant* optimal. Cette approche est appelée l'*analyse de compétitivité* (competitive analysis). Nous décrivons ces deux approches.

Simulation

Cette approche est ancienne puisqu'on retrouve d'abondants résultats de simulation dans [22]. Le livre de Kenneth Baker consacre deux chapitres sur ce sujet concernant respectivement les problèmes de job shop et les problèmes d'ordonnancement de projet avec contraintes de ressource [3]. Notons toutefois que les résultats de ces études sont fortement dépendants :

- des distributions de probabilité utilisées pour générer les instances de problèmes.
- de la taille des problèmes.

Le principe consiste à générer aléatoirement des tâches arrivant dynamiquement dans le système à ordonnancer. La méthode est ensuite appliquée sur l'instance. L'expérience est répétée suffisamment de fois afin d'obtenir des résultats satisfaisants du point de vue des statistiques. Ceci impose donc qu'une analyse statistique soit conduite pour extraire des comparaisons pertinentes des résultats numériques.

Analyse compétitivité avec un algorithme hors-ligne

Le reproche principal que l'on peut adresser à la simulation est que les résultats obtenus sont évalués au sein d'un modèle stochastique donné [10]. Cette approche est valable si, et seulement si, les lois de distribution utilisées, qui sont obtenues par observations du passé, modéliseront toutes les instances à venir. Très souvent, une telle hypothèse est incompatible avec l'environnement d'un algorithme en-ligne.

Une approche alternative consiste à comparer dans le pire cas l'algorithme en-ligne avec un algorithme *clairvoyant* optimal. Ces deux algorithmes utilisent une machine ayant la même puissance de traitement. Cette technique est appelée l'analyse de compétitivité. Un algorithme clairvoyant est un adversaire de l'algorithme en-ligne étudié. Un bon adversaire génère des instances de problème conduisant l'algorithme en-ligne à ses pires performances. Il existe plusieurs

types d'adversaire qui sont équivalents du point de vue de l'analyse :

- l'adversaire qui génère hors-ligne une instance et la sert optimalement (oblivious adversary).
- l'adversaire qui génère en-ligne des tâches, mais les traite sans attente (on-line adaptative adversary).

Un algorithme en-ligne qui minimise un critère de performance est c-compétitif s'il obtient un résultat qui est c fois plus important que l'adversaire, à une constante a près (pour les instances comportant une infinité de tâches). La valeur c est appelée le *ratio comparé*. Un algorithme est dit *compétitif* s'il existe une constante c telle qu'il soit c-compétitif. Plus formellement, étant donnée une instance I, soit $A(I)$ la valeur du critère obtenue par l'algorithme en-ligne et $OPT(I)$ la valeur optimale, alors l'algorithme en-ligne A est c-compétitif si, et seulement si, il existe une constante c et un entier a tels que :

$$A(I) \leq c \times OPT(I) + a$$

Le ratio comparé c_A de l'algorithme A s'obtient en considérant le pire scénario auquel il peut être confronté parmi toutes les instances possibles :

$$c_A = \sup_{tout\ I} \frac{A(I)}{OPT(I)}$$

Notons que ce ratio est toujours supérieur ou égal à 1 pour un problème de minimisation. Si $c_A = 1$ alors l'algorithme est optimal. Si c_A n'est pas une constante, mais une fonction de données de l'instance, alors l'algorithme n'est pas compétitif.

Pour un problème de maximisation, le ratio comparé se définit par :

$$c_A = \inf_{tout\ I} \frac{A(I)}{OPT(I)}$$

Dans ce cas, $0 \leq c_A \leq 1$. Si $c_A = 0$ alors l'algorithme n'est pas compétitif, si $c_A = 1$ alors l'algorithme est optimal.

L'inconvénient de l'analyse de compétitivité est de ne capturer la performance d'un algorithme en-ligne que par son ratio comparé. Très souvent, cela conduit à une vision très pessimiste du comportement de l'algorithme évalué. Par exemple, il est connu qu'il n'existe pas d'algorithme compétitif pour ordonnancer sans préemption des tâches sur une machine en minimisant le temps de traversé moyen. Plusieurs relaxations de l'analyse de compétitivité ont été proposées dans la littérature afin de limiter cette vision pessimiste comme l'augmentation de la vitesse des machines utilisées par l'algorithme en-ligne par rapport à celles utilisées par l'adversaire clairvoyant [47]. Cette technique est appelée *technique d'augmentation de ressource.*

Un état de l'art sur l'utilisation de l'analyse de compétitivité pour des problèmes d'ordonnancement est présenté dans [94]. Cet article concerne principalement l'ordonnancement à une machine et de machines parallèles identiques. Nous avons présenté un état de l'art sur les problèmes à une machine à l'École d'Automne de Recherche Opérationnelle *[92].* La tableau 5.1 donne la

Problèmes	Modèles	Bornes Inférieures		Bornes Supérieures	
$\min \sum C_j$	Préemption	1		1	
	Standard	2	[68]	2	[68, 38]
	Redémarrage	1.2108	[26]	3/2	[102]
$\min \sum F_j$	Préemption	1		1	
	Standard	$\Omega(n)$			
	Redémarrage	$\Omega(\sqrt{n})$	[26]	$\Theta(n)$	[26]
$\min \sum w_j C_j$	Préemption	1.0730	[26]	2	
	Standard	2	[68]	2	[1]
	Redémarrage	1.2232	[26]	2	[1]
$\min \sum w_j F_j$	Préemption	2	[26]	$\Theta(n)$	[26]
	Standard	$\Omega(n)$			
	Redémarrage	$\Omega(n)$	[26]	$\Theta(n^2)$	[26]
$\min L_{max}$ $(d_i \leq 0)$	Préemption	1		1	
	Standard	1.618	[39]	1.618	[39]
	Redémarrage	3/2	[101]	3/2	[101]
$\max \sum(1 - U_j)$	Préemption			0	[6, 7]
	Standard			0	[6, 7]
	Redémarrage	1/2	[37]	1/2	[37]

TAB. 5.1 – *Garanties de performance des algorithmes d'ordonnancement en-ligne à une machine.*

synthèse des résultats des problème d'ordonnancement en-ligne à une machine. Dans ce tableau, le modèle standard interdit la préemption, le modèle préemption l'autorise tandis que le modèle redémarrage nécessitera la reprise de la tâche depuis son début après chaque préemption.

5.5 Ordonnancement en-ligne des machines à traitement par lot

Les machines d'analyse présentes dans les laboratoires sont souvent à traitement par lot. Les échantillons sont disposés sur une plaque et sont traités simultanément (i.e., en parallèle). Nous avons donc étudié plus spécifiquement ce problème à l'aide de l'analyse de compétitivité. Nous utiliserons dans la suite la terminologie classique de l'ordonnancement d'atelier.

Une machine à traitement par lot est une machine qui peut traiter simultanément jusqu'à b tâches simultanément. Les tâches d'un même lot se terminent simultanément. La durée de traitement d'un lot est égale à la durée de la plus longue tâche contenue dans le lot. La minimisation de la durée de l'ordonnancement est connue $\mathcal{NP}$-Difficile au sens fort [13]. Ce problème a été étudié dans sa version en-ligne [104]. Nous proposons deux algorithmes lorsque la capacité des lots est infinie ($b = \infty$, ou $b > n$, où n est le nombre de tâche) et que toutes les tâches ont une durée égale (i.e.$p_i = p$). Nous avons aussi proposé un algorithme pour les systèmes de tâches de durées quelconques ; ce problème est noté dans la littérature $1|p - batch, r_i, b = \infty|C_{max}$ [12].

5.5.1 Un algorithme hors-ligne optimal pour les lots de taille non bornée

Le problème $1|p-batch,r_i,b=\infty|C_{max}$ peut être résolu en temps polynomial ($O(n^2)$, où n est le nombre de tâches) en utilisant la programmation dynamique [50]. Une implémentation plus efficace a été proposée dans [69] conduisant à un algorithme en $O(n \log n)$. Ainsi si les durées de traitement sont égales, minimiser le makespan peut être résolu hors-ligne en temps polynomial.

Borne inférieure

Théorème 13 *Tout algorithme déterministe d'ordonnancement d'une machine à traitement par lot pour minimiser la durée totale de l'ordonnancement a un ratio comparé d'au moins* $\frac{1+\sqrt{5}}{2} \approx 1{,}618$.

L'article [104] propose une preuve de ce résultat fondée sur un adversaire hors-ligne, nous avons proposé dans *[90]* une preuve fondée sur un adversaire en-ligne.

Algorithmes en-ligne pour les tâches de durées égales

Lorsque la machine est libre et qu'une nouvelle tâche arrive, il peut être bénéfique d'attendre un peu pour vérifier si de nouvelles tâches ne vont encore arriver dans un futur proche. Si une tâche de durée p arrive dans le système, nous ne pouvons pas attendre plus de p unités de temps puisque sinon si aucune nouvelle tâche n'arrive le ratio comparé sera supérieur à 2 (moins efficace que le meilleur algorithme déjà proposé dans [57]). De plus il est simple de montrer qu'attendre exactement p unités de temps lorsqu'une tâche de durée p arrive quand la machine est libre conduit à un ratio comparé de 2.

Nous étudions maintenant une généralisation de ces approches autorisant de laisser la machine inoccupée durant un intervalle de temps proportionnel à la durée de la tâche qui arrive dans le système. Lorsqu'une tâche de durée p arrive et que la machine est libre, alors elle est laissée inoccupée durant αp unités de temps, $0 \leq \alpha \leq 1$. Nous appelons cette règle αH. C'est une extension de la règle H, proposée dans [57], en fixant $\alpha = 0$.

Définition 6 *αH: Lorsque la machine est libre et qu'il existe des tâches disponibles à ordonnancer, soit S cet ensemble de tâches, alors le prochain lot n'est pas ordonnancé avant $min_{j \in S}(r_j + \alpha p_j)$ unités de temps et alors toutes les tâches disponibles sont ordonnancées.*

L'algorithme αH est un des meilleurs algorithmes déterministes lorsque α est fixé à la valeur $(-1+\sqrt{5})/2$ pour le problème $1|p-batch,r_i,p_i=p,b=\infty|C_{max}$. Ainsi, nous supposons que les tâches ont des durées égales.

Théorème 14 *αH est 1,618-compétitif pour le problème $1|p-batch,r_i,p_i=p,b=\infty|C_{max}$ lorsque $\alpha = (-1+\sqrt{5})/2$.*

Nous avons aussi défini un deuxième algorithme en-ligne, nommé $\alpha H2$, qui sera lui aussi optimal.

Définition 7 *$\alpha H2$: Lorsque la machine est libre et qu'il existe des tâches disponibles à ordonnancer, soit S cet ensemble de tâches, alors le prochain lot n'est pas ordonnancé avant $max_{j \in S}(r_j + \alpha p_j)$ unités de temps et alors toutes les tâches disponibles sont ordonnancées.*

Théorème 15 *$\alpha H2$ est 1,618-compétitif pour le problème $1|p-batch,r_i,p_i=p,b=\infty|C_{max}$ lorsque $\alpha=(-1+\sqrt{5})/2$.*

Lorsque des tâches sont disponibles ($j \in S$), l'attente avant le début d'un lot peut être choisie dans l'intervalle $[minj \in S(r_j+\alpha p_j),max_{j\in S}(r_j+\alpha p_j)]$ sans altérer l'optimalité de l'algorithme en-ligne dans le pire cas. Ceci permet une certaine flexibilité dans la mise en œuvre de l'ordonnancement.

5.5.2 Algorithme en-ligne pour les tâches de durées inégales

Nous considérons toujours le problème de machine à traitement par lot, avec des tailles de lots non bornées, mais avec des tâches de durées quelconques. Nous avons montré que dans ce contexte αH et $\alpha H2$ ne sont que 2-compétitifs [91]. Nous avons proposé un algorithme optimal, c'est-à-dire conduisant au ratio de performance 1,618.

Cet algorithme améliore le meilleur algorithme connu (présenté dans [104], non en terme de ratio de performance, mais dans la durée d'attente avant le lancement d'un lot). Dans l'algorithme de [104], uniquement deux lots pourront être exécutés consécutivement sans attente. Notre algorithme conduit toujours à introduire moins d'attente et pourra ainsi comporter des séquences de plus de deux lots consécutifs sans attente. En introduisant moins d'attente avant le début d'un lot, nous ne pouvons terminer l'ordonnancement plus tard. L'algorithme 4 présente cet algorithme qui est une adaptation de [104] puisque l'unique différence est dans le calcul de λ. Dans l'algorithme de [104], $\lambda=(1+\alpha)r_k+\alpha p_k$. Dans notre version, le facteur multiplicateur de r_k par $(1+\alpha)$ a été supprimé. Dans [91], nous montrons que cette évolution du calcul de λ ne détériore pas la ratio comparé.

Algorithme 4: Algorithme αH^{∞}

```
t=0;
U(t) = φ;
while True do
    U(t) est l'ensemble des tâches disponibles à la date t ;
    while U(t) = φ do
        Ne rien faire et mettre à jour t;
    end
    Trouver une tâche J_k ∈ U(t) telle que p_k = max{p_j / J_∈U(t)} ;
    λ = r_k + αp_k ;
    s = max(t,λ) ;
    Soit J_h une tâche réveillée dans l'intervalle [t,s] à l'instant t';
    if p_h > p_k then
        k = h ;
        λ = r_h + αp_h ;
        t = t' ;
        s = max(t,λ) ;
    U(t) = U(t) ∪ {J_h} ;
    A l'instant s, ordonnancer toutes les tâches de U(s) dans un lot ;
    t = s + p_k;
end
```

5.6 Développement du projet : contribution au moteur de planification

5.6.1 Déroulement du projet

Le projet iW4L a été subventionné par l'ANVAR Poitou-Charentes sur 2 ans. Une part importante du travail a consisté à collecter les informations de gestion du laboratoire dans une base de données et enfin à planifier les demandes des clients. Une partie importante du développement a été confiée à des étudiants en informatique de l'Université de Poitiers. Le moteur de planification a été programmé par un ancien doctorant de l'équipe IHM, employé sur le contrat. Je ne suis intervenu que sur deux points :

- la méthode de planification mis en œuvre dans le logiciel iW4L. Ce problème est au cœur du projet et a fait l'objet d'une expertise par un spécialiste du domaine.
- le problème de la propriété intellectuelle en liaison avec le cabinet Breeze et Majerowicz, à Paris. Cette étude a été menée à la demande de l'ANVAR qui subventionnait le projet.

Un projet de création d'entreprise (Enginn Software) est en cours avec l'aide de l'incubateur régional.

5.6.2 Le moteur de planification

Le logiciel développé dans le projet iW4L repose sur une méthode d'aide à la décision. L'utilisateur, expert en planification (il réalise d'ordinaire ces opérations sans logiciel particulier), va guider l'ordre de passage des analyses complexes, et le moteur de planification est chargé de vérifier la disponibilité des ressources et de les placer au plus tôt (à partir de l'instant choisi par l'utilisateur). Les choix suivants ont été faits durant le développement du projet :

- Les analyses déjà placées dans l'ordonnancement ne seront pas déplacées dans le temps.
- Les analyses ne pouvant pas être insérées dans l'ordonnancement sont retournées à l'utilisateur. Celui-ci devra donc arbitrer les analyses devant être différées (souvent en négociation avec les clients) ou rejetées.
- La constitution des lots, et la date de lancement d'un lot sont du ressort de l'utilisateur. Etant donnée l'étude théorique sur les machines à traitement par lot, il nous est apparu important de ne pas lancer les lots à partir d'un seuil de remplissage.

Nous pouvons constater que les techniques mises en œuvre dans le moteur de planification sont plus du ressort du développement que de la recherche. L'unique *transfert* des travaux scientifiques sur les méthodes d'ordonnancement en-ligne mis en œuvre dans le logiciel de planification est la prise de conscience sur la nécessité d'insérer des temps d'oisiveté dans l'ordonnancement en présence de machines à traitement par lot.

5.7 Collaborations et diffusion des résultats

Ce projet s'est conduit en collaboration avec de nombreuses personnes. Le projet était porté par le Pr. Patrick Girard, responsable de l'équipe Interface Homme Machine (IHM) du LISI. Comme déjà signalé dans le chapitre, je ne me suis occupé uniquement que de l'aspect scientifique de la partie ordonnancement (Rapport scientifique pour l'ANVAR et l'expertise et les problèmes de propriété intellectuelle de la méthode de planification en vue de son financement).

Le développement du moteur de planification a été fait par un ingénieur employé sur le projet (Dr. Guillaume Texier, ancien membre de l'équipe IHM du LISI).

L'étude sur l'ordonnancement en-ligne des machines à traitement par lot à fait l'objet d'une présentation à la conférence internationale Industrial Ingeneering and Production Management (IEPM'03, Porto, Portugal) *[90]*. Ce travail a été fait en collaboration avec le Pr. Patrick Martineau (Laboratoire d'Informatique, Tours) et a aussi fait l'objet du stage de DEA de Mr Frédéric Ridouard (DEA T3IA, Poitiers) sur l'année universitaire 2002-2003 [91], sous ma direction. Nous avons présenté un état de l'art sur l'ordonnancement en-ligne des problèmes d'ordonnancement à une machine à l'École d'Automne de Recherche Opérationnelle (EARO, Tours) *[92]*.

Les perspectives de ce travail seraient d'étudier les problèmes d'ordonnancement en-ligne des machines à traitement par lot, en optimisant d'autres critères de performance et en intégrant des facteurs pratiques comme les temps de préparation des lots. Malgré l'intérêt personnel, cette thématique étant éloignée des activités de l'équipe temps réel, nous ne pensons pas poursuivre dans ce domaine. Toutefois, nous pensons réutiliser l'analyse de compétitivité sur des problèmes d'ordonnancement temps réel.

Par contre, les porteurs du projet iW4L m'ont dores et déjà contactés pour travailler sur des extensions du moteur de planification dans le cadre d'un nouveau contrat avec le laboratoire.

Chapitre 6

Réseaux de Petri

6.1 Résumé

De 1998 à 2000, j'ai continué de travailler sur les réseaux de Petri. Deux articles de synthèse ont été écrits et une étude sur la longueur pondérée des séquences de tirs a été faite. Nous détaillons succinctement ces travaux.

6.1.1 Démarche et outils

Notre principale contribution porte sur le calcul de la plus courte séquence de tirs pour atteindre un marquage. Les poids sont associés aux transitions afin de définir d'intérêt de tirer une transition par rapport à une autre. l'objectif est de minimiser le somme des poids des transitions tirées pour atteindre le marquage. L'approche utilisée repose sur deux étapes successives :

- le calcul du nombre de transitions à tirer (i.e., le vecteur caractéristique) à l'aide de méthodes performantes fondées sur la théorie des graphes,
- la définition de propriétés analytiques sur les séquences de tirs dominantes pour séquencer les tirs des transitions afin d'atteindre un marquage. Ces propriétés sont ensuite utilisées afin d'accélérer la recherche recherche d'une séquence franchissable respectant le vecteur caractéristique calculé dans la première étape.

6.2 Modélisation, validation et analyse des performances de systèmes

6.2.1 Les réseaux de Petri

Les réseaux de Petri définissent un modèle formel pour analyser les systèmes à événements discrets. Leur puissance d'expression est largement reconnue en terme de modélisation de ces systèmes. Ils permettent d'analyser les modèles des points de vue qualitatif (vérification de propriétés) et quantitatif (évaluation des performances).

Un réseau de Petri est un graphe biparti composé de deux types de nœuds, les places (P) et les transitions (T). Une place n'est jamais reliée à une autre place, et il en va de même pour les transitions. l'ensemble des arcs orienté est : $F \subseteq (P \times T) \cup (T \times P)$. Soit W l'ensemble des poids des arcs : $W : F \to \mathbb{N}^*$. Alors $N = (P,T,F,W)$ est un réseau de Petri généralisé.

Nous utiliserons les notations suivantes pour désigner l'ensemble de prédécesseurs et successeurs d'un nœud : ${}^\bullet x = \{y|(y,x) \in F\}$ et $x^\bullet = \{y|(x,y) \in F\}$. Par extension, cette notation sera appliquée aussi sur des ensembles de nœuds.

La matrice d'incidence du graphe $C : (P \times T) \to \mathbb{Z}$ est définie par :

$$C(p,t) = \begin{cases} 0 & \text{si} \quad (p,t) \notin F \quad \text{et} \quad (t,p) \notin F \\ -W(p,t) & \text{si} \quad (p,t) \in F \\ W(p,t) & \text{si} \quad (t,p) \in F \end{cases}$$

Les colonnes de la matrice d'incidence sont associées aux transitions. Soit t une transition, alors $C(t)$ est la colonne de C associée à la transition t. Les colonnes de C peuvent aussi être identifiées par leur numéro: C_i est la colonne numéro i de la matrice d'incidence.

L'état d'un réseau de Petri est défini par une fonction de marquage qui à chaque place associe le nombre de jetons (ou marques) : $M : P \to \mathbb{N}$. Une place est dite marquée si elle contient au moins un jeton. Une transition t est dite validée si toutes les places $p \in {}^\bullet t$ contiennent au moins $W(p,t)$ jetons. Le franchissement d'une transition validée t retire $W(p,t)$ jetons dans chaque place de $p \in {}^\bullet t$ et ajoute $W(p',t)$ jetons dans les places de $p \in {}^\bullet t$. Tirer une séquence de transitions σ est noté $M_0 \stackrel{\sigma}{\longmapsto} M$; M est le marquage atteint depuis le marquage initial M_0. Les marquages accessibles suivent l'équation d'état: $M = M_0 + C\overline{\sigma}$, où C est la matrice d'incidence du graphe et $\overline{\sigma}$ est le vecteur caractéristique de la séquence de tirs (i.e., $\overline{\sigma}_i$ est le nombre d'occurrences de t_i dans la séquence σ). L'équation d'état est une condition nécessaire d'accessibilité.

Un réseau de Petri marqué est une paire (N,M_0), où N est un réseau et M_0 un marquage inital de N. $R(M_0)$ est l'ensemble des marquages accessibles depuis M_0. Le graphe des marquages (V,E) de (N,M_0) est un graphe orienté, où V est l'ensemble des sommets et E est l'ensemble des arcs :

$$\begin{aligned} V &= R(M_0) \\ E &= \{(M_i,M_j) \in (E \times E) | M_i \stackrel{t}{\longmapsto} M_j, t \in T\} \end{aligned}$$

L'ensemble des marquages potentiellement accessibles $PR(M_0)$ est l'ensemble des vecteurs d'entiers M tels que $M = M_0 + C\overline{\sigma}, \overline{\sigma} \geq 0$ [21]. Nous vérifions que $R(M_0) \subseteq PR(M_0)$.

Un graphe d'événements est un réseau de Petri connexe vérifiant $|{}^\bullet t| = |t^\bullet|$ et $W(x,y) = 1$, $\forall (x,y) \in F$. Ainsi, dans cette classe de réseau, chaque place a exactement une transition d'entrée et une de sortie. En pratique, la place p_{ij} sera la place connectant t_i à t_j.

Une machine à état est un réseau de Petri connexe telle que $|{}^\bullet p| = |p^\bullet|$ et $W(x,y) = 1, \forall (x,y) \in F$. Donc, dans une machine à état toute transition a une place en entrée et une en sortie. Nous noterons, t_{ij} la transition reliant p_i à p_j.

Les propriétés qualitatives classiques sont : la vivacité et la bornitude. Un réseau de Petri est vivant si, pour tout marquage accessible M et toute transition t, il existe un marquage M' tel

que $M \rightarrow M'$ qui valide t. Soit $b \in \mathbb{N}^*$, alors le réseau est b-borné si pour toute place p et pour tout marquage $M \in R(M_0)$: $M(p) \leq b$,$p \in P$.
Les réseaux de Petri peuvent être utilisés à tous les niveaux de la conception d'un système :

- la conception du modèle,
- l'analyse qualitative du modèle,
- l'anayse quantitative pour vérifier les performances,
- la génération de code pour définir le squelette d'une application.

Il existe plusieurs approches pour modéliser en fonction de la sémantique qui sera attachée aux places et aux transitions. Une place peut être une ressource, un compteur, un événement ou une mémoire tampon avec une capacité ou bien une condition logique modélisée par la présence d'une jeton. Les transitions sont usuellement utilisées pour représenter les actions : allocation des ressources, incrémentation et décrémentation de compteur, occurrence d'événements, vérification d'une condition. Enfin les jetons représentent les instances de ressources, le contenu d'une mémoire tampon. De façon générale, les jetons seront associés aux données.

6.2.2 Modélisation et analyse

Dans *[19]*, nous avons présenté la modélisation par un réseau de Petri du comportement d'un programme informatique de la gestion de commandes auprès de clients. Cette même étude de cas a été aussi spécifiée par de nombreuses méthodes de spécification formelle dans les autres chapitres de l'ouvrage.

Dans *[88]*, nous avons présenté l'application des réseaux de Petri à :

- l'analyse d'un système en cours de conception. Le système analysé est un atelier de production avec un fonctionnement cyclique. Nous avons montrer la modélisation, l'analyse des performances et le dimensionnement de ce système.
- l'utilisation des réseaux de Petri pour mettre en œuvre un système. Un réseau de Petri modélise un programme mathématique permettant la planification de la production des usines fabriquant des bouteilles en verre de BSN/Emballage (aujourd'hui BSN/Glasspack). L'étude complète est présentée dans *[79]* et repose sur la capacité des réseaux de Petri à modéliser tous les programmes mathématiques *[81]*.

6.3 Plus courte séquence dans le graphe de marquage

Dans ce paragraphe, nous étudions le chemin pondéré le plus court dans le graphe de marquage. Ce problème à des applications pratiques notamment pour concevoir des logiciels de vérification de modèle (Model-Checking) [23, 24]. Ce problème peut aussi être utile lorsque le réseau de Petri modélise des problèmes d'optimisation *[81]*.

Soient M et M' deux marquages, nous voulons trouver la plus courte séquence (pondérée) de tirs conduisant du marquage M à M'. Ce problème peut être décomposé en deux sous-problèmes : premièrement calculer le vecteur caractéristique, puis construire une séquence qui lui corresponde. Nous étudierons les machines à état et les graphes d'événements vivants. Nous présenterons des algorithmes efficaces fondés sur la programmation mathématique et la théorie des graphes pour résoudre la première étape. Nous donnerons aussi un résultat sur une classe de

séquence de tirs qui est dominant pour les graphes d'événements vivants (et sous une certaine condition est aussi valide pour les réseaux de Petri généralisés).

Soit $d : T \to \mathbb{Z}$ le coût d'une transition, alors d_i est le coût d'un franchissement de la transition t_i.

6.3.1 Machine à état

Optimisation du vecteur caractéristique

Une borne supérieure du nombre de tirs nécessaires pour atteindre un marquage dans un machine à état b-bornée est $b.m$, où m est le nombre de transitions [23, 24]. Une borne inférieure peut être facilement déterminée en relaxant l'accessibilité des marquages en utilisant l'équation d'état pour déterminer les marquages potentiellement accessibles. Nous obtiendrons une borne inférieure puisque l'ensemble des marquages potentiellement accessibles inclut l'ensemble des marquages accessibles par une séquence de tirs [21].

Nous considérons le même problème mais en associant des coûts à chaque transition. Nous souhaitons minimiser le coût total de la séquence de tirs, c'est-à-dire la fonction $z = \sum_{i=1}^{m} \overline{\sigma}_i d_i$. La borne inférieure du critère sera calculée avec le programme mathématique (6.1).

$$\begin{aligned} &\min z = \sum_{i=1}^{m} \overline{\sigma}_i d_i \\ &\text{Soumis à} \\ &\qquad M_0 + C\overline{\sigma} = M \\ &\qquad \overline{\sigma} \geq 0 \end{aligned} \tag{6.1}$$

Nous avons montré que ce programme mathématique est optimal pour les machines à état. Ce résultat n'est pas trivial puisque des marquages solutions de l'équation des marquages ne seront pas nécessairement accessibles sur le réseau, même si c'est une machine à état.

Théorème 16 *Soit (N,M_0) une machine à état, alors le programme linéaire 6.1 calcule la valeur optimale du vecteur caractéristique pour le problème de la séquence de coût minimum.*

Nous proposons un algorithme plus efficace que la programmation linéaire pour résoudre ce problème en utilisant la théorie des graphes. A toute machine à état nous associons un réseau de transport (c.f. problème de recherche opérationnelle).

Définition 8 *Soit N une machine à état, nous définissons un réseau de transport $G(N)$ associé à N tel que :*

- *l'ensemble des sommets: $V_s \cup V_t \cup \{s_0,s_{n+1}\}$,$V_s \cap V_t = \emptyset$*
  - *$V_s = \{p \in P | M_0(p) > 0\}$ est l'ensemble des sommets sources.*
  - *$V_t = \{p \in P | M(p) > 0\}$ est l'ensemble des sommets puits.*
  - *Nous ajoutons une super-source s_0 et un super-puits s_{n+1}.*
- *l'ensemble des arcs:*
  - *$\forall p \in V_s$, nous ajoutons un arc (s_0,p) avec une capacité $M_0(p)$ et un coût de transport nul*

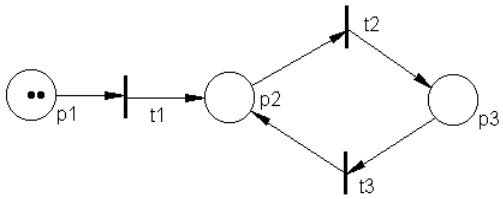


FIG. 6.1 – *Une machine à état*

- $\forall p \in V_t$, *nous ajoutons un arc* (p,s_{n+1}) *avec une capacité* $M(p)$ *et un coût nul*
- $\forall t_{ij} \in T$, *nous ajoutons un arc* (p_i,p_j) *avec une capacité infinie et un coût* d_i
- *nous ajoutons aussi une boucle implicite de* s_{n+1} *à* s_0.

La définition précédence suppose que $V_s \cap V_t = \emptyset$. Alors la propriété correspondante dans la machine à état est : $M(p) > 0 \Rightarrow M_0(p) = 0$ et $M_0(p) > 0 \Rightarrow M(p) = 0$. Nous montrons que nous pouvons toujours transformer le problème afin de respecter $V_s \cap V_t = \emptyset$. Supposons $M_0(p_i) > 0$ and $M(p_i) > 0$, alors nous effectuons le changement de variable suivant :

- $i \in V_s$, si $M_0(p_i) - M(p_i) \geq 0$ alors la capacité de l'arc (s_0,p_i) est $M'_0(p_i) = M_0(p_i) - M(p_i)$.
- $i \in V_t$ si $M_0(p_i) - M(p_i) < 0$ alors la capacité de l'arc (p_i,s_{n+1}) is $M'(p_i) = M(p_i) - M_0(p_i)$.

Nous donnons maintenant un exemple de réseau de transport associé à une machine à état. Du marquage $M_0 = (2 \quad 0 \quad 0)^t$ nous voulons atteindre le marquage $M = (0 \quad 0 \quad 2)^t$. Le vecteur des coûts associés aux transitions est $D = (1 \quad 3 \quad 2)^t$. Les valuations des arcs du réseaux de transport sont indiquées entre crochets : [coût,capacité]. La figure 6.2 donne le réseau de transport associé à la machine à état de la figure 6.1. Notons que l'arc (p_2,s_{n+1}) n'est pas représenté car sa capacité et son coût sont nuls.

Dans une machine à état, le nombre de jetons reste toujours constant. La propriété correspondant dans le réseau de transport est la loi de Kirchhoff (pour les nœuds). Le vecteur caractéristique $\overline{\sigma}$ de l'équation d'état est clairement relié au flot φ dans le réseau de transport.

Nous pouvons indiquer:

$$\sum_{i \in V_s} M_0(p_i) = \sum_{j \in V_t} M(p_j) \tag{6.2}$$

Ainsi, calculer la séquence de coût minimum dans un réseau de Petri N revient à déterminer le flot maximal de coût minimum dans $G(N)$. Des algorithmes très efficaces sont connus pour ce problème [64]. Sur l'exemple, le vecteur caractéristique optimal est $\overline{\sigma} = (2 \quad 2 \quad 0)^t$ et le coût total de la séquence est 8. Il est simple de vérifier que la séquence $s = t_1t_1t_2t_2$ est tirable, ainsi que $s = t_1t_2t_1t_2$.

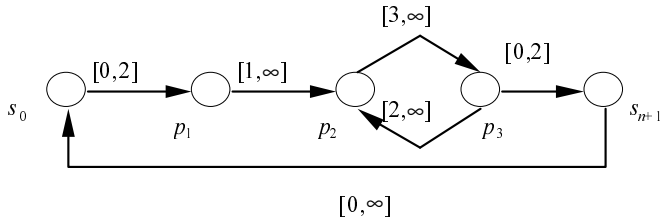


FIG. 6.2 – *Le réseau de transport $G(N)$ associé à la machine à état N de la figure 6.1 (l'arc (p_2,s_{n+1} n'a pas été représenté puisque sa capacité est nulle)*

Calculer une séquence de tirs

Nous ne décrivons pas une procédure pour calculer une séquence de tirs correspondant à un vecteur caractéristique. Ceci peut être facilement obtenu en utilisant un parcours en profondeur d'abord du graphe des marquages. L'explosion combinatoire correspondante peut être limitée par deux facteurs:

- le nombre de tirs de chaque transition est connu (c.f. paragraphe précédent).
- les séquences ordonnées (Ordered firing sequence [24]) permettent d'atteindre tout marquage accessible dans une machine à état.

6.3.2 Graphes d'événements vivants

Optimisation du vecteur caractéristique

Dans [24] est prouvé qu'une borne supérieure du nombre de tirs pour atteindre un marquage dans un graphe d'événement b-borné est $b.m(m-1)/2$, où m est le nombre de transitions du réseau. Comme pour les machines à état, le programme mathématique (6.1) donne une borne inférieure du nombre optimal de tirs pour minimiser la fonction objectif. Mais ce programme est aussi optimal puisque l'équation des marquages est une condition nécessaire et suffisante d'accessibilité pour les graphes d'événements vivants [24]. Ainsi, à chaque solution du programme linéaire pourra être construite une séquence de tirs. De plus, la matrice d'incidence d'un graphe d'événements est totalement unimodulaire. En conséquence, la résolution du programme (6.1) pourra être faite par solveur sur des nombres réels (i.e., l'intégrité des tirs peut être relâchée). Notons, qu'un meilleur algorithme peut être proposé pour résoudre l'équation des marquages d'un graphe d'événements vivant [20].

6.3.3 Calculer une séquence de tirs

Comme pour les machines à état, un parcours en profondeur du graphe des marquages permet de conduire à une séquence de tirs franchissables. Nous détaillons ci-après un sous-ensemble dominant de séquences de tirs qui permettra de limiter l'explosion combinatoire durant la construction du graphe des marquages.

Nous rappelons tout d'abord la définition d'une séquence de permutation.

Définition 9 *Une séquence σ est une permutation d'une séquence τ si toute transition apparaît le même nombre de fois dans τ et σ (i.e. $\overline{\sigma} = \overline{\tau}$)*

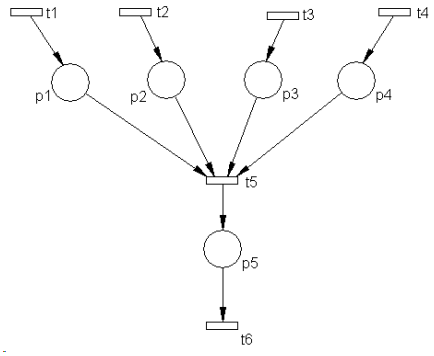


FIG. 6.3 – *Un graphe d'événements vivant*

Nous introduisons la notion de *séquence équilibrée*, qui définit une sous-classe des séquences de tir. Une séquence équilibrée impose une structure particulière des tirs des transitions. Les tirs des transitions ne pourront pas survenir n'importe quand, mais seulement dans des blocs disjoints de la séquence.

Définition 10 *Une séquence σ de longueur $L = \sum_{i=1}^{m} \overline{\sigma}(i)$ est équilibrée si elle peut être décomposée en L sous-séquences σ_0,σ_1...σ_{L-1} telles que la transition t_i apparaît dans σ_k si, et seulement*

$$\left\lceil \frac{k+1}{L}\overline{\sigma}_i \right\rceil - \left\lceil \frac{k}{L}\overline{\sigma}_i \right\rceil = 1 \tag{6.3}$$

Dans cette définition, les transitions survenant dans σ_k sont précisément définies. Mais l'ordre des tirs des transitions au sein des sous-séquences n'est pas fixé. En conséquence, cette classe de séquences peut être utilisée pour limiter l'explosion combinatoire de la construction du graphe des marquages, mais ne permet pas de l'éviter complètement (les séquences équilibrées vont jouer le même rôle que les séquences ordonnées pour les machines à état). Nous avons montré que tout marquage dans un graphe d'événements vivant peut être atteint par une séquence équilibré.

Théorème 17 *Soit (N,M_0) un graphe d'événements et σ une séquence telle que $M \stackrel{\sigma}{\longmapsto} M$, alors il existe une séquence équilibrée τ, permutation de σ, telle que $M_0 \stackrel{\tau}{\longmapsto} M$.*

Considérons le réseau de la figure 6.3.3 avec :

- le marquage initial $M_0 = (0 \quad 0 \quad 0 \quad 0 \quad 0)^t$
- le marquage final $M = (0 \quad 0 \quad 0 \quad 0 \quad 5)^t$.

Alors il est simple de montrer que le vecteur caractéristique optimal et le coût de la séquence optimal sont:

$$\begin{aligned} X &= (5 \quad 5 \quad 5 \quad 5 \quad 5 \quad 0) \\ L &= 25 \end{aligned}$$

k	$\triangle_1(k)$	$\triangle_2(k)$	$\triangle_3(k)$	$\triangle_4(k)$	$\triangle_5(k)$	$\triangle_6(k)$
0	1	1	1	1	1	0
1..4	0	0	0	0	0	0
5	1	1	1	1	1	0
6..9	0	0	0	0	0	0
10	1	1	1	1	1	0
11..14	0	0	0	0	0	0
15	1	1	1	1	1	0
16..19	0	0	0	0	0	0
20	1	1	1	1	1	0
21..24	0	0	0	0	0	0

TAB. 6.1 – *Calcul d'une séquence équilibrée*

Nous montrons sur un exemple la limitation de l'explosion combinatoire en se limitant aux séquences équilibrées pendant la construction du graphe de marquages. La construction de la séquence est donnée dans le tableau 6.1. La grandeur $\triangle_i(k)$ indique si une transition t_i doit être tirée à l'étape k. En accord avec la définition d'une séquence équilibrée, nous avons :

$$\triangle_i(k) = \left\lceil \frac{k+1}{L}\overline{\sigma}_i \right\rceil - \left\lceil \frac{k}{L}\overline{\sigma}_i \right\rceil \qquad L = \sum_{i=1}^{m} \overline{\sigma}_i \tag{6.4}$$

La structure de la séquence de tirs obtenue consiste à répéter 5 fois la sous-séquence : $\sigma = (t_1 t_2 t_3 t_4 t_5)$. Énumérer toutes les séquences franchissables conduit à $5 \times 5!$ séquences différentes alors que l'énumération complète nécessitera d'étudier 25! séquences. Un parcours en profondeur se limitant à la séquence équilibrée conduira à une séquence tirable dès la première branche explorée.

6.4 Collaborations et diffusion des résultats

Deux articles de synthèse sur la modélisation de système à l'aide de réseaux de Petri ont été publiés :

- un chapitre de livre dans l'ouvrage "An Introduction to Formal Specification", (éditions Springer Verlag) *[19]*. Ce chapitre a été écrit en collaboration avec A. Geniet (LISI, Poitiers).
- un article dans la collection Informatique Industrielle des Techniques de l'Ingénieur (éditions Techniques de l'Ingénieur) *[88]*. Cet article a été écrit en collaboration avec C. Haro (Laboratoire d'Informatique, Tours)

La détermination des plus courtes séquences de tir dans un graphe des marquages d'un réseau de Petri a été présentée à la conférence internationale Emerging Technologies and Factory Automation (ETFA'99, éditions IEEE) *[80]*.

Troisième partie

Perspectives de recherches

Chapitre 7

Perspectives de recherche

7.1 Introduction

L'évolution des technologies matérielles et des langages applicatifs, le déploiement massif des systèmes informatiques autonomes intégrant des fonctions *temps réel* sont des sources inépuisables de nouveaux problèmes de recherche en informatique temps réel. Il serait illusoire de vouloir dresser une liste exhaustive des problèmes et des techniques émergentes pour les affronter.

La théorie de l'ordonnancement développée dans ce document repose principalement sur le paramétrage et la validation après conception du système. Bien sûr, l'inconvénient est de démontrer très *tardivement* que l'application respectera ses spécifications temporelles. Mais le comportement temporel et la prédictibilité d'un système dépendent de la plateforme d'exécution. Par exemple, le temps de réponse à un événement dépendra des durées d'exécution des tâches le traitant, des caractéristiques temporelles des cartes d'entrée/sortie, du fonctionnement du noyau temps réel.... Tout cela ne pourra jamais être défini avant l'étape d'implantation de l'application.

Nous souhaitons continuer à travailler sur l'ordonnancement d'un système après conception et choix de la plateforme d'exécution. Nous insistons sur le fait que le problème d'ordonnancement survient quels que soient les outils de développement (méthodes de spécifications, langage de programmation...). Les choix d'implantation reviennent à définir l'allocation des ressources du système (y compris les processeurs) aux tâches. Ces choix sont la nature même de tout problème d'ordonnancement.

Nous nous limiterons ci-après à quatre thématiques sur l'ordonnancement en-ligne. Elles constituent un prolongement des travaux présentés dans la première partie de ce document :

- les techniques d'analyse de l'ordonnançabilité des tâches,
- l'ordonnancement monoprocesseur avec suspension des tâches,
- l'ordonnancement monoprocesseur à vitesse variable,
- l'ordonnancement dans les systèmes distribués.

7.2 Analyse d'ordonnançabilité

Les techniques d'analyse de l'ordonnançabilité conduisent soit à une analyse exacte, soit à une analyse approchée. Dans le dernier cas, le comportement pire cas n'est pas connu. Par exemple,

l'analyse du temps de réponse dans les systèmes distribués (analyse holistique) est souvent désignée comme pessimiste. Mais aucune étude démontre quel est le *niveau de pessimisme*. Il nous apparaît très important d'établir une mesure quantitative de ce pessimisme. Nous pensons que l'analyse de compétitivité permettra de mesurer l'efficacité des méthodes d'analyse de l'ordonnançabilité des tâches avec des méthodes optimales (i.e., construction d'un ordonnancement hors-ligne).

7.3 Ordonnancement avec suspension des tâches

Dans le chapitre 2, nous avons étudié la complexité du problème d'ordonnancement de tâches pouvant se suspendre durant leur exécution. Contrairement au problème de gestion des accès concurrent aux ressources partagées, la suspension des tâches n'a pas reçu de solution adaptée. Les algorithmes d'ordonnancement classiques en monoprocesseur, tels que Rate Monotonic, Deadline Monotonic, Earliest Deadline First et Least Laxity First, ne fournissent pas un moyen satisfaisant pour ordonnancer des tâches pouvant se suspendre. Ces problèmes sont actuellement en cours d'étude dans le cadre de la thèse de Mr Frédéric Ridouard.

Tout d'abord, la complexité du problème d'ordonnancement de tâches périodiques à démarrage simultané et à échéance sur requête est ouvert. Nous pensons résoudre ce problème rapidement et conjecturons dores et déjà sa $\mathcal{NP}$-complétude au sens fort. En conséquence, même en hors-ligne, le problème sera très difficile à résoudre.

Dans un second temps, nous allons chercher à mesurer l'inefficacité des algorithmes d'ordonnancement en-ligne. Nous pensons qu'il est toujours possible de construire des instances de tâches, avec un facteur d'utilisation inférieur à une constante c arbitrairement petite, telles que l'algorithme d'ordonnancement en-ligne n'exécute pas fiablement les tâches alors qu'un algorithme hors-ligne le fera. Nous pensons rapidement démontrer ces résultats pour les algorithmes Rate Monotonic, Deadline Monotonic, Earliest Deadline First et Least Laxity First. Le même sort est peut-être réservé à tout algorithme en-ligne ! Nous souhaitons aborder cette catégorie de problèmes avec l'analyse de compétitivité (c.f. chapitre 4) afin de déterminer une analyse pire cas.

Dans un troisième temps, nous pensons important de proposer de nouveaux algorithmes d'ordonnancement afin de les comparer statistiquement avec les méthodes classiques d'ordonnancement. L'idée est de prendre en compte explicitement les pires durées de suspension des tâches dans l'algorithme en-line. Nous espérons ainsi faire émerger de nouvelles techniques d'ordonnancement en-ligne.

7.4 Ordonnancement de processeur à vitesse variable

L'ordonnancement de processeur à vitesse variable est actuellement très largement étudié. La technologie existe, mais les méthodes permettant d'optimiser la consommation d'énergie ne répondent pas de façon optimale à ce problème. Les constructeurs proposent des outils propriétaires, opaques, laissant peu de degré de liberté aux concepteurs du systèmes (LongRun System chez Transmeta et XScale Technology chez Intel). Les travaux actuels étudient la minimisation de l'énergie pour des systèmes de tâches simplistes au regard des applications industrielles.

La majorité de ces travaux supposent en effet des systèmes de tâches pour lesquels une condition nécessaire et suffisante d'ordonnançabilité est connue (tâches à démarrage simultané et à échéance sur requête). Le calcul des vitesses des tâches induit une dimension supplémentaire dans le problème d'ordonnancement.

Le premier point à aborder serait donc de mesurer la dépendance entre le calcul des vitesses d'exécution des tâches avec la méthode d'ordonnancement. Nous pouvons faire l'analogie avec l'ordonnancement des systèmes distribués par la dépendance entre le placement des tâches et leur ordonnancement. Nous désirons plus précisément proposer une méthode affectant une priorité et une vitesse constante pour chaque tâche. Ces deux opérations devant être réalisées simultanément afin de garantir l'optimalité (dans la classe des algorithmes à priorité fixe et instances des tâches à vitesse constante).

Un second point nous semble très intéressant : l'influence du choix de la vitesse du processeur dans les protocoles d'accès au ressource. De la même façon qu'une tâche détenant une ressource voit sa priorité augmentée, est-il tout aussi pertinent d'augmenter sa vitesse d'exécution ?

7.5 Ordonnancement de système distribué

Les systèmes distribués demeurent un large domaine d'étude en ordonnancement temps réel. L'intégration de nouvelles technologies telles que des plateformes multiprocesseurs et des nouveaux réseaux, nécessite de nombreuses recherches afin de proposer des méthodes efficaces de validation (analyse d'ordonnançabilité) afin d'éviter le surdimensionnement systématique du système.

Dans le cadre de la thèse de Michaël Richard, nous avons proposé une méthode de placement et d'affectation de priorité fixe aux tâches. La méthode considère des contraintes de placement simples puisque les tâches ne peuvent être affectées que sur des processeurs identiques. Il serait très intéressant de relaxer cette contrainte et de considérer en plus des placements imposés par le concepteur (tâches dédiées). Un autre axe de développement serait de minimiser l'espace mémoire nécessaire sur chaque processeur afin d'optimiser le placement. Ce dernier critère est très important pour réduire les coûts de production en grande série du système, comme dans l'industrie automobile. En conséquence, la méthode optimiserait aussi l'architecture du système (Hardware/Software CoDesign).

Bibliographie

[1] Anderson, E., and Potts, C. On-line scheduling of a single machine to minimize total weighted completion time. *in: proc. ACM-SIAM Symposium on Discrete Algorithms* (2002), 548–557.

[2] Audsley, N. On priority assignment in fixed priority scheduling. *Information Processing Letters 79*, 1 (2001), 39–44.

[3] Baker, K. *Introduction to Sequencing and Scheduling.* John Wiley and Sons, 1984.

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

[5] Baruah, S., and Goossens, J. Scheduling real-time tasks: algorithms and complexity. *in Handbook of scheduling: algorithms, models and performance analysis J.Y.T Leung (Ed), Chapman and Hall, CRC Press* (2003).

[6] Baruah, S., Haritsa, J., and Sharma, N. On-line scheduling to maximize task completions. *proc. Real-Time Systems Symposium* (1994), 228–236.

[7] Baruah, S., Haritsa, J., and Sharma, N. On-line scheduling to maximize task completions. *Journal of Combinatorial Mathematics and Combinatorial Computing 39* (2001), 65–78.

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

[9] Baruah, S. K. A general model for recurring real-time tasks. *proc. IEEE Real-Time Systems Symposium* (1998).

[10] Borodin, A., and R.El-Yaniv. *Online Computation and Competitive analysis.* Cambridge University Press, 1998.

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

[12] Brucker, P. *Scheduling algorithms.* Springer Verlag, 2001.

[13] Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M., Potts, C., Tautenhahn, T., and de Velde, S. V. Scheduling a batching machine. *Journal of Scheduling 1*, 1 (1998), 31–58.

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

[15] Carlier, J., and Chrétienne, P. *Problèmes d'ordonnancement: modélisation/complexité/algorithmes.* Masson, 1988.

[16] Cassez, F., and Roux, O. Automates hybrides à files. Seminaires du GDR/ARP/STS, 11 décembre 1998.

[17] Castelpietra, P., Song, Y., Simonot-Lion, F., and Cayrol, O. Performance evalutation of a multiple networked in-vehicle embedded architecture. *proc. Workshop on Factory Communication Systems, Porto* (2000).

[18] Charon, I., Germa, A., and Hudry, O. *Méthode d'optimisation combinatoire.* Masson, Paris, 1996.

[19] Choquet-Geniet, A., and Richard, P. Petri nets : a graphical tool for system modelling. *An Introduction to Formal Specification, Springer Verlag* (2000), 241–257.

[20] Chrétienne, P. Timed petri nets: a solution to the minimum-time-reachability problem between two states of a timed event graph. *The Journal of Systems and Software 1,2* (1986), 95–101.

[21] Colom, J., and Silva, M. Improving the linearly based characterization of p/t nets. *in: Advances in Petri nets'90, Springer Verlag*, 1 (1990), 79–112.

[22] Conway, R., Maxwell, W., and Miller, L. *Theory of scheduling.* Addison-Wesley, 1967.

[23] Desel, J., and Esparza, J. Shortest paths in reachability graphs. *in: proc. of Application and theory of Petri nets'93, LNCS 691* (1993), 224–241.

[24] Desel, J., and Esparza, J. *Free-Choice Petri nets.* Cambridge University Press, 1995.

[25] Devi, U. An improved schedulability test for uniprocessor periodic task systems. *IEEE Euromicro Conf. on Real-Time Systems (ECRTS'03)* (2003), 23–30.

[26] Epstein, L., and Van-Stee, R. Lower bounds for on-line single-machine scheduling. *Theoretical Computer Science*, 299 (2003), 439–450.

[27] Finta, L., and Liu, Z. Single-machine scheduling subjected to precedence delays. *Discrete Applied Mathematics 70* (1996), 247–266.

[28] Garey, M. R., and Johnson, D. S. *Computers and intractability: a guide to the theory of NP-Completeness.* W.H. Freeman and co, New York, 1979.

[29] Gaujal, B., and Navet, N. Ordonnancement sous contrainte de temps et d'énergie. In *Actes de l'école d'été temps réel (ETR'03)* (9-12 Septembre 2003), pp. 263–276.

[30] Geniet, A., and Grolleau, E. Minimal schedulability interval for real time systems of periodic tasks with offsets. *Theoretical Computer Science* (2003), .

[31] Goossens, J., and Richard, P. Performance optimization for hard real-time fixed priority tasks. *proc. Real-Time and Embedded Systems (RTS'04), Paris* (2004).

[32] Graham, R. Bounds on multiprocessing anomalies. *SIAM Journal of Applied Mathematic 17* (1969), 263–269.

[33] Gutierrez-Garcia, J., and Gonzalez-Harbour, M. Optimized priority assignment for tasks and messages in distributed hard real-time systems. *proc. Workshop on Parallel and Distributed Hard Real-Time Systems, Santa Barbara* (1995).

[34] Gutierrez-Garcia, J., Palencia-Gutierrez, J., and Gonzalez-Harbour, M. Schedulability analysis of distributed hard real-time systems with multiple-event synchronization. In *Euromicro Real-Time Systems* (2002), pp. 15–24.

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

[36] Harbour, M., Klein, M., and Lehoczky, J. Fixed priority scheduling of periodic tasks with varying execution priority. In *Proceedings of Real-Time Systems Symposium* (1991), San Antonio (Texas), pp. 116–128.

[37] Hoogeveen, J., Potts, C., and Woeginger, G. On-line scheduling on a single machine: maximizing the number of early jobs. *Operations Research Letters 27* (2000), 193–197.

[38] Hoogeveen, J., and Vestjens, A. Optimal on-line algorithms for single-machine scheduling. *in: proc. 5th Conference on Integer Programming and Combinatorial Optimization* (1996), 404–414.

[39] Hoogeveen, J., and Vestjens, A. A best possible determinitic on-line algorithm for minimizing maximum delivery time on a single machine. *SIAM Journal of Discrete Mathematics 13*, 1 (2000), 56–63.

[40] ISO. Road vehicles - low speed serial data communication - part 2: low-speed controller area network. *ISO 11519-2* (1994).

[41] ISO. Vehicle area network, serial data communication - road vehicle, serial data communication for automotive application. *ISO 11519-3* (1994).

[42] Jeannenot, S. Ordonnancement temps réel avec contrainte de consommation d'énergie dans les systèmes embarqués. *Rapport de stage de DEA T3iA, ENSMA et Université de Poitiers* (2003).

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

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

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

[46] Kaiser, C. Description et critique d'un système temps réel pour le suivi d'un laminoir. *Technique et Science Informatiques 20*, 2 (2001), 179–211.

[47] Kalyanasundaram, B., and Pruhs, K. Speed is as powerful as clairvoyance. *in: proc. IEEE Foundations of Computer Science* (1995), 214–223.

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

[49] Lawler, E., Lenstra, J., RinnooyKan, A., and Shmoys, D. *Sequencing and Scheduling: Algorithms and Complexity.* Eindhoven University of Technology, Dpt of Mathematics and Computing Science, PO Box 513, 1989.

[50] Lee, C., and Uzsoy, R. Minimizing makespan on a single batch processing machine with dynamic job arrivals. *International Journal of Production Research 37*, 1 (1999), 219–236.

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

[52] Lehoczky, J. P. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *proc. IEEE Real-Time System Symposium* (1990), 201–209.

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

[54] Leung, J., and Whitehead, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation 2* (1982), 237–250.

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

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

[57] Liu, Z., and Yu, W. Scheduling one batch processor subject to job release dates. *Discrete Applied Mathematics 105* (2000), 129–136.

[58] MEJIA-ALVAREZ, P., LEVNER, E., and MOSSE, D. An integrated heuristic approach to power aware real-time scheduling. In *Workshop on Power-Aware Computer Systems (PACS'02), LNCS 2325, Springer Verlag* (2002).

[59] MEJIA-ALVAREZ, P., LEVNER, E., and MOSSE, D. Power-optimized scheduling server for real-time tasks. In *Eighth IEEE Real-Time and Embedded Technology and Applications Symposium* (25-27 Septembre 2002), pp. 239–253.

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

[61] Munier, A. Régime asymptotique optimal d'un graphe d'événements temporisés généralisés : application à un problème d'assemblage. *RAIRO APII 27*, 5 (1993), 487–513.

[62] Munier, A. The basic cyclic scheduling problem with linear precedence constraints. *Discrete Applied Mathematics 64* (1996), 219–238.

[63] Nakamura, N., and Silva, M. Cycle time computation in deterministically timed weighted marked graphs. In *Proc: IEEE Conf. on Emerging technologies and Factory Automation* (1999), pp. 1037–1046.

[64] Nemhauser, G., and Wolsey, L. *Integer and Combinatorial optimization.* John Wiley & Sons, New York, 1979.

[65] Palletwuori, K., Luostarinen, P., Muurinen, K., and Nevalainen, O. On the scheduling of a multipurpose laboratory analysis instrument. *Technical Report 34, Turku Centre for Computer Science, Finland* (july 1996).

[66] Palletwuori, K., Luostarinen, P., Muurinen, K., and Nevalainen, O. On the scheduling of a multipurpose laboratory analysis instrument. *in: Euromicro Conference on Real-Time Systems (WIP session)* (1997), 181–181.

[67] Parain, F., Banatre, M., Cabiliic, G., Higuera, T., Issarny, V., and Lseot, J. Tenchiques de réduction de la consommation dans les systèmes embarqués temps réel. *INRIA Research report*, 3932 (May 2000).

[68] Phillips, C., Stein, C., and Wein, J. Scheduling jobs that arrive over time. *in: proc. 4th Workshop on Algorithms and Data Structures, LNCS 955, Springer Verlag* (1995), 86–97.

[69] Poon, C., and Zhang, P. Minimizing makespan in batch machine scheduling. *in: proc ISAAC 2000, LNCS, Springer Verlag 1969* (2000), 386–397.

[70] Puaut, I. Méthodes de calcul du wcet (worst-case execution time) - etat de l'art. *Ecole d'été temps réel, Toulouse (France)*, 1 (2003), 263–276.

[71] Redell, O., and Sanfridson, M. Exact best-case response time analysis of fixed-priority scheduled tasks. In *Euromicro Real-Time Systems* (2002).

[72] Richard, M. *Contribution à la Validation des Systèmes Temps Réel Distribués : Ordonnancement à Priorités Fixes et Placement.* PhD thesis, École Nationale Supérieure de Mécanique et d'Aérotechnique – Université de Poitiers, Novembre 2002.

[73] Richard, M., and Richard, P. Une approche graphique pour l'aide à la conception d'applications temps réel ordonnançables. *Technique et Science Informatiques 21*, 3 (2002), 315–343.

[74] Richard, M., and Richard, P. Méthode de placement et d'affectation des priorités pour les systèmes temps réel distribués. *Technique et Science Informatiques* (2003, à paraître).

[75] Richard, M., Richard, P., and Cottet, F. Task and message priority assignment in automotive systems. In 4^{th} *FeT IFAC Conference on Fieldbus Systems and their Applications* (15,16 November 2001), Elsevier, Ed., pp. 105–112.

[76] Richard, M., Richard, P., and Cottet, F. Allocating and scheduling tasks in multiple fieldbus real-time systems. *IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA'03), Lisbon (Portugal) 1* (2003), 137–144.

[77] Richard, M., Richard, P., and Cottet, F. Placement et validation dans les systèmes temps réel distributes. *proc. Real-Time and Embedded Systems (RTS'03), Paris* (2003).

[78] Richard, M., Richard, P., Grolleau, E., and Cottet, F. Contraintes de précédences et ordonnancement mono-processeur. *proc. Real-Time and Embedded Systems (RTS'02), Paris* (2002), 121–138.

[79] Richard, P. *Contribution des réseaux de Petri à l'étude des problèmes de recherche opérationnelle.* PhD thesis, Université de Tours, 1997.

[80] Richard, P. Optimal shortest path in reachability graph. *IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA'99), Barcelone (Spain) 1* (1999), 303–312.

[81] Richard, P. Modelling integer linear programs with petri nets. *Rairo/Operations Research 34*, 3 (2000), 305–312.

[82] Richard, P. Controlling response time in real-time systems. In *Computer Performance Evaluation: Modelling Techniques and Tools, LNCS 2324, Springer Verlag* (2002), pp. 339–348.

[83] Richard, P. Analyse du temps de réponse des systèmes temps réel. *Ecole d'été temps réel, Toulouse (France)*, 1 (2003), 241–262.

[84] Richard, P. On the complexity of scheduling real-time tasks with self-suspensions on one processor. *Euromicro Conf. on Real-Time Systems (ECRTS'03), IEEE Computer Press* (2003), 187–194.

[85] Richard, P., Cottet, F., and Kaiser, C. Validation temporelle d'un logiciel temps réel : application à un laminoir industriel. In *IEEE Conférence Internationale Francophone d'Automatique (CIFA'00), Lille (France)* (2000), pp. 687–692.

[86] Richard, P., Cottet, F., and Kaiser, C. On-line scheduling of real-time distributed computers with complex communication constraints. In *IEEE International Conference on Emerging Complex Computer Systems, (ICECCS'01), Skövde (Sweden), IEEE Computer Press* (2001), pp. 26–34.

[87] Richard, P., Cottet, F., and Kaiser, C. Précédences généralisées et ordonnançabilité des tâches de suivi temps réel d'un laminoir. *Journal Européen des Systèmes Automatisés 35*, 9 (2001), 1055–1071.

[88] Richard, P., and Haro, C. Application des réseaux de petri. *Techniques de l'ingénieur, traité Informatique Industriel*, S 7 254 (2001), 12p.

[89] Richard, P., Richard, M., and Cottet, F. Analyse holistique des systèmes temps réel distribués: principes et algorithmes. *in: ordonnancement pour l'informatique parallèle, Traité IC2, Hermès* (juin 2003), .

[90] Richard, P., Ridouard, F., and Martineau, P. On-line scheduling on a single batching machine to minimize the makespan. *Industrial Ingeneering and Production Management (IEPM'03), Porto (Portugal)* (2003).

[91] Ridouard, F. Ordonnancement temps réel : une machine à traitement par lot. *Rapport de stage de DEA T3iA, ENSMA et Université de Poitiers* (2003).

[92] Ridouard, F., Richard, P., and Cottet, F. Ordonnancement en-ligne à une machine. *Ecole d'Automne de Recherche Opérationnelle, Tours* (2003).

[93] Rusu, C., Melhelm, R., and MOSSE, D. Maximizing the system value while satisfying time and energy constraint. In *proc Real-Time Systems Symposium (RTSS'02)* (2002).

[94] Sgall, J. On-line scheduling - a survey. *in: On-line algorithms. The state of the art, LNCS, Springer Verlag 1442* (1998), 196–231.

[95] Spuri, M., and Stankovic, J. A. How to integrate precedence constraints and shared resources in real-time scheduling. *IEEE Transactions on Computers 43*, 12 (1994), 1407–1412.

[96] Stankovic, J., Spuri, M., DiNatale, M., and Buttazzo, G. Implications of classical scheduling results for real-time systems. *IEEE Computer 28*, 6 (1995), 16–25.

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

[98] Tindell, K., Burns, A., and Wellings, A. Allocating hard real-time tasks: an np-hard problem made easy. *J. Real-Time Time Systems 4* (1992), 145–165.

[99] Trinquet, Y. Noyaux temps réel : le cas osek/vdx. *Ecole d'été temps réel, Toulouse (France)*, 1 (2003), 227–240.

[100] Ullman, J. D. Np-complete scheduling problems. *Journal of Computer and System Sciences 10* (1975), 384–393.

[101] van den Akker, M., Hoogeveen, H., and Vakhania, N. Restarts can halp in the on-line minimization of the maximum delivery time on a single machine. *proc. European Symposium on Algorithms* (2000), 427–436.

[102] Van-Stee, R., and Poutré, J. L. Minimizing total completion time on-line on a single machine, using restarts. *10th European Symposium on Algorithms, Lecture Notes in Computer Science, Springer Verlag* (2002).

[103] Xu, J., and Parnas, D. L. Priority scheduling versus pre-run-time scheduling. *Journal of Real-Time Systems 18*, 1 (2000), 7–23.

[104] Zhang, G., Cai, X., and Wong, C. On-line algorithms for minimizing makespan on batch processing machines. *Naval Research Logistics 48* (2001), 241–258.