Optimisation de l'ordonnancement sous contrainte de faisabilité

Mathieu Grenier

Nancy-Université
INPL

Département de formation doctorale en informatique

Institut National
Polytechnique de Lorraine

École doctorale IAEM Lorraine

Optimisation de l'ordonnancement sous contrainte de faisabilité

THÈSE

présentée et soutenue publiquement le 26 octobre 2006

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine

(spécialité informatique)

par

Mathieu Grenier

Composition du jury

Président :	Maryline Silly-Chetto
Rapporteurs :	Pascal Richard
	Gilles Muller
Examinateurs :	Françoise Simonot-Lion
	Nicolas Navet
	Stéphan Merz
	Maryline Silly-Chetto
	Joël Goossens

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria

Mis en page avec la classe thloria.

Remerciements

Je remercie toute l'équipe TRIO et plus particulièrement Françoise Simonot-Lion et Nicolas Navet.

A Emilie
pour son soutien.
Et á toute ma famille.

Table des matières

Chapitre 5
Nouvelles politiques à priorité fixe dans les systèmes "offset free"

Chapitre 6
Configuration d'ordonnancement sous Posix 1003.1b

Annexe C
Chapitre 5

Annexe D
Chapitre 6

Première partie

Introduction

Chapitre 1

Contexte et problématique

Le travail présenté dans ce document porte sur l'ordonnancement temps réel optimisé d'activités sur une ressource unique. Deux cas sont étudiés : le cas de tâches indépendantes périodiques s'exécutant sur un processeur et le cas de flux de messages indépendants périodiques sur un réseau de terrain avec accès au médium priorisé.

Par temps réel, nous entendons respect des contraintes de temps exprimées sur les activités du système ; ceci amène à traiter un premier problème qui est celui de la **faisabilité**. Par ailleurs, le deuxième problème pris en compte porte sur la notion d'**optimisation de l'ordonnancement** et recouvre deux concepts :

- optimisation de l'utilisation de la plate-forme d'exécution : utiliser au mieux le potentiel de la plate-forme d'exécution tout en garantissant le respect des contraintes temporelles imposées au système. Concrètement, cela permet soit d"'accepter" des systèmes induisant une charge élevée sur une plate-forme donnée, soit d'utiliser une plate-forme composée de ressources moins puissantes pour un système donné. Dans les deux situations, l'usage de tout le potentiel de la plate-forme permet d'utiliser des composants moins coûteux et de réduire de manière significative les coûts de production,
- optimisation des critères applicatifs de qualité de service propres à l'application : il s'agit de garantir la faisabilité tout en optimisant les performances de l'application. Par exemple, un objectif généralement visé dans les applications de contrôle-commande est de minimiser la variabilité sur les dates de fin d'exécution des tâches (ou gigues).

L'objectif de la thèse est de concevoir des algorithmes d'ordonnancement temps réel en-ligne qui soient faisable et qui "optimisent" l'ordonnancement. Dans ce contexte, nous proposons d'aborder le problème en fournissant des méthodes et des politiques d'ordonnancement **génériques**, *i.e.*, non-spécifiques à un problème donné et qui, de plus, doivent pouvoir être mises en oeuvre sur des **composants standards du commerce**.

1.1 Temps réel et ordonnancement

Avant d'introduire les modèles utilisés et la problématique traitée, nous définissons l'ordonnancement temps réel.

1.1.1 Définitions du temps réel

Une définition des systèmes temps réel a été introduite par Stankovic [2] :

Définition 1 *[2] Un système temps réel est un système informatique dont le bon fonctionnement ne dépend pas uniquement de la correction logique des résultats, mais aussi des instants de production de ces résultats.*

Elloy [3] propose une définition alternative à celle-ci :

Définition 2 *[3] Peut être qualifié de "temps réel" toute application mettant en oeuvre un système informatique dont le fonctionnement est assujetti à l'évolution dynamique de l'état d'un environnement (appelé ici procédé) qui lui est connecté et dont il doit contrôler le comportement.*

Pour suivre et piloter l'état du procédé, le système informatique de contrôle doit alors généralement contenir un ensemble d'actions (tâches) dont les instants de début d'exécution sont, directement ou indirectement, assujettis aux signaux émis (événements) ou prélevés (mesures) sur ce procédé, et dont les instants de terminaison d'exécution sont soumis à des contraintes temporelles imposées par le procédé.

Ainsi, un système temps réel est donc composé d'applications ayant des interactions fortes avec leur environnement. Comme l'environnement change avec le temps, les systèmes temps réel doivent réagir "assez vite" et , plus précisemment, au bon moment pour contrôler le procédé.

Comme il est souligné par Elloy [3] dans la définition 2, les systèmes temps réel sont composés d'un ensemble d'actions (tâches) que la spécification peut faire apparaître comme parallèles (parallélisme d'expression [3]).

Enfin, le dernier point important concernant ces systèmes est leur sûreté. Leur exécution doit être maîtrisée et prévisible. En particulier, le parallélisme d'expression des actions (tâches) lorsque celles-ci sont "repliées" [3] sur une plate-forme d'exécution impose d'assurer que les contraintes de temps sont respectées par les politiques d'ordonnancement de ces activités/tâches sur les ressources.

1.1.2 Ordonnancement temps réel

Pour les systèmes composés d'un ensemble d'activités parallèles, les ressources d'exécution doivent donc être partagées par ces activités dans le temps ; il faut donc donner un ordre sur ces activités (exécution des tâches, transmission de messages), c'est-à-dire de résoudre des problèmes d'**ordonnancement**. Dans le cadre des systèmes temps réel, le respect des contraintes de temps, généralement des contraintes de dates au plus tard appelées échéances, est imposé par l'environnement. Le problème de l'ordonnancement temps réel est donc de donner un ordre sur les activités du système qui respecte les contraintes de temps imposés par l'environnement. En fonction des conséquences du non-respect des contraintes de temps, on distingue deux classes d'activités temps réel :

- les activités à contraintes dites *dures (hard)* où le non-respect des contraintes de temps est catastrophique et est donc de ce fait interdit,
- les activités à contraintes *souples (soft)* où le non-respect occasionnel des contraintes de temps dégrade seulement les performances du système.

Dans cette étude, nous considérons les **systèmes temps réel à contraintes dures** et un ordonnancement sera dit faisable si toutes les contraintes de temps sont respectées.

1.1.3 Classification des algorithmes d'ordonnancement

Un algorithme d'ordonnancement définit à tout instant quelle tâche active, *i.e.*, "ayant du travail", exécuter sur la ressource. Un algorithme d'ordonnancement peut être oisif ou non-oisif. Un algorithme d'ordonnancement non-oisif doit exécuter une tâche s'il y a du travail en attente, *i.e.*, s'il y a une tâche active. Un algorithme d'ordonnancement oisif peut ne pas allouer le processeur même s'il y a du travail en attente. Dans cette étude, nous considérons les **algorithmes d'ordonnancement non-oisif**.

Les algorithmes d'ordonnancement sont développés autour de trois paradigmes : exécuté selon une table *(table driven)*, à priorités constantes et à priorités dynamiques. Généralement, on distingue deux classes d'algorithmes d'ordonnancement ; les algorithmes sont dits hors-ligne ou en-ligne si la majorité des décisions d'ordonnancement sont prises respectivement avant ou pendant l'exécution du système.

ordonnancement hors-ligne : une séquence d'ordonnancement "valide", *i.e.*, qui respecte les contraintes de temps, est construite. Le plus fréquemment une table d'ordonnancement est construite pour déterminer à chaque instant quelle tâche exécuter, cf. [4] par exemple.

ordonnancement en-ligne : une priorité est allouée aux instances des tâches et à chaque instant la tâche ayant la plus grande priorité est exécutée. Dans cette classe, on distingue deux sous-classes de politiques d'ordonnancement :

- indépendantes du temps : la priorité d'une instance d'une tâche est constante. Dans cette classe, on peut distinguer deux types de politiques :
  - à priorité fixe *(fixed priority)* : politiques qui attribuent une priorité unique à toutes les instances d'une tâche. Les politiques à priorité fixe sont au coeur des systèmes d'exploitation temps réel. Les politiques d'ordonnancement bien connues tel que RM (*Rate Monotonic* - plus petite la période, plus grande la priorité), DM (*Deadline monotonic* - plus petite l'échéance relative, plus grande la priorité) font partie de cette classe.
  - à priorité dynamique (*dynamic priority*) : politiques qui attribuent une priorité constante à chaque instance d'une tâche en fonction de cette instance. La politique EDF (*Earliest Deadline First* - plus petite l'échéance absolue, plus grande la priorité) appartient à cette classe.
- dépendantes du temps : la priorité d'une instance d'une tâche peut changer au cours du temps. Les politiques d'ordonnancement non-préemptives, *i.e.*, une instance garde la ressource jusqu'à la fin de son exécution, appartiennent à cette classe.

Des méthodes ont été développées pour tester la faisabilité de l'ordonnancement avec ces politiques. Afin de comprendre l'intérêt de nos contributions, certains de ces résultats seront détaillés en fonction des besoins de nos études.

1.2 Modélisation du système

Dans cette étude, nous considérons des systèmes temps réel où les caractéristiques des activités du système (ex : pire temps d'exécution, échéances, etc.) sont parfaitement identifiées. Nous faisons les

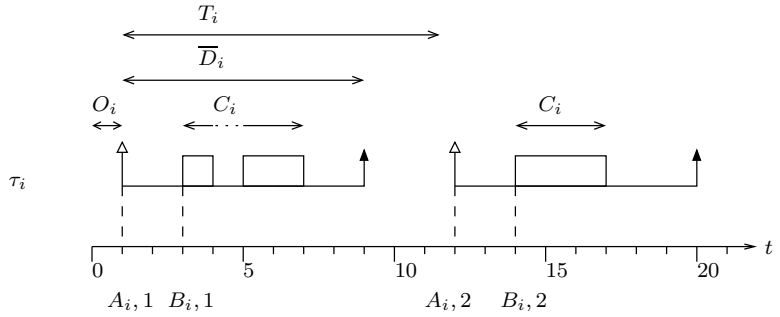


FIG. 1.1 – Modèle de tâche périodique τ_i caractérisée par le tuple $(C_i, T_i, \overline{D}_i, O_i)$ où $\tau_{i,j}$ la $j^{\text{ème}}$ activation de τ_i, appelée instance, arrive à l'instant $A_{i,j} = O_i + (j-1)$ et débute son exécution à l'instant $B_{i,j}$.

hypothèses suivantes :

- il y a une seule ressource (dans notre étude le processeur ou le réseau de communication),
- toutes les activités sont périodiques (ou parfois sporadiques avec un temps minimal inter-arrivée),
- toutes les activités sont indépendantes, (ex. : pas de contraintes de précédence, pas de partage de ressource autre que la ressource principale, etc.),
- toutes les activités ont un pire temps d'exécution (ou de transmission) connue,
- aucune activité ne peut interrompre elle-même son exécution,
- dans le cas préemptif, les surcharges dues aux changements de contexte sont négligées.
- sans perdre de généralité, selon Baruah et *al.* [5], toutes les caractéristiques des activités sont des multiples d'un tic (le temps est discret).

La résolution des problèmes d'ordonnancement présentés précedemment demande de disposer d'un modèle du système et de sa plate-forme d'exécution. Nous utilisons le modèle classique suivant où sont explicitées les activités du systèmes (tâches ou flux) ; la plate-forme est prise en compte dans le temps d'exécution/transmission des activités.

1.2.1 Système mono-processeur : modèle de tâche

Dans le cadre de l'ordonnancement de tâches sur une ressource mono-processeur, nous utilisons le modèle introduit par Liu et Layland [6]. Les activités sont des tâches périodiques indépendantes τ_i. Le système est modélisé par un ensemble de n tâches $\mathcal{T}$, $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$. Une tâche périodique τ_i est caractérisée par un tuple $(C_i, T_i, \overline{D}_i, O_i)$ où chaque requête de τ_i, appelée instance, a un pire temps d'exécution C_i, une échéance relative $\overline{D}_i$. T_i unités de temps séparent deux instances consécutives de τ_i, T_i est donc la période de la tâche. Nous notons $\tau_{i,j}$ la $j^{\text{ème}}$ activation de τ_i. La première instance $\tau_{i,1}$ de τ_i arrive O_i unités de temps après l'initialisation du système. Dans la suite, O_i est appelé l'*offset* de la tâche, *i.e.*, le "décalage initial". Par définition, l'instance $\tau_{i,j}$ arrive à l'instant $A_{i,j} = O_i + (j-1) \cdot T_i$. L'instant auquel une instance$\tau_{i,j}$ gagne l'accès au processeur pour la première fois est noté $B_{i,j}$. La figure 1.1 illustre ces notations.

1.2.2 Système distibué : modèle de flux

Nous étendons le modèle précédent au cas des systèmes distribués. Dans ces systèmes, la ressource partagée est le réseau de communication. Comme nous l'avons dit précedemment, nous considérons que les stations du réseau échangent des flux de messages périodiques. Le système est modélisé par un ensemble de n stations $\mathcal{T} = \{\mathcal{T}^1, \mathcal{T}^2, ..., \mathcal{T}^n\}$. Chaque station k est composée d'un ensemble de m_k flux $\mathcal{T}^k = \{\tau_1^k, \tau_2^k, ..., \tau_{m_k}^k\}$ où τ_i^k est le $i^{\text{ème}}$ flux de la station k. Le flux τ_i^k génère et met en file d'attente périodiquement une trame, où on note $\tau_{i,j}^k$ la $j^{\text{ème}}$ trame du flux τ_i^k. Un flux τ_i^k est caractérisé par un tuple $(T_i^k, C_i^k, \overline{D}_i^k, P_i^k, O_i^k)$ où :

- k est l'indice de la station qui envoie les trames,
- T_i^k est la période de transmission du flux τ_i^k,
- C_i^k est le pire temps de transmission (incluant le bit-stuffing et le temps inter-trame maximal),
- $\overline{D}_i^k$ est l'échéance relative de la trame (ex. : la trame doit être reçue 10ms après avoir été crée),
- P_i^k est la priorité des trames du flux τ_i^k utilisé dans le réseau priorisé considéré. Nous faisons l'hypothèse que la priorité de chaque flux est unique,
- O_i^k est l'offset, qui est la date de première arrivée d'une trame de τ_i^k sur la station k. Dit de manière différente, O_i^k est la durée entre l'instant où la station a été allumée et l'instant où la première trame de τ_j^k fût mise dans la file d'attente. Donc, le flux τ_i^kmet dans la file d'attente sa $j^{\text{ème}}$ trame$\tau_{i,j}^k$ à l'instant $A_{i,j}^k \stackrel{\text{def}}{=} O_i^k + (j-1) \cdot T_i^k$. Comme sur un réseau, les stations ne sont pas synchronisées globalement, nous devons considérer le cas où les décalages sont fixés localement sur les stations, relativement à un référentiel qui est l'horloge interne de la station. Le début de transmission d'une trame est noté $B_{i,j}^k$.

Lorsque la référence à la station n'a pas d'incidence sur la résolution du problème, celle-ci sera omise dans les notations (cf. chapitre 4). Le modèle est alors identique à celui présenté au §1.2.1.

1.2.3 Génération aléatoire des ensembles de tâches/flux

Dans le courant du document, nous utiliserons des ensembles de tâches/flux afin d'évaluer les performances des solutions proposées. Nous décrivons l'algorithme générique utilisé dans ce document dans l'annexe A.1. Dans chaque chapitre, des précisions seront données lorsque cela sera nécessaire. Il est important de noter qu'il existe d'autres algorithmes de génération de tâches. Par exemple, le lecteur intéréssé pourra se référer au papier de Bini et Buttazzo [7] qui analyse l'impact de la génération de tâches et proposent un algorithme de génération de tâches.

1.3 Problématique traitée

Dans ce travail, nous avons étudié l'ordonnancement temps réel optimisé :

- de tâches indépendantes périodiques s'exécutant sur un processeur ;
- de flux de messages périodiques sur un réseau de terrain avec accès au médium priorisé.

Les seules contraintes de temps considérées sont les contraintes d'échéance. Deux sortes d'optimisation sont traitées :

- l'optimisation de l'utilisation de la plate-forme d'exécution ;

– l'optimisation des critères applicatifs de qualité de service propres à l'application.

Il s'agit donc de spécifier et développer des méthodes, techniques et outils pour construire, valider, configurer et optimiser des politiques d'ordonnancement génériques. Techniquement les problèmes ouverts sont :

1. développer des analyses d'ordonnançabilité « génériques » qui permettront de décider de la faisabilité d'un ensemble d'activités pour n'importe quelle politique candidate. Cela permet de fournir au concepteur de l'application temps réel un support d'analyse permettant de vérifier les caractéristiques temporelles de son application.
2. construire et configurer des politiques d'ordonnancement génériques, *i.e.*, des politiques faciles à mettre en oeuvre ou disponibles sur des exécutifs temps réel standard, dans le but d'optimiser l'usage des ressources.
3. mettre au point les techniques pour évaluer la qualité des politiques candidates relativement aux autres critères que la faisabilité ; ce pourra être réalisé par analyse, par des mesures sur plate-forme, par simulation à événements discrets ou même en temps continu pour des métriques de performances relatives aux régulations industrielles.

Chapitre 2

État de l'art

Ainsi que nous l'avons présenté dans le chapitre précédent, le problème traité dans ce travail est celui de l'ordonnancement temps réel optimisé. Par temps réel, nous entendons respect des contraintes de temps exprimés sur les activités du système et ceci conduit à la notion de faisabilité du système. La notion d'optimisation recouvre deux concepts : l'optimisation de critères applicatifs (ex. : minimisation de la gigue) et la maximisation du nombre de configurations faisables pour une plate-forme donnée. Aussi, dans un premier temps, nous rappelons brièvement les résultats concernant la faisabilité des systèmes temps réel. Ensuite, nous détaillons les principales approches d'ordonnancement temps réel visant à optimiser des critères de qualité de service dépendant de l'application.

2.1 Faisabilité et optimalité

Avant d'introduire les résultats classiques d'ordonnancement, nous rappelons les principaux concepts utilisés en ordonnancement temps réel.

Ensemble de tâches concrètes et non-concrètes (Jeffay et *al.* [8]) Une tâche τ_i est une tâche concrète si l'offset O_i de cette tâche est connu avant l'exécution du système. τ_i est une tâche non-concrète lorsque O_i n'est pas connu avant l'exécution du système. Un ensemble de n tâches non-concrètes est noté $\Omega = \{\tau_1, \tau_2, ..., \tau_n\}$, où $\tau_i = (C_i, T_i, D_i, O_i)$ est la $i^{\text{ème}}$ tâche de Ω et $\forall i, i \in [1, n], O_i$ n'est pas défini. Dans un ensemble ω de n tâches concrètes les offsets O_i sont définis pour tout i. Sans restriction sur les offsets des tâches, il y a une infinité d'ensembles de tâches concrètes pouvant être générés à partir d'un ensemble de tâches non-concrètes Ω.

Faisabilité dans la littérature, un ensemble de tâches (ou un système) est dit faisable s'il existe un ordonnancement qui respecte les contraintes de temps. Un ensemble de tâches est ordonnançable sous une politique donnée si toutes les contraintes de temps sont respectées en appliquant cette politique. Dans la suite, lorsqu'il n'y a pas s'ambiguîté, nous confondons les deux termes. De plus, comme les seules contraintes de temps auxquelles nous nous intéressons sont les contraintes d'échéances, nous parlerons plus généralement de faisabilité pour dire que l'ordonnancement fourni par la politique proposée respecte ces contraintes. Un ensemble de tâches concrètes, *i.e.*, un ensemble où toutes les tâches sont concrètes, est dit faisable si toutes les tâches respectent les contraintes de temps, *i.e.*, si

	Ordonnancement préemptif		Ordonnancement non-préemptif	
Priorité	dynamique	fixe	dynamique	fixe
Optimalité	EDF [9], LLF [11]	RM ($\overline{D_i} = T_i$) [6], DM ($\overline{D_i} \leq T_i$) [12], Audsley ($\forall \overline{D_i}$) [14]	NP-EDF : cas ($\overline{D_i} = T_i$) [13, 8], cas ($\forall \overline{D_i}$) [15]	Audsley ($\forall \overline{D_i}$) [10]

TAB. 2.1 – Principaux résultats d'optimalité dans le cas des systèmes non-concrets (issue de [10]).

aucune tâche ne termine son exécution après son échéance absolue $D_{k,n} = A_{k,n} + \overline{D_k}$. Un ensemble de tâches non-concrètes Ω est faisable, si tous les ensembles concrets possibles, qui peuvent être générés à partir de Ω, sont faisables.

Optimalité Une politique d'ordonnancement est dite optimale en regard d'un certain critère (ex. : faisabilité, minimisation temps de réponse) dans sa classe si aucune politique de sa classe n'a de meilleure performance pour ce critère. Dans la suite, quand rien n'est précisé, optimal signifie optimal en regard de la faisabilité. Une politique est non-concrète optimale (en regard de la faisabilité) si elle ordonnance avec succès tous les ensembles ordonnançables avec les politiques de sa classe. Par exemple, Jeffay et *al.* [8] montre que *NP-EDF* (*Non-Preemptive Earliest Deadline First*) est non-concrète optimale en regard de la faisabilité dans la classe des politiques non-oisives. Nous reviendrons plus précisemment sur cette notion d'optimalité dans chaque chapitre présentant les contributions techniques de ce travail.

Les principaux résultats (ceci n'est pas une liste exhaustive, mais résume juste les principaux résultats) concernant l'optimalité dans le cas de systèmes non-concrets sont listés dans le tableau 2.1 . Dans la classe de politiques préemptives à priorité dynamique, les politiques EDF et Least Laxity First (LLF), plus petite la laxité (*i.e.*, l'échéance moins le travail restant à exécuter), plus grande la priorité, sont optimales, cf. respectivement Dertouzos [9] et Mok [11]. Dans la classe des politiques préemptives à priorité fixe, RM est optimale dans le cas de tâches à échéance sur requête, *i.e.* $\overline{D_i} = T_i$, cf. Liu et Layland [6], Leung et Whitehead [12] ont montré l'optimalité de DM dans le cas d'échéances inférieures ou égales à la période, *i.e.*, $\overline{D_i} \leq T_i$, et l'algorithme d'allocation de priorité d'Audsley, quant à lui, est optimal quelles que soient les échéances, *i.e.*, $\forall \overline{D_i}$ cf. Audsley [14]. Dans la classe des politiques non-préemptives à priorité dynamique, NP-EDF, la version non-préemptive de EDF, est optimale quelles que soient les échéances, *i.e.*, $\forall \overline{D_i}$ (prouvée optimale dans le cas $\overline{D_i} = T_i$ par Kim et Naghibzadeh [13] et Jeffay et *al.*[8], et $\forall \overline{D_i}$ par George et *al.* [15]). L'algorithme d'Audsley [14] est optimal dans la classe des politiques non-préemptives à priorité fixe (preuve donnée par George et *al.* [10] pour la version non-préemptive).

2.2 Optimisation de la QdS

Les différents travaux réalisés pour prendre en compte les critères de performances dépendants de l'application agissent aux niveaux :

1. des tâches en :
   (a) modélisant/ajustant les paramètres des tâches périodiques,

(b) proposant de nouveaux modèles de tâche,

2. de l'algorithme d'ordonnancement en :

   (a) ajustant les paramètres des politiques d'ordonnancement,

   (b) utilisant des algorithmes d'ordonnancement autre que les politiques reposant sur les priorités.

Nous présentons les travaux selon ce classement. Pour chaque méthode, nous précisons si cette méthode est en-ligne ou hors-ligne et quelle technique d'exploration de l'espace de recherche est utilisée, s'il y en a une (ex. : recherche exhaustive, algorithme génétique, etc.).

2.2.1 Modélisation et ajustement des paramètres des tâches périodiques

Les méthodes présentées supposent que l'application est modélisée par un ensemble de tâches/flux périodiques (dans la suite, nous employons le terme tâche pour désigner une tâche ou un flux), cf. §1.2. Deux types de problèmes sont traités :

1. le modèle de tâche périodique est utilisé, le problème est de synthétiser les paramètres des tâches afin de respecter les besoins de l'application temps réel,
2. l'ensemble de tâches modélisant l'application temps réel est connu mais il est possible de modifier certains attributs (ex. : échéance, date d'arrivée) pour optimiser les performances de l'application.

Ces méthodes proposent des solutions pour générer ou ajuster les tâches périodiques alors que nous, nous faisons l'hypothèse qu'un ensemble de tâches périodiques modélisant le système temps réel est fourni. Ces propositions sont donc "en amont" de nos propositions et peuvent être utilisées au préalable pour générer l'ensemble de tâches périodiques utilisé par nos méthodes. Nous donnons ici un aperçu des méthodes proposées dans la littérature.

2.2.1.1 Synthétiser l'ensemble de tâches

Ces contributions font l'hypothèse que les spécifications de l'application sont données sous une autre forme qu'un ensemble de tâches ; le but étant de synthétiser l'ensemble de tâches périodiques afin de respecter ces spécifications.

Gerber et *al.* [16] : les besoins de l'application temps réel sont modélisés par un ensemble : de "ports" d'entrées et de sorties avec les contraintes sur ces ports (e.g, contraintes de bout-en-bout entre un port de sortie et un port d'entrée, contrainte de "séparation" entre deux productions consécutives d'une même sortie), de "tâches fonctionnelles" définies par un pire temps d'exécution et un graphe de tâches modélisant les chemins de communications entre les ports d'entrée, les tâches et les ports de sortie. Le problème traité est la détermination des attributs des tâches périodiques (ex. périodes, échéances) afin de respecter les contraintes. Pour résoudre ce problème, un ensemble de contraintes sur les attributs des tâches (ex. $\overline{D}_i - O_i \leq \text{cst}$) est généré automatiquement à partir des contraintes et du graphe de tâches. Une méthode de résolution (utilisant des solutions inspirées des techniques de programmation linéaire), décomposant le problème en sous-problèmes moins complexes, permet d'affecter les attributs des tâches afin de respecter les contraintes.

Bate et Burns [17] : les auteurs distinguent les besoins de l'application en deux catégories : les contraintes sur les tâches périodiques, les contraintes sur les "transactions" où les transactions sont définies comme une séquence de tâches périodiques s'exécutant en respectant un ordre prédéterminé. Une transaction est activée périodiquement et a des contraintes de bout-en-bout (contrainte de temps entre l'activation et la fin d'exécution de la dernière tâche) et de gigue sur la fin d'exécution (*i.e.*, variabilité entre deux fins d'exécution consécutives). Le problème traité est l'allocation des attributs des tâches périodiques composant la transaction afin de respecter les contraintes (ex. : bout-en-bout, précédences). Les auteurs proposent un algorithme allouant les offsets (satisfaisant aux contraintes de de précédence) et les échéances (satisfaisant aux contraintes de bout-en-bout). Par exemple, l'idée, pour respecter les contraintes de bout-en-bout, est de fixer l'échéance en commençant par la dernière tâche de la transaction (afin de respecter la contrainte de bout-en-bout) puis de travailler "en marche arrière" jusqu'à la première tâche ($\forall i, i = n-1, .., 1$ fixer l'échéance de τ_i pour que τ_{i+1} respecte son échéance).

2.2.1.2 Ajuster les paramètres des tâches

Dans ces travaux, un ensemble de tâches est donné mais il est possible de modifier certain attributs des tâches afin d'optimiser les performances de l'application.

Di Natale et Stankovic [18, 19] : les tâches considérées ont des contraintes de précédence et d'accès à des ressources critiques. Les auteurs proposent d'utiliser une technique d'optimisation appelée recuit simulé pour fixer les offsets des tâches afin de minimiser la gigue sur les débuts d'exécution. La qualité d'une solution est évaluée à l'aide d'une fonction de coût définie par les auteurs.

Coutinho et *al.* [20] : les auteurs proposent d'utiliser un algorithme génétique pour définir les offsets des flux périodiques afin de réduire la gigue sur réseaux CAN.

Baruah et *al.* [21] : les auteurs considèrent un ensemble de tâches périodiques (*i.e.*, tous les attributs sont connus). Les auteurs proposent d'avancer les échéances des tâches dans le but de limiter la gigue sur la fin d'exécution et le délai entre le début et la fin d'exécution de chaque tâche.

David [22] : l'auteur propose de fixer les offsets et de réduire les échéances relatives des tâches afin de réduire la gigue. En introduisant la notion de "fenêtre d'exécution" (intervalle de temps pendant lequel la tâche est susceptible de s'exécuter), l'auteur traite un problème de satisfaction de contraintes afin de "désynchroniser" les "fenêtres d'exécution" des tâches. Un état de l'art très complet sur les méthodes traitant de la minimisation de la gigue peut être trouvé dans cette thèse.

Dans cette thèse, nous avons fait l'hypothèse que le système temps réel et ses contraintes étaient données. Cependant, les méthodes synthétisant un ensemble de tâches (Gerber et *al.* [16] et Bate et Burns [17]) présentées ci-dessus sont en amont de nos propositions car elles modifient ou génèrent un ensemble de tâches périodiques et elles peuvent dans certains cas être utilisées au préalable dans l'étape de spécification détaillée du système (modélisation sous forme d'un ensemble de tâches concrètes).

Nous verrons, cf. section 3, que nous proposons, dans le cadre des systèmes *offset free* (systèmes où la première date d'activation d'une tâche peut être choisie, cf. chapitre 5 pour plus de détails), des heuristiques pour optimiser l'usage de la plate-forme et pour réduire le temps de réponse. Nos propositions s'inscrivent donc dans la même démarche que celles proposées par Di Natale et Stankovic [18, 19],

Coutinho et *al.* [20], Baruah et *al.* [21] et David [22] excepté que l'objectif est différent (minimiser la gigue dans [18, 19, 21], minimiser le temps de réponse dans notre approche, cf. section 3 et chapitre 5). Notons, cependant que l'approche de Baruah et *al.* [21] peut être utilisée conjointement à nos contributions.

2.2.2 Utilisation de nouveaux modèles de tâches

Certains auteurs proposent de nouveaux modèles de tâches permettant de mieux modéliser les systèmes temps réel, leurs contraintes et les algorithmes d'ordonnancement appropriés à ces modèles.

Han et Lin [23] : le système considéré est modélisé par un ensemble de tâches à contraintes de séparation (*Distance Constrained Task System - DCTS*), *i.e.*, la fréquence d'arrivée des tâches n'est pas représentée par une période mais par une contrainte de séparation entre la fin d'exécution de deux instances consécutives de la même tâche. Han et Lin transforment ce problème en un problème "pinwhell" (*i.e.*, étant donné un ensemble $\{a_1, a_2, ..., a_n\}$ trouver une séquence infinie de symboles $\{1, 2, ..., n\}$ telle qu'il y ait au moins un symbole i dans chaque sous-séquence de a_i symboles consécutifs). Ils ont donné des tests d'ordonnançabilité reposant sur la densité (*i.e.,* le taux d'utilisation) pour RM et proposent d'autres ordonnanceurs spécifiques.

Crespo et *al.* [24] : les auteurs utilisent une pratique fréquemment employée (en automatique) pour réduire la gigue en séparant chaque tâche de contrôle en trois sous-tâches différentes pour l'acquisition des données, le calcul de la commande et l'application de la commande. La réduction de la gigue est réalisée en fixant l'exécution des tâches d'entrée et de sortie.

L'utilisation de ces nouveaux modèles offre la possibilité de mieux prendre en compte les besoins de l'application et donc de mieux répondre à ceux-ci. Malheureusement, le modèle de tâches devient spécifique à une classe de problèmes d'ordonnancement. Donc, la solution n'est pas générique et implique souvent des efforts d'implémentation supplémentaires. C'est pourquoi, nous avons considéré dans notre étude le modèle périodique classique.

2.2.3 Politiques d'ordonnancement en-ligne

Les politiques d'ordonnancement à priorités fixes et dynamiques sont très utilisées. Nous montrons des méthodes de la littérature définissant une règle d'allocation de priorité permettant d'optimiser le comportement de la politique en ce qui concerne des critères dépendants de l'application autres que la faisabilité.

Goossens et Richard [25] : les auteurs proposent d'utiliser une technique de recherche par séparation évaluation (*branch and bound*) pour fixer les priorités dans le contexte de l'ordonnancement à priorités fixes afin de minimiser le temps de réponse moyen. La méthode présentée est générique et peut être utilisée pour d'autres problèmes d'optimisation. Cependant, la principale difficulté est de trouver les fonctions de coût permettant d'évaluer une solution.

Navet et Migge [26] : un algorithme génétique est proposé pour allouer les politiques et les priorités dans les systèmes conformes au standard Posix 1003.1b afin de minimiser les critères dépendants de l'application tels que la gigue.

Schrage et Smith [27, 28] : (cité par Bunde [29]) la politique préemptive à priorités fixes allouant la plus forte priorité à l'activité ayant la quantité de travail la plus faible (*Shortest Remaining Processing Time First - SRPTF*) est prouvé optimale en ce qui concerne les temps de réponse moyens (*i.e.*, pour minimiser la moyenne des temps de réponse).

La méthode proposée par Goossens et Richard [25] permet d'effectuer la meilleure attribution de priorités (fixes) possible pour optimiser un critère donné. Néanmoins, cette technique est limitée au cas des priorités fixes. Nous verrons que nos contributions exploitent la même idée que celle utilisée par Schrage et Smith [27, 28] : allouer les priorités dynamiques aux instances des tâches en fonction des caractéristiques de la tâche. Dans le cadre des systèmes conformes Posix 1003.1b considérés dans Navet et Migge [26], nous proposons des politiques de configuration d'ordonnancement optimales en regard de la faisabilité.

2.2.4 Construction d'algorithmes d'ordonnancement

Nous nous intéressons aux techniques utilisant des algorithmes d'ordonnancement (*i.e.*, ordonnanceurs) autres que les ordonnanceurs classiques reposant sur les priorités. Nous présentons tout d'abord les méthodes hors-ligne qui, à partir de la description du système temps réel construisent un ordonnanceur. Ensuite, nous montrons quelques méthodes en-ligne utilisant des mécanismes d'ordonnancement spécialisés.

2.2.4.1 Méthodes hors-ligne

Les premières approches (Mok [4], Xu et Parnas [4]) partent de tâches et déterminent un ordonnancement statique tandis que les suivantes prennent un modèle d'application (ex. : sous forme d'automate) et génére un ordonnanceur (ex. : sous forme d'automate).

Mok [11], Xu et Parnas [4] : ces approches proposent des méthodes hors-ligne de construction de "tables" d'ordonnancement (les techniques possibles sont traitées en détails par Mok [11]). La faisabilité et le respect des contraintes sont assurés par construction, *i.e.*, on regarde dans la table les décisions à prendre pendant l'exécution du système.

Altisen et *al.* [30] : cette méthode est appelée synthèse d'ordonnanceur. En synthèse d'ordonnancement, l'application est modélisée à l'aide d'automates temporisés. Chaque place représente un état du système (par exemple, la tâche 1 a la ressource et les autres sont en attente). On peut passer d'un état à un autre grâce aux transitions. Les transitions représentent les actions : elles peuvent être contrôlables (le début d'une tâche peut être différé par exemple) et non-contrôlables (la fin d'exécution d'une tâche). Les contraintes du système sont représentées par des invariants : deux tâches ne peuvent utiliser une ressource critique en même temps, une tâche doit finir de s'exécuter avant son échéance. La synthèse d'ordonnanceur consiste à créer un ordonnanceur à partir de ces invariants : l'ordonnanceur déclenchera les actions contrôlables en fonction de l'état du système pour garantir le respect des invariants.

Grolleau et *al.* [31] : un séquenceur est réalisé hors-ligne à l'aide de réseaux de Pétri. Le principe est de construire une séquence d'ordonnancement qui est amenée à se répéter. Dans cette étude la description de l'application est réalisée à l'aide de réseau de Pétri. On parcourt ensuite toutes les

séquences d'ordonnancement (une séquence est représentée par la séquence des marquages du réseau de Pétri) et on sélectionne une séquence qui garantit la faisabilité.

Ces méthodes sont capables de prendre en compte des applications temps réel avec des contraintes complexes. Une "simple" analyse dans l'ordonnanceur (ex., table, automate) permet de prendre la décision d'ordonnancement. Cependant, la connaissance a priori de toutes les activités du système est nécessaire pour créer l'ordonnanceur. Si l'application évolue (ex. : insertion de nouvelles tâches), l'ordonnanceur n'est plus valable. On ne peut donc utiliser ces méthodes pour une application qui peut évoluer avec le temps comme c'est très souvent le cas dans l'industrie. Un autre problème, souligné par les auteurs eux-même dans [30], est la complexité. C'est pourquoi, nous n'avons pas envisagé d'approche hors-ligne dans cette thèse. Fondamentalement, notre solution est un algorithme d'ordonnancement, indépendant de l'application, alors que ces approches donnent des solutions spécifiques pour une application particulière.

2.2.4.2 Méthodes en-lignes

Les méthodes en-ligne présentées dans la suite utilisent des mécanismes d'ordonnancement spécialisés.

Cervin et Eker [32] : une version modifiée du *Constant Bandwith Server* (*CBS*), proposée initialement par Abeni et Buttazzo [33], est utilisée pour éliminer la gigue dans les applications de contrôle-commande. L'acquisition et la production des données sont réalisées à des instants fixés prédéfinis (implémentées à l'aide d'interruptions) et les tâches de contrôle s'exécutent dans un CBS, qui est une abstraction d'un CPU dédié offrant une fraction choisie du CPU original, afin de garantir que la tâche de contrôle finisse avant la production du signal de sortie. Un CBS crée l'abstraction d'un CPU virtuel et apparaît "de l'extérieur" comme une tâche EDF ordinaire.

Cena et Valenzano[34], Nölte et *al.* [35] : l'utilisation de "serveurs d'ordonnancement" sur un réseau priorisé est proposée pour garantir l'équité et l'isolation de bande passante. Un serveur maître conserve ou estime l'état des serveurs "esclaves" distribués sur les noeuds du réseau et alloue la bande passante en fonction de ces informations.

Ces méthodes sont efficaces et permettent d'atteindre les objectifs désirés. Cependant, la mise en place de ces techniques impliquent un effort de développement, car les mécanismes proposés ne sont pas disponibles sur les composants standards du commerce. Nous proposons dans cette thèse des méthodes reposant sur les standards (ex. : standard Posix 1003.1b [36] définissant des extensions temps réel au standard Posix) afin de permettre une implémentation peu coûteuse et possible sur des composants du commerce.

2.3 Conclusion

Dans nos travaux, nous considérons des activités périodiques et indépendantes. En effet, nous voulons utiliser un modèle générique qui n'est pas spécifique à l'application contrairement aux modèles introduits dans [23, 24] (cf. §2.2.2).

Nous proposons deux techniques de configuration de l'ordonnancement. La première est une méthode d'allocation des offsets comme dans [18, 19, 20] et la seconde une d'allocation de priorités comme dans [25, 26] (nous verrons que l'affectation ne sera pas limitée aux priorités). La différence avec ces approches est que le système considéré n'est pas le même ou l'objectif est différent. De plus, nous exhibons une

nouvelle classe de politiques d'ordonnancement en-lignes dynamiques. L'idée, comme pour les politiques RM et EDF, est d'attribuer les priorités aux instances des tâches en fonctions des caractéristiques de ces instances. La principale contribution est que les politiques de cette classe permettent de faire un compromis entre le respect des échéances et l'amélioration de critères de performance. De plus, ces politiques sont faciles à mettre en oeuvre et nous donnons une analyse d'ordonnancement générique pour cette classe.

Chapitre 3

Présentation des contributions

Nos contributions sur l'ordonnancement temps réel optimisé se sont déclinées sur trois problèmes particuliers, ce qui sructure le plan de la suite de ce document (chapitres 4, 5 et 6 de la partie II "Contributions"). Le chapitre 4 propose une nouvelle classe de politiques à priorité dynamique pouvant être implémentée sur un bus priorisé CAN (*Controller Area Netwok*) optimisant des critères applicatifs de qualité de service propres à l'application. Les chapitres 5 et 6 présentent des méthodes de configuration de l'ordonnancement respectivement dans le cadre des systèmes "offset free", *i.e.*, des systèmes où les décalages initiaux des activités peuvent être choisis par l'utilisateur, et dans le cadre des systèmes conformes au standard Posix 1003.1b afin d'optimiser l'utilisation de la plate-forme d'exécution. Nous résumons ci-dessous les contributions figurant dans chacun de ces chapitres.

3.1 Optimisation d'une messagerie sur bus priorisé (chapitre 4)

La qualité, pour chaque type d'application, peut être evaluée et mesurée solon des métriques de performances particulières. Ainsi, dans le contexte spécifique des applications de contrôle-commande, des métriques importantes concernant l'ordonnancement sont (voir [37], par exemple) sont :

- le délai entre l'échantillonnage (la mesure) et l'application de la consigne (action sur le système de contrôle) et la variabilité au cours du temps de ce délai,
- la période d'échantillonnage et la variabilité entre deux instants d'échantillonnage successifs.

Dans le chapitre 4, nous identifions une classe de politiques d'ordonnancement, appelée classe Dépendantes des Dates d'Arrivée (DDA), qui est importante en pratique, car les politiques qui la composent sont facilement implémentables et performantes du point de vue respect des échéances temporelles (les politiques EDF Earliest deadline First, FIFO First-In First-Out appartiennent à cette classe). Précisément, la priorité d'une instance $\tau_{i,j}$ dans la classe DDA est inversement proportionnelle à la date d'arrivée de l'instance $A_{i,j}$ et d'un facteur constant f_k propre à toutes les instances d'une tâche, *i.e.*, plus petit $A_{k,n} + f_k$, plus grande la priorité. Nous proposons une version préemptive (cas mono-processeur) et une version non-préemptive (cas distribué sur un réseau priorisé) des politiques DDA.

Le premier résultat significatif de cette thèse est une analyse d'ordonnançabilité générique pour toutes les politiques de la classe considérée. Pratiquement, cela permet d'envisager chacune des politiques en étant toujours capable de déterminer si les échéances du système vont être respectées. Pour évaluer

la satisfaction des autres critères que la faisabilité, un simulateur d'ordonnancement a été développé. Nous montrons qu'utiliser une politique DDA permet par exemple de réduire de 10.2% la moyenne de la gigue sur le temps de réponse comparée à NP-EDF. L'étude de l'influence de l'ordonnancement sur le contrôle d'un système, obtenue par simulation du système de contrôle et de l'ordonnancement à l'aide du logiciel TrueTime, a été réalisée dans le cadre distribué. Cette étude montre une très nette amélioraton des performances du système de contrôle considéré (moteur électrique contrôlé par un contrôleur PD, *i.e.*, Proportionnel Dérivé, réparti sur un réseau CAN), par exemple, sur un réseau chargé à 50% le dépassement est réduit de 30.1% comparé à NP-EDF.

Ces résultats ont été présenté dans un article accepté à la conférence ETFA 2005 [38], pour le cas mono-processeur préemptif. Un autre article [39], portant sur la "version" non-préemptive des politiques Dépendantes des Dates d'Arrivée, est en cours de révision pour le journal IEEE Transactions on Industrial Informatics. Dans cette thèse, nous présentons les résultats portant sur la partie non-préemptive.

3.2 Nouvelles politiques à priorité fixe dans les systèmes "offset free" (chapitre 5)

Dans ce chapitre, nous étudions le problème de l'ordonnancement à priorité fixe de tâche à contraintes temps réel dures. Nous considérons des systèmes "offset free" (systèmes modélisés par un ensemble de tâches/flux périodiques indépendantes ayant un offset, *i.e.*, l'instant de la première requête, pouvant être choisi par l'utilisateur). Ces travaux proposent une technique d'allocation d'offsets et de priorités. Le cas où toutes les tâches/flux sont activées de manière synchrone (*i.e.*, toutes les tâches/flux sont activées au démarrage) est la pire situation en terme de faisabilité dans le cas préemptif et est une très mauvaise situation dans le cas non-préemptif. Ne pas considérer d'autres allocations d'offsets s'avère pessimiste.

Nous étudions le cas préemptif (la ressource est un processeur) et distribué non-préemptif (la ressource partagée est un réseau de communication constitué d'un bus priorisé CAN). Dans le cas préemptif, nous proposons une nouvelle technique, reposant sur l'algorithme d'Audsley, permettant de réduire significativement l'espace de recherche du problème d'allocation des offsets : nos expérimentations ont montré que le nombre d'allocations d'offsets différentes à considérer pouvait être réduit jusqu'à 50% avec une utilisation appropriée de l'algorithme d'Audsley.

Ensuite, nous proposons des algorithmes d'allocation d'offset dans le cas préemptif et distribué non-préemptif pour choisir les offsets afin de réduire les temps de réponse. Ces heuristiques optimisent l'utilisation de la plate-forme d'exécution au sens où elles fournissent des situations asynchrones alternatives, qui permettent d'améliorer de manière significative le nombre de systèmes ordonnançables en regard du cas pessimiste synchrone. En effet, l'usage combiné des heuristiques permet d'ordonnancer au moins 40.5% et jusqu'à 97% des ensembles de tâches selon nos expérimentations, réalisées avec des ensembles de 5 à 11 tâches sous une charge moyenne de 0.8 . Pour le cas des systèmes offset free non-préemptif distribués sur un réseau priorisé (le réseau CAN dans notre étude), aucune analyse d'ordonnancement existe à notre connaissance. Nous avons donc fourni une analyse d'ordonnancement pour ces sytèmes. De plus, nos expérimentations montrent que l'usage de nos heuristiques permet de réduire en moyenne d'un facteur 2.5 les temps de réponse comparé au cas synchrone ; ces expérimentations ont été appliquées à des messageries automobiles sur un réseau CAN (*i.e.*, cas distribué non-préemptif).

Dans le cadre des systèmes "offset free" préemptifs, les résultats obtenus ont été publiés à la conférence RTNS'06, cf. [40] et ce papier a obtenu le Best Paper Award (sélectionné pour publication dans un numéro spécial du "Journal of Embedded Computing"). Les autres travaux ont été rédigés dans un délivrable dans le cadre d'un projet industriel avec PSA, mais le contrat que nous avons signé avec PSA ne nous autorise pas à diffuser ces travaux.

3.3 Configuration d'ordonnancement sous Posix 1003.1b (chapitre 6)

Les systèmes conformes au standard Posix 1003.1b offrent deux politiques d'ordonnancement bien spécifiées, à savoir *sched_rr* (*i.e., Round-Robin* - RR où chaque tâche est autorisée à s'exécuter pendant un quantum, *i.e.* un intervalle de temps, avant d'être préemptée) et *sched_fifo* (*i.e., First-In First-out* - FIFO). Nous étudions dans ce chapitre l'utilisation combinée de ces deux politiques dans l'optique de maximiser la faisabilité et proposons des algorithmes d'allocation de priorités, de politiques d'ordonnancement et de quantum pour les systèmes Posix 1003.1b (cf. point 2 §1.3). Les algorithmes d'allocation sont montrés optimaux vis-à-vis du pouvoir du test de faisabilité, c'est-à-dire la capacité du test à distinguer les configurations faisables des configurations non-faisables. Bien que les algorithmes proposés réduisent la complexité du problème, celle-ci reste exponentielle en le nombre de tâches ce qui limite l'utilisation de ces algorithmes à des problèmes de petites et moyennes tailles. C'est pourquoi, nous proposons une heuristique peu complexe capable de traiter des problèmes de grandes tailles.

Par simulation, nous constatons que nos propositions permettent d'obtenir un gain significatif en faisabilité en comparaison à FPP (*Fixed Preemptive Priority* - politiques préemptive à priorités fixes). Dans de nombreuses configurations non-faisables avec FPP, l'utilisation combinée de FPP et RR (*Round-Robin*) permet de conduire à un ordonnancement faisable et ainsi de fournir une alternative à EDF, politique qui, le plus souvent, n'est pas disponible dans les systèmes d'exploitation. Par exemple, selon nos expérimentations, l'utilisation combinée de FPP et RR permet d'ordonnancer presque 100% des ensembles, ayant une charge de 0.75, non-faisables avec FPP.

Deux algorithmes ont été proposés. Le premier, appelé *Audsley-RR-FPP*, traite du cas où le quantum est une valeur globale constante au système. Le papier présentant cet algorithme est en cours de révision pour le journal IEEE Transactions on Industrial Informatics. L'algorithme suivant, appelé *Audsley-RR-FPP**, est une extension de *Audsley-RR-FPP* prenant en compte le cas où les valeurs des quanta sont spécifiques à chaque tâche. Les résultats obtenus ont été publiés à la conférence RTNS'07 [41]. La synthèse de ces deux papiers et notre heuristique capable de traiter des problèmes de grandes tailles ont été décrites dans un rapport de recherche INRIA [42].

Deuxième partie

Contributions

Chapitre 4

Optimisation d'une messagerie sur bus priorisé

4.1 Introduction

Contexte. Dans les systèmes temps réel distribués, le respect des contraintes temporelles est obligatoire (*i.e.*, faisabilité), mais des critères dépendants de l'application autre que la faisabilité sont importants du point de vue des performances de l'application. Par exemple, dans les systèmes de contrôle-commande distribués sur un réseau (NCS - *Network Controlled Systems*), les échanges de messages induisent des délais, qui peuvent être non constants, dans la boucle de contrôle et qui affectent les performances (ex : dépassement, IAE - *Integral of the Absolute magnitude Error*) et même la stabilité des systèmes contrôlés [43, 37, 44]. La gestion de l'accés au médium de communication a un fort impact sur ces délais et donc la configuration des trames circulant sur le réseau doit être choisie afin de minimiser ce délai et/ou le rendre constant.

Dans cette étude, nous considérons l'ordonnancement d'un ensemble de flux périodiques sur le bus priorisé CAN (*Controller Area Network*). Nous rappelons qu'un flux périodique génére périodiquement une trame. Le protocol MAC (*Medium Access Control*) de CAN est tel que n'importe quel noeud peut commencer à émettre quand le bus est au repos. Les conflits possibles sont résolus à l'aide d'un mécanisme d'arbitrage reposant sur les identifiants. Chaque trame a un identifiant unique, servant de priorité, et en cas de transmissions simultanées, la trame ayant la plus grande priorité est émise en dépit de la contention avec les trames de plus faibles priorités. L'arbitrage est déterminé par l'identifiant des trames, *i.e.*, la priorité, avec la convention "plus petite la valeur numérique, plus grande la priorité". Il n'est pas obligatoire que les trames successives d'un même flux aient le même identifiant. Cela permet l'implémentation de politiques à priorités dynamiques où la priorité de chaque trame est codée en-ligne dans l'identifiant de celle-ci.

Dans la suite, nous étudions la politique, *i.e.*, l'algorithme, utilisée pour choisir l'identifiant des trames au niveau MAC et considérons que l'arbitrage natif du réseau CAN basé sur les identifiants, *i.e.*, priorités, est inchangé.

Définition du problème. Le problème traité est la conception d'un algorithme d'ordonnancement, *i.e.*, d'allocation de priorités, pour une messagerie (c'est-à-dire l'ensemble des trames qui sont émises sur le réseau) sur un réseau CAN qui :

- respecte la contrainte de faisabilité, *i.e.*, respecte l'échéance sur le temps de réponse exprimée sur chaque trame ; le temps de réponse est la durée écoulée entre la demande d'émission de la trame et la réception de cette trame par le contrôleur de communication de la station réceptrice ; notons que plusieurs stations peuvent être destinataires de la même trame. Dans ce travail, le temps de propagation entre ces stations est intégré au pire temps de transmission (C_i),
- optimise des critères dépendants de l'application (ex : minimisation temps de réponse, minimisation de la gigue sur le temps de réponse).

De plus, ces algorithmes devront s'implémenter autant que possible sur du matériel standard du commerce en induisant une faible surcharge. Dans la suite, nous considérons que l'algorithme d'ordonnancement est implémenté au-dessus d'un bus priorisé CAN [45, 46]. Les bus priorisés ont prouvé qu'ils étaient performants dans un contexte temps réel, et le bus CAN est très couramment utilisé comme réseau embarqué dans les automobiles [47]. Les mécanismes présentés dans la suite peuvent être facilement adaptés à tous les autres bus priorisés tel que le VAN [48] et le J1850 [49].

Notre approche Dans cette étude, nous allons :

1. définir les propriétés qu'une politique d'ordonnancement de messageries "acceptable" pour le temps réel doit posséder (dans notre cas, il s'agit de définir les propriétés de l'allocation de priorités),
2. identifier une sous-classe de politiques dynamiques prenant en compte les attributs des flux (ex : arrivées, échéances, périodes) prometteuse en termes de performances (*i.e.,* qui permettra de respecter la faisabilité et d'optimiser l'ordonnancement selon des critères propres à l'application),
3. réaliser une analyse d'ordonnancement générique pour cette sous-classe de politiques,
4. évaluer l'implémentabilité.

Nous réaliserons ensuite des expérimentations dans le cadre des systèmes controlés en réseaux (NCS - *Network Controlled Systems*) afin de montrer l'efficacité de la classe de politiques proposée. Deux cas seront distingués. Le cas où la politique doit être efficace en moyenne (elle peut être employée avec diffèrent types de trafics dont les caractéristiques ne sont pas *a priori* connues) et le cas où la politique peut être optimisée pour une application particulière.

Organisation du chapitre. Dans la section 4.2, nous rappelons les méthodes proposant d'implémenter NP-EDF sur un réseau priorisé (ex. : CAN) car NP-EDF appartient à la classe de politiques dynamiques que nous proposons d'étudier. Ensuite, la section 4.3 présente le modèle du système considéré dans cette étude. Par la suite, nous exhibons dans la section 4.4 les propriétés requises par une politique d'ordonnancement "acceptable" dans un contexte temps réel et définissons parmi ces politiques "acceptables" la sous-classe considérée dans la suite. La section 4.5 donne l'analyse d'ordonnancement. Finalement, les expérimentations montrant l'efficacité de notre proposition sont présentées dans la section 4.6.

4.2 Etat de l'art

De nombreuses études ont été réalisées pour trouver des solutions d'ordonnancement bien adaptées aux besoins de l'application, pour l'ordonnancement de messages et de tâches. Ces méthodes ont été présentées dans l'état de l'art de l'introduction générale (cf. chapitre 2). Nous complétons ici cet état de l'art général par une étude des techniques spécifiques au problème traité. Notons que dans le domaine de l'automobile, les configurations de messageries appliquent généralement des politiques d'allocation de priorité statique de type NP-DM ou NP-RM.

Des travaux proposent néanmoins d'implémenter NP-EDF sur un réseau priorisé (ex. : CAN) afin d'atteindre un taux d'utilisation du réseau élevé tout en garantissant le respect des contraintes de temps. Pedreiras et Almeida [50] implémentent NP-EDF au-dessus du protocole FTT-CAN (*Flexible Time-Trigerred communication on CAN*). Avec FTT-CAN, des cycles élémentaires sont utilisés pour gérer l'accès au bus : tous les noeuds sont synchronisés au début du cycle élémentaire par la réception d'un message particulier, qui est envoyé par un noeud spécial appelé maître. De plus, le maître indique les messages devant être envoyés pendant chaque cycle élémentaire dans le message envoyé aux autres noeuds pour marquer le début d'un cycle. Si l'on considère le cycle élémentaire comme unité de temps, une politique NP-EDF peut être utilisée par le maître pour décider les messages à envoyer dans chaque cycle élémentaire. Notons qu'au sein de chaque cycle élémentaire, la politique d'accés au médium native du réseau CAN est utilisée. Dans notre étude, nous n'avons pas utilisé cette méthode, elle repose sur le protocole FTT-CAN et en conséquence a une "granularité" de l'ordonnancement NP-EDF d'un cycle élémentaire. Nous verrons que la méthode que nous utilisons a une granularité au niveau de la transmission d'un bit.

Zuberi et Shin [51] proposent un compromis entre NP-DM et NP-EDF. L'échéance d'une trame est codée dans une partie de l'identifiant (l'autre est réservée pour garantir l'unicité des identifiants imposée par CAN). Le temps est divisé en "époques" et un processus périodique met à jour sur chaque station les échéances des trames relativement au début de cette époque. Les auteurs avancent qu'il n'est pas possible de mettre à jour les priorités à chaque début de transmission de trames et leur solution permet de préserver la capacité du processeur utilisé pour l'implémentation d'une politique NP-EDF. Cependant, Meschi et *al.* [52] montrent qu'il est possible de le faire. Nous détaillons leurs travaux dans la suite de cet état de l'art.

Meschi et *al.* [52] proposent une implémentation de NP-EDF sur CAN, qu'ils ont améliorée dans [53, 54] ; nous adapterons cette méthode aux politiques considérées dans notre étude. L'idée de base, déjà suggérée par Zuberi et Shin [51], est d'encoder l'échéance de la trame en utilisant une grande partie de l'identifiant. Cette méthode permet de transformer les échéances de trames en valeurs données sur un nombre limité de bits. Comme il est imposé que toutes les trames aient un identifiant distinct sur CAN, quelques bits sont réservés pour assurer l'unicité des identifiants. Une solution est d'allouer une valeur distincte à ces bits pour chaque flux. Cependant, ordonnancer les trames avec NP-EDF sur CAN pose certains problèmes. Ces problèmes, explicitement énoncés dans [51], sont :

1. les échéances absolues (comptées à partir de l'intialisation du système) croissent au fur et à mesure de l'exécution du sytème,
2. le codage des échéances avec un nombre limité de bits induit des "erreurs de codage" (cf. annexe B.2) :

deux échéances distinctes peuvent avoir le même identifiant.

Sans perdre de généricité, nous pouvons considèrer le temps comme discret où la granularité est le "temps-bit" (*i.e*, le temps entre l'émission de deux bits successifs de la même trame). Pour résoudre le problème 1, les auteurs dans [51, 52, 53, 54] proposent d'exprimer l'échéance relativement à une origine de temps qui est mise à jour (*i.e.*, augmentée) au cours du temps. Précisément, l'origine des temps t_{start} est mise à jour à chaque début de transmission (*i.e.*, $t_{start} = \max_{i,j}(B_{i,j} \mid B_{i,j} \leq t)$, où $B_{i,j}$ est le début de transmission de la trame $\tau_{i,j}$) : il est égal à l'instant du tout premier bit de la trame émise (bit *Start of Frame*). L'échéance de chaque trame en attente doit être mis à jour avant chaque nouveau début de transmission entre l'émission de deux trames sur le réseau. Ce calcul induit peu de surcharge. Par exemple, elle a été estimée [54] à moins de 3% pour le microcontrôleur Siemens C167CR.

Meschi et *al.* [52] ont développé une solution efficace au problème 2. Les auteurs utilisent une échelle logarithmique pour coder les échéances. La ligne du temps est divisée en blocs dont la taille est augmentée d'une manière exponentielle (cf. figure B.1 dans l'annexe B.1). Tous les blocs sont divisés en un nombre identique de slots. L'identifiant de la trame est fixé à l'indice du slot dans lequel son échéance tombe. Meschi et *al.* [52] ont formellement prouvé que cette technique améliore significativement la faisabilité du système en limitant les erreurs de codage. Cependant celles-ci ne peuvent pas être complètement évitées et doivent être prises en compte. Ainsi qu'énoncé précédemment, nous appliquerons cette technique dans la suite de ce chapitre.

Nolte et *al.* [55] restreignent les identifiants donc les priorités qui peuvent être allouées aux flux sur le réseau CAN afin de réduire la gigue sur le temps de réponse due au bit-stuffing. Il serait possible d'intégrer cette solution sous certaines conditions sur l'espace restreint des identifiants.

4.3 Modélisation du système considéré

Ce chapitre traite de l'ordonnancement des trames sur le réseau priorisé CAN au niveau du protocole MAC ; en particulier la segmentation est réalisée dans les couches supérieures de la pile de communication et elle n'est pas considérée ici.

4.3.1 Modèle du trafic

Nous considérons un ensemble de n flux périodiques $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$. Chaque flux est envoyé par un et un seul noeud. Il peut y avoir plusieurs flux envoyés par le même noeud. Par analogie avec le modèle des tâches périodiques utilisé pour l'ordonnancement périodique, chaque flux périodique τ_i est modélisé par un tuple $(C_i, T_i, \overline{D}_i)$ où chaque trame de τ_i a un pire temps de transmission C_i, une échéance relative $\overline{D}_i$ et où τ_i génére une trame toutes les T_i unités de temps, T_i est donc la période du flux. Nous notons $\tau_{i,j}$ la $j^{\text{ème}}$ activation de τ_i. La date d'activation de $\tau_{i,j}$ est notée $A_{i,j}$. Nous considérons des ensembles de flux non-concrets (*i.e.*, où tous les offsets sont possibles).

4.3.2 Définition de l'allocation des priorités avec les fonctions de priorités

L'accès au médium repose sur les priorités, *i.e.*, la trame en attente ayant la plus grande priorité gagne l'accés au médium, et nous appelons l'algorithme d'allocation de priorités, *politique d'ordonnancement*

dans la suite. Nous utilisons les fonctions de priorité, introduite pour la première fois par Migge [56], pour décrire formellement d'une façon non-ambiguë les politiques d'ordonnancement. Une fonction de priorité $\Gamma_{i,j}(t)$ indique la priorité de la trame $\tau_{i,j}$ à l'instant t. Le réseau est alloué, à chaque instant, selon le paradigme "plus grande priorité d'abord" (*HPF - Highest Priority First*).

La fonction $\Gamma_{i,j}(t)$ prend ses valeurs dans un ensemble totalement ordonné $\mathcal{P} = \{(p_1, ..., p_h) \in \mathbb{R}^h \mid h \in \mathbb{N}^+\}$ où h est un entier naturel positif ; $\mathcal{P}$ est en fait l'ensemble de vecteurs à valeurs réelles muni de l'ordre lexicographique. Entre deux vecteurs, les coordonnées sont comparées une à une en commençant par la gauche, la première coordonnée différente décide de l'ordre de priorité avec la convention "plus petite la valeur numérique, plus grande la priorité". Par exemple, $\Gamma_{i,j}(t) = (3, 4, 5)$ et $\Gamma_{k,n}(t) = (3, 4, 6)$ implique que $\tau_{i,j}$ a une priorité plus importante que $\tau_{k,n}$ à l'instant t. Les vecteurs de priorité de différentes tailles peuvent être comparés avec la même règle que ci-dessus et avec la convention qu'une coordonnée manquante est remplacée par la plus petite valeur numérique possible (ex : $\Gamma_{i,j}(t) = (3, 4, 5)$ et $\Gamma_{k,n}(t) = (3, 4)$ implique que $\tau_{k,n}$ a une priorité plus importante que $\tau_{i,j}$ à l'instant t). Finalement, deux vecteurs sont égaux si et seulement si ils ont la même taille et leurs coordonnées sont égales une à une.

La plupart des politiques d'ordonnancement peuvent être définies par une fonction de priorité. En particulier dans notre problème, nous montrons ci-dessous comment exhiber une politique d'ordonnancement qui définisse une politique d'allocation de priorités des trames. Comme il a été expliqué dans la section 4.2, sur un bus priorisé tel que CAN, le vecteur de priorité peut être codé dans l'identifiant des trames en utilisant la méthode proposée dans [52, 53, 54]. Cela permet d'utiliser le mécanisme d'arbitrage "natif" du bus priorisé pour réaliser la politique d'ordonnancement définie par la fonction $\Gamma_{i,j}(t)$. Par exemple, la priorité de la trame $\tau_{i,j}$ sous la politique Deadline Monotonic non-préemptive (NP-DM - plus petite l'échéance relative, plus grande la priorité) est :

$$\Gamma_{i,j}^{NP-DM}(t) = \begin{cases} (\overline{D}_i, i, j) \text{ si } t < B_{i,j} \\ (-\infty, i, j) \text{ si } t \geq B_{i,j} \end{cases},$$

où $B_{i,j}$ est le début de transmission de $\tau_{i,j}$ (les deux dernières coordonnées i, j sont utilisées pour garantir le déterminisme , cf. définition 4). La valeur $-\infty$ de la première coordonnée du vecteur de priorité après le début d'exécution (*i.e.,* $t \geq B_{i,j}$) garantit la non-préemption. De même, la priorité de $\tau_{i,j}$ sous NP-EDF est :

$$\Gamma_{i,j}^{NP-EDF}(t) = \begin{cases} (A_{i,j} + \overline{D}_i, i, j) \text{ si } t < B_{i,j} \\ (-\infty, i, j) \text{ si } t \geq B_{i,j} \end{cases}.$$

Au-delà de fournir une définition non-ambiguë de la politique d'ordonnancement, les fonctions de priorité permettent de distinguer des classes de politiques d'ordonnancement et d'exhiber des résultats génériques valides pour toutes les politiques de certaines classes. La section suivante présente une sous-classe de politiques d'ordonnancement non-préemptive qui sera étudiées dans la suite de ce chapitre.

4.4 Domaine d'étude

Dans le contexte étudié, nous définissons les propriétés qu'une fonction de priorité doit avoir afin de définir une politique d'ordonnancement "acceptable" pour le temps réel, c'est-à-dire une politique assurant la prévisibilité du système (ex. : pour le calcul du pire temps de réponse) et pouvant être implémentée.

Ensuite, parmi l'ensemble de toutes les politiques d'ordonnancement "acceptables", nous proposons d'en étudier une sous-classe particulière qui nous semble intéressante en pratique car elle permet de prendre en compte tous les attributs des trames et permettra, cf. section 4.6, d'optimiser certains critères propres à l'application (ex. : gigue sur les temps de réponse des trames d'un flux).

4.4.1 Politiques non-préemptives

Dans notre étude, nous considérons l'ordonnancement non-préemptif de trames sur un réseau priorisé CAN. Les politiques d'ordonnancement non-préemptives (ex. : NP-EDF, NP-DM) appartiennent à la classe des politiques à *Promotion de Priorité au Début d'Exécution* (PPDE) introduite par Migge [56].

Définition 3 *[56] Une politique d'ordonnancement est à Promotion de Priorité au Début d'Exécution (PPDE) si la priorité de la trame peut changer uniquement en début de transmission $B_{i,j}$ de la trame :*

$$\Gamma_{i,j}^{PPEB}(t) = \begin{cases} \overrightarrow{P}_{i,j} \text{ si } t < B_{i,j} \\ \overrightarrow{Q}_{i,j} \text{ si } t \geq B_{i,j} \end{cases},$$

où le vecteur de priorité de la trame $\tau_{i,j}$ est $\overrightarrow{P}_{i,j}$ (resp. $\overrightarrow{Q}_{i,j}$) avant (resp. après) son début d'exécution.

Une politique est non-préemptive quand le vecteur de priorité $\overrightarrow{Q}_{i,j}$, après le début d'exécution (*i.e.*, $t \geq B_{i,j}$), est :

$$\overrightarrow{Q}_{i,j} = (-\infty, i, j)\,.$$

En effet, la valeur $-\infty$ de la première coordonnée de ce vecteur de priorité garantit la non-préemption.

4.4.2 Politiques d'ordonnancement "acceptables" pour le temps réel

Déterminisme. Les politiques doivent être déterministes : à chaque instant t, il y a exactement une trame de priorité maximale parmi l'ensemble des trames actives (*i.e.*, trames qui sont en concurrence pour l'accès au canal de transmission). Ce concept permet en outre de vérifier la faisabilité du système.

Définition 4 *[56] Une fonction de priorité est déterministe ssi, à chaque instant t où il y a des trames en attente de transmission, il y a exactement une trame ayant la plus grande priorité.*

Par exemple, les deux dernières coordonnées du vecteur $\overrightarrow{P}_{i,j}^{NP-EDF} = (A_{i,j} + \overline{D}_i, i, j)$ garantissent le déterminisme dans le cas d'échéances identiques. En effet, considérons deux trames $\tau_{1,1}$ et $\tau_{1,2}$ arrivées respectivement en $A_{1,1} = 0$ et $A_{2.1} = 5$ ayant pour échéances relatives $\overline{D}_1 = 10$ et $\overline{D}_2 = 5$. Sous la politiques NP-EDF, les vecteurs de priorités $\overrightarrow{P}_{i,j}^{NP-EDF}$ sont donc $\overrightarrow{P}_{1,1}^{NP-EDF} = (10, 1, 1)$ et $\overrightarrow{P}_{2,1}^{NP-EDF} = (10, 2, 1)$ et nous avons $(10, 1, 1) \succ (10, 2, 1)$ grâce à la coordonnée i de $\overrightarrow{P}_{i,j}^{NP-EDF}$.

Invariance temporelle. Dans cette étude, pour assurer la prévisibilité du système, les seules politiques prises en compte sont celles dont l'ordonnancement du système est le même si on "décale" l'arrivée de toutes les trames vers la gauche ou vers la droite. Ces politiques sont dites *indépendantes aux décalages temporels* (IDT). NP-EDF est une politique IDT car la priorité entre deux trames dépend seulement des échéances relatives à l'arrivée de la trame. En revanche, une politique définie par $\overrightarrow{P}_{i,j} = (C_i \cdot A_{i,j}, i, j)$

n'est pas IDT ; en effet, en considérant $\tau_{1,1}$ et $\tau_{2,1}$ avec $A_{1,1} = 0$, $C_1 = 10$ et $A_{2,1} = 0$ avec $C_2 = 1$ (dans ce cas $\overrightarrow{P}_{1,1} = (0,1,1)$ et $\overrightarrow{P}_{2,1} = (0,2,1)$ donc $(0,1,1) \succ (0,2,1)$) et en considérant les deux mêmes trames dont les arrivées auraient été décalées d'une unité de temps(dans ce cas $\overrightarrow{P}_{1,1} = (10,1,1)$ et $\overrightarrow{P}_{2,1} = (1,2,1)$ donc $(10,1,1) \prec (1,2,1)$).

Définition 5 *Soit un ensemble de flux $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$. Soit deux allocations d'offsets pour l'ensemble $\mathcal{T}$: $\omega = \{O_1, O_2, ..., O_n\}$ et $\omega' = \{O_1 + \varepsilon, O_2 + \varepsilon, ..., O_n + \varepsilon\}$ où ω' est une version "décalée" de ω (avec $\varepsilon \in \mathbb{Z}$) .*

Une politique d'ordonnancement A est IDT ssi pour tous les $\varepsilon \in \mathbb{Z}$ possibles, $\forall i, j, k, n$ tels que $(k,n) \neq (i,j)$ (c'est-à-dire deux trames distinctes), on ait :

$$\begin{array}{rll} \forall t\, \Gamma_{k,n}^{\mathcal{A},\omega}(t) & \succ (resp \prec) & \Gamma_{i,j}^{\mathcal{A},\omega}(t) \implies \\ \Gamma_{k,n}^{\mathcal{A},\omega'}(t+\varepsilon) & \succ (resp \prec) & \Gamma_{i,j}^{\mathcal{A},\omega'}(t+\varepsilon) \end{array}$$

où $\Gamma_{i,j}^{\mathcal{A},\omega}(t)$ est la priorité de $\tau_{i,j}$ de l'ensemble $\mathcal{T}$ avec l'allocation d'offsets ω à l'instant t sous la politique A.

Politiques implémentables. Pour pouvoir être utilisée en pratique, une politique doit être implémentable. Dans cette étude, une politique est implémentable si :

- les coordonnées de son vecteur de priorité sont "représentables" par un nombre limité de bits (par exemple, 11 bits sur le réseau CAN standard). Dans la suite, les coordonnées du vecteur de priorité sont des valeurs rationnelles ($\mathbb{Q}$) ; les imprécisions dues au nombre fini de bits pour coder la priorité, appelées "erreurs de codage" peuvent être prises en compte dans l'analyse d'ordonnancement tel que nous le verrons au §4.5.
- dans le cas où la politique spécifie que les priorités des trames sont mises à jour avant le début de chaque transmission, le calcul et l'affectation des nouvelles priorités doivent être effectués en un temps borné par la spécification du protocole.

Toute politique d'ordonnancement "acceptables" devant être implémentée au-dessus d'un bus priorisé doit avoir les propriétés mentionnées précédemment : la politique doit être déterministe, indépendante aux décalages temporels et implémentable. Dans le paragraphe suivant, nous définissons la classe des politiques non-préemptive étudiée dans le reste du chapitre.

4.4.3 Espace de recherche

Dans notre étude, nous considérons l'ordonnancement non-préemptif de trames sur un réseau priorisé CAN. Dans un premier temps, nous définissons notre espace de recherche et ensuite nous justifions nos choix.

Politiques non-préemptives "dépendantes des dates d'arrivée". L'appellation de politiques non-préemptives "dépendantes des dates d'arrivée" (NP-DDA) vient du fait que la priorité d'une trame dépend de son instant d'arrivée. Les politiques NP-DDA appartiennent à la classe de politique PPDE introduite précédemment, cf. définition 3.

Définition 6 *Une politique non-préemptives "dépendantes des dates d'arrivée" est une politique dont la fonction de priorité peut être mise sous la forme :*

$$\Gamma_{i,j}(t) = \begin{cases} \overrightarrow{P}_{i,j} = (A_{i,j} + f_i, i, j) & \text{if } t < B_{i,j} \\ \overrightarrow{Q}_{k,n} = (-\infty, i, j) & \text{if } t \geq B_{i,j} \end{cases} \tag{4.1}$$

où f_i est un nombre rationnel positif propre à chaque flux τ_i.

f_i est un nombre rationnel positif identique pour toutes les trames du flux τ_i. Ce nombre peut être calculé en fonction des caractéristiques du flux (*i.e.* $\overline{D}_i$, T_i ou C_i). Par exemple, pour NP-EDF, f_i est égal à l'échéance relative $\overline{D}_i$ des trames du flux τ_i.

Pourquoi identifier les politiques non-préemptives "dépendantes des dates d'arrivée". Avant tout, les politiques NP-DDA sont de bonnes politiques :

- le déterministe est assuré par les deux dernières coordonnées du vecteur de priorité,
- les politiques NP-DDA sont IDT, cf. définition 5, car la priorité relative entre deux trames $\tau_{k,n}$ et $\tau_{i,j}$ dépend seulement des décalages relatifs des dates d'arrivée et des valeurs f_k et f_i,
- ces politiques sont implémentables en utilisant la technique intialement proposée par Meschi et *al.* [52, 53, 54]. En effet, cette technique peut être utilisée pour coder les priorités exprimées de manière générique par $(A_{i,j} + f_i)$. L'annexe B.1 présente les détails de l'implémentation du codage utilisant une échelle logarithmique appliquée aux politiques NP-DDA.

De plus, les politiques NP-DDA sont prometteuses en termes de performances. En effet, NP-EDF, qui est optimale (cf. [8]) en regard de la faisabilité avec des ensembles non-concrets de flux (où tous les offsets sont possibles), appartient à cette classe. Cependant, il peut exister d'autres politiques ayant des performances proches de celles de NP-EDF en terme de faisabilité et ayant un meilleur comportement en ce qui concerne les critères dépendants de l'application. Les expérimentations présentées au §4.6 confirment cette intuition. Enfin, un test de faisabilité générique (dans le cas non-concret où tous les offsets sont possibles), reposant sur le calcul de borne supérieure sur le pire temps de réponse peut être réalisé pour toutes les politiques NP-DDA, cf. §4.5.3.

4.5 Analyse d'ordonnancement des politiques NP-DDA

L'ordonnançabilité dans le cas non-concret (où tous les offset sont possibles, cf. chapitre 2 pour une définition formelle) peut être analysée sous NP-EDF en utilisant des tests d'ordonnancement [8] et des calculs de borne sur le pire temps de réponse (*WCRT - Worst Case Response Time*) [10, 57]. L'objectif de cette section est de présenter une analyse d'ordonnancement calculant les bornes supérieures sur les pires temps de réponse avec les politiques NP-DDA (*i.e.*, équivalent au cas NP-EDF avec échéances relatives quelconques) prenant en compte les erreurs de codage. Nous étendons les analyses faites par Meschi et *al.* [52, 53, 54] où seul le cas des échéances inférieures ou égales à la période était traité. Le paragraphe 4.5.1 résume les travaux existants sur le calcul du pire temps de réponse. Ensuite, nous prenons en compte dans ce calcul les imprécisions dues aux erreurs de codage dans le §4.5.2. Nous étendons cette analyse, dans le §4.5.3, au cas des politiques NP-DDA.

4.5.1 Analyse du pire temps de réponse sous NP-EDF : rappel

Le temps de réponse $r_i(\mathtt{a})$ d'une trame est le temps écoulé entre son arrivée et sa fin de transmission. L'ensemble des flux est faisable sous une politique donnée si le temps de réponse de chaque trame est inférieur ou égal à son échéance relative. En général, il n'est pas possible de calculer le temps de réponse de toutes les trames pour tous les scénarios possibles du système ; une solution pour tester la faisabilité est de calculer des bornes sur le temps de réponse. Spuri [58] a développé une telle analyse pour EDF, puis Spuri et *al.* [10, 57] l'ont étendue au cas NP-EDF. George et *al.* [10] montrent que le pire temps de réponse d'une trame d'un flux arrive après une certaine situation d'arrivées appelé la situation "dès que possible" (*ASAP - "As soon As Possible"*). Les auteurs utilisent le concept de *"deadline busy period"* pour une trame $\tau_{i,j}$ d'échéance absolue $D_{i,j}$, qui est une période d'occupation du réseau sans interruption dans laquelle seul les trames avec une priorité supérieure ou égale à celle de $\tau_{i,j}$ (c'est-à-dire, ayant une échéance absolue inférieure ou égale) sont exécutées.

Lemme 1 *[10] Une borne sur le temps de réponse d'une trame, appartenant au flux τ_i, activée $\mathtt{a}$ unités de temps après le début de sa deadline busy period peut être calculée dans la deadline busy period générée dans la situation ASAP. La situaton ASAP est la situation où :*

- *τ_i génère une trame à l'instant $\mathtt{a}$ (d'autres trames peuvent avoir été générées avant $\mathtt{a}$),*
- *toutes les flux τ_k, dont l'échéance relative $\overline{D}_k$ est inférieure ou égale à $\mathtt{a}+\overline{D}_i$, sont activés à partir de l'instant $t = 0$ à leur "taux" maximal,*
- *la plus grande trame de tous les flux dont l'échéance relative est plus grande que $\mathtt{a}+\overline{D}_i$, s'il y en a, est générée à l'instant $t = -1$. Cette trame constitue le facteur de blocage dû à la non-préemption.*

Ce lemme permet de calculer une borne sur le temps de réponse $r_i(\mathtt{a})$ d'un flux τ_i activé à l'instant $\mathtt{a}$, noté $\tau_i(\mathtt{a})$ dans la suite. Il a été prouvé dans [10] que l'exécution de $\tau_i(\mathtt{a})$ commence, au plus tard, à l'instant t solution de l'équation suivante qui peut être résolue par récurrence :

$$t = W_i(\mathtt{a}, t) + \mathcal{B}_i(\mathtt{a}) + \left\lfloor \frac{\mathtt{a}}{T_i} \right\rfloor \cdot C_i. \tag{4.2}$$

dans laquelle la longueur $L_i(\mathtt{a})$ de la *deadline busy period* est la valeur de t solution de l'équation 4.2, et où $W_i(\mathtt{a}, t)$ est une borne supérieure sur la "charge de travail de plus haute priorité" (*i.e.,* le travail induit par les trames ayant une échéance inférieure ou égale) dans l'intervalle de longueur t ($(x)^+ \stackrel{\text{def}}{=} \min(x, 0)$) :

$$W_i(\mathtt{a}, t) = \sum_{k \neq i} \underbrace{\left(\min\left(\left\lfloor \frac{t}{T_k} \right\rfloor + 1, \left\lfloor \frac{\mathtt{a} + \overline{D_i} - \overline{D_k}}{T_k} \right\rfloor + 1\right)\right)^+}_{\substack{\text{nombre maximal de trames dans l'intervalle } t \\ \text{ayant une échéance inférieure ou égale à } \mathtt{a} + \overline{D}_i}} .C_k \tag{4.3}$$

avec $\mathcal{B}_i(\mathtt{a})$ le facteur de blocage (*i.e.,* la plus grande trame ayant une échéance relative inférieure à celle de $\tau_i(\mathtt{a})$) :

$$\mathcal{B}_i(\mathtt{a}) = \max_{k=1..n} \{C_k \mid \overline{D}_k > \mathtt{a} + \overline{D}_i\} - 1 \tag{4.4}$$

et où le travail des autres trames du flux τ_i est :

$$\left\lfloor \frac{\mathtt{a}}{T_i} \right\rfloor \cdot C_i \tag{4.5}$$

Donc, une borne sur le temps de réponse d'une trame $\tau_i(\mathtt{a})$ est :

$$r_i(\mathtt{a}) = \max\left\{C_i, L_i(\mathtt{a}) + C_i - \mathtt{a}\right\}. \tag{4.6}$$

La borne sur le temps de réponse pour τ_i est $\max_{\mathtt{a}} r_i(\mathtt{a})$. Il serait très long de calculer les valeurs de $r_i(\mathtt{a})$ pour toutes les valeurs possibles de $\mathtt{a}$, mais il a été prouvé dans [58, 10] que les seules valeurs "significatives" de $\mathtt{a}$ sont les éléments de l'ensemble $\mathbf{A}_i$ suivant (correspond aux instants où les fonctions $W_i(\mathtt{a}, t)$, $\mathcal{B}_i(\mathtt{a})$ et $\left\lfloor \frac{\mathtt{a}}{T_i} \right\rfloor \cdot C_i$ changent de valeur) :

$$\mathbf{A}_i = \{t = h \cdot T_k + \overline{D}_k - \overline{D}_i \mid t \geq 0,\, t \leq \mathcal{L} - C_i,\, h \in \mathbb{N},\, k = 1...n\} \tag{4.7}$$

où $\mathcal{L}$ est la plus longue période d'utilisation (plus longue durée pendant laquelle la ressource est occupée sans interruption, cf. [58] pour les détails du calcul). Les valeurs significatives sont les valeurs qui impliquent des changements de valeurs dans la charge de travail plus prioritaire calculée avec l'équation 4.3.

La section suivante montre comment cette analyse peut être adaptée quand des inversions de priorité peuvent arriver à cause du nombre limité de bits pour coder la priorité.

4.5.2 Pire temps de réponse sous NP-EDF avec des erreurs de codage

Quand on utilise une échelle logarithmique (ou un nombre limité de bits pour coder l'échéance), les trames qui ont des échéances différentes peuvent appartenir au même slot. L'ordre initial donné par les priorités est alors perdu et une inversion de priorité peut se produire (*i.e.*, donner la même valeur). Dans le pire des cas, durant le calcul de pire temps de réponse d'une trame $\tau_i(\mathtt{a})$, nous supposons que toutes les trames ayant une échéance dans le même slot que $\tau_i(\mathtt{a})$ ont une priorité plus importante. Précisément, la charge de travail plus prioritaire est maintenant composée de toutes les trames ayant une échéance inférieure ou égale à $A_{i,j} + \overline{D}_i + S_i(\mathtt{a}, t)$, où $S_i(\mathtt{a}, t)$ est une borne sur la taille du slot où $\mathtt{a} + \overline{D}_i - t_{start}$ tombe. Donc, la charge de travail $W_i(\mathtt{a}, t)$ plus prioritaire devient :

$$W_i(\mathtt{a}, t) = \sum_{k \neq i} \left(\min\left(\left\lfloor \frac{t}{T_k} \right\rfloor + 1, \left\lfloor \frac{\mathtt{a} + \overline{D}_i + S_i(\mathtt{a}, t) - \overline{D}_k}{T_k} \right\rfloor + 1 \right) \right)^+ \cdot C_k \tag{4.8}$$

La justification de la formule est donnée en annexe B.3. La difficulté est que la longueur du slot $S_i(\mathtt{a}, t)$ n'est pas constante au cours du temps : elle dépend de la valeur de l'échéance qui doit être codée (*i.e.*, $\mathtt{a} + \overline{D}_i$), et de l'origine de l'échelle logarithmique (t_{start}), qui est mise à jour à chaque nouveau début de transmission. Comme l'explique l'annexe B.3, la taille de $S_i(\mathtt{a}, t)$ a une croissance monotone avec $\mathtt{a} + \overline{D}_i - t_{start}$. Pour calculer une borne supérieure à $S_i(\mathtt{a}, t)$, on doit trouver une borne inférieure pour la valeur t_{start} à laquelle $\tau_i(\mathtt{a})$ gagne l'accès au canal de communication. Comme $\tau_i(\mathtt{a})$ arrive en $\mathtt{a}$, une borne inférieure évidente est a, mais on peut améliorer le résultat selon la méthode proposée par Di Natale [53].

Pour quelles valeur s de a doit on calculer $r_i(a)$. Comme Spuri et *al.* [58, 10] l'ont montré les seules valeurs significatives de a qui doivent être analysées sont les valeurs qui impliquent un changement dans la charge de travail de plus grande priorité calculée avec l'équation 4.8. Précisément, ces valeurs sont des valeurs entières de la forme $h \cdot T_k + \overline{D}_k - \overline{D}_i - S_i(\mathtt{a}, t)$; le problème est que la longueur du slot $S_i(\mathtt{a}, t)$ n'est pas a priori connue car il dépend également de a. La méthode de codage logarithmique présentée dans l'annexe B.1 assure que, à tout instant, la longueur du slot prend sa valeur dans l'intervalle $[S, 2^{k_{max}}S]$, où S est la longueur du premier slot (cf. annexe B.1 pour plus de détails). Une solution pour prendre en compte un changement dans la charge de travail de plus grande priorité est donc de considérer toutes les valeurs possibles de la longueur du slot ; l'ensemble de a à analyser devient :

$$\mathbf{A}_i = \{t = h \cdot T_k + \overline{D}_k - \overline{D}_i - x \mid t \geq 0,\ t \leq \mathcal{L} - C_k,\ h \in \mathbb{N},\ x \in \mathbb{N},\ x = S...2^{k_{max}}S,\ k = 1...n\}. \quad (4.9)$$

La section suivante montre comment cette analyse peut être facilement adaptée à la classe des politiques NP-DDA.

4.5.3 Pire temps de réponse sous NP-DDA avec erreurs de codage

Une trame $\tau_{i,j}$ sous NP-EDF possède un vecteur de priorité $\overrightarrow{P}_{i,j}$ égal à $(A_{i,j} + \overline{D}_i, i, j)$; NP-EDF est donc un cas particulier de politique NP-DDA où $f_i = \overline{D}_i$ (cf. définition des NP-DDA du § 4.4.3). Dans la suite, il est montré que le lemme 1 aussi bien que l'ensemble des dates d'arrivée a à considérer après la situation ASAP (cf. équation 4.7) reste valide sous condition de remplacer $\overline{D}_i$ par f_i. Les deadline busy periods deviennent les priority busy periods pour une trame $\tau_{i,j}$ qui sont des intervalles où le réseau est utilisé sans interruption et durant lesquelles seules les trames plus prioritaires que $\tau_{i,j}$ sont exécutées.

Lemme 2 *Une borne sur le temps de réponse d'une trame, appartenant au flux τ_i, activée a unités de temps après le début de la priority busy period peut être trouvée dans une priority busy period induit par la situation ASAP. La situation ASAP est la situation où :*

- *τ_i génère une trame à l'instant a (d'autres trames peuvent avoir été générées avant a),*
- *tous les flux, dont la valeur f_k est inférieure ou égale à $\mathtt{a} + f_i$, sont activés à partir de l'instant $t = 0$ à leur "taux" maximal,*
- *la plus grande trame de tous les flux dont le f_k est plus grand que $\mathtt{a} + f_i$, s'il y en a, est générée à l'instant $t = -1$. Cette trame constitue le facteur de blocage dû à la non-préemption.*

Ebauche de preuve:

Considérons une politique que nous appelerons par la suite NP-EDF-virtuelle. NP-EDF-virtuelle est une version modifiée de NP-EDF qui ordonnance les trames non pas en prenant en compte les échéances $A_{i,j} + \overline{D}_i$ des trames, mais une échéance arbitraire "virtuelle" $A_{i,j} + f_i$. La fonction de priorité de cette politique NP-EDF-virtuelle est :

$$\Gamma_{i,j}^{virtual\ NP-EDF}(t) = \begin{cases} \overrightarrow{P}_{i,j} = (A_{i,j} + f_i, i, j) \text{ si } t < B_{i,j} \\ \overrightarrow{Q}_{i,j} = (-\infty, i, j) \text{ si } t \geq B_{i,j} \end{cases},$$

où f_i, comme $\overline{D}_i$, possède une valeur positive égale pour toutes les trames d'un flux τ_i. En effet, cette propriété sur f_i est nécessaire pour que le lemme 4.1 dans [10] soit valide (précisément, quand la situation ASAP est

construite, décaler vers la gauche une trame doit augmenter la charge de travail plus prioritaire).

Selon le lemme 1, une borne sur le temps de réponse de τ_i sous NP-EDF-virtuelle arrive après la situation ASAP, telle que définie par George et *al.* [10], où $\overline{D}_i$ est remplacée par f_i dans les équations 4.3 et 4.7. Pour tester la faisabilité, les bornes sur le temps de réponse sont comparées avec l'échéance relative actuelle $\overline{D}_i$.

■

La méthode de calcul des pires temps d'exécution sous les politiques NP-DDA est la même que sous NP-EDF excepté que $\overline{D}_i$ est remplacé par la valeur f_i dans les équations 4.4 et 4.8.

4.6 Expérimentations

Dans la suite, les performances des politiques d'ordonnancement NP-DDA sont évaluées dans le contexte des systèmes contrôlés en réseaux (*Networked Control Systems - NCS*), qui est un important domaine d'application de nos propositions.

4.6.1 Espace de recherche

L'objectif est de trouver des politiques qui ont de bonnes performances en termes de faisabilité, pour optimiser l'utilisation de la bande passante,, mais qui sont également efficaces vis-à-vis du respect des critères importants pour les NCS (qui seront définis dans les §4.6.2.1 et 4.6.3.2). Dans la suite, les simulations sont faites au sein d'une sous-classe des politiques NP-DDA ayant un vecteur de priorité $\overrightarrow{P}_{i,j}$ égal à :

$$\overrightarrow{P}_{i,j} = (A_{i,j} + c \cdot C_i + d \cdot \overline{D_i},\ i,\ j) \text{ avec } c \in [0, 50] \text{ et } d \in [0, 1], \tag{4.10}$$

qui signifie que dans la définition 6, on a $f_i = c \cdot C_i + d \cdot \overline{D_i}$. Donc, une politique appartenant à notre espace de recherche est définie par une fonction de priorité ayant la forme décrite par l'équation 4.10. Les simulations ont montré que les paramètres $c \in [0, 50]$ et $d \in [0, 1]$ définissent un espace de recherche où, en général, les politiques intéressantes sont trouvées.

Cette sous-classe de politiques NP-DDA a été choisie parce que nous conjecturons qu'elle contient des politiques fournissant un bon compromis entre la faisabilité et la satisfaction des critères importants dans les NCS (critères qui seront formellement décris dans le § 4.6.2.1). En effet, NP-EDF appartient à cette classe ($f_i = \overline{D}_i$) et nous espérons que les politiques dont la fonction de priorité est "proche" de NP-EDF ont un comportement similaire en terme de faisabilité. D'un autre côté, introduire un terme dépendant du temps de transmission pourrait permettre d'améliorer les autres critères. En particulier, il a été montré que la politique préemptive allouant la plus forte priorité à l'activité ayant la quantité de travail la plus faible (*Shortest Remaining Processing Time First - SRPTF*) est optimale en ce qui concerne les temps de réponse moyens (*i.e.*, pour minimiser la moyenne des temps de réponse), cf. [27, 28] cité dans [29]) et, dans nos simulations, la version non-préemptive de cette politique (*Non-Preemptive Shortest Maximum Processing Time First - NP-SMPTF*) définie par $\overrightarrow{P}_{i,j} = (C_i,\ i,\ j)$, réalise de meilleures performances que NP-EDF pour tous les autres critères autres que la faisabilité.

Dans la suite, les performances des politiques NP-DDA sont évaluées et comparées à celles de NP-SMPTF et NP-EDF. En plus, *Non-Preemptive Deadline Monotonic* (*NP-DM*) est également considérée,

car cette politique est connue pour ses bonnes performances en termes de faisabilité (George et *al.* [10] ont montré que NP-DM est optimal en regard de la faisabilité pour le cas non-préemptif avec $\overline{D}_i \leq T_i$ quand $\overline{D}_i \leq \overline{D}_k$ implique $C_i \leq C_k$) et pour sa facilité de mise en oeuvre.

4.6.2 Performances en regard de l'ordonnancement

Les expériences suivantes étudient l'impact des politiques NP-DDA sur les performances en regard de critères d'ordonnancement tels que le temps de réponse des flux.

4.6.2.1 Critères de performances

Un système de contrôle commande en boucle fermée est composé de trois parties : l'échantillonnage des sorties du systèmes à contrôler (*i.e.*, les données sont lues sur des capteurs), le calcul de la consigne à appliquer aux actionneurs et l'application de cette consigne aux actionneurs (*i.e.*, transmission du signal de contrôle aux actionneurs), voir figure 4.1. Des délais spécifiques ont été identifiés pour avoir

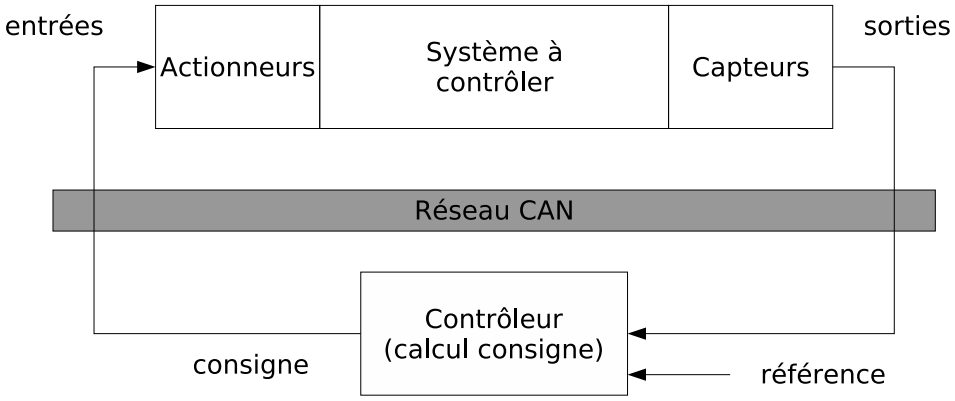


FIG. 4.1 – Représentation schématique d'un NCS.

un impact sur la stabilité du système et, plus généralement, sur ses performances (cf., par exemple, les études dans [59, 43, 44] et le §4.6.3). Ces délais incluent :

- *la latence d'entrée-sortie* : le temps écoulé entre la lecture d'une sortie et de l'application de la consigne correspondante aux actionneurs,
- *l'intervalle d'échantillonnage* : l'intervalle de temps entre deux lectures consécutives des sorties,
- *la latence d'échantillonnage* : le temps écoulé entre l'instant théorique de lecture d'une sortie et son occurence réelle.

Dans les NCS, où la boucle de contrôle est distribuée sur un réseau, les données des capteurs au contrôleur et du contrôleur aux actionneurs sont par exemple échangées par le réseau. Les communications des capteurs au contrôleur (temps de réponse noté t_{sc}) et du contrôleur aux actionneurs (temps de réponse noté t_{ca}) induisent des délais non-constants dans la latence d'entrée-sortie (égale à t_{sc}+temps de réponse+t_{ca}), et sa variabilité est bien connue pour avoir une influence cruciale sur les performances de la boucle de contrôle (cf., par exemple, [43] et les expérimentations du § 4.6.3). Un système de communication efficace doit donc minimiser les délais de communications et leurs variabilités. Les expériences suivantes étudient

l'impact des politiques NP-DDA sur les temps de réponse des flux (*i.e.*, le délai entre l'instant de mise en attente et la fin de transmisson).

4.6.2.2 Conditions de simulation

Nous considérons des flux échangés sur un réseau CAN. Selon nos connaissances, il n'y a pas de techniques analytiques exactes pour évaluer la valeur de nos critères[1], c'est pourquoi nous utilisons la simulation. Un critère donné pour une politique est évalué comme la valeur moyenne de tous les flux.

Le temps de transmission d'un octet est la plus petite unité de temps, ce qui signifie que le délai de propagation est négligé et que les effets de bit-stuffing sont négligés (comme le bit-stuffing dépend des données transmises, le bit-stuffing n'entre pas dans le cadre de notre étude). Les ensembles de flux sont générés avec l'algorithme 4 décrit dans l'annexe A selon le tuple $(n, U, 8, 16, 1, 1, +\infty)$, c'est-à-dire des ensembles de n flux avec une charge globale dans l'intervalle $[U - 0.02, U + 0.02]$, où les flux sont à échéance sur requête ($\overline{D}_i = T_i$) et ont un temps d'exécution dans l'intervalle $[8, 16]$. Dans les simulations, nous cherchons des ensembles faisables dans le cas non-concret (où tous les offsets sont possibles) ; néanmoins pour l'évaluation des critères de performances, des ensembles concrets de flux (où les offsets sont connus) sont générés à partir de ceux non-concrets en choisissant un offset pour chaque flux τ_k dans l'intervalle $[0, T_k[$. Le calcul de bornes sur les temps de réponse et un simulateur ont été implémentés en C++ (une version applet du simulateur est disponible à l'adresse suivante *http ://www.loria.fr/~grenier/logiciel/SimApplet.html*).

4.6.2.3 Influence des paramétres

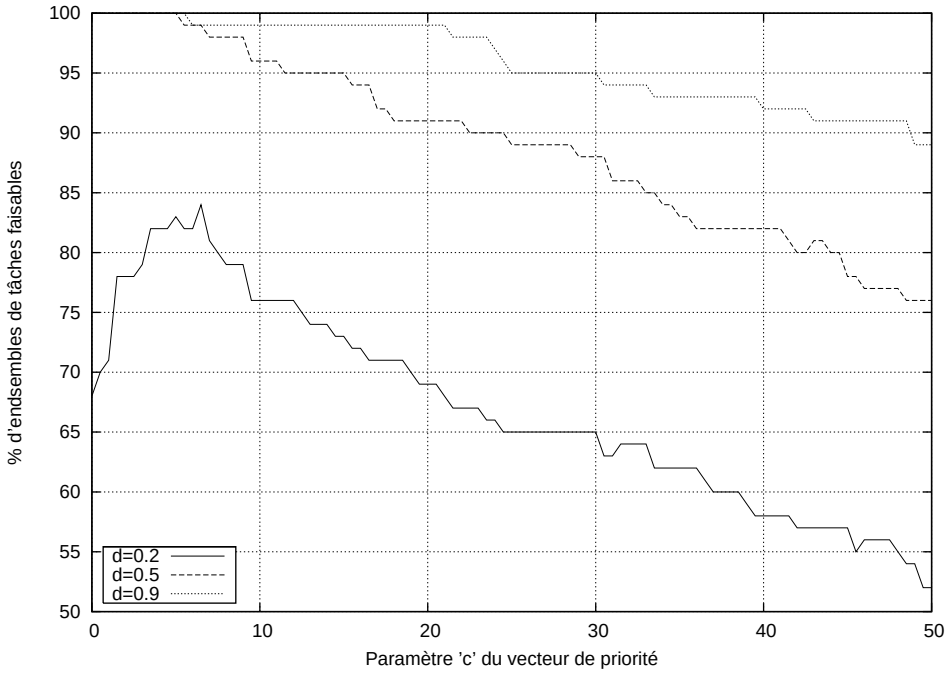
Dans ce paragraphe, nous évaluons de quelle manière les valeurs des paramètres c et d dans le vecteur de priorité $\overrightarrow{P}_{i,j}$ décrit par l'équation 4.10 (*i.e.*, $\overrightarrow{P}_{i,j} = (A_{i,j} + c \cdot C_i + d \cdot \overline{D_i}, i, j)$ avec $c \in [0, 50]$ et $d \in [0, 1]$) influencent la faisabilité et la moyenne des gigues sur les temps de réponse, où la gigue sur le temps de réponse pour un flux est l'écart type des temps de réponse des trames de ce flux. Pour la faisabilité, la performance d'une politique $\mathcal{A}$ est le pourcentage d'ensemble de flux faisables sous NP-EDF qui sont faisables sous $\mathcal{A}$.

Les figures 4.2(a) et 4.2(b) montrent les performances de l'ensemble des politiques définies par l'équation 4.10 où $d \in \{0.2, 0.5, 0.9\}$ et c prend ses valeurs dans $[0, 50]$ avec un pas égal à 0.5 . Les performances sont présentées en termes de faisabilité dans la figure 4.2(a) et de moyenne de gigues sur les temps de réponse dans la figure 4.2(b). Les valeurs moyennes sont calculées sur 1500 simulations : 100 ensembles non-concrets de 10 flux (*i.e.*, $n = 10$) avec 15 allocations différentes d'offsets avec une charge globale comprise dans l'intervalle $[0.6, 0.7]$. Seuls les ensembles de tâches faisables sous NP-EDF sont considérés.

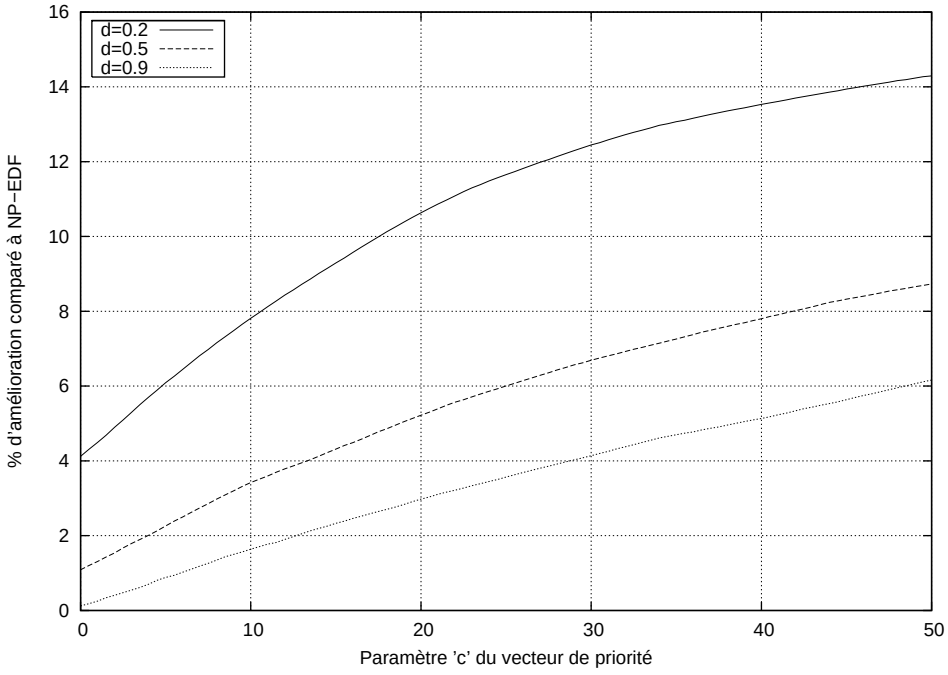
D'après les figures 4.2(a) et 4.2(b), on peut observer que :

- plus la valeur de c dans $\overrightarrow{P}_{k,n}$ est importante, plus la moyenne des gigues sur les temps de réponse est faible. La contrepartie est que la faisabilité décroît de manière significative quand c augmente.
- plus la valeur de d dans $\overrightarrow{P}_{k,n}$ est importante, meilleure est la faisabilité avec la contrepartie que la moyenne des gigues sur le temps de réponse augmente.

[1] Il existe des méthodes analytiques pour évaluer des bornes sur la gigue, cf. [60] par exemple, en utilisant des calculs de meilleurs et pires temps de réponse.



(a) Faisabilité



(b) Moyenne des gigues sur les temps de réponse

FIG. 4.2 – Faisabilité et gigue moyenne sur les temps de réponse pour les politiques définies par l'équation 4.10 avec $d \in \{0.2, 0.5, 0.9\}$ et c allant de 0 à 50. Chaque point est la valeur moyenne de 1500 simulations d'ensemble de 10 trames avec une charge globale dans $[0.6, 0.7]$.

En effet, quand c devient important dans l'équation 4.10, $c \cdot C_k$ a plus d'influence que $A_{k,n}$ et $d \cdot \overline{D}_k$ sur la priorité des trames. Donc, la politique a tendance à se comporter d'une manière similaire à Non-Preemptive Shortest Maximum Processing Time First (cf. § 4.6.1). Pour la même raison, quand d augmente, la politique agit de façon similaire à NP-EDF en termes de faisabilité. Un autre résultat est qu'il n'y a pas de différence significative concernant le temps de réponse moyen (ex. : dans nos simulations, la meilleure politique NP-DDA améliore le temps de réponse moyen de 1.4% comparée à NP-EDF). Ce phénomène est dû aux relativement faibles variations des temps de transmissions C_k imposées par le protocole CAN.

Le tableau 4.1 présente les performances de NP-SMPTF et NP-DM en comparaison à NP-EDF avec les mêmes ensembles de tâches simulés. De la figure 4.2 et du tableau 4.1, nous voyons que les politiques

	Faisabilité (%)	moyenne des gigues sur les temps de réponse : amélioration par rapport à NP-EDF
NP-SMPTF	57.5	17.9
NP-DM	100	−4.5

TAB. 4.1 – Faisabilité et moyenne des gigues sur les temps de réponse (en %) pour NP-SMPTF et NP-DM.

NP-DDA ont de meilleures performances que NP-SMPTF en termes de faisabilité (quand c est inférieur à 40), et sont meilleures que NP-DM et NP-EDF pour la gigue moyenne sur le temps de réponse. Par exemple, la politique définie par $\overrightarrow{P}_{i,j} = (A_{i,j} + 18 \cdot C_i + \frac{2}{10}\overline{D}_i, i, j)$ permet de trouver un ordonnancement faisable pour 70% des ensembles de flux tandis qu'une amélioration de 10.2% sur la gigue sur le temps de réponse comparée à NP-EDF est réalisée. Dans les mêmes conditions d'expérimentation, NP-SMPTF est ordonnançable pour 57.5% des ensembles de flux tandis qu'elle améliore la gigue de 17.9%. D'un autre côté, NP-DM ordonnance 100% des ensembles de flux, mais est pire que NP-EDF en regard de la gigue. Ces résultats confirment que les politiques NP-DDA que nous avons proposées offrent un bon compromis entre faisabilité et amélioration de critères propres à l'application.

En pratique, la plupart des NCS auront de meilleures performances (*i.e.*, plus petite gigue sur les temps de réponse dans nos simulations) avec une politique NP-DDA bien choisie assurant la faisabilité. De plus, la faisabilité avec les politiques NP-DDA est bien meilleure que sous NP-SMPT. Les politiques NP-DDA permettent au designer de l'application d'implémenter un protocole au niveau MAC fournissant un bon compromis ente faisabilité et les critères dépendants de l'application tels que la minimisation de la gigue.

4.6.3 Performances de la boucle de contrôle-commande

Dans ce paragraphe, un NCS particulier est considéré et les performances de la meilleure politique NP-DDA trouvée est comparée aux performances réalisées avec NP-EDF. Le processus est un moteur électrique (*DC servo*) ayant une fonction de transfert définie par $G(s) = \frac{1000}{s^2+s}$. Le système, cf. figure 4.3, est composé d'une boucle de contrôle où le capteur, le contrôleur et l'actionneur sont distribués sur un réseau CAN à 125kbit/s. Il y a également un noeud générant du trafic sur le réseau pour modéliser les autres flux temps réel échangés sur le bus. Ce système est simulé avec la boîte à outils TrueTime sous Matlab/Simulink et est décrit plus en détail dans [61]. Pour les besoins de cette étude, l'allocation de

priorité NP-DDA que nous avons proposée a été implémentée dans TrueTime.

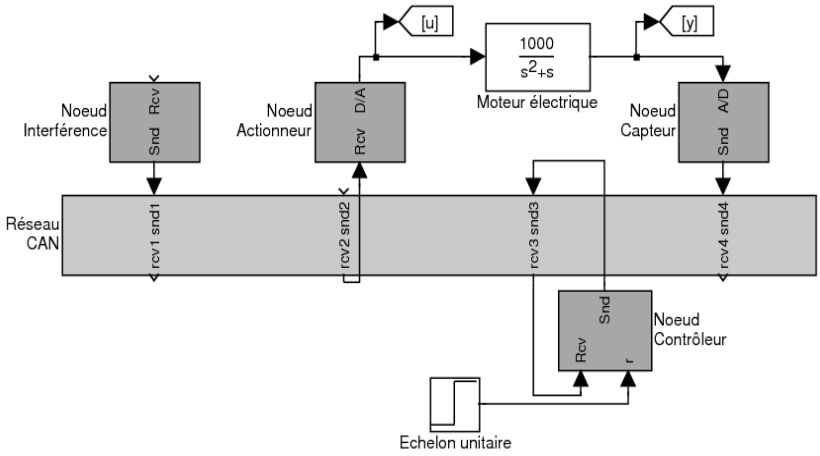


FIG. 4.3 – L'architecture du NCS composée d'une boucle de contrôle où le capteur, le contrôleur et l'actionneur sont distribués sur un réseau CAN à 125kbit/s. Un noeud générant du trafic temps réel sur le réseau modélise les autres flux temps réel échangés sur le bus.

4.6.3.1 Aperçu de l'architecture

Précisément, comme dans [61], le NCS est composé de :

- un noeud "capteur" (une tâche - *time driven*) où la seule tâche de ce noeud prélève un échantillon avec une période de 0.01s. Ensuite, la tâche envoie le résultat $y(k)$ au contrôleur par le réseau ($C = 80$bits). Le délai entre l'échantillonnage et l'instant où le message est mis dans la file d'attente est de $9 \cdot 10^{-4}$s.
- un noeud "contrôleur" (une tâche - *event driven*) où la seule tâche, chaque fois qu'elle reçoit une donnée du capteur, calcule le signal de sortie $u(k)$ et l'envoie à l'actionneur. Le délai entre la réception du message et l'instant où le signal de contrôle ($C = 80$bits) est mis dans la file d'attente est de $5 \cdot 10^{-4}$s. Un contrôleur PD est utilisé et est implémenté selon les équations suivantes :

$$u(k) = P(k) + D(k),$$

$$P(k) = 1.5 \cdot (r(k) - y(k)),$$

$$D(k) = 3.5 \cdot 10^{-5} \cdot D(k-1) + 5.25 \cdot (y(k-1) - y(k)),$$

  où $r(k)$, la consigne, est un échelon unitaire (cf. [61] et [62] pour de plus amples explications).
- un noeud "actionneur" (une tâche - *event driven*) contrôle l'actionneur selon les données reçues dans la trame émise par le contrôleur. Le délai entre la réception de la trame et l'application effective de l'action est de $5 \cdot 10^{-4}$s.
- un noeud "interférence" (une tâche - *time-driven*) génère du trafic perturbant le réseau. L'ensemble de ses flux est généré aléatoirement : pour une charge U donnée, un flux aléatoire est ajouté

itérativement jusqu'à ce que la charge du noeud dépasse U. La période de chaque flux est fixée aléatoirement selon une loi de distribution uniforme dans l'intervalle $\{0.01, 0.02, 0.05, 0.1, 0.5\}$ seconde, et le nombre d'octets de données prend une valeur aléatoire dans l'intervalle $[1, 8]$.

Nous devons remarquer ici que les flux envoyés ont des contraintes de précédances : le flux périodique du noeud "contrôleur" est envoyé après que le message émis par le noeud "capteur " soit reçu (même situation entre le noeud "contrôleur" et "actionneur"). Comme Tindell et Clark [63] l'ont montré, ce délai entre l'arrivée (*arrival*) et l'activation (*release*) du flux peut être pris en compte par l'introduction pour chaque flux τ_i d'une gigue d'activation J_i dépendant du temps de réponse du flux précédant (cf. analyse holistique de Tindell et Clark [63]).

4.6.3.2 Critères de performance

La réponse du système à un échelon unitaire (*i.e.*, si $t < 0$, $r(t) = 0$ sinon $r(t) = 1$) est utilisé avec le dépassement $\overline{O}$, le temps de réponse (point de vue automatique) $T_{\%}$ et l'IAE (Integral of the Absolute magnitude Error) comme métriques de performances. Dans cette étude, $IAE = \int_0^{0.5} |e(t)|\, dt$, où la durée de simulation est de 0.5s et le temps de réponse $T_{\%}$ est le temps requis par le système pour que la réponse du système reste à 10% de la valeur désirée.

4.6.3.3 Résultats

La figure 4.4 montre la réponse indicielle d'un système (ayant une charge $U = 0.5$) avec la meilleure politique NP-DDA de l'espace de recherche et avec la politique NP-EDF. Comme on peut le constater, le dépassement passe de 1.155 à 1.085, tandis que $T_{10\%}$ chute de 46 millisecondes à 169 millisecondes avec la meilleure politique NP-DDA trouvée.

La figure 4.5 présente les améliorations de performances moyennes (sur 100 simulations pour chaque point) de la meilleure politique NP-DDA trouvée comparée à NP-EDF pour les trois métriques utilisées. Cette figure met en évidence de l'amélioration apportée par les politiques NP-DDA comparé à NP-EDF pour les NCS étudiés. En effet, l'amélioration est significative quelque soit la charge et augmente avec elle. Par exemple, pour une charge de 0.6, l'amélioration comparée à NP-EDF est de 35.5% pour le temps de réponse à 10%, 30.1% pour le dépassement et 14.3% pour l'IAE.

4.7 Conclusion et perspectives

Ce chapitre propose la classe des politiques Dépendantes des Dates d'Arrivée (NP-DDA). Nous montrons qu'elles peuvent être utilisées pour l'ordonnancement de trames au niveau MAC sur CAN et qu'elles fournissent un bon compromis entre la faisabilité et la satisfaction d'autres critères dépendants de l'application tels que la gigue sur le temps de réponse. Une politique NP-DDA peut-être "construite" pour les performances requises par une application particulière ou peut être choisie pour donner de bonnes

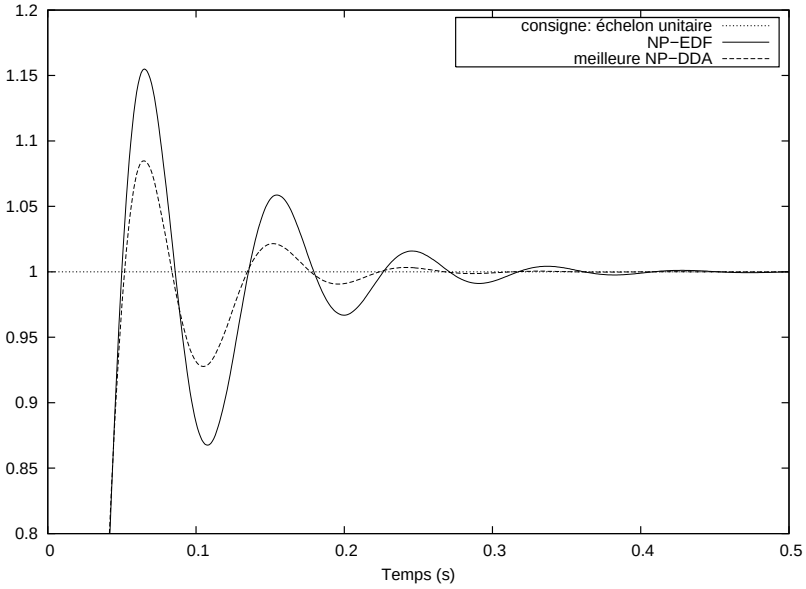


FIG. 4.4 – Réponse indicielle d'un NCS avec NP-EDF et avec la meilleure politique NP-DDA trouvée avec une recherche exhaustive ($c = 35$ et $d = 0.4$). La charge est de 0.5 . Sur cette figue ne figure pas l'évolution du système entre 0 et $t_{0.8}$ où $t_{0.8}$ est le premier instant où la réponse du système atteint pour la première fois 0.8 fois la consigne.

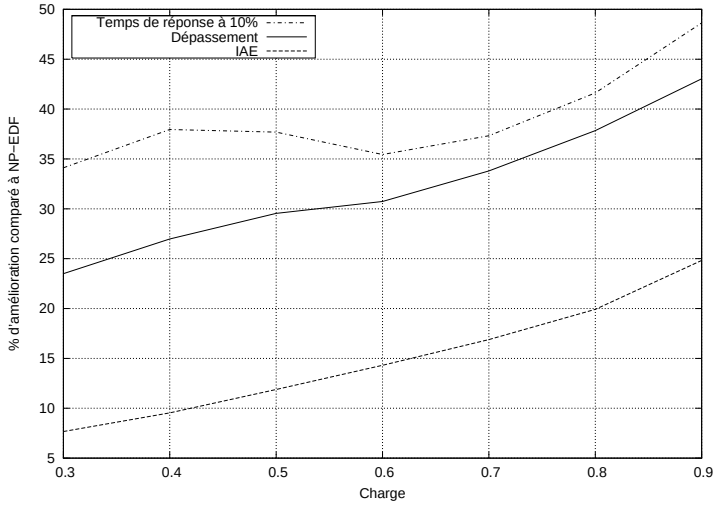


FIG. 4.5 – Améliorations moyennes du dépassement, du temps de réponse et de l'IAE apportées par l'utilisation de
la meilleure politique NP-DDA trouvée avec une recherche exhaustive comparé à l'utilisation de NP-EDF.

performances moyennes sur des ensembles de flux. Un intérêt de ces politiques est qu'elles sont implémentables sur des composants du commerce (ex. :, les contrôleurs CAN) en utilisant la méthode proposée par Meschi et *al.* [52].

Pour tester la faisabilité du système utilisant les politiques NP-DDA, une analyse d'ordonnancement générique pour toutes les politiques de la classe a été proposée. Dans le contexte des NCS où les délais et les gigues ont un impact important sur les performances de la boucle de contrôle, nos simulations ont montré que bien choisir les politiques peut améliorer significativement les performances comparé à NP-EDF et NP-DM.

Une perspective serait d'évaluer plus précisément l'impact de nos politiques d'ordonnancement sur un système contrôlé s'exécutant sur une plateforme réelle. Une autre amélioraton serait d'utiliser des techniques de recherche plus efficaces ; des expérimentations préliminaires ont montré qu'une simple méthode approchée d'optimisation par voisinnage de recherche d'optimum local (*i.e., hill-climbing*) est plus efficace qu'une recherche exhaustive. Finalement, ce travail pourrait être étendu aux autres classes de politiques telles que les politiques "partageant le temps" (*Time sharing*) (ex. : Round-Robin [1], Pfair [64]) ; le principal problème sera de développer une analyse d'ordonnancement générique pour les politiques de ces classes.

Chapitre 5

Nouvelles politiques à priorité fixe dans les systèmes "offset free"

5.1 Introduction

Contexte. Ce chapitre traite de l'ordonnancement des tâches/flux dans les systèmes temps réel à contraintes dures, *i.e.*, des systèmes dans lesquels le respect des contraintes de temps est obligatoire. Nous considérons plus spécifiquement les systèmes offset free où l'offset de chaque tâche/flux peut être choisi par l'utilisateur. Les modèles utilisés pour les tâches et les flux sont ceux décrit dans le chapitre 1 §1.2 où une tâche/flux est modélisé par un tuple (C_i, T_i, D_i, O_i) où chaque requête de τ_i a un pire temps d'exécution C_i, une échéance relative arbitraire $\overline{D}_i$ (*i.e.*,$\forall \overline{D}_i$), une période T_i et un offset O_i.

Trois types d'ensembles de tâches/flux périodiques concrets (les offsets des tâches sont connus avant l'exécution du système, cf. chapitre 2 §2.1) peuvent être distingués : les ensembles synchrones, où tous les offsets sont égaux à 0, les ensembles asynchrones, dans lesquels les contraintes du système déterminent les offsets et finalement les systèmes offsets free. Dans les systèmes offset free, il n'y a pas de contraintes sur les offsets. Ils peuvent donc être choisis hors-ligne. Nous voulons rappeler que considérer seulement le cas synchrone est très pessimiste, car le cas synchrone est le pire en terme de faisabilité pour l'ordonnancement préemptif, dans le sens où, si un système est ordonnançable dans le cas synchrone il est également ordonnançable dans toutes les situations asynchrones (cf. [65], par exemple). Le cas synchrone pour l'ordonnancement non-préemptif n'est pas le pire, mais reste un très mauvais scénario car tous les tâches/flux sont activés en même temps.

Définition du problème. Le problème d'ordonnancement est le suivant : étant donné un ensemble de tâches/flux dont les caractéristiques T_i, C_i et $\overline{D}_i$ sont connues, trouver un ensemble d'offsets et une allocation de priorité fixe permettant d'atteindre les objectifs définis dans la suite de ce paragraphe en fonction du contexte. Dans cette étude, nous avons considéré des applications s'exécutant dans un contexte :

- mono-processeur préemptif (modèle de tâche) où l'objectif est de trouver une allocation d'offset et de priorité faisable,

– distribué non-préemptif (modèle de flux) où l'objectif est de minimiser le temps de réponse des trames échangées sur le réseau.

Dans ces deux situations, l'objectif est de fixer les offsets afin de réduire l'interférence entre les activités périodiques du système.

État de l'art. Avant toute chose, nous rappelons qu'une règle d'allocation de priorité est optimale vis-à-vis de l'ordonnançabilité pour les systèmes concrets (où les offsets sont fixés) si : quand une allocation de priorité ordonnançable existe pour un système concret, ce système concret est également ordonnançable avec la règle d'allocation de priorité considérée.

Leung et Whitehead [12] prouvent la non-optimalité de DM dans le cas concret quand les échéances des tâches sont quelconques et suggérent une régle d'allocation considérant $n!$ différentes allocations de priorités. Audsley [14, 66] propose un algorithme d'allocation optimal considérant au plus $\mathcal{O}(n^2)$ allocations. Dans la littérature, cet algorithme est souvent appelé l'algorithme d'Audsley. Cet algorithme sera présenté dans le §5.3.2.

Des résultats plus récents concernent l'optimalité des systèmes offset free. Goossens et Devillers [67, 65] montrent l'intérêt des systèmes offset free et la non-optimalité de l'allocation RM quand les systèmes offset free sont considérés [67]. Bien qu'il y ait un nombre infini de situations asynchrones pour un ensemble de tâches, Goossens [65] énonce une allocation d'offsets optimale en ne considérant que les allocations d'offsets non équivalentes. Cependant le nombre d'allocations reste exponentiel, Goossens [65] fournit une heuristique, appelée *dissimilar offset*, performante avec une complexité faible (une seule allocation est prise en compte). Les résultats de [65] seront rappelés dans la section 5.2.

Des analyses de faisabilité bien connues existent pour les systèmes mono-processeurs dans le cas synchrones (cf., Lehoczky [68] ou Tindell et *al.* [69]). Des analyses existent également pour les systèmes mono-processeurs asynchrones mais nous ne donnons pas de détails ici car ce chapitre ne traite pas de l'analyse de faisabilité dans le cas mono-processeur. Néanmoins, on peut noter que, par exemple (cf. Leung et Whitehead [12]), $[0, O_{\max} + 2P)$, où P est le ppcm des périodes et $O_{\max} = \max_j(O_j)$, est un intervalle d'étude pour la faisabilité (cf [67, 70] pour plus de détails concernant l'intervalle minimal d'étude à considérer pour tester la faisabilité). Il n'existe pas, à notre connaissance, d'analyse d'ordonnancement des offset free systèmes distribués (cf. section 5.6 pour une définition précise d'un offset free système distribué).

Contributions. Dans ce chapitre, nous montrons comment utiliser l'algorithme d'Audsley pour réduire la complexité de l'allocation des offsets dans le cas préemptif en réduisant le nombre de tâches à considérer dans l'allocation. Nous verrons dans la section 5.3 que la méthode proposée ne peut pas être utilisée dans le cas non-préemptif. Selon nos tests, l'utilisation de l'algorithme d'Audsley permet dans le cas préemptif, pour un nombre significatif de systèmes (plus de 38% dans nos expérimentations), de réduire le nombre d'allocations d'offset différentes à considérer d'au moins 50%. Malgré cette réduction, l'allocation optimal des offsets ne peut pas toujours être utilisée du fait de sa complexité exponentielle. Nous proposons donc de nouvelles heuristiques (valables pour l'ordonnancement préemptif et non-préemptif) d'allocation d'offsets qui améliorent celle proposée dans [65]. Par exemple, quand on considère des ensembles de 9 tâches, 37.9% des ensembles non-faisables dans le cas synchrone sont faisables avec au moins une des heuristiques tandis que seulement 26.3% sont faisables avec l'allocation proposée dans [65]. L'efficacité

et l'utilité de ces heuristiques sont évaluées dans le cas préemptif et dans le cas distribué non-préemptif. L'étude du cas distribué a été réalisée dans le cadre d'un projet industriel avec PSA. Dans ce projet, nous avons développé une nouvelle analyse de temps de réponse très performante et réalisé un logiciel (prototype) de calcul de temps de réponse et d'allocation d'offset. Ce prototype a donné naissance au logiciel NetCar-Analyzer enregistré à l'APP sous le numéro IDDN FR 001 300015 000 S P 2007 000 10000 dont je suis co-auteur à 33%, cf. annexe 5 section C.1.

Organisation. La section 5.2 rappelle les résultats de [65] utiles à la compréhension de nos contributions. La section 5.3 montre comment l'usage de l'algorithme d'Audsley permet de réduire la complexité de l'allocation des offsets dans le cas préemptif. Ensuite, nous présentons nos nouvelles heuristiques dans la section 5.4. Nous évaluons leur efficacité dans la section 5.5 dans le cas mono-processeur préemptif. La section 5.6 étudie dans quelle mesure l'usage d'offset (et donc de nos heuristiques) permet de minimiser les temps de réponse dans les réseaux de communication (cas distribué non-préemptif). Dans la suite, nous employons le terme tâche pour désigner une tâche ou un flux excepter dans les sections spécifiques aux tâches, cf. sections 5.3 et 5.5 et aux flux, cf. section 5.6.

5.2 Allocation des offsets : rappel

Dans cette section, nous résumons les principaux résultats sur l'ordonnancement des systèmes offset free ("à décalages libres"). Nous résumons plus particulièrement l'approche développée par Goossens [65].

5.2.1 Allocation optimale d'offset

Nous définissons ce qu'est une allocation d'offset optimale et présentons celle proposée par Goossens [65].

Supposons que la priorité des tâches est déjà fixée à l'aide d'une règle d'allocation de priorité spécifique $\mathcal{P}^{rule}$, qui peut être par exemple Deadline Monotonic. Comme nous considérons un ordonnanceur à priorité fixe, l'ordonnanceur donne, à chaque instant, le CPU à l'instance de la tâche ayant la plus grande priorité (s'il y a au moins une tâche). Supposons que le système n'est pas ordonnançable dans le cas synchrone avec $\mathcal{P}^{rule}$. Nous voulons trouver une situation asynchrone dans laquelle le système est ordonnançable. Pour faciliter la compréhension, nous dirons qu'une allocation de priorités (ou une allocation d'offsets) est faisable (ou ordonnançable) quand les tâches du système dans la configuration obtenue respectent leur échéance. Dans la suite, on distingue deux sortes d'optimalité.

Définition 7 *Une règle d'allocation de priorité $\mathcal{P}^{rule}$ est optimale dans le cas asynchrone, si quand une allocation de priorité ordonnançable existe dans une situation asynchrone, $\mathcal{P}^{rule}$ fournit une allocation de priorité ordonnançable dans la même situation.*

Définition 8 *Une règle d'allocation d'offset $\mathcal{O}$ est optimale sous une règle d'allocation de priorité $\mathcal{P}^{rule}$, si quand une allocation d'offset faisable existe avec $\mathcal{P}^{rule}$, $\mathcal{O}$ fournit une situation asynchrone faisable avec la même allocation de priorité $\mathcal{P}^{rule}$.*

L'allocation optimale d'offset décrite dans [65] est résumée dans cette section. L'idée principale est de tester toutes les situations différentes “non-équivalentes” d'un ensemble de tâches. Toutes les combinaisons

différentes peuvent être trouvées en restreignant les offsets tel que $O_1 = 0$ et $\forall i \in [2, n] \mid O_i \in [0, T_i)$. Par conséquent, le nombre de combinaisons est borné supérieurement par $\prod_{i=2}^{n} T_i = \mathcal{O}((\max_{2 \leq j \leq n} T_j)^{n-1})$.

Afin de réduire encore le nombre d'allocations d'offset à tester, il est possible de considérer uniquement les allocations d'offset produisant des situations asynchrones différentes. Deux situations asynchrones sont dites *équivalentes*, si l'ordonnancement dans ces deux situations a le même comportement périodique. En effet, l'ordonnancement devient périodique avec une période $P = \text{ppcm}\{T_1, .., T_n\}$. Ce comportement périodique dépend seulement des phases relatives des instances des tâches, *i.e.*, du tuple $(O_1 \pmod{T_1}, O_2 \pmod{T_2}, \ldots, O_n \pmod{T_n})$. Ce tuple caractérise les décalages relatifs des instances des divers tâches [67].

Pour deux tâches τ_1 et τ_2, deux choix ($O_2 = O_1 + v_1$ et $O_2 = O_1 + v_2$) sont dits équivalents s'ils définissent une phase relative identique :

$$\begin{aligned} &\exists k_1, k_2 \in \mathbb{N} : (O_1 + v_1 + k_1 \cdot T_2) \bmod T_1 \\ &\qquad = \\ &\qquad (O_1 + v_2 + k_2 \cdot T_2) \bmod T_1 \ , \end{aligned} \tag{5.1}$$

qui est équivalent à :

$$v_1 \equiv v_2 (\text{mod } \text{pgcd}\{T_1, T_2\}). \tag{5.2}$$

D'après les équations 5.1 et 5.2 seul les valeurs $0, 1, ..., \text{pgcd}\{T_1, T_2\} - 1$ doivent être prises en compte et sont des choix non-équivalents pour O_1 et O_2.

L'algorithme d'allocation optimal d'offset construit de manière itérative des situations afin d'explorer toutes les situations asynchrones non-équivalentes possibles. Tout d'abord, il fixe les choix non-équivalents pour O_2 (l'offset O_1 est arbitrairement fixé à 0) en considérant pour O_2 toutes les valeurs entières dans l'intervalle $[0, \text{pgcd}\{T_1, T_2\})$. Ensuite, en faisant l'hypothèse qu'à chaque étape les offsets $O_1, O_2, ...O_{i-1}$ sont fixés, il considère pour l'offset O_i l'intervalle $[0, \text{pgcd}\{T_i, \text{ppcm}(T_1, ..., T_{i-1})\})$ (les instances du sous-ensemble de tâches $\{\tau_1, ..., \tau_{i-1}\}$ ayant une "période" de $\text{ppcm}(T_1, ..., T_{i-1})$).

5.2.2 Allocation "Dissimilar offset"

La méthode de Goossens [65], présentée dans la section 5.2.1, permet de considérer uniquement les allocations d'offset non-équivalentes, *i.e.*, de considérer $\frac{\prod_{i=2}^{n} T_i}{P}$ allocations au lieu de $\prod_{i=2}^{n} T_i$. Malgré cette réduction significative, le nombre d'allocations considérées par l'algorithme optimal reste exponentiel. Goossens [65] définit donc une heuristique qui donne une seule allocation d'offset pour un ensemble de tâches.

L'idée basique de l'heuristique est d'écarter le plus possible les offsets des tâches pour éviter que leurs instances soient en conflit pour l'usage du CPU. Précisément, une mesure est introduite pour estimer la proximité d'une allocation avec le cas synchrone. L'algorithme dissimlar offset alloue les offsets des tâches périodiques afin de maximiser cette mesure. Pour un couple de tâches, cette mesure est définie comme le plus court intervalle qui contient au moins une instance de chaque tâche.

Cette technique considère la distance (minimale) entre deux instances des tâches τ_i et τ_j. Le calcul de cette distance est réalisée selon le théorème 1.

Théorème 1 *[65] Soit $r \in [0, \text{pgcd}\{T_i, T_j\})$. Si $O_i = O_j + r$ (ou $O_j = O_i + r$), la distance minimale*

entre une instance de τ_i et une instance de τ_j est égale à $\min\{r, \text{pgcd}\{T_i, T_j\} - r\}$.

On déduit du théorème 1 que la distance minimale entre deux instances de τ_i et τ_j est bornée par $\left\lfloor \frac{\text{pgcd}\{T_i,T_j\}}{2} \right\rfloor$ et correspond à l'allocation d'offset $O_i = O_j + \left\lfloor \frac{\text{pgcd}\{T_i,T_j\}}{2} \right\rfloor$ (ou $O_j = O_i + \left\lfloor \frac{\text{pgcd}\{T_i,T_j\}}{2} \right\rfloor$). Dans ce cas, r est égal à $\left\lfloor \frac{\text{pgcd}\{T_i,T_j\}}{2} \right\rfloor$ ou $\left\lceil \frac{\text{pgcd}\{T_i,T_j\}}{2} \right\rceil$.

L'algorithme d'allocation dissimilar offset fixe les offsets des tâches périodiques dans le but de maximiser la mesure définie précédemment. L'algorithme trie les couples de tâches (τ_i, τ_j) par valeur décroissante de $\text{gcd}\{T_i, T_j\}$. Ensuite, il assigne itérativement les offsets O_i et O_j des couples triés (τ_i, τ_j) pour obtenir la plus grande distance minimale (i.e., $r = \left\lfloor \frac{\text{pgcd}\{T_i,T_j\}}{2} \right\rfloor$). Durant cette allocation, trois cas peuvent se produire :

1. quand O_i et O_j ne sont pas encore alloués, un offset aléatoire est choisi pour O_i et $O_j = O_i + \left\lfloor \frac{\text{pgcd}\{T_i,T_j\}}{2} \right\rfloor$,
2. quand O_i (resp. O_j) est fixé et O_j (resp. O_i) ne l'est pas, $O_j = O_i + \left\lfloor \frac{\text{pgcd}\{T_i,T_j\}}{2} \right\rfloor$ (resp. $O_i = O_j + \left\lfloor \frac{\text{pgcd}\{T_i,T_j\}}{2} \right\rfloor$),
3. quand O_j et O_i sont déjà choisis, il n'y a rien à faire.

La complexité temporelle maximale est $\mathcal{O}(n^2 \cdot (\log T^{\max} + \log n^2)$ où $T^{\max} \stackrel{\text{def}}{=} \max_{1 \leq k \leq n}(T_k)$.

5.3 Réduire la complexité dans le cas préemptif

Dans cette section, nous proposons une technique, reposant sur l'allocation de priorité d'Audsley, pour réduire de manière significative l'espace de recherche dans le cas préemptif. Nous présentons avant tout des propriétés de l'ordonnancement reposant sur les priorités fixes et l'algorithme d'Audsley [14] utiles à la compréhension de notre technique.

5.3.1 Propriétés de l'ordonnancement à priorité

Sous un ordonnancement reposant sur les priorités (préemptif ou non-préemptif), les instances prêtes à s'exécuter avec une priorité supérieure à τ_i retardent la fin d'exécution des instances de τ_i :

Lemme 3 *Le temps de réponse de τ_i dépend de l'ensemble de tâches de priorité supérieure, mais est indépendant de l'ordre relatif des priorités parmi ces tâches de priorité supérieure.*

De plus, lorsque l'ordonnancement est préemptif, les instances des tâches de plus basse priorité n'interfèrent pas avec l'exécution des instances de τ_i et donc ne retardent pas leur fin d'exécution :

Lemme 4 *Sous l'ordonnancement préemptif à priorité fixe, le temps de réponse de τ_i est indépendant des tâches de priorité inférieure.*

Lehoczky et *al.* [71] ont montré que le pire temps de réponse sous un ordonnancement préemptif à priorité fixe peut être trouvé dans le scénario suivant.

Lemme 5 *[71] Le pire temps de réponse sous un ordonnancement préemptif à priorité fixe arrive dans un scénario dans lequel toutes les tâches sont activées à l'instant* 0 *(i.e., situation synchrone).*

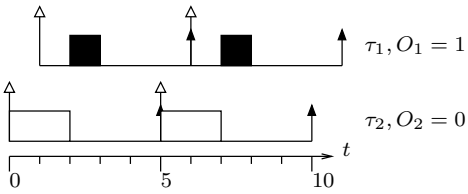


FIG. 5.1 – Ordonnancement non-préemptif de l'ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2\}$ où τ_1 est plus prioritaire que τ_2 et où $\tau_1 = (C_1 = 1, T_1 = 5, \overline{D}_1 = 5, O_1)$ et $\tau_2 = (2, 5, 5, O_2)$ avec l'allocation d'offsets : $O_1 = 1$, $O_2 = 0$.

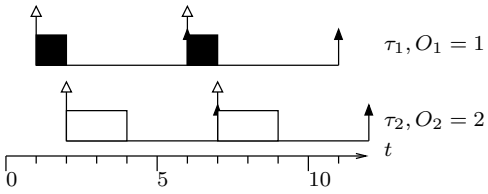


FIG. 5.2 – Ordonnancement non-préemptif de l'ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2\}$ où τ_1 est plus prioritaire que τ_2 et où $\tau_1 = (C_1 = 1, T_1 = 5, \overline{D}_1 = 5, O_1)$ et $\tau_2 = (2, 5, 5, O_2)$ avec l'allocation d'offsets : $O_1 = 1$, $O_2 = 2$.

C'est pourquoi, comme nous l'avons déjà rappelé dans l'introduction, le cas synchrone est très pessimiste en ce qui concerne la faisabilité, car le cas synchrone est le pire en terme de faisabilité pour l'ordonnancement préemptif, dans le sens où, si un système est ordonnançable dans le cas synchrone il est également ordonnançable dans toutes les situations asynchrones.

Par ailleur, notons que le lemme 4 n'est pas valide sous un ordonnancement non-préemptif. En effet, le temps de réponse d'une tâche dépend des tâches de priorité inférieure (ex. : de leur offset). Par exemple, les figures 5.1 et 5.2 présentent l'ordonnancement non-préemptif de l'ensemble $\mathcal{T} = \{\tau_1, \tau_2\}$ où τ_1 est plus prioritaire que τ_2 et où $\tau_1 = (C_1 = 1, T_1 = 5, \overline{D}_1 = 5, O_1)$ et $\tau_2 = (2, 5, 5, O_2)$ avec les allocations d'offsets : $O_1 = 1$, $O_2 = 0$ (figure 5.1) et $O_1 = 1$, $O_2 = 2$ (figure 5.2).

Comme on peut le constater sur les figures 5.1 et 5.2, le temps de réponse de τ_1 est 2 avec la première allocation d'offsets (figure 5.1) et est 2 avec la seconde (figure 5.2). Le temps de réponse de la tâche la plus prioritaire τ_1 est donc dépendante de l'offset de la tâche τ_2 moins prioritaire.

5.3.2 Algorithme d'Audsley avec des offsets fixés

L'algorithme d'Audsley [14] réalise une allocation de priorité statique qui est optimale dans le cas asynchrone (dans le cas préemptif et non-préemptif) selon la définition 7.

Une allocation de priorité est définie par un ensemble $\mathcal{P}$ de n couples (τ_j, p_j), un pour chaque tâche, où p_j est la priorité de la tâche τ_j dans l'allocation de priorité $\mathcal{P}$. La fonction d'allocation $\mathcal{P}(i)$ donne la tâche τ_j allouer au niveau de priorité i en utilisant la convention : plus petit le niveau de priorité, plus grande est la priorité.

L'algorithme d'Audsley, dont le pseudo-code est donné dans l'algorithme 1, considère au plus $\mathcal{O}(n^2)$ allocations de priorités distinctes. Les priorités sont allouées de la plus faible priorité à la plus importante,

Entrée : ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2..., \tau_n\}$
Sortie : ensemble de tâches dont les priorités ne sont pas allouées
Données : i : niveau de priorité à allouer
$\mathcal{R}_i$: ensemble de tâches n'ayant pas encore de priorité
$\mathcal{P}$: allocation partielle de priorité

```
1  R_i = T;
2  P = ∅;
3  pour i = n à 1 faire
4  |  essayer d'allouer la priorité i :
5  |  chercher une tâche τ_j faisable au niveau de priorité i dans R_i
6  |  si pas de tâche faisable au niveau de priorité i alors
7  |  |  échec, retourner l'ensemble de tâches dont les priorités ne sont pas
   |  |  allouées :
8  |  |  retourner R_i;
9  |  sinon
10 |  |  soit τ_j une tâche faisable à la priorité i ;
11 |  |  allouer la priorité i :
12 |  |  P = P ∪ (τ_j, i);
13 |  |  enlever τ_j de R_i :
14 |  |  R_i = R_i \ τ_j;
15 |  finsi
16 finpour
17 succès, une allocation faisable a été trouvée :
18 retourner ∅;
```

Algorithme 1 : Algorithme optimal d'Audsley réalisant une allocation de priorités fixes.

cf. algorithme 1 ligne 3. A chaque niveau de priorité, l'algorithme cherche une tâche faisable. Il commence par chercher une tâche τ_i dans $\mathcal{T}$ faisable à la plus faible priorité, *i.e.*, essaye d'allouer le niveau de priorité n, cf. algorithme 1 lignes 4 et 5. Ensuite, à chaque niveau de priorité i, l'algorithme cherche une tâche faisable dans l'ensemble $\mathcal{R}_i$ composé de l'ensemble des tâches n'ayant pas encore de priorité, cf. algorithme 1 lignes 4 et 5. Après l'exécution de l'algorithme d'Audsley, deux cas peuvent se produire :

1. (cf. lignes 17 et 18) L'allocation de priorité réalisée par l'algorithme d'Audsley mène à un système ordonnançable (i.e., l'allocation a réussi) : l'ensemble de tâches $\mathcal{T}$ est ordonnançable avec l'allocation de priorité donnée par $\mathcal{P}$.
2. (cf. lignes 7 et 8) Sinon, l'algorithme d'Audsley ne trouve pas de tâche faisable au niveau de priorité i où $i \in [1, n]$ (*i.e.*, l'allocation de priorité échoue). Notez que quand l'algorithme d'Audsley est utilisé les instances de l'ensemble de tâches $\{\mathcal{P}(i+1), \mathcal{P}(i+2), ..., \mathcal{P}(n)\}$ respectent leur échéance. En effet, l'ordonnançabilité d'une tâche à un niveau de priorité, avec un ordonnancement à priorité fixe, ne dépend pas de l'ordre de priorité parmi l'ensemble des tâches de plus hautes priorités [14, 66] (cf., lemme 3).

5.3.3 Utilisation de Audsley pour réduire l'espace de recherche sous un ordonnancement préemptif

Dans cette section, nous expliquons comment allouer ensemble les priorités et les offsets en réduisant le nombre d'allocations d'offsets à considèrer sous un ordonnancement préemptif. La figure 5.3 présente

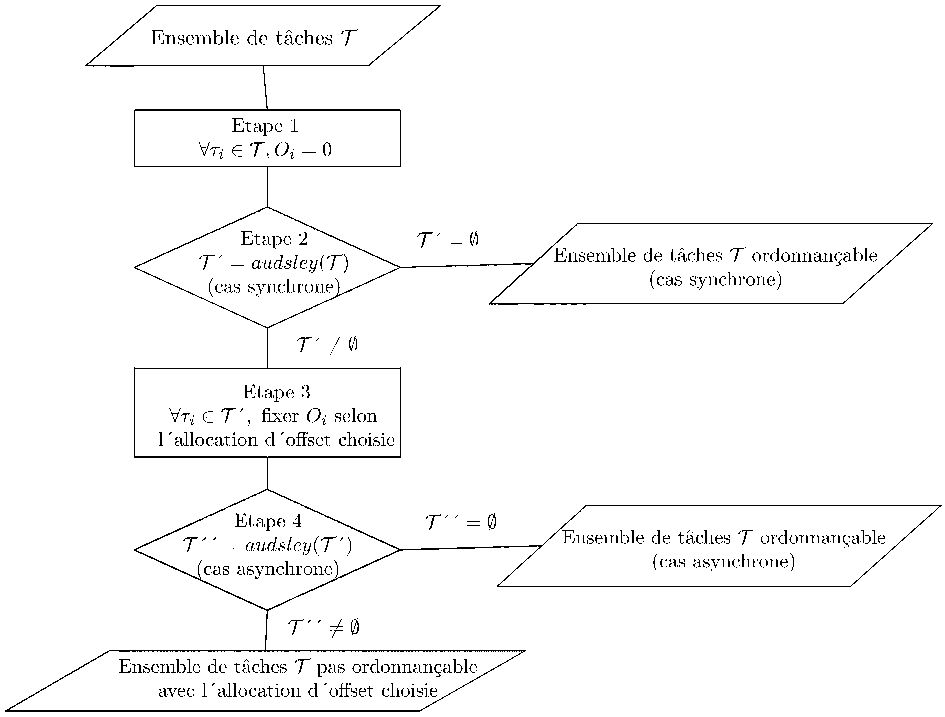


FIG. 5.3 – Algorithme d'allocation d'offsets et de priorités où $\mathcal{T}' = audsley(\mathcal{T})$ signifie que l'algorithme d'Audsley est appliqué à l'ensemble $\mathcal{T}$ et $\mathcal{T}'$ est l'ensemble de tâches dont les priorités n'ont pas été allouées par Audsley. Ici, la faisabilité (étape 2 et 4) est évaluée en simulant le système sur l'intervalle d'étude $[0, O_{\max} + 2P)$ proposé par Leung et Whitehead [12],

notre approche dans une forme pseudo-algorithmique.

Dans un premier temps, nous initialisons la valeur des offsets afin d'être dans une situation *synchrone* ($\forall i, O_i = 0$). Ensuite, nous utilisons l'algorithme d'Audsley, cf. §5.3.2 précédent, pour allouer les priorités (dans le cas synchrone). Si l'algorithme d'Audsley réussit à allouer toutes les priorités (cas 1, section 5.3.2), le système est ordonnançable dans le cas synchrone. Sinon, l'algorithme d'Audsley a échoué dans le cas synchrone (cas 2, section 5.3.2), une situation asynchrone ordonnançable doit être cherchée. Pour tester la faisabilité (étape 2 et 4), le système est étudié sur l'intervalle d'étude $[0, O_{\max} + 2P)$ proposé par Leung et Whitehead [12],

En conséquence, nous utilisons à cette étape une règle pour choisir les offsets — pour le sous-ensemble de tâches $\mathcal{T}'$ n'ayant pas encore de priorité allouée par Audsley : $\mathcal{T}' \stackrel{\text{def}}{=} \mathcal{T} \backslash \{\mathcal{P}(i+1), \mathcal{P}(i+2), ..., \mathcal{P}(n)\}$— et nous utilisons ensuite Audsley une seconde fois sur le sous-ensemble $\mathcal{T}'$ (pas sur l'ensemble de tâches d'origine) pour fixer les priorités de 1 à i. Nous pouvons uniquement considérer dans l'allocation des offsets et des priorités de 1 à i les tâches de l'ensemble $\mathcal{T}'$ car :

1. les tâches de priorité inférieure (*i.e.*, tâches du sous-ensemble de tâches $\{\mathcal{P}(i+1), \mathcal{P}(i+2), ..., \mathcal{P}(n)\}$) respectent leurs contraintes de temps quels que soit les offsets et l'ordre de priorité parmi les tâches de priorité supérieure. En effet, les tâches dans $\{\mathcal{P}(i+1), \mathcal{P}(i+2), ..., \mathcal{P}(n)\}$ sont faisables aux plus bas niveaux de priorités dans le cas synchrone. Comme le cas synchrone est le pire pour la faisabilité en ce qui concerne l'ordonnancement préemptif (cf. lemme 5) et que le temps de réponse d'une tâche dépend de l'ensemble de tâches de priorité supérieure, mais est indépendant de l'ordre relatif des priorités parmi ces tâches (cf. lemme 3), ces tâches restent faisables aux plus bas niveaux de priorité dans n'importe quelle situation asynchrone et quel que soit l'ordre de priorité parmi les tâches de plus haute priorité.
2. sous l'ordonnancement préemptif à priorité fixe, le temps de réponse de τ_i est indépendant des tâches de priorité inférieure (cf. lemme 4), c'est-à-dire que le temps de réponse des tâches de $\mathcal{T}'$ ne dépend pas des offsets des tâches de l'ensemble $\{\mathcal{P}(i+1), \mathcal{P}(i+2), ..., \mathcal{P}(n)\}$.

Avec cette méthode le nombre de tâches à considérer dans l'allocation d'offset est plus petit (plus de détails seront donnés en section 5.5.1). Comme la complexité en temps de l'allocation d'offset dépend du nombre de tâches et de leur période, la complexité est donc réduite.

Nous voulons faire remarquer que nous ne pouvons pas utiliser cette technique avec l'ordonnancement non-préemptif. La principale raison est due au fait que sous l'ordonnancement non-préemptif, comme nous l'avons montré dans le §5.3.1, le temps de réponse d'une tâche est dépendant des tâches de priorité inférieure. Donc, si Audsley échoue (dans le cas synchrone), il faut également considèrer les tâches de l'ensemble $\{\mathcal{P}(i+1), \mathcal{P}(i+2), ..., \mathcal{P}(n)\}$ dans l'allocation d'offsets car la valeur des offsets de ces tâches a un impacte sur le temps de réponse des tâches de priorité supérieure (*i.e.*, tâches de l'ensemble $\mathcal{T}'$).

5.4 Nouvelles heuristiques d'allocation d'offset

Nous proposons plusieurs heuristiques, qui fournissent des allocations alternatives à l'allocation dissimlar offset. Ces heuristiques sont efficaces dans le cas préemptif et non-préemptif, car elles fixent les offsets de façon à éviter les conflits indépendamment du fait que l'ordonnancement soit préemptif ou non-préemptif.

Le schéma de fonctionnement de ces nouvelles heuristiques est très similaire à celui du dissimilar offset : les couples de tâches sont classés selon un critère, les offsets des tâches sont ensuite choisis en considérant les couples du début de la liste à sa fin. Les nouvelles heuristiques proposent des allocations différentes de celle du dissimilar offset, car elles ne considèrent pas seulement la distance minimale entre les tâches. Par exemple, quelques unes essayent de séparer les tâches ayant le plus grand taux d'utilisation (*i.e.*, $\frac{C_k}{T_k}$). Nous proposons quatre nouvelles allocations d'offset qui prennent en compte des caractéristiques de tâche autres que la distance minimale entre les couples de tâches. Nos quatre heuristiques considèrent les couples (τ_k, τ_i) par valeur décroissante de :

1. $\left(\frac{C_k}{T_k} + \frac{C_i}{T_i}\right) \cdot \text{pgcd}(T_k, T_i)$
2. $\max\left(\frac{C_k}{T_k}, \frac{C_i}{T_i}\right) \cdot \text{pgcd}(T_k, T_i)$
3. $\frac{C_k}{T_k} + \frac{C_i}{T_i}$
4. $-\text{pgcd}(T_k, T_i)$

Les heuristiques 1,2 et 3 considèrent le taux d'utilisation des tâches pour classer les couples de tâches. Introduire de différentes façons le taux d'utilisation dans le classement des couples génère plusieurs situations asynchrones pouvant fournir une situation ordonnançable. Dans l'heuristique 1 (resp. 2), le taux d'utilisation des couples de tâches (resp. le taux d'utilisation maximal) pondéré par le pgcd des périodes est pris en compte. L'heuristique 3 classe les (τ_k, τ_i) selon la valeur décroissante de leur taux d'utilisation.

L'heuristique 4 s'intéresse d'abord aux couples (τ_k, τ_i) pour lesquels la distance minimale entre les tâches est petite. Les (τ_k, τ_i) sont donc classés par valeur décroissante de $-\text{pgcd}(T_k, T_i)$. Les couples de tâches sont considérées dans l'ordre inverse à celui de l'algorithme dissimilar offset. Le but est de fixer les offsets des couples ayant le moins de choix possibles dans l'allocation d'offset.

Nous prenons en compte l'ensemble de ces nouvelles heuristiques. L'utilisation combinée de ces heuristiques (section 5.5.4) fournit des allocations d'offset très performantes. La complexité de ces nouvelles heuristiques est identique à celle de l'allocation dissimilar offset (i.e., $\mathcal{O}(n^2 \cdot (\log T^{\max} + \log n^2))$ car l'algorithme utilisé est le même excepté pour le critère de classement des couples.

5.5 Expérimentations : le cas mono-processeur préemptif

Dans cette section nous présentons les résultats d'expérimentation dans le cas mono-processeur préemptif concernant la faisabilité. Nous utilisons l'algorithme 5.3 défini dans la section 5.3.3. Des tests similaires pour le cas mono-processeur non-préemptif n'ont pas été réalisés. Cependant, des tests et des études de cas ont été réalisés, cf. section 5.6, dans le cadre d'un projet industriel avec PSA dans un contexte distribué non-préemptif. L'objectif de ce projet était d'étudier l'impact des offsets sur le temps de réponse des trames et de minimiser ces temps de réponse. La génération des ensembles de tâches utilisés dans les expérimentations est décrite dans l'annexe A et nous utilisons le tuple $(n, U, c_{\min}, c_{\max}, d_{\min}, d_{\max}, t_{\max})$ pour préciser les paramètres utiliser pour générer aléatoirement ces ensembles, c'est-à-dire des ensembles de n flux avec une charge globale dans l'intervalle $[U - 0.02, U + 0.02]$, et où le temps d'exécution C_i de chaque tâche τ_i est uniformément répartie dans l'intervalle $[c_{\min}, c_{\max}]$, l'échéance relative $\overline{D_i}$ est aléatoirement choisie dans l'intervalle $[d_{\min}, d_{\max}]$ et la période est bornée par $t_{\max}$.

5.5.1 Algorithme d'Audsley pour réduire la complexité

Dans ce paragraphe, nous étudions la réduction de l'espace de recherche apportée par l'utilisation de l'algorithme d'Audsley. L'amélioration est évaluée avec des ensembles de tâche générés selon le tuple $(n, 0.8, 2, 30, T_k - 0.9 \times (T_k - C_k), T_k + 0.9 \times (T_k - C_k), 200)$ où n est le nombre de tâches dans l'intervalle $[5, 17]$. Nous avons fait 14000 simulations pour chaque graphique (14 histogrammes par graphique).

Dans la figure 5.4, les histogrammes noires montrent le pourcentage d'ensembles de tâches non-faisables dans le cas synchrone pour lesquels Audsley alloue avec succès au moins la plus petite priorité. Ce pourcentage représente le nombre d'ensembles pour lesquels l'usage d'Audsley permet de réduire la complexité de l'allocation d'offset, *i.e.*, le nombre de tâches à considérer dans l'allocation d'offsets. Les histogrammes gris représentent les ensembles restants, *i.e.*, pour lesquels Audsley alloue aucune priorité et donc pour lesquels la complexité n'est pas réduite.

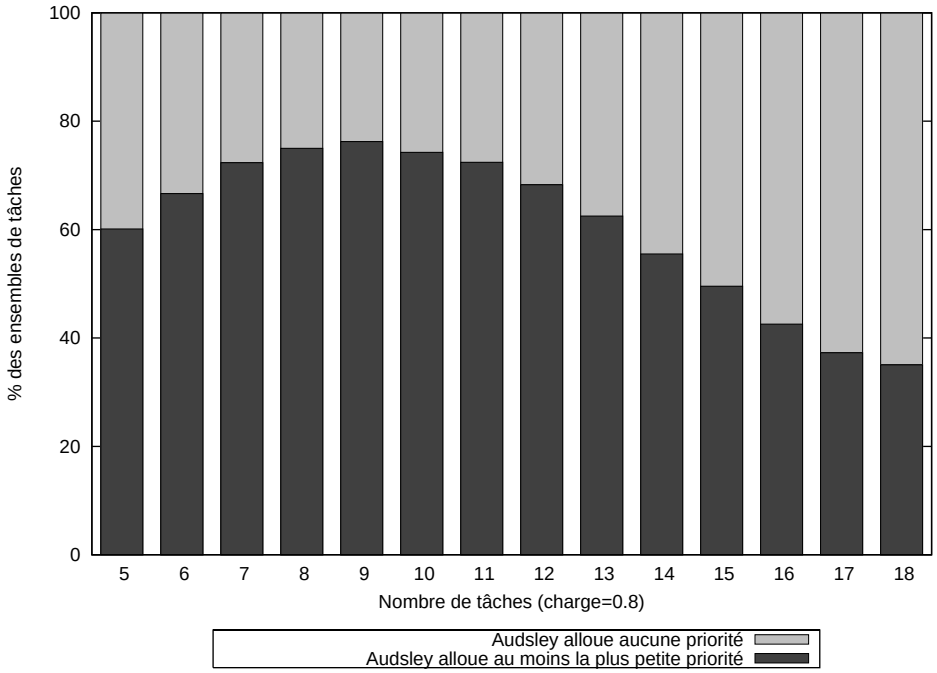


FIG. 5.4 – Pourcentage d'ensembles de tâches non-faisables dans le cas synchrone qui ont au moins une tâche faisable au plus bas niveau de priorité dans le cas synchrone, *i.e.*, ayant une priorité allouée par l'algorithme d'Audsley. Ce pourcentage représente le nombre d'ensembles pour lesquels l'usage d'Audsley permet de réduire la complexité de l'allocation d'offset.

On peut observer que au moins 38% des ensembles inclus une tâche faisable au plus bas niveau de priorité. Pour ces ensembles, Audsley (étape 2, algorithme 5.3) permet de réduire le nombre de tâches à considérer dans l'allocation d'offset. On peut également remarquer de la figure 5.4 que le pourcentage décroît avec le nombre de tâches. Ce phénomène est probablement dû à notre algorithme de génération de tâche. En effet, des restrictions sont faites sur les caractéristiques des tâches pour réduire le ppcm des périodes des tâches, dans des bornes qui permettent d'évaluer la faisabilité par simulation. Quand le nombre de tâches devient important, les tâches ont tendances à avoir les mêmes caractéristiques et donc à se comporter d'une manière similaire. C'est pourquoi, quand une tâche n'est pas faisable à la plus faible priorité, la probabilité de trouver une autre tâche faisable est plutôt faible.

Dans la figure 5.5, nous prenons en compte uniquement les ensembles de tâches non-faisables dans le cas synchrone qui ont une tâche faisable au plus bas niveau de priorité. Les histogrammes en noire montrent le pourcentage de tâches faisables aux plus bas niveaux de priorité après l'étape 2 de l'algorithme 5.3 (*i.e.*, les tâches de l'ensemble $\mathcal{T} \setminus \mathcal{T}'$). Les histogrammes en gris représentent le pourcentage de tâche τ_j, qui ne le sont pas, (*i.e.*, les tâches de l'ensemble $\mathcal{T}'$).

Comme on peut le constater sur la figure 5.5, au moins 30% des tâches sont faisables au plus bas niveau de priorité (dans le cas synchrone). Donc, moins de 70% des tâches ont besoin d'être considérées dans l'allocation d'offset.

Dans le but d'évaluer plus précisément l'efficacité de note proposition, nous étudions la réduction de l'espace de recherche apportée par l'usage de l'algorithme d'Audsley. Dans la figure 5.6, nous considérons encore seulement les ensembles de tâches ayant au moins une tâche faisable au plus bas niveau de priorité

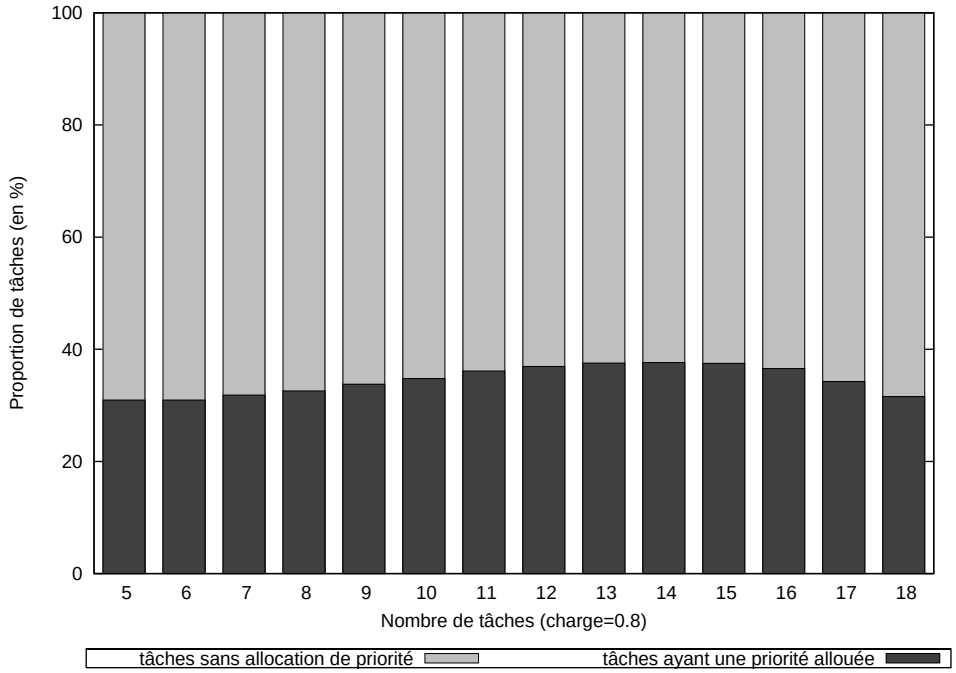


FIG. 5.5 – Pourcentage des tâches faisables au plus bas niveau de priorité, *i.e.*, ayant une priorité allouée par l'algorithme d'Audsley, dans le cas synchrone. Seul les ensembles de tâches non-faisables dans le cas synchrone qui ont une tâche faisable au plus bas niveau de priorité sont considérés, *i.e.*, ensembles pour lesquels l'usage d'Audsley permet de réduire la complexité de l'allocation d'offset.

(dans le cas synchrone). La courbe montre le pourcentage de réduction de l'espace de recherche évalué selon l'équation :

$$100 * \frac{Comp_{\text{sans}} - Comp_{\text{avec}}}{Comp_{\text{sans}}},$$

où $Comp_{\text{sans}}$ (resp. $Comp_{\text{avec}}$) est le nombre d'allocations de priorité considérées sans (resp. avec) l'usage de l'algorithme d'Audsley.

Les résultats de simulation présentés dans la figure 5.6 montrent que la réduction de l'espace de recherche est d'environ 50% quel que soit le nombre de tâches.

Nous pouvons conclure de ces expérimentations que pour un nombre significatif de systèmes (plus de 38% dans nos expérimentations) la priorité d'au moins 30% des tâches peut être allouée dans le cas synchrone à l'aide de l'algorithme d'Audsley. Cela permet de réduire l'espace de recherche de l'allocation d'offset d'environ 50%.

5.5.2 Améliorer la faisabilité avec les systèmes à offset free

Ce paragraphe montre l'intérêt concernant la faisabilité de l'allocation optimale dans les systèmes à offsets free. Les ensembles de tâche sont générés aléatoirement selon le tuple $(5, U, 2, 30, T_k - \frac{T_k - C_k}{2}, T_k, 30)$ avec U choisie dans l'intervalle $[0.73, 0.95]$. Nous réalisons 6000 simulations pour chaque graphique (6

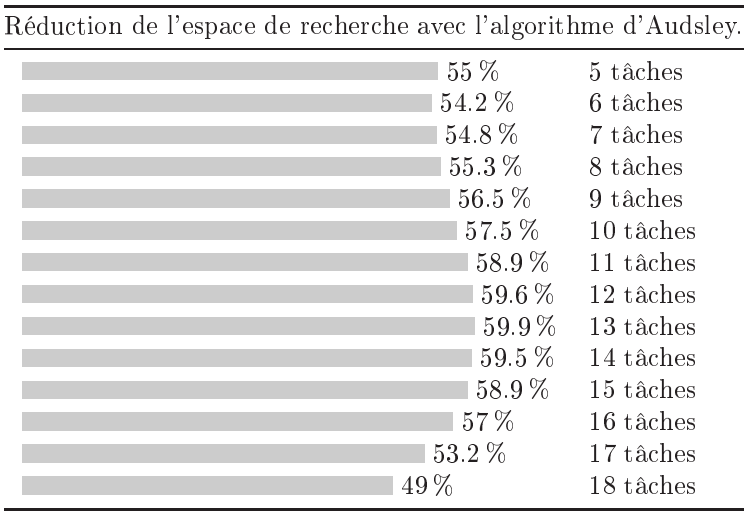


FIG. 5.6 – Réduction de l'espace de recherche avec l'algorithme d'Audsley en fonction du nombre de tâches (charge=0.8). Seul les ensembles de tâches non-faisables dans le cas synchrone qui ont une tâche faisable au plus bas niveau de priorité sont considérés, *i.e.*, ensembles pour lesquels l'usage d'Audsley permet de réduire la complexité de l'allocation d'offset.

points par graphique). Remarquons que la complexité temporelle de la règle d'allocation optimale utilisée dans ces expérimentations est élevée. Trouver si un ensemble est ordonnançable ou pas peut donc demander de très long calcul (car nous avons à considérer—dans le pire des cas—toutes les allocations d'offset qui ne sont pas équivalentes). Pour cette raison, nous réduisons fortement le nombre de tâches n et la valeur maximale des périodes dans nos simulations. Ainsi, nous diminuons le nombre d'allocations d'offset n'étant pas équivalentes et la durée du test de faisabilité qui dépend de ppcm$\{T_1, .., T_n\}$.

On évalue maintenant le pourcentage de systèmes n'étant pas faisables dans le cas synchrone qui le deviennent dans une situation asynchrone (*i.e.*, faisables en utilisant l'allocation optimale). Une fois encore, nous utilisons l'algorithme 5.3 pour déterminer ces pourcentages.

La figure 5.7 présente le pourcentage de système n'étant pas faisables dans le cas synchrone (courbe pointillé) et le pourcentage de ceux faisables uniquement dans une situation asynchrone (courbe continue). De la figure 5.7, on remarque que le pourcentage de tâche n'étant pas faisable dans le cas synchrone augmente avec la charge. Cette tendance confirme l'intuition qu'il est plus difficile de trouver un système faisable quand la charge est élevée. De plus, le pourcentage de système faisable dans un cas asynchrone augmente avec la charge (jusqu'à 18%) jusqu'à une charge de 0.87 et commence ensuite à diminuer. Intuitivement, il est clair que l'ensemble des tâches a tendance à ne pas être faisable quelle que soit leur allocation d'offset quand la charge devient trop élevée.

5.5.3 Utilisation combinée des heuristiques : comparaison avec l'optimal

La figure 5.8 montre le pourcentage d'ensemble de tâches faisables dans un cas asynchrone particulier (non équivalent au cas synchrone) qui reste faisable avec la règle d'allocation d'offset dissimilar offset (courbe en pointillée) et avec au moins une de nos nouvelles heuristiques (courbe continue).

Comme on peut le constater sur la figure 5.8, les heuristiques d'allocation trouvent une situation asynchrone faisable pour au moins 51% et jusqu'à 95% des ensembles de tâches dans lequel une telle

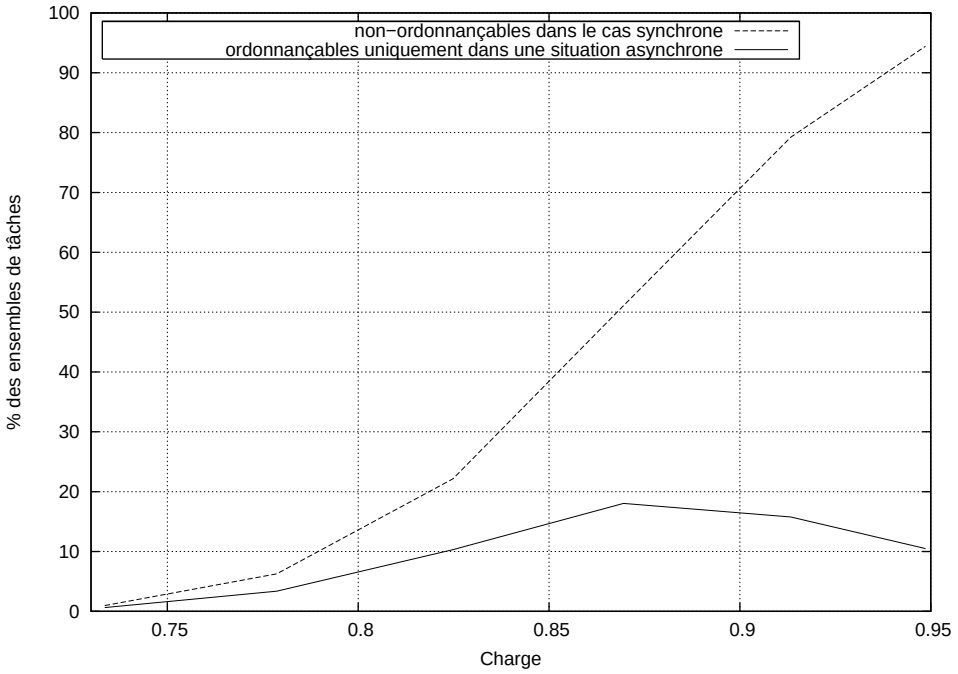


FIG. 5.7 – Pourcentage des systèmes n'étant pas faisables dans le cas synchrone (courbe pointillée) et faisables dans une situation asynchrone (courbe continue). La charge CPU varie de 0.7 à 0.95 .

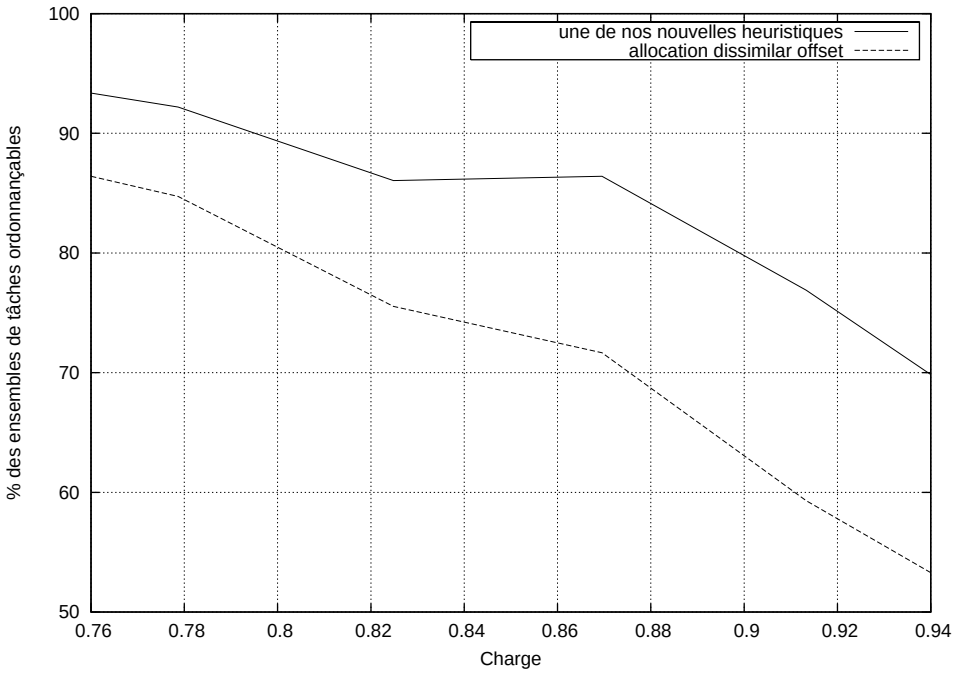


FIG. 5.8 – Pourcentage d'ensembles de tâches faisables avec la règle d'allocation d'offset dissimilar offset (courbe en pointillée) et avec au moins une de nos nouvelles heuristiques (courbe continue). Les ensembles de tâches considérés sont faisables dans un cas asynchrone particulier mais ne sont pas faisables dans le cas synchrone. La charge CPU varie de 0.76 à 0.94 .

situation existe. Les chances de trouver une situation faisable diminuent logiquement avec la charge. En effet, quand la charge est faible, il y a beaucoup de situations asynchrones faisables. Inversement, quand la charge augmente, il y a peu de ces situations.

L'utilisation combinée de ces heuristiques nous permettent de trouver un pourcentage important de situation asynchrone faisable. Il est évident, selon la figure 5.8, que l'usage combiné de nos heuristiques surpasse l'allocation dissimilar offset.

5.5.4 Performances relatives des heuristiques

Dans cette section, l'amélioration apportée par nos heuristiques est examinée plus en détails. Les ensembles de tâches sont générés selon le tuple $(n, U, 2, 30, T_k - \frac{T_k - C_k}{2}, T_k, 30)$ avec U choisi dans l'ensemble $\{0.8, 0.9\}$ et n dans l'intervalle $[5, 11]$. Nous réalisons 7000 simulations pour chaque graphique (7 points par graphique).

L'allocation des offsets et des priorités des tâches est réalisée à l'aide de l'algorithme 5.3 de la section 5.3.3. A l'étape 4, les situations asynchrones correspondes aux allocations d'offset produites par l'allocation dissimilar offset et par nos nouvelles heuristiques.

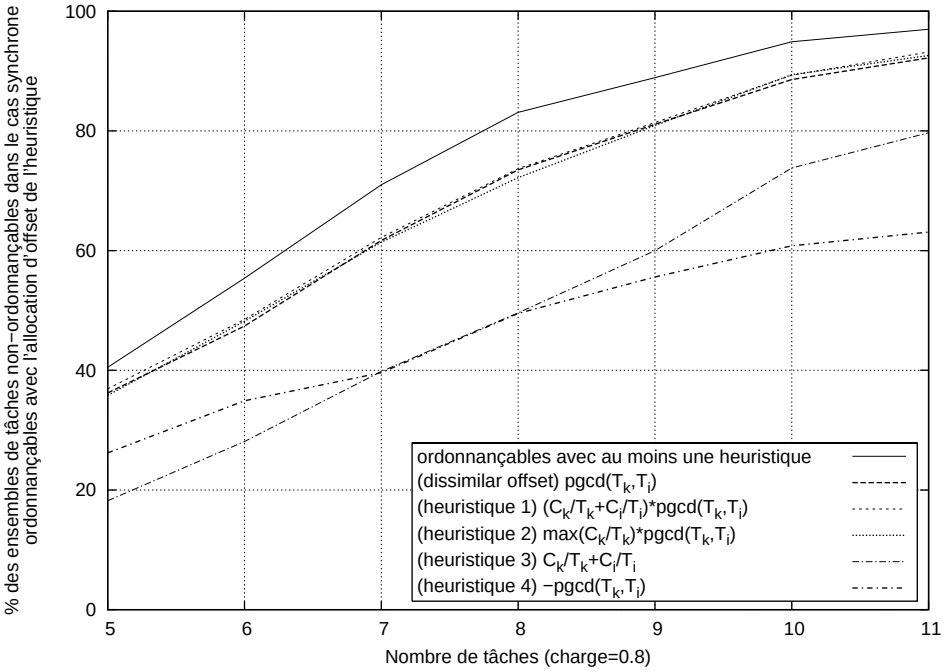


FIG. 5.9 – Pourcentage des ensembles de tâches n'étant pas faisables dans le cas synchrone qui le deviennent avec les différentes heuristiques d'allocation d'offsets (charge CPU de 80%).

Les figures 5.9 et 5.10 montrent le pourcentage des ensembles de tâches n'étant pas faisables dans la cas synchrone qui le deviennent dans la situation asynchrone produite par chacune des heuristiques. Les expérimentations sont faites avec une charge de 0.8 dans la figure 5.9 et de 0.9 dans la figure 5.10. A partir de ces figures, on constate que les heuristiques d'allocation d'offset améliorent significativement la faisabilité comparée au cas synchrone. Par exemple, dans la figure 5.9, le pourcentage de tâche faisable dans une situation asynchrone produit par une heuristique est au moins égal à 40.5% et atteint 97%.

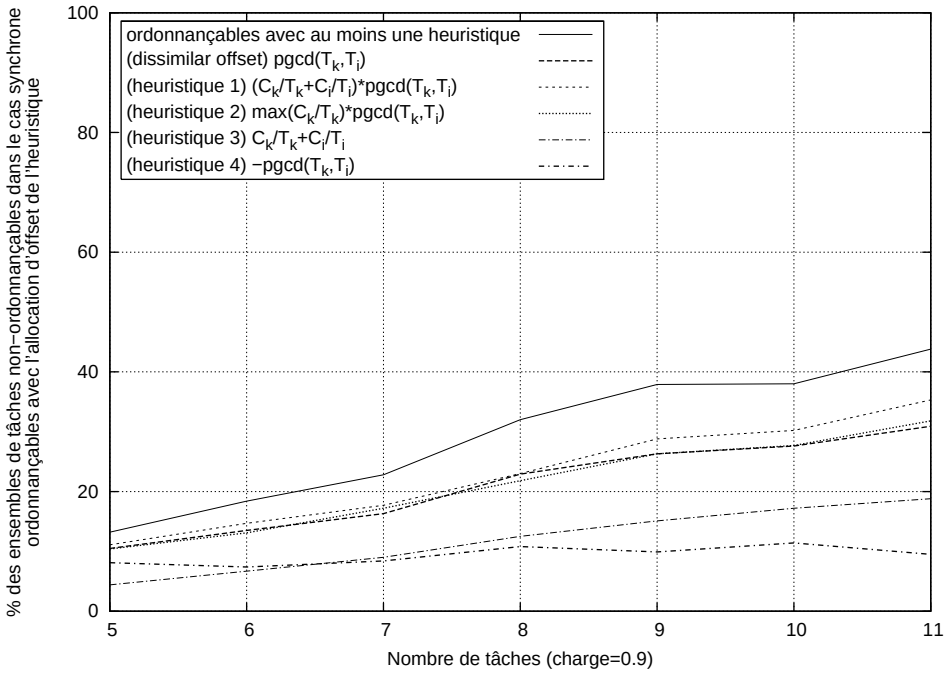


FIG. 5.10 – Pourcentage des ensembles de tâches n'étant pas faisables dans le cas synchrone qui le deviennent avec les différentes heuristiques d'allocation d'offsets (charge CPU de 90%).

L'amélioration augmente avec le nombre de tâches. Par exemple, dans la figure 5.9, le pourcentage de tâche faisable dans une situation asynchrone est égal à 71% pour 7 tâches tandis qu'il est de 88.9% pour 9 tâches. Cette tendance peut être expliquée de manière intuitive par le fait que plus le nombre de tâches est important, plus le degré de liberté pour fixer les offsets est grand et donc, plus on peut se placer "loin" du cas asynchrone.

On peut également observer que, logiquement, le pourcentage de système faisable dans une situation asynchrone décroît fortement quand la charge est élevée. Par exemple, ce pourcentage pour des ensembles de 8 tâches est de 83.1% avec une charge de 0.8 et de 32% pour une charge de 0.9 (figure 5.10).

Les différentes heuristiques peuvent être comparées en utilisant les figures 5.9 et 5.10. On observe que l'allocation dissimilar offset est très performante, souvent meilleure que nos heuristiques. Cependant, utiliser toutes les heuristiques ensemble (*i.e.*, essayer l'allocation fournit par chacune des heuristiques) permet de surpasser l'allocation dissimilar offset. Les heuristiques (incluant l'allocation dissimilar offset) sont très complémentaires. Par exemple, 37.9% des ensembles sont faisables avec au moins une des heuristiques pour 9 tâches figure 5.10) tandis que seulement 26.3% sont faisables avec l'allocation dissimilar offset. Il est important de noter que la complexité de chaque nouvelle heuristique est la même que celle du dissimilar offset et, en pratique, le temps de calcul ne pose pas de problème quel que soit le nombre de tâches.

En conclusion, nos expérimentations montrent que l'utilisation combinée de toutes les heuristiques (*i.e.*, essayer l'allocation fournie par chacune des heuristiques) permet d'augmenter considérablement le pourcentage de système faisable comparée au cas où une seule situation asynchrone est prise en compte.

5.6 Expérimentations : le cas distribué non-préemptif

Comme nous venons de le voir, l'usage de déphasages initiaux pour optimiser la faisabilité du système (et donc minimiser les temps de réponse) est très efficace dans le cas de l'ordonnancement processeur préemptif. Il est loin d'être évident que l'on retrouve des résultats aussi intéressants dans le cas de réseaux de communication, i.e., le cas non-préemptif. En effet, les stations n'étant pas synchronisées temporellement, les déphasages seront nécessairement "locaux" aux stations. Plus explicitement, cela veut dire que l'on ne peut désynchroniser que les flux envoyés par une même station et donc, l'efficacité sera d'autant plus faible, que le trafic est uniformément réparti sur un grand nombre de stations. Des travaux réalisés dans le cadre d'un projet industriel avec PSA nous ont permis d'observer que l'ajout de décalages initiaux sur des messageries réelles améliore grandement les performances (i.e., temps de réponse) du système de communication. Néanmoins, le contrat que nous avons signé avec PSA ne nous autorise pas à publier ces résultats. Nous présentons donc dans la suite, seulement les travaux que nous pouvons publiés selon les termes de ce contrat. Dans ce paragraphe, nous définissons formellement les offset free systèmes dans le cadre distribué. Ensuite, comme il n'existe pas à notre connaissance, d'analyse d'ordonnancement des offset free systèmes distribués ; nous montrons comment modéliser le travail généré par l'ensemble des flux d'une station à l'aide d'une transaction unique afin de permettre une analyse d'ordonnancement dans le cadre bien connu des transactions, cf. [72, 73]. Finalement, les résultats d'expérimentation sont présentés.

5.6.1 Définition des systèmes offset free dans le cadre distribué

Nous étendons ici le modèle des systèmes offset free dans le cadre distribué. Le modèle de flux utilisé est celui décrit dans le chapitre 1 au §1.2. Comme sur un réseau CAN, les stations ne sont pas synchronisées globalement, nous devons considérer le cas où les décalages sont fixés localement sur les stations, *i.e.*, considérer les offsets ($\forall O_i^k$) des tâches de chaque station k, relativement à un référentiel qui est l'horloge interne de la station.

Comme introduit précédemment, un flux τ_i^k est entièrement défini par un tuple $(T_i^k, C_i^k, \overline{D}_i^k, P_i^k, O_i^k)$ où k est l'indice de la station qui envoie les trames du flux, T_i^k est la période de transmission du flux τ_i^k, C_i^k est le pire temps de transmission, $\overline{D}_i^k$ est l'échéance relative de la trame (ex. : la trame doit être reçue 10ms après avoir été créée), O_i^k est l'offset (*i.e.*, date de première arrivée d'une trame de τ_i^k sur la station k). Le flux τ_i^k met dans la file d'attente sa $j^{\text{ème}}$ trame $\tau_{i,j}^k$ à l'instant $A_{i,j}^k \stackrel{\text{def}}{=} O_i^k + (j-1) \cdot T_i^k$.

Sans perdre de généralité, nous faisons l'hypothèse que sur chaque station k, le plus petit offset est nul : $\min_k(O_i^k) = 0$ (i.e. les offsets peuvent tous être décalés vers la "gauche" afin que le plus petit soit nul). La charge de travail générée par un flux τ_i^k est totalement définie par l'ensemble

$$\mathcal{A}(\tau_i^k) = \{\tau_{i,j}^k \mid \forall j \in \mathbb{N}^+\}$$

ou, dans notre cas, par

$$\mathcal{A}(\tau_i^k) = \{(T_i^k, C_i^k, \overline{D}_i^k, P_i^k, A_{i,j}^k) \mid \forall n \in \mathbb{N}^+,\ A_{i,j}^k = O_i^k + (j-1)T_i^k\}.$$

La charge globale générée par un noeud k (aussi appelée "patron d'arrivée de travail") est l'ensemble des

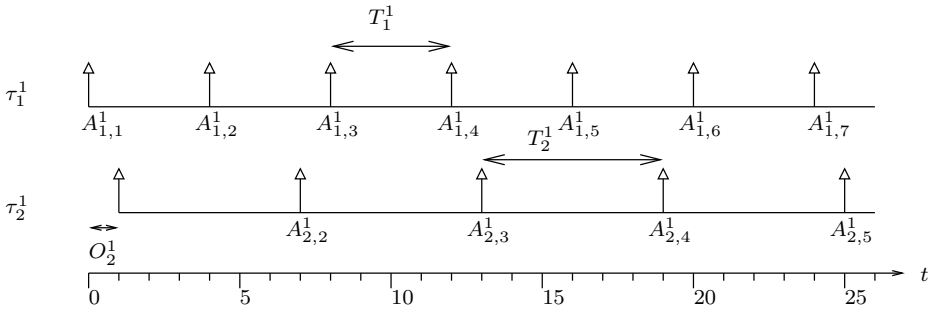


FIG. 5.11 – Charge de travail induite par les flux τ_1^1 et τ_2^1 sur le noeud 1, où les périodes sont respectivement égales à 4 et 5 et les offsets sont 0 et 1.

flux émis par cette station sur le bus :

$$\mathcal{A}(k) = \bigcup_i \mathcal{A}(\tau_i^k).$$

La figure 5.11 illustre les notations proposées avec deux flux : τ_1^1 défini par $(4, 1, 4, 1, 0)$ et τ_2^1 défini par $(6, 2, 6, 2, 1)$ sur le noeud 1. Le patron des arrivées de travail est présenté jusqu'à l'instant 25.

5.6.2 Temps de réponse dans les systèmes offset free distribués

Nous expliquons le modèle de transaction introduit par Tindell [72] formalisé par Palencia et Harbour [73] et comment modéliser le travail généré par l'ensemble des flux d'une station à l'aide d'une transaction unique. Nous verrons que la modélisation à l'aide des transactions permet d'analyser le système à l'aide d'une analyse d'ordonnancement connue. Cependant, comme nous le verrons, cette analyse a une complexité exponentielle. Notre principale contribution fût de réduire de manière significative la complexité de cette analyse afin de pouvoir l'appliquer dans des cas réels. Comme nous l'avons déjà précisé le contrat que nous avons signé avec PSA ne permet pas de présenter la technique de réduction de complexité mais nous pouvons montrer les résultats d'expérimentations.

5.6.2.1 Modèle des transactions

Une transaction Υ_k est constituée d'un sous-ensemble de trame $\mathcal{G}_k$. Chaque transaction Υ_k est périodiquement (ou sporadiquement) activée (*released*) avec une période T_k où $\Upsilon_{k,n}$ est la $n^{\text{ème}}$ activation de la transaction. La $i^{\text{ème}}$ trame de la transaction Υ_k est notée tr_i^k où k est l'indice de la transaction. Remarquez qu'il existe une infinité de ces trames comme la transaction est périodique. Chaque trame tr_i^k a un pire temps de transmission C_i^k, une échéance relative $\overline{D}_i^k$, une priorité P_i^k et un offset O_i^k. Par définition, tr_i^k est activée O_i^k unités de temps après l'activation de chaque instance de Υ_k. tr_i^k est définie par le tuple $(C_i^k, \overline{D}_i^k, P_i^k, O_i^k)$. La figure 5.12 illustre ces notations.

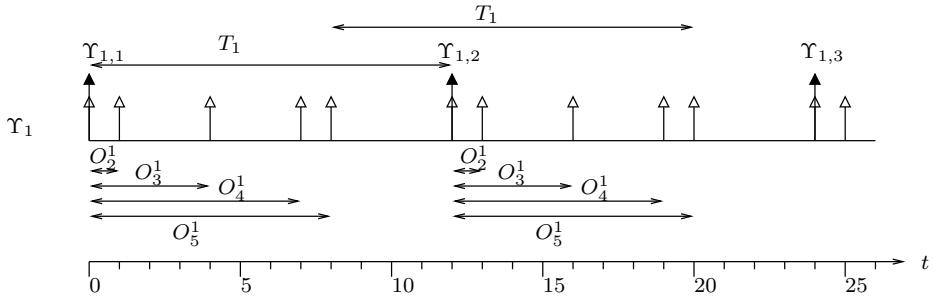


FIG. 5.12 – Charge de travail générée par la transaction Υ_1 de période $T_1 = 12$. Υ_1 est constituée des instances $\mathcal{G}_1 = \{tr_1^1, tr_2^1, tr_3^1, tr_4^1, tr_5^1\}$ où les offsets de tr_1^1, tr_2^1, tr_3^1, tr_4^1 et tr_5^1 sont respectivement $0, 1, 4, 7, 8$.

Pendant toute la durée de vie du système (c'est-à-dire, durant les instances successives de la transaction), la charge de travail générée par tr_i^k est :

$$\mathcal{A}(tr_i^k) = \{(C_i^k, \overline{D}_i^k, P_i^k, A_{i,n}^k) \mid \forall n \in \mathbb{N}^+,\ A_{i,j}^k = O_i^k + (n-1)T_k\}$$

où n est l'indice de l'instance de la transaction. Le patron d'arrivée de travail apporté par la transaction Υ_k est :

$$\mathcal{A}(\Upsilon_k) = \bigcup_i \mathcal{A}(tr_i^k).$$

5.6.2.2 Des flux au modèle de transaction

Nous montrons comment modéliser l'ensemble des flux d'une station k avec une transaction Υ_k *unique*. A l'aide de cette transformation, le système est modélisé dans le cadre bien connu des transactions [72, 73], qui, avec quelques adaptations, permet de calculer les pires temps de réponse. L'objectif est donc de trouver une transaction Υ_k qui agrège la totalité de la charge de travail générée par le noeud k. Pour démontrer l'équivalence entre le modèle des transactions et le modèle classique des flux périodiques, nous montrons que le patron d'arrivée de travail est identique dans chaque cas (*i.e.*, $\mathcal{A}(\Upsilon_k) = \mathcal{A}(k)$).

Patron d'arrivée de travail avec le modèle de flux. Soit ppcm_k le plus petit commun multiple des périodes des flux appartenant à la station k : $\text{ppcm}_k = \text{ppcm}_i\{T_i^k\}$. Par exemple, dans le noeud montré dans la figure 5.11, ppcm_k est égale à 12. Comme on peut le constater sur la figure 5.13, le patron d'arrivée de travail du flux τ_i^k est identique dans chaque intervalle de la forme $[O_i^k + (n'-1)\text{ppcm}_k, O_i^k + (n')\text{ppcm}_k[$ où n' est un nombre entier positif arbitraire. En effet, dans chaque intervalle $[O_i^k + (n'-1)\text{ppcm}_k, O_i^k + (n')\text{ppcm}_k[$, τ_i^k génère une trame exactement au début de l'intervalle $O_i^k + (n'-1)\text{ppcm}_k$, ensuite les $(\frac{\text{ppcm}_k}{T_i^k} - 1)$ trames suivantes sont générées aux instants $O_i^k + (n'-1)\text{ppcm}_k + (j-1)T_i^k$ où $1 \leq j \leq \frac{\text{ppcm}_k}{T_i^k}$. Par exemple, sur la figure 5.13, τ_2^1 génère $\frac{\text{ppcm}_1}{T_2^1} = \frac{12}{6} = 2$ trames dans les deux intervalles $[1, 13[$ et $[13, 25[$ avec une trame générée à l'instant 1 et une à l'instant 13. Par définition, le patron d' arrivée de travail est l'ensemble des trames étant transmises sur le bus durant toute la "vie" du système ; pour le flux i du noeud k, le patron d'arrivée de travail est égale à :

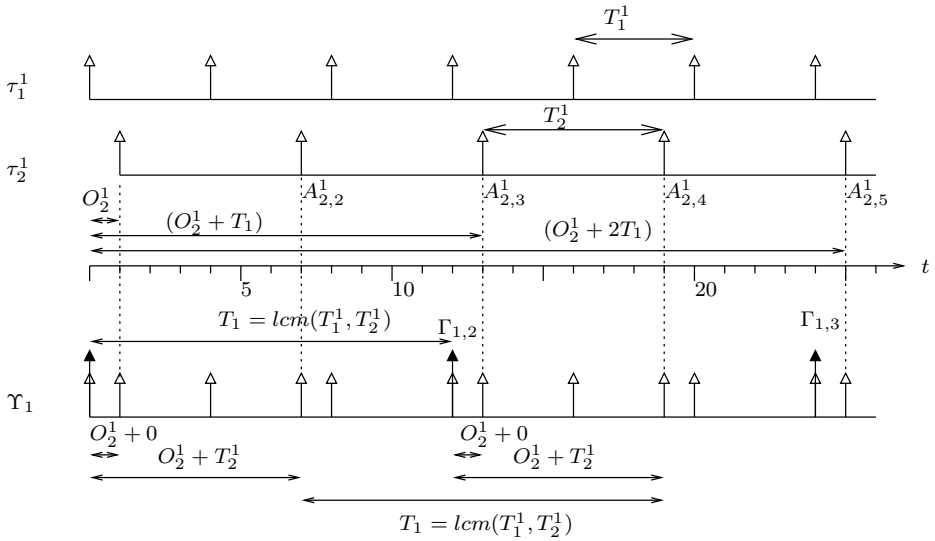


FIG. 5.13 – Modélisation des flux τ_1^1 et τ_2^1 avec la transaction Υ_1. Les périodes de τ_1^1 et τ_2^1 sont respectivement 4 et 6, tandis que les offsets sont 0 et 1.

$$\mathcal{A}(k) = \bigcup_{\forall j \in \mathbb{N}^+ \text{ et } j \leq \frac{\text{ppcm}_i}{T_k^i}} \{(C_i^k, \overline{D}_i^k, P_i^k, O_i^k + (j-1)T_i^k + (n'-1)\text{ppcm}_k) \mid \forall n' \in \mathbb{N}^+\} \tag{5.3}$$

où n' est l'indice de l'intervalle $[O_i^k + (n'-1)\text{ppcm}_k, O_i^k + (n')\text{ppcm}_k[$ considéré (*i.e.*, n' correspond au $n'^{\text{ème}}$ ppcm des périodes), j est la $j^{\text{ème}}$ trame de τ_i^k dans l'intervalle.

Patron d'arrivée de travail avec le modèle des transactions. Nous voulons modéliser le trafic généré par les différents flux du noeud k avec une transaction *unique* Υ_k. Dans le paragraphe précédent, nous avons observé que le patron d'arrivée de travail de chaque flux τ_i^k du noeud k se répète périodiquement dans chaque intervalle $[O_i^k + (n'-1)\text{ppcm}_k, O_i^k + (n')\text{ppcm}_k[$ de longueur ppcm_k. Nous utilisons cette propriété pour modéliser l'ensemble du trafic à l'aide d'une transaction Υ_k de période $T_k = \text{ppcm}_k$ qui inclut toutes les instances générées dans chaque intervalle $[O_i^k + (n'-1)\text{ppcm}_k, O_i^k + (n')\text{ppcm}_k[$. Un flux particulier τ_i^k apporte à la transaction Υ_k l'ensemble suivant de trames $\mathcal{G}_i^k$:

$$\mathcal{G}_i^k = \{(C_i^k, \overline{D}_i^k, P_i^k, O_i^k + (j-1)T_k^i) \mid \forall j \in \mathbb{N}^+ \text{ et } j \leq \frac{\text{ppcm}_k}{T_i^k}\} .$$

Par exemple, dans la figure 5.13, les arrivées de travail de τ_2^1 dans les transactions successives Υ_1 comprennent deux trames avec respectivement les offsets $1+0$ et $1+6$.

Comme Υ_k agrège l'ensemble des flux du noeud k, l'ensemble des trames générées par chaque instance de Υ_k est

$$\mathcal{G}_k = \bigcup_i \mathcal{G}_i^k.$$

Priorité (seulement les 5 dernières)	1	2	3	4	5
Nombre de scénarios testés (sans réduction de complexité)	$1.6 \cdot 10^{13}$	$1.6 \cdot 10^{13}$	$2 \cdot 10^{13}$	$2.1 \cdot 10^{13}$	$2.3 \cdot 10^{13}$
Nombre de scénarios testés (en utilisant notre techniques de réduction)	775.8	1798.4	1635.9	1884.8	1961.8

TAB. 5.1 – Complexité moyenne (nombre de scénarios à considérer) du calcul de temps de réponse d'un réseau habitacle.

Le patron d'arrivée de travail de Υ_k est donc :

$$\{(C_i^k, \overline{D}_i^k, P_i^k, O_i^k + (j-1)T_k^i + (j'-1)T_k) \mid \forall j \in \mathbb{N}^+ \text{ et } j \leq \frac{\text{ppcm}_i}{T_k^i}, \forall j' \in \mathbb{N}^+\}$$

qui est équivalent à l'ensemble défini dans l'équation 5.3. Donc, la charge de travail générée par Υ_k et le noeud k sont identiques.

A l'aide de la transformation proposée dans ce paragraphe, nous pouvons utiliser l'analyse d'ordonnancement développée par Palencia et Harbour [73] pour calculer le temps de réponse des différents flux.

5.6.2.3 Analyse des temps de réponse

Le calcul du pire temps de réponse d'une trame est possible en modélisant le trafic à l'aide de transaction et en utilisant l'analyse d'ordonnancement développée par Palencia et Harbour [73]. Comme les offsets des flux sont fixés localement sur les stations, l'idée est de trouver un scénario défavorable. Le problème est qu'il est impossible d'identifier a priori quelle trame de chacune des stations va conduire à la pire trajectoire du système et donc à la trajectoire qu'il faut étudier dans une analyse pire-cas.

En termes de complexité, l'espace de recherche est globalement le produit du PPCM de chaque station et, dans le cas général, il est impossible de trouver le scénario pire cas en un temps raisonnable. C'est pourquoi, nous avons proposé des techniques visant à réduire la complexité qui nous permettent dans le cadre particulier des messageries véhicules, où le nombre de périodes différentes des trames est restreint (donc le PPCM de ces périodes est petit), de calculer effectivement les pires temps de réponse. Des méthodes moins complexe de calcul de bornes supérieures sur le temps de réponse dans le cadre des transactions ont été proposées, cf. par exemple Maki-Turja et Nolin [74], et de calcul exacte lorsque les transactions ont des propriétés spécifiques, cf. Traore [75].

La figure 5.1 présente la réduction de l'espace de recherche moyenne calculée sur 100 ensembles de tâches. Les conditions d'expérimentation sont données dans le §5.6.3. On constate que la réduction de complexité est très importante (d'au moins 9 ordres de grandeur).

5.6.3 Minimiser les temps de réponse : le cas des messageries véhicules

L'algorithme dissimilar offset et les heuristiques sont très performants lorsque les périodes des flux sont aléatoires [40]. Néanmoins, un réel problème se pose lorsque les périodes différentes sont peu nombreuses et que le pgcd des couples sont similaires[2]. En effet, dans ce cas, il y a de multiples couples avec le même pgcd et des offsets identiques sont attribués à de nombreux flux. Les algorithmes n'ont tout simplement pas été conçus et testés sur des cas où il pouvait y avoir plusieurs couples de flux avec des pgcd identiques. La section C.2 de l'annexe C présente un exemple illustrant ce problème avec l'algorithme dissimilar offset. Dans le cadre du projet industriel, nous avons proposé des heuristiques "améliorées" afin de pallier à ce problème (heuristiques confidentielles). Dans la suite, nous présentons tout de même l'efficacité de ces algorithmes avec des messageries représentatives d'un réseau habitacle actuel générées aléatoirement. Dans ces expérimentations, nous évaluons les performances des algorithmes "améliorés" dissimilar offset, de l'heuristique 1 présentée au §5.4.

5.6.3.1 Conditions d'expérimentation

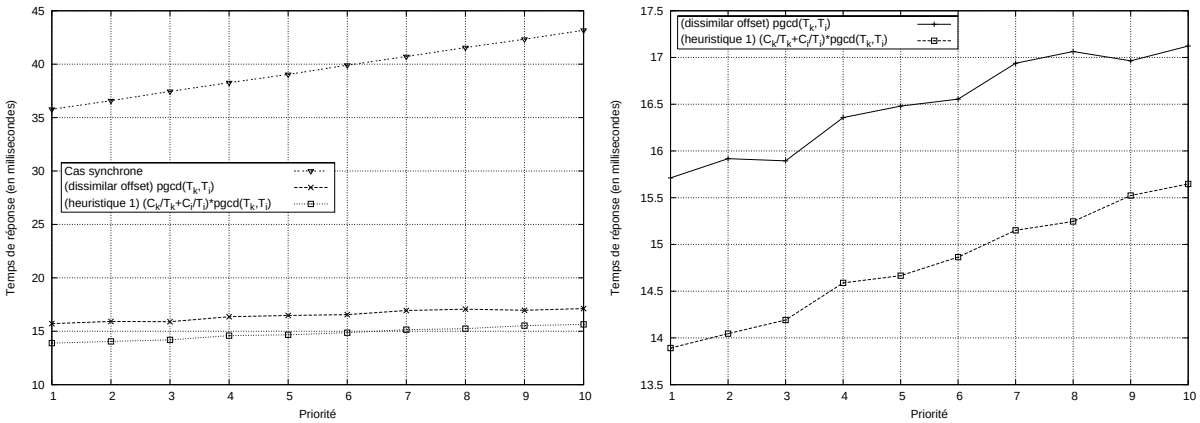
Afin d'obtenir une vision statistiquement plus exacte des performances relatives de chacun des algorithmes, nous avons généré 100 messageries aléatoires représentatives d'un réseau habitacle et collecté les temps de réponse obtenus avec chacun des algorithmes de déphasage. Pour la génération aléatoire des messageries, le logiciel NETCARBENCH (enregistré à l'APP sous le numéro IDDN FR 001 260007 000 S P 2007 000 10000) a été utilisé ; les paramètres suivants ont été utilisés :

- 8 stations connectées sur un réseau à 125kbit/s,
- charge réseau totale (trafic périodique) entre 30 et 35%,
- données utiles entre 1 et 8 octets par trame,
- périodes choisies aléatoirement dans l'ensemble $\{100ms, 200ms, 500ms, 1000ms\}$,
- en moyenne 50 trames de priorités différentes sur l'ensemble du réseau, chaque trame est allouée à une station choisie au hasard.

5.6.3.2 Performances relatives des heuristiques

Les figures 5.14(a) et 5.14(b) présentent la moyenne du temps de réponse pour les 10 trames les moins prioritaires sur l'ensemble des messageries "artificielles" avec les deux algorithmes de déphasage et avec le cas synchrone (cf. figure 5.14(a)). On peut constater sur la figure 5.14 que l'utilisation d'offset permet de diminuer en moyenne d'un facteur 2.5 les temps de réponse. Par exemple, le temps de réponse de la trame de priorité 5 le temps de réponse dans le cas synchrone est 38.2ms alors qu'il est, pour la plus mauvaise heuristique de 16.4ms. On observe sur la figure 5.14(b) qu'en moyenne l'heuristique 1 est toujours la meilleure. En moyenne, l'heuristique 1 est sensiblement meilleure avec une amélioration d'environ 10% par rapport à l'allocation dissimilar offset. Il faut bien avoir à l'esprit qu'il s'agit ici de valeurs moyennes sur des messageries simulées, sur une messagerie réelle particulière les résultats respectifs de chacun des algorithmes pourront être différents. En pratique, sauf connaissance imparfaite des tailles des trames qui

[2] Par exemple, dans les messageries réelles étudiées dans le cadre du projet industriel avec PSA, il n'y avait que 5 valeurs différentes de périodes et de pgcd.



(a) Pire cas et asynchrones

(b) Zoom sur situations asynchrones

FIG. 5.14 – Moyenne des temps de réponse pire cas des 10 trames les moins prioritaires de 100 messageries représentatives d'un réseau habitacle générées aléatoirement (30 à 35% de charge - réseau à 125kbit/s). La figure (a) présente les résultats du cas synchrone et des différentes situations asynchrones, la figure (b) fait un zoom sur les situations asynchrones.

ne nous permet pas d'appliquer l'heuristique 1, on pourra utiliser les 2 algorithmes différents (et même les autres heuristiques présentées au §5.4) et choisir la solution la meilleure qui sera généralement celle obtenue à l'aide de l'heuristique 1.

Intuitivement la raison qui permet d'expliquer les performances de l'heuristique 1 est que la disparité de la taille des trames et donc de la charge générée par chacun des flux, est très importante sur les messageries générées. Or l'heuristique 1 a été dès le début conçue comme une amélioration de l'allocation dissimilar offset dans le cas de trames dont la taille varie et pour lesquelles, outre la période, il devient donc nécessaire de considérer la taille. Sur des messageries pour lesquelles la variabilité de la taille des trames est moindre, les performances de l'heuristique 1 et de l'allocation dissimilar offset seront plus proches.

5.7 Conclusion

Dans ce chapitre, nous avons étudié le problème de l'ordonnancement préemptif à priorité fixe des systèmes offset free dans les situations mono-processeur préemptif et distribué sur un réseau priorisé. Pour le cas préemptif, nous avons montré que la complexité, *i.e.*, le nombre d'allocations d'offsets différentes à considérer pouvait être réduit d'au moins 50% avec une utilisation appropriée de l'algorithme d'Audsley. Ensuite, nous avons proposé de nouvelles heuristiques d'allocation d'offset pour améliorer celle du dissimilar offset introduite dans [65]. Ces heuristiques fournissent des situations asynchrones alternatives, qui permettent d'améliorer de manière significative le nombre de systèmes ordonnançables en regard du cas pessimiste synchrone. En effet, l'usage combiné des heuristiques permet d'ordonnancer au moins 40.5% et jusqu'à 97% des ensembles de tâches selon nos expérimentations, réalisées avec des ensembles de 5 à 11 tâches avec une charge moyenne de 0.8 .

A notre connaissance, aucune analyse d'ordonnancement existe pour le cas des systèmes offset free non-préemptif distribué sur un réseau priorisé. Nous avons fourni ici une analyse d'ordonnancement pour

ces sytèmes. De plus, nos expérimentations montrent que l'usage de nos heuristiques permet de réduire en moyenne d'un facteur 2.5 les temps de réponse comparé au cas synchrone dans les messageries automobiles sur un réseau CAN (*i.e.*, cas distribué non-préemptif).

Dans de prochains travaux, nous envisagerons l'utilisation d'algorithme d'optimisation locale (e.g., hill-climbing, algorithme génétique, recuit simulé) pour réaliser l'allocation d'offset. Une méthode hill-climbing a déjà été implémentée dans NetCar-Analyzer. Le principal problème serait de trouver les fonctions de coût appropriées pour évaluer les bonnes solutions. De plus, des métaheuristiques ont déjà été utilisées pour réduire la gigue (pour plus de détails voir chapitre 2). Les préliminaires à toute étude serait donc d'étudier l'impact des métaheuristiques existantes (visant à réduire la gigue) sur le temps de réponse. Une autre approche serait d'essayer d'utiliser d'autres méthodes d'optimisation telles que la programmation par contraintes ou la programmation linéaire.

Chapitre 6

Configuration d'ordonnancement sous Posix 1003.1b

6.1 Introduction

Contexte. Ce chapitre traite de l'ordonnancement des systèmes temps-réel implémentés sur un système d'exploitation conforme au standard Posix 1003.1b. Posix 1003.1b [36], anciennement Posix4, définit des extensions temps réel au standard Posix concernant principalement les signaux, les communications inter-processus, les entrées sorties synchrones et asynchrones, les timers et les politiques d'ordonnancement (un rappel des principales caractéristiques de Posix relatives à l'ordonnancement est donné en §6.2.1). Ce standard est devenu très populaire et la plupart des systèmes d'exploitation s'y conforment au moins partiellement.

Les systèmes d'exploitation conformes au standard Posix 1003.1b fournissent en standard deux politiques d'ordonnancement qui s'appliquent aux processus, appelées *sched_fifo* et *sched_rr*. Ces deux politiques sont respectivement équivalentes, sous certaines hypothèses détaillées au §6.2.1, à une politique à allocation de priorité fixe (*Fixed Preemptive Priority* - FPP) et à une politique à allocation de priorité dite à "tourniquet" (Round-Robin - RR). Sous Posix 1003.1b, une priorité, une politique d'ordonnancement et dans le cas de Round-Robin, un quantum sont alloués à chaque processus. A chaque instant, le processus en exécution appartient à l'ensemble des processus de plus haute priorité et est sélectionné parmi cet ensemble selon la politique d'ordonnancement.

Définiton du problème. Habituellement, la politique RR est appliquée principalement aux processus de faible priorité réalisant des calculs secondaires "quand rien d'autre d'important ne s'exécute". Dans ce chapitre, nous montrons comment maximiser la faisabilité par l'utilisation combinée de RR et FPP en ordonnançant avec succés un nombre important de systèmes non-ordonnançables avec seulement FPP. Le problème traité est en fait d'allouer la priorité, la politique d'ordonnancement et le quantum à chaque processus, *i.e.* tâche, de façon à respecter les contraintes d'échéance.

État de l'art. Audsley dans [14, 66] propose un algorithme d'ordonnancement, optimal pour FPP seule, qui est maintenant connu dans la littérature sous la dénomination d'algorithme d'Audsley ; cet algorithme a été présenté dans le §5.3.2 du chapitre 5. Nous rappelons que l'algorithme d'Audsley est optimal dans la classe des politiques préemptives à priorité fixe pour des systèmes de tâches concrètes et non-concrètes ayant des échéances arbitraires. Plus tard dans [10], cet algorithme a été montré optimal pour le cas non-préemptif à priorité fixe. Le problème étudié ici est différent du cas FPP seul, car l'utilisation de RR mène à l'occurrence d"'anomalies" d'ordonnancement, souvent contre-intuitives. Par exemple, comme il sera montré dans le §6.2.5, augmenter la valeur du quantum d'une tâche peut quelquefois augmenter le temps de réponse de cette tâche. Ce problème de l'allocation des priorités et des politiques a été traité pour la première fois par Navet et Migge [26], où les auteurs proposent un algorithme génétique cherchant des allocations faisables optimisant des critères dépendants de l'application (ex. : minimiser la gigue sur la fin d'exécution, maximiser la fraîcheur d'une donnée) ; la solution repose sur des heuristiques et n'est donc pas optimale dans le cas général. Trouver une solution optimale dans le cas général revient à faire une recherche exhaustive dans l'espace des solutions où chaque solution est l'allocation de priorités, de politiques et de quanta à chaque processus (*i.e.*, tâche). Comme nous le verrons dans le §6.3.2 et le §6.4.2, la complexité de ce problème est exponentielle et une recherche exhaustive n'est pas possible même avec des problèmes de petite taille. Par exemple, un ensemble de 10 tâches pour lequel le quantum de chaque tâche peut être choisi parmi 5 valeurs différentes requiert d'analyser la faisabilité de plus de 10^{11} configurations différentes.

Contributions. Pour réduire la complexité du problème nous proposons des extensions de l'algorithme d'Audsley dans les cas suivants : (1) la valeur du quantum est une variable constante globale au système et (2) le quantum de chaque processus peut être choisi. La première extension [39], appelée *Audsley-RR-FPP*, traite du cas (1) et la deuxième [41], appelée *Audsley-RR-FPP**, étend *Audsley-RR-FPP* pour prendre en compte le cas (2).

L'existence d"'anomalies" d'ordonnancement sous RR nous empêche de tester la faisabilité de l'allocation de priorité, de politiques et de quanta par simulation. De plus, dans la littérature, à notre connaissance, il n'existe pas d'analyse d'ordonnançabilité qui repose sur une condition nécessaire et suffisante dans ce cas.

Dans cette étude, nous supposons que l'on sait exhiber une fonction de calcul de borne supérieure sur le pire temps de réponse (qui peut ne pas être atteinte), telle que la borne supérieure sur le pire temps réponse d'une tâche τ_i ne croît pas quand l'ensemble de tâches de priorité supérieure ou égale à celle de τ_i est réduit. Alors, nous dirons qu'une configuration d'ordonnancement est faisable si, pour chaque tâche, la borne supérieure sur le pire temps de réponse, calculée avec cette fonction de calcul, est inférieure ou égale à son échéance. On appelle puissance de l'analyse d'ordonnançabilité la capacité de l'analyse à distinguer les configurations faisables des configurations non-faisables à l'aide du test de faisabilité qui repose sur une telle fonction.

Nos nouveaux algorithmes d'allocation (*Audsley-RR-FPP* et *Audsley-RR-FPP**) seront montrés optimaux relativement à la fonction de calcul de borne sur le temps de réponse, dans le sens où les algorithmes trouvent une solution faisable (au sens défini précédemment) s'il en existe au moins une. En effet, nous verrons que la fonction de calcul utilisée dans l'analyse d'ordonnancement doit posséder des propriétés

spécifiques, détaillées dans le §6.2.5, afin qu'un algorithme similaire à celui d'Audsley ne "rejette" pas des allocations ordonnançables à cause des "anomalies" de l'ordonnancement RR et ne soit donc pas optimal.

Dans la suite, nous utilisons l'analyse de borne supérieure sur les pires temps de réponse proposée par Migge et *al.* [1] pour distinguer les solutions faisables et non-faisables. Pour chacun des algorithmes proposés, la complexité pire cas est étudiée et un ensemble d'optimisations est fourni pour réduire l'espace de recherche.

Malgré la réduction significative de la complexité, les algorithmes sont limités aux problèmes de petites et moyennes tailles. C'est pourquoi nous spécifions, dans le cas (1) où le quantum est une valeur globale au système, une heuristique, appelée *Load-RR-FPP*, pour les problèmes de plus grande taille.

La troisième contribution de ce chapitre est de mettre en évidence l'efficacité de l'usage combiné de FPP et RR pour trouver des ensembles ordonnançables même quand la charge du système est élevée (efficacité encore meilleure quand le quantum individuel de chaque tâche peut être choisi).

Organisation. La section 6.2 résume les principales caractéristiques de l'ordonnancement Posix 1003.1b, introduit le modèle et les notations. Dans les sections 6.3 et 6.4, nous présentons les deux algorithmes optimaux d'allocation de priorité, de politique et de quantum : *Audsley-RR-FPP* (i.e., quantum constant global au système) et *Audsley-RR-FPP** (i.e., quantum spécifique à chaque tâche). Ensuite, une heuristique, appelée *Load-RR-FPP,* dont le but est de traiter de plus grand ensemble de tâches est proposée en section 6.5. L'efficacité de nos propositions est évaluée dans la 6.6.

6.2 Ordonnancement sous Posix 1003.1b : modèle et propriétés

Nous présentons dans cette section le modèle du système et résumons les principales caractéristiques de l'ordonnancement Posix 1003.1b. Ensuite, nous donnons les hypothèses faites dans cette étude et les propriétés de base de l'ordonnancement Posix 1003.1b qui seront utilisées dans les sections suivantes.

6.2.1 Ordonnancement sous Posix 1003.1b

On appelle "processus" l'agent responsable de l'exécution d'un programme. Un processus est caractérisé par un état et un espace d'adressage dans lequel programme et données peuvent être adressés. La notion de *thread*, *i.e.*, de "processus léger" , défini dans le standard Posix 1003.1c [36] permet de réaliser plusieurs "tâches" (ou threads) dans un seul processus. Un thread partage l'espace d'adressage du processus dans lequel il a été crée. Les threads peuvent être ordonnancés de deux façons :

- au niveau système (*system contention scope*) : indépendamment du processus dans lequel ils ont été créés, ces threads sont en concurrence avec tous les autres processus et tous les autres threads ordonnancés au niveau système.
- au niveau processus (*process contention scope*) : les threads sont en concurrence avec les autres threads du même processus pendant le temps processeur qui est alloué au processus. Cependant, le standard ne précise pas comment ces threads sont ordonnancés par rapport aux threads ordonnancés globalement et aux processus. L'ordonnancement de ces threads étant dépendant de l'implémentation, nous ne les considérons pas dans la suite.

Au niveau du système d'exploitation, nous définissons une tâche périodique comme une activité récurrente qui est réalisée soit en lançant un processus ou un thread (ordonnancés au niveau système) de manière répétitive soit en utilisant un seul processus ou un thread (ordonnancés au niveau système) s'exécutant en cycle.

Ainsi qu'énoncé dans l'introduction, Posix 1003.1b définit trois politiques d'ordonnancement : *sched_rr*, *sched_fifo* et *sched_other*. Ces politiques sont appliquées aux processus : chaque processus s'exécute avec une priorité et une politique données. Chaque processus hérite des paramètres d'ordonnancement de son père, mais peut les changer en cours d'exécution. A chaque instant, le processus en exécution appartient à l'ensemble des processus de plus haute priorité et est sélectionné parmi cet ensemble selon la politique d'ordonnancement. Les trois politiques définies par Posix 1003.1b sont :

- *sched_fifo* : politique préemptive à priorité fixe avec un ordre "FIFO" (First-In First-out, premier arrivé, premier servi) parmi les tâches de même priorité.
- *sched_rr* : politique dite à "tourniquet", ou Round-Robin (RR) qui permet à des processus de même priorité de partager équitablement le CPU. Notons qu'un processus aura le CPU jusqu'à ce qu'un processus plus prioritaire s'exécute. La valeur du quantum de temps peut être une constante globale au système, ou spécifique à chaque processus.
- *sched_other* : propre au système d'exploitation. Elle peut être équivalente à *sched_fifo* ou *sched_rr*, ou bien mettre en oeuvre une politique Unix classique de temps partagé. Étant donné qu'il n'est pas possible de garantir un comportement identique d'un système à l'autre, l'utilisation de cette politique n'est pas recommandée pour des questions de portabilité. Nous ne la considérons donc pas dans cette étude.

A chaque politique d'ordonnancement est associé un intervalle de priorité. En fonction de l'implémentation, les intervalles peuvent ou non se chevaucher, mais la plupart des implémentations permettent le chevauchement. Notez que ces mécanismes d'ordonnancement expliqués précédemment s'appliquent de manière similaire aux threads Posix ordonnancés globalement (system contention scope) comme spécifié dans le standard [36].

6.2.2 Modèle du système

Les activités du système sont modélisées par un ensemble de tâches $\mathcal{T}$ composé de n tâches indépendantes et périodiques. Une tâche périodique τ_i est caractérisée par un tuple $(C_i, T_i, \overline{D}_i, O_i)$ où T_i est la période et O_i son offset et où toutes les requêtes de τ_i, appelées instances, ont le même pire temps d'exécution C_i et la même échéance relative $\overline{D}_i$ (cf. introduction chapitre 1 § 1.2). Nous notons $\tau_{i,j}$ la $j^{\text{ème}}$ activation de τ_i. Dans cette étude nous considérons des ensembles non-concrets de tâches où toutes les valeurs d'offset sont enviseagables.

Sous Posix 1003.1b, cf. §6.2.1, chaque tâche τ_i possède une priorité p_i et une politique d'ordonnancement $sched_i$. Dans cette étude, nous choisissons la convention "plus petite la valeur numérique p_i, plus grande la priorité de τ_i". Sous RR, en plus de la priorité, une valeur de quantum ψ_i est allouée à chaque tâche τ_i. Une allocation de priorités et de politiques d'ordonnancement, notée $\mathcal{P}$, est entièrement définie par un ensemble de n tuples $(\tau_i, p_i, sched_i^{\mathcal{P}})$ (un pour chaque tâche). Par exemple, pour un ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2, \tau_2\}$, une allocation $\mathcal{P}$ possible est $\{(\tau_1, 1, sched\_fifo), (\tau_2, 2, sched\_rr), (\tau_3, 2, sched\_rr)\}$. Sous $\mathcal{P}$, une allocation de quantum, appelé $\Psi_{\mathcal{P}}$, définit l'ensemble des valeurs de quantum $\psi_i^{\Psi_{\mathcal{P}}}$, où $\psi_i^{\Psi_{\mathcal{P}}}$

est le quantum d'une tâche τ_i qui est ordonnancé avec RR dans $\mathcal{P}$. Dans notre exemple, une allocation de quantum $\Psi_\mathcal{P}$ possible est $\{\psi_2^{\Psi_\mathcal{P}}, \psi_3^{\Psi_\mathcal{P}}\}$ où $\psi_2^{\Psi_\mathcal{P}} = 2$ et $\psi_3^{\Psi_\mathcal{P}} = 5$. La totalité de l'ordonnancement, appelée *configuration* du système dans la suite, est donc définie par le couple $(\mathcal{P}, \Psi_\mathcal{P})$

Sous l'allocation $\mathcal{P}$, l'ensemble de tâches $\mathcal{T}$ est partitionné en "couches", une couche par niveau de priorité j où la couche $\mathcal{T}_j^\mathcal{P}$ est le sous-ensemble de tâches ayant la priorité j. Sous $\mathcal{P}$, $\mathcal{T}_{hp(j)}^\mathcal{P}$ (resp. $\mathcal{T}_{lp(j)}^\mathcal{P}$) est l'ensemble de tâches possédant une priorité supérieure (resp. inférieure) à j. Une couche dans laquelle toutes les tâches sont ordonnancées avec RR (resp. FPP) est appelée une couche RR (resp. FPP). Dans la suite, $\mathcal{P}$ ou $\Psi_\mathcal{P}$ sont omis quand aucune confusion n'est possible.

Dans la suite de ce chapitre, une tâche τ_i est dite faisable (*i.e.*, ordonnançable) si sa borne supérieure sur le pire temps de réponse, calculée avec la fonction proposée par Migge et *al.* [1] pour les systèmes Posix 1003.1b, est toujour inférieure ou égale à son échéance relative. Le système est faisable si toutes les tâches sont faisables. Il faut remarquer que l'analyse de Migge et *al.* [1] donne une condition suffisante mais pas nécessaire : il y a des ensembles de tâches qui ne sont pas classifiés comme ordonnançables alors qu'il existe pour ces ensembles des configurations où aucune échéance n'est ratée.

6.2.3 Hypothèses

Tel qu'expliqué dans le *§6.2.1,* on considère, dans cette étude, uniquement les politiques *sched_fifo* et *sched_rr* pour des raisons de portabilité. Du fait de la complexité du problème de l'allocation des priorités, des quanta et des politiques d'ordonnancement, les restrictions suivantes sont faites :

1. les latences dues aux changements de contexte sont négligées. Néanmoins, elles peuvent être incluses d'une manière classique dans l'analyse d'ordonnancement [1] (cf., par exemple, [76]),
2. nous considérons, pour faciliter les calculs, des rangs contigus de priorité, car un niveau de priorité sans tâche n'a pas d'influence sur l'ordonnancement ; tout système répondant aux autres hypothèses peut être transformé en un système équivalent répondant à celle-ci,
3. deux tâches ayant des politiques différentes ont des priorités différentes, c-à-d, $\forall i \neq j$, $sched_i \neq sched_j \implies p_i \neq p_j$,
4. les tâches ayant la même priorité sont nécessairement ordonnancées avec la politique *sched_rr* (RR). Toutes les tâches *sched_fifo* doivent donc posséder des priorités fixes différentes ($sched_i = sched_j = sched\_fifo \implies p_i \neq p_j$). En fait, la politique *sched_fifo* n'est pas supportée dans le sens où une tâche *sched_fifo* est seule à son niveau de priorité. Avec cette hypothèse et sans changement de priorité en cours d'exécution, *sched_fifo* est équivalente à la politique préemptive à priorité fixe (FPP - Fixed Preemptive Priorty),
5. la valeur du quantum est soit une constante unique spécifique au système, soit une valeur constante spécifique à chaque tâche,
6. dans une couche RR, l'ordre dans lequel l'ordonnanceur considère les tâches est toujours le même.

6.2.4 Rappel : analyse d'ordonnancement sous Posix [1]

Dans ce paragraphe, nous résumons l'analyse d'ordonnancement [1] d'une configuration $(\mathcal{P}, \Psi_\mathcal{P})$ sous Posix. L'ordonnancement de tâches sous Posix peut se décrire comme une superposition de couches

de priorité [1]. A chaque instant, une des instances prêtes avec la plus grande priorité (disons p_i) est exécutée dès que et aussi longtemps qu'aucune instance dans une couche de plus haute priorité (instance d'une tâche dans $\mathcal{T}_{hp(p_i)}$) soit en attente. Au sein de chaque couche de priorité, les instances des tâches sont ordonnancées selon la politique d'ordonnancement (selon FPP ou RR dans cette étude) avec la contrainte que toutes les instances des tâches appartenant à la même couche aient la même politique d'ordonnancement.

La politique FPP est réalisée quand, à chaque instant, l'instance en exécution appartient à l'ensemble des processus de plus haute priorité. Sous RR, l'ordonnancement est divisé en cycle et chaque tâche τ_i se voit attribuer un quantum $\psi_i^{\Psi_\mathcal{P}}$. Dans chaque cycle, l'ordonnanceur offre l'occasion à chaque tâche τ_i de s'exécuter pendant au plus une durée égale à $\psi_i^{\Psi_\mathcal{P}}$, où $\psi_i^{\Psi_\mathcal{P}}$ est le quantum de la tâche. Un cycle est donc un intervalle de temps durant lequel chaque tâche de la couche RR a l'occasion de s'exécuter. Notons que selon l'hypothèse 6,nous supposons que l'ordre dans lequel l'ordonnanceur considère les tâches est toujours le même. Si l'ordonnanceur considère une tâche qui n'a plus d'instance en attente ou moins de travail que la durée de son quantum, alors le reste de son quantum est perdu, car l'ordonnanceur considérera la tâche suivante et la tâche devra attendre le prochain cycle pour continuer. Le temps entre deux opportunités consécutives varie en fonction des demandes actuelles des autres tâches, mais est borné dans chaque intervalle où la tâche considérée a du travail en attente à chaque instant par : la somme des quantum des autres tâches de la couche RR $\overline{\psi}_i^{\Psi_\mathcal{P}} = \sum_{\tau_k \in \mathcal{T}_{p_i}^\mathcal{P}} \psi_k^{\Psi_\mathcal{P}}$ plus le travail éventuel des tâches plus prioritaires $\widetilde{s}_i(t)$. Dans [1], une fonction de calcul de borne sur le pire temps de réponse a été développée pour les couches de priorité indépendamment des politiques d'ordonnancement des autres couches. Cette analyse est basée sur le concept de fonction d'arrivée de travail majorante, qui mesure une borne sur la demande processeur, pour chaque tâche, sur un intervalle commençant à un "instant critique". La fonction d'arrivée de travail majorante sur un intervalle de longueur t pour une tâche périodique τ_i est :

$$s_i(t) = C_i \cdot \left\lceil \frac{t}{T_i} \right\rceil . \tag{6.1}$$

Une borne sur le pire temps de réponse est calculée avec la fonction suivante :

$$\max_{j<j^*}(e_{i,j} - a_{i,j}) \ , \tag{6.2}$$

où $j^* = \min\{j \mid e_{i,j} \leq a_{i,j+1}\}$, $a_{i,j}$ la $j^{\text{ème}}$ activation de τ_i après l'instant critique et $e_{i,j}$ est une borne sur la fin d'exécution de cette instance. Comme τ_i est une tâche périodique, $a_{i,j} = (j-1) \cdot T_i$ $(j = 1, 2, ...)$. Si τ_i est dans une couche FPP, alors

$$e_{i,j} = \min\{t > 0 \mid \widetilde{s}_i(t) + s_{i,j} = t\} \ , \tag{6.3}$$

où $\widetilde{s}_i(t) = \sum_{\tau_k \in \mathcal{T}_{hp(p_i)}^\mathcal{P}} s_k(t)$ est la demande des tâches de plus haute priorité (*i.e.*, tâches dans $\mathcal{T}_{hp(p_i)}^\mathcal{P}$) et $s_{i,j} = \sum_{i=1}^{j} C_i$ est la demande des instances précédentes et courante de τ_i. Si τ_i est dans une couche RR, alors

$$e_{i,j} = \min\{t > 0 \mid \overline{\Psi}_i(t) + s_{i,j} = t\} \ , \tag{6.4}$$

où la demande des tâches de plus haute priorité et de toutes les autres tâches dans la couche RR est :

$$\overline{\Psi}_i(t) = \min\left(\left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_\mathcal{P}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_\mathcal{P}} - \psi_i^{\Psi_\mathcal{P}}) + \widetilde{s}_i(t), s_i^*(t)\right) , \tag{6.5}$$

où $\overline{\psi}_i^{\Psi_\mathcal{P}} - \psi_i^{\Psi_\mathcal{P}}$ est la somme des quanta de toutes les autres tâches dans la couche RR et

$$s_i^*(x) = \max_{u \geq 0} \left(s_i(u) + \widetilde{s}_i(u+x) + \overline{s}_i(u+x) - u\right), \tag{6.6}$$

où $\overline{s}_i(u+x) = \sum_{\tau_k \in \mathcal{T}_{p_i}^\mathcal{P} \setminus \{\tau_i\}} s_k(u+x)$ est la demande des autres tâches que τ_i dans $\mathcal{T}_{p_i}^\mathcal{P}$.

Ainsi que dit dans l'introduction, nous utiliserons cette fonction de calcul (cf. équation 6.2) de borne supérieure sur le temps de réponse proposée par Migge et *al.* [1] pour tester la faisabilité.

6.2.5 Ordonnancement sous Posix 1003.1b : propriétés de base

Sous FPP et sous RR, les tâches de haute priorité préemptent les tâches de plus basses priorités. C'est pourquoi, les propriétés suivantes, pour toute tâche τ_i de priorité p_i, sont respectées :

1. les instances prêtes à s'exécuter avec une priorité plus grande que p_i retardent la fin d'exécution des instances de τ_i. Il est important de noter que ce délai ne dépend ni de l'ordre relatif des priorités parmi les instances de plus hautes priorités ni de la valeur de leurs quanta,
2. les instances de plus basse priorité n'interfèrent pas avec l'exécution des instances de τ_i et donc ne retardent pas leur fin d'exécution.

Ces deux propriétés assurent que le lemme suivant, bien connu dans le cas FPP, soit vrai.

Lemme 6 *[66] Le pire temps de réponse de τ_i dépend seulement de l'ensemble des tâches de plus haute priorité et, le cas échéant (ordonnancement de τ_i sous RR), de l'ensemble des tâches de même priorité, en particulier des valeurs de leur quantum. L'ordre relatif des priorités et la valeur des quantum parmi les tâches de plus hautes priorités n'a pas d'influence.*

Cependant, malgré le lemme 6, il existe des anomalies d'ordonnancement avec RR. En effet, l'ordonnancement avec Posix est souvent contre-intuitif. Par exemple, il a été montré dans [77] que réduire le temps d'exécution d'une tâche dans une configuration qui aurait été faisable avec les pires temps d'exécution peut entraîner des dépassements d'échéance. Ici, nous montrons que l'augmentation de la taille du quantum d'une tâche peut augmenter son temps de réponse. Les figures 6.1 et 6.2 présentent l'ordonnancement de l'ensemble $\mathcal{T} = \{\tau_1, \tau_2\}$ où $\tau_1 = (C_1 = 2, T_1 = 5, \overline{D}_1 = 5)$ et $\tau_2 = (4, 10, 10)$. Toutes les tâches appartiennent à la même couche et les allocations de quantum choisies sont $\Psi = \{\psi_1 = 2, \psi_2 = 2\}$ (figure 6.1) et $\Psi' = \{\psi_1 = 2, \psi_2 = 3\}$ (figure 6.2) .

Comme on peut le constater sur les figures 6.1 et 6.2 le temps de réponse de τ_2 est 6 avec un quantum de 2 et 8 avec un de 3. Cependant, les bornes supérieures sur les temps de réponse de τ_2 calculées avec la fonction proposée dans [1] sont de 4 avec l'allocationΨ ainsi qu'avec l'allocation Ψ'. Dans ce cas l'augmentation du quantum n'a pas modifié la borne supérieure sur le temps de réponse. A la suite de cette observation, nous énonçons la propriété d'une fonction de calcul de bornes supérieures de temps de réponse.

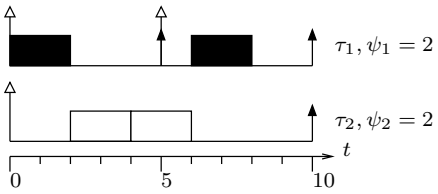


FIG. 6.1 – Ordonnancement de l'ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2\}$ avec Round-Robin et l'allocation de quantum $\Psi = \{\psi_1 = 2, \psi_2 = 2\}$.

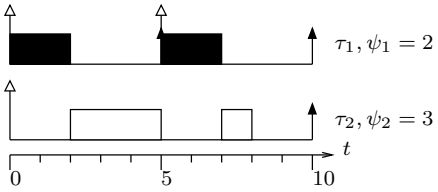


FIG. 6.2 – Ordonnancement de l'ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2\}$ avec Round-Robin et l'allocation de quantum $\Psi' = \{\psi_1 = 2, \psi_2 = 3\}$.

Propriété 1 *Soit f une fonction de calcul de bornes supérieures de temps de réponse et τ_i une tâche dans une couche RR. La borne supérieure sur le temps de réponse de τ_i calculée avec f est telle que :*

1. *augmenter (resp. réduire) ou conserver le quantum de τ_j,*
2. *tout en réduisant (resp. augmentant) ou conservant la valeur du quantum des autres tâches de la couche RR,*

ne change pas ou diminue (resp. augmente) la borne supérieure sur le temps de réponse de τ_i calculée avec f.

Nous montrons de manière général en annexe D.1 que la fonction [1] utilisée dans notre analyse d'ordonnancement garantit la propriété 1. De plus, nous utiliserons cette propriété pour réduire l'espace de recherche dans la section 6.4.

De plus, nous pouvons remarquer également que le temps de réponse d'une tâche peut augmenter quand on réduit l'ensemble de tâches ayant une priorité supérieure ou égale. L'ordonnancement de l'ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ où $\tau_1 = (C_1 = 1, T_1 = 14, \overline{D}_1 = 14)$, $\tau_2 = (3, 7, 7)$ et $\tau_3 = (6, 14, 14)$ avec une valeur de quantum globale au système égale à 3 est présenté sous les allocations :

- $\mathcal{P} = \{(\tau_1, 1, sched\_fifo), (\tau_2, 2, sched\_rr), (\tau_3, 2, sched\_rr)\}$ sur la figure 6.3,
- $\mathcal{Q} = \{(\tau_1, 2, sched\_fifo), (\tau_2, 1, sched\_rr), (\tau_3, 1, sched\_rr)\}$ sur la figure 6.4.

Comme on peut le constater sur les figures 6.3 et 6.4, le temps de réponse de τ_2 est 4 sous $\mathcal{P}$ tandis qu'il est égal à 5 sous $\mathcal{Q}$. Ici aussi, nous pouvons remarquer que les bornes supérieures calculées avec la fonction décrite dans [1] pour τ_2 et τ_3 sont respectivement de 7 et 13 avec les priorités $p_1 = 1, p_2 = p_3 = 2$ et sont respectivement de 6 et 12 avec les priorités $p_2 = p_3 = 1$, $p_1 = 2$. Dans ce cas la réduction de l'ensemble de tâches ayant une priorité supérieure a bien diminué la borne supérieure sur le temps de réponse. A la suite de cette observation, nous énonçons la propriété suivante :

Propriété 2 *Soit f une fonction de calcul de bornes supérieures de temps de réponse et τ_i une tâche d'une couche FPP ou RR. La borne supérieure sur le temps de réponse de τ_i calculée avec f est telle que :*

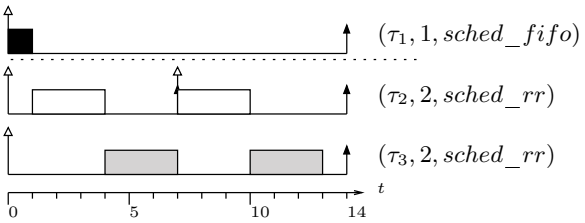


FIG. 6.3 – Ordonnancement de l'ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ avec les priorités $p_1 = 1$, $p_2 = p_3 = 2$ et les politiques $sched_1 = sched\_fifo$, $sched_2 = sched_3 = sched\_rr$ (allocation $\mathcal{P}$).

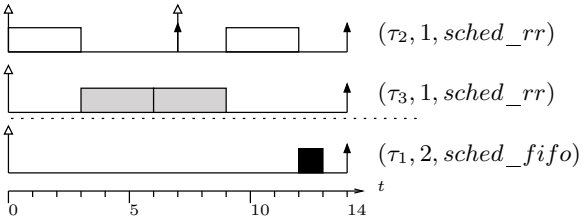


FIG. 6.4 – Ordonnancement de l'ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ avec les priorités $p_2 = p_3 = 1$, $p_1 = 2$ et les politiques $sched_2 = sched_3 = sched\_rr$, $sched_1 = sched\_fifo$ (allocation $\mathcal{Q}$).

1. *réduire ou conserver l'ensemble de tâches de priorité supérieure ou égale,*
2. *tout en conservant l'allocation de quantum inchangée dans sa couche RR (si τ_i est ordonnancée sous RR),*

diminue ou laisse inchangé la borne sur le pire temps de réponse de τ_i calculée avec f.

Nous proposons dans ce travail d'utiliser la fonction de calcul de borne supérieure [1] pour l'analyse d'ordonnancement. L'annexe D.1 et D.2 prouvent respectivement que cette fonction vérifie les propriétés 1 et 2. De plus, comme il est montré dans l'annexe D.2, un test d'ordonnancement utilisant une fonction de calcul de borne supérieure sur le temps de réponse vérifiant la propriété 2, permet d'utiliser une extension de l'algorithme d'Audsley (telle que réalisée dans les sections 6.3 et 6.4) et préserve l'optimalité, dans le sens où, s'il existe une allocation de priorités, de politiques et de quantum faisable qui peut être identifiée par le test considéré, l'algorithme trouve une solution faisable avec ce test s'il en existe au moins une avec ce test. Nous pourrions obtenir de meilleurs résultats (c'est-à-dire, trouver plus de configurations d'ordonnancement faisables) en utilisant une extension de l'algorithme d'Audsley avec un autre - meilleur - test d'ordonnancement tant qu'il possède la propriété 2.

6.3 Algorithme d'allocation optimale : cas du quantum constant

Cette section présente un algorithme d'allocation de priorité et de politique d'ordonnancement pour les systèmes Posix 1003.1b. Cette allocation est optimale quand l'analyse d'ordonnancement utilisée repose sur une fonction de calcul de borne supérieure sur le pire temps de réponse vérifiant la propriété 2 énoncée dans le §6.2.5. Nous étudions le cas où le quantum des tâches est constant pour tout le système,

tel qu'imposé dans la plupart des systèmes d'exploitation[3]. Dans ce contexte, la politique est induite par le nombre de tâches ayant le même niveau de priorité. En effet, si une seule tâche est au niveau de priorité i alors sa politique est FPP (*i.e.*, une couche RR ayant une seule tâche est strictement équivalente à une couche FPP, cf. §6.2.1), sinon sa politique est nécessairement RR. Le problème est donc réduit à l'allocation de priorité. Notre algorithme est une extension de l'algorithme d'Audsley [14], c'est pourquoi nous l'appelons *Audsley-RR-FPP* .

6.3.1 Algorithme *Audsley-RR-FPP*

De la même manière que pour l'algorithme d'Audsley, noté AA dans la suite (cet l'algorithme est donné §5.3.2), les priorités sont allouées de la plus faible priorité n à la plus importante 1 (ligne 3 dans l'algorithme 3). Cependant, à la différence de AA, l'algorithme ne cherche pas une seule tâche à chaque niveau de priorité, mais un ensemble de tâches (ligne 5 dans l'algorithme 2).

Principe de l'algorithme. L'idée principale est de mettre la plus grande charge de travail possible aux niveaux de faible priorité et d'ordonnancer ces tâches avec RR. Quand une instance $\tau_{i,j}$ a la même priorité que $\tau_{k,h}$ et que ces tâches sont ordonnancées sous RR, l'interférence de $\tau_{i,j}$ sur $\tau_{k,h}$ est moindre que si elle avait été ordonnancée avec une plus haute priorité. Le même argument est valide pour l'interférence de $\tau_{k,h}$ sur $\tau_{i,j}$. Donc, un ensemble de tâches non faisable avec l'algorithme FPP seul peut être faisable avec RR ; un exemple peut être trouvé dans [26]. Cependant, dans le cas général, l'utilisation combinée des deux politiques est plus efficace et permet de trouver une allocation de priorités et de politiques optimale.

Étapes de l'algorithme. A chaque niveau de priorité, l'algorithme *Audsley-RR-FPP* cherche un sous-ensemble $\mathcal{T}_i$ faisable dans l'ensemble $\mathcal{R}$ (ligne 5 de l'algorithme 2) où $\mathcal{R}$ est composé de l'ensemble des tâches n'ayant pas encore de priorité et de politique. L'algorithme considère un à un tous les ensembles possibles dans $\mathcal{R}$ jusqu'à en trouver un de faisable ou jusqu'à ce que tous les ensembles soient envisagés. Dans ce dernier cas, le système n'est pas ordonnançable (lignes $7-8$ de l'algorithme 2). Sinon, un ensemble ordonnançable $\mathcal{T}_i$ a été trouvé. Précisément, $\mathcal{T}_i$ est ordonnançable quand toutes les tâches de $\mathcal{T}_i$ sont faisables à la priorité i (ordonnancées avec RR si $\#\mathcal{T}_i > 1$ ou avec FPP sinon) tandis que toutes les tâches n'ayant pas d'allocation (*i.e.*, tâches dans $\mathcal{R}\backslash\mathcal{T}_i$) ont une priorité plus grande que i. A chaque itération, *Audsley-RR-FPP* alloue la priorité et la politique d'au moins une tâche (lignes 11 à 16 de l'algorithme 2). Notons que lorsque RR est utilisé au moins une fois, moins de n niveaux de priorité sont utilisés (sortie anticipée ligne 20 de l'algorithme 2).

Chercher un ensemble faisable $\mathcal{T}_i$. Dans l'ensemble $\mathcal{R}$, il y a a $2^{\#\mathcal{R}}-1$ ensembles $\mathcal{T}_i$ possibles qui peuvent être alloués au niveau de priorité i (ligne 5 de l'algorithme 2). Nous utilisons un arbre binaire pour refléter les choix de priorité. La recherche d'un sous-ensemble faisable à la priorité i est réalisée en explorant cet arbre. Nous expliquons tout d'abord le principe de base de la recherche, puis nous présenterons dans le §6.3.2 une méthode permettant d'accélérer la recherche en coupant des branches de l'arbre qui ne peuvent contenir des solutions. Afin d'illustrer nos propos, la figure 6.5 montre l'arbre de

[3]Dans cette section, le quantum est une valeur unique constante pour tout le système, les notations distinguant les valeurs de quantum pour chaque tâche sont omises dans cette section.

Entrée : ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2..., \tau_n\}$
Sortie : allocation de priorité et de politiques d'ordonnancement ordonnançable $\mathcal{P}$
Données : i : niveau de priorité à allouer
$\mathcal{R}$: ensemble de tâche sans priorité allouée
$\mathcal{P}$: allocation de priorité et de politiques partielle

1 $\mathcal{R} = \mathcal{T}$;
2 $\mathcal{P} = \emptyset$;
3 **pour** $i = n$ **à** *1* **faire**
4 `essayer d'allouer la priorité` i `:`
5 chercher un sous-ensemble de tâche ordonnançable $\mathcal{T}_i$ dans $\mathcal{R}$
6 **si** *pas de sous-ensemble* $\mathcal{T}_i$ *ordonnançable à la priorité* i **alors**
7 `échec, retourner l'allocation partielle :`
8 **retourner** $\mathcal{P}$;
9 **sinon**
10 soit $\mathcal{T}_i$ un sous-ensemble ordonnançable à la priorité i;
11 `allouer la priorité et la politique :`
12 **si** $\#\mathcal{T}_i = 1$ **alors**
13 $\mathcal{P} = \mathcal{P} \cup \{(\tau_k, i, sched\_fifo)\}_{\tau_k \in \mathcal{T}_i}$;
14 **sinon**
15 $\mathcal{P} = \mathcal{P} \cup \{(\tau_k, i, sched\_rr)\}_{\tau_k \in \mathcal{T}_i}$;
16 **finsi**
17 `enlever` $\mathcal{T}_i$ `de` $\mathcal{R}$ `:`
18 $\mathcal{R} = \mathcal{R} \backslash \mathcal{T}_i$;
19 **finsi**
20 **si** $\mathcal{R} = \emptyset$ **alors retourner** $\mathcal{P}$;
21 **finpour**

Algorithme 2 : algorithme *Audsley-RR-FPP* avec un quantum constant global au système.

recherche correspondant à l'ensemble $\mathcal{R} = \{\tau_1, \tau_2, \tau_3\}$. Chaque arc est étiqueté soit avec l'étiquette "$= i$" (*i.e.*, priorité égale à i) soit avec "$> i$" (*i.e.*, priorité supérieure à i). Une étiquette "$= i$" (resp. "$> i$") sur l'arc entre un noeud de profondeur k et un noeud de profondeur $k+1$ signifie que la $(k+1)^{\text{ème}}$ tâche de $\mathcal{R}$ appartient à la couche de priorité i (resp. appartient à une couche de priorité supérieure à i). Donc, un noeud de profondeur k modélise les choix effectués pour les k premières tâches de $\mathcal{R}$. Par exemple, sur la figure 6.5, le noeud e implique que τ_1 appartient à la couche de priorité i tandis que τ_2 a une plus grande priorité. Chaque feuille modélise une allocation complète pour un niveau de priorité i, par exemple la feuille g correspond à l'ensemble $\mathcal{T}_i = \{\tau_1, \tau_2\}$.

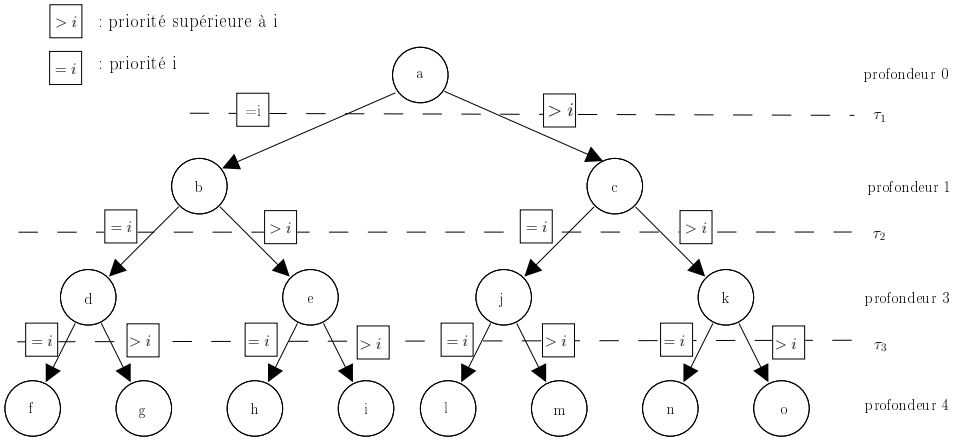


FIG. 6.5 – Arbre construit lors de la recherche d'un sous-ensemble faisable dans $\mathcal{R} = \{\tau_1, \tau_2, \tau_3\}$ à la priorité i. Par exemple, le noeud b modélise l'allocation partielle où τ_1 a la priorité i tandis que le noeud c signifie que τ_1 a une plus grande priorité.

Nous utilisons une recherche en profondeur. L'algorithme considère le premier enfant du noeud qu'il rencontre et descend de plus en plus profondément jusqu'à atteindre une feuille, *i.e.*, jusqu'à ce que l'ensemble $\mathcal{T}_i$ soit complètement défini. Quand une feuille est atteinte, la faisabilité de l'ensemble $\mathcal{T}_i$ est évaluée. Si $\mathcal{T}_i$ est faisable, l'algorithme termine en retournant l'ensemble $\mathcal{T}_i$, sinon il remonte jusqu'à atteindre un noeud dont il n'a pas exploré tous les enfants.

6.3.2 Complexité et amélioration

Taille de l'espace de recherche. Navet et Migge [26] ont montré qu'il y a $\sum_{k=1}^{n} \sum_{i=0}^{k} (-1)^{(k-i)} \binom{k}{i} i^n$ configurations d'ordonnancement différentes pour un ensemble de n tâches (ex. : $\approx 10^8$ configurations avec 10 tâches). Comme on peut le constater sur la figure 6.6, l'espace de recherche, *i.e.*, le nombre de configurations différentes, croît plus qu'exponentiellement avec le nombre de tâches. Il est donc évident qu'une recherche exhaustive n'est pas envisageable en pratique pour la plupart des systèmes temps réel.

Complexité de l'algorithme *Audsley-RR-FPP*. A chaque niveau de priorité, l'algorithme *Audsley-RR-FPP* cherche un sous-ensemble $\mathcal{T}_i$ faisable dans l'ensemble $\mathcal{R}$ (ligne 5 de l'algorithme 2). Il y a $2^{\#\mathcal{R}} - 1$ différents sous-ensembles $\mathcal{T}_i$ dans $\mathcal{R}$ à chaque niveau de priorité. Comme au moins une tâche est allouée à chaque niveau de priorité en commençant par la plus faible n (sinon l'allocation n'est pas faisable, cf.

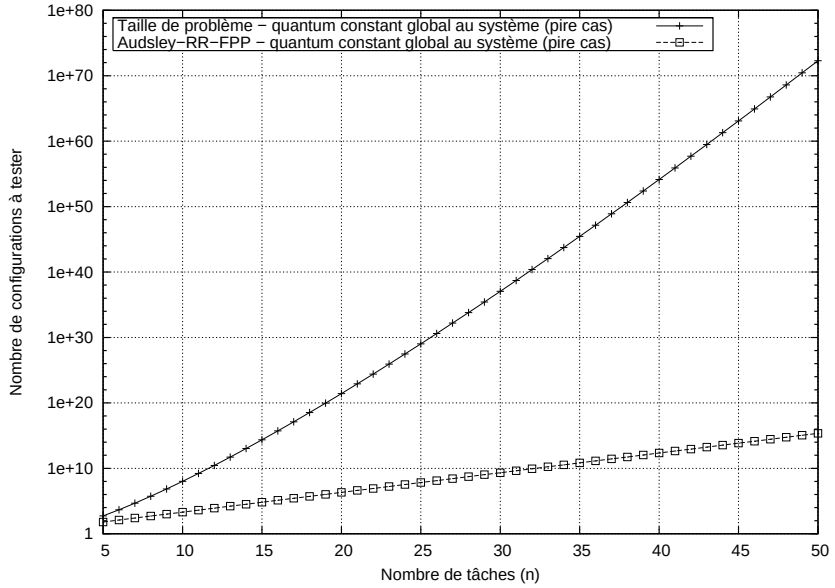


FIG. 6.6 – Complexité du problème pour un nombre de tâches variant de 5 à 50 (échelle log. pour l'axe des ordonnées) quand le quantum des tâches est constant pour tout le système.

lignes 8 de l'algorithme 2), le nombre de tâches appartenant à $\mathcal{R}$ est plus petit ou égal à i quand le niveau de priorité i est considéré. L'algorithme considère au pire $\sum_{i=1}^{n}(2^i-1) = 2^{n+1}-(n+2)$ allocations. Cette complexité est présentée en fonction du nombre de tâches sur la figure 6.6. Par exemple, la complexité est égale à 2036 pour un ensemble de 10 tâches. La figure 6.6 présente également la taille de l'espace de recherche. Bien qu'une réduction significative de l'espace de recherche soit effectuée comparée à un algorithme de recherche exhaustif naïf, la complexité reste exponentielle en le nombre de tâches. En pratique, notre proposition ne convient donc pas pour des ensembles de grande taille (*i.e.*, plus de 50 tâches) qui seront, par exemple, mieux traité par des heuristiques guidant la recherche vers des parties prometteuses de l'espace de recherche. Une telle heuristique est proposée en section 6.5.

Réduction de complexité. Comme nous l'avons vu précédemment, l'algorithme *Audsley-RR-FPP* effectue une recherche exhaustive à chaque niveau de priorité i. Il est possible de réduire dans une certaine limite le nombre d'ensembles à considérer. En effet, nous pouvons identifier et couper les branches de l'arbre de recherche qui représente nécessairement des ensembles $\mathcal{T}_i$ non faisables grâce à la propriété 2.

Avec l'algorithme expliqué au §6.3.1, la faisabilité est évaluée aux feuilles quand toutes les tâches se sont vu attribuer une priorité. Ici, l'idée est d'évaluer la faisabilité aux noeuds intermédiaires en assignant une priorité plus faible aux tâches pour lesquelles le choix de priorité n'a pas encore été effectué. Avec cette configuration, si une tâche τ_i ayant la priorité i est non-faisable, il n'est pas utile de considérer les enfants de ce noeud. En effet, selon la propriété 2, le temps de réponse de cette tâche ne peut pas diminuer, car l'allocation de priorité des enfants de ce noeud peut seulement ajouter des tâches à l'ensemble des tâches de même ou de plus haute priorité de cette tâche. Tous les enfants de ce noeud correspondent donc à des allocations de priorité non-ordonnançable.

La procédure de réduction de l'espace de recherche, proposée dans ce paragraphe, permet de diminuer le nombre moyen de configurations à tester par *Audsley-RR-FPP* d'une manière significative. Par

exemple[4], comme on peut le constater sur la figure 6.7, le nombre moyen de configurations que l'on a besoin de tester avec notre méthode pour 20 tâches est à peu près 300 fois moins important que le nombre de configurations à tester sans notre amélioration. Il est important de noter que le ratio d'amélioration augmente avec le nombre de tâches (de 3 à 300 ici).

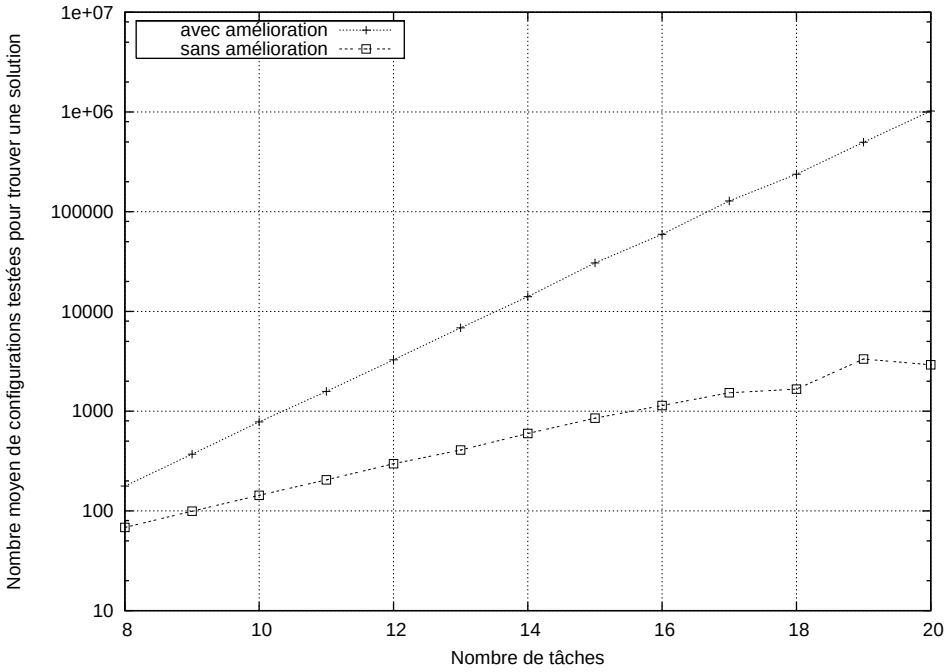


FIG. 6.7 – Nombre moyen de configurations testées pour trouver une solution avec et sans la procédure de réduction de l'espace de recherche. La charge du CPU moyenne est égale à 0.84 avec un nombre de tâches compris entre 8 et 20. Chaque point est la valeur moyenne obtenue avec 100 simulations (échelle log. pour l'axe des ordonnés).

Cependant, malgré la procédure de réduction, la limite de notre proposition est sa complexité exponentielle. D'autres simulations montrent que des ensembles de tâches de 30 à 35 tâches peuvent être traités en quelques heures. Au-delà de ce seuil, la complexité et donc la durée d'exécution posent problème sur certains ensembles de tâches pour lesquels l'algorithme n'est pas approprié. Des ensembles de tâches plus grands peuvent être traités en pratique par des heuristiques au dépens de l'optimalité du résultat, cf. section 6.5.

6.4 Algorithme d'allocation optimal : cas du quantum spécifique aux tâches

Nous étendons l'algorithme proposé dans la section 6.3 au cas où la valeur du quantum est spécifique à chaque tâche et peut être choisie dans l'intervalle $[\psi_{\min}, \psi_{\max}]$, où $\psi_{\min}$ et $\psi_{\max}$ sont des nombres naturels

[4]Les conditions d'expérimentations sont données en section 6.6.

spécifiques au système d'exploitation ou choisis par le concepteur de l'application. L'algorithme correspondant est appelé *Audsley-RR-FPP**. Ce nouvel algorithme reste optimal, cf. annexe D section D.3, quand la faisabilité est évaluée avec un test de faisabilité utilisant une fonction de calcul de borne supérieure sur le pire temps de réponse vérifiant la propriété 2 décrite dans le §6.2.5. Avec les hypothèses faites en section 6.2, la politique est impliquée par le nombre de tâches ayant le même niveau de priorité. En effet, si une seule tâche a le niveau de priorité i alors sa politique est FPP (*i.e.*, une couche RR ayant une tâche est strictement équivalente à une couche FPP, cf. §6.2.1), sinon sa politique est nécessairement RR . Le problème est donc réduit à l'allocation des priorités et des quanta aux tâches appartenant à une couche RR.

6.4.1 Algorithme *Audsley-RR-FPP**

De la même façon que l'algorithme *Audsley-RR-FPP, Audsley-RR-FPP**cherche un ensemble de tâches faisable à chaque niveau de priorité. Cependant, comme la valeur du quantum peut-être choisie, l'algorithme examine, pour chaque ensemble de tâches possible, toutes les allocations de quantum possibles jusqu'à en trouver une qui corresponde (ligne 5 de l'algorithme 3).

Entrée : ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2..., \tau_n\}$
Sortie : allocation de priorités, de politiques et da quantum $\mathcal{P}_k = (\mathcal{P}, \Psi_k)$
Données : i : niveau de priorité à allouer
$\mathcal{R}$:ensemble de tâches sans priorité allouée
$\mathcal{P}$: allocation partielle de priorités et de politiques
$\Psi_{\mathcal{P}}$: allocation partielle de quantum

1 $\mathcal{R} = \mathcal{T}$;
2 $\mathcal{P} = \emptyset$;
3 **pour** $i = n$ **à** *1* **faire**
4 `essayer d'allouer la priorité` i `:`
5 chercher un sous-ensemble de tâches $\mathcal{T}_i$ ordonnançables sous l'allocation $\{\psi_k\}_{\tau_k \in \mathcal{T}_i}$ dans $\mathcal{R}$
6 **si** *pas de sous-ensemble* $\mathcal{T}_i$ *ordonnançable à la priorité* i **alors**
7 `échec, retourner l'allocation partielle :`
8 **retourner** $(\mathcal{P}, \Psi_{\mathcal{P}})$;
9 **sinon**
10 soit $\mathcal{T}_i$ un sous-ensemble ordonnançable à la priorité i avec l'allocation de quantum $\{\psi_k\}_{\tau_k \in \mathcal{T}_i}$;
11 `allouer les priorités, les politiques et les quantum :`
12 **si** $\#\mathcal{T}_i = 1$ **alors**
13 $\mathcal{P} = \mathcal{P} \cup \{(\tau_k, i, sched\_fifo)\}_{\tau_k \in \mathcal{T}_i}$;
14 **sinon**
15 $\mathcal{P} = \mathcal{P} \cup \{(\tau_k, i, sched\_rr)\}_{\tau_k \in \mathcal{T}_i}$;
16 $\Psi_{\mathcal{P}} = \Psi_{\mathcal{P}} \cup \{\psi_k\}_{\tau_k \in \mathcal{T}_i}$;
17 **finsi**
18 `enlever` $\mathcal{T}_i$ `de` $\mathcal{R}$ `:`
19 $\mathcal{R} = \mathcal{R} \backslash \mathcal{T}_i$;
20 **finsi**
21 **si** $\mathcal{R} = \emptyset$ **alors retourner** $(\mathcal{P}, \Psi_{\mathcal{P}})$;
22 **finpour**

Algorithme 3 : algorithme *Audsley-RR-FPP** avec un quantum spécifique à chaque tâche.

Étape de l'algorithme. A chaque niveau de priorité, l'algorithme *Audsley-RR-FPP* cherche un sous-ensemble $\mathcal{T}_i$ faisable dans l'ensemble $\mathcal{R}$ (ligne 5 de l'algorithme 3) où $\mathcal{R}$ est composé de l'ensemble des tâches n'ayant pas encore de priorité, de politique et de quantum. L'algorithme considère une à une toutes les allocations possibles de quantum pour tous les ensembles possibles dans $\mathcal{R}$ jusqu'à trouver une configuration faisable ou jusqu'à avoir considéré toutes les configurations possibles. Dans le dernier cas, le système est non-faisable (lignes 7 and 8 de l'algorithme 3). Sinon, nous avons trouvé un sous-ensemble faisable, noté $\mathcal{T}_i$, qui dans le cas de RR, possède l'allocation de quantum $\{\psi_k\}_{\tau_k \in \mathcal{T}_i}$ (ligne 10 de l'algorithme 3). Précisément, $\mathcal{T}_i$ est ordonnançable quand toutes les tâches de $\mathcal{T}_i$ sont faisables à la priorité i (ordonnancées avec RR si $\#\mathcal{T}_i > 1$ ou avec FPP sinon) tandis que toutes les tâches n'ayant pas d'allocation (*i.e.*, tâches dans $\mathcal{R} \backslash \mathcal{T}_i$) ont une priorité plus grande que i. A chaque itération, *Audsley-RR-FPP* alloue la priorité et la politique d'au moins une tâche (lignes 11 à 16 de l'algorithme 3). Notons que lorsque RR est utilisée au moins une fois, moins de n niveaux de priorité sont utilisés (sortie anticipée ligne 21 de l'algorithme 3).

Chercher un ensemble faisable $\mathcal{T}_i$. Il y a $2^{\#\mathcal{R}} - 1$ sous-ensembles $\mathcal{T}_i$ de $\mathcal{R}$ pouvant être alloués à la priorité i (ligne 5). Comme le quantum peut prendre $\|\psi\| = \psi_{\max} - \psi_{\min} + 1$ valeurs différentes, il existe $\|\psi\|^{\#\mathcal{T}_i}$allocations de quantum pour chaque ensemble $\mathcal{T}_i$. La recherche exhaustive en arbre utilisée pour fixer les priorités est la même que pour l'algorithme *Audsley-RR-FPP*, cf. section 6.3. Dans la suite, nous appelons l'arbre modélisant les choix de priorité *arbre-prio*. Nous expliquons ici comment nous utilisons un arbre similaire pour choisir l'allocation de quantum pour chaque ensemble $\mathcal{T}_i$ possible. Nous présentons au §6.4.2 une méthode permettant d'accélérer la recherche en coupant des branches de l'arbre qui ne peuvent pas contenir de solutions.

D'une manière similaire à l'algorithme *Audsley-RR-FPP*, la recherche d'un sous-ensemble $\mathcal{T}_i$ est réalisée en explorant l'*arbre-prio* selon une stratégie en profondeur d'abord. Quand une feuille est atteinte, la faisabilité est évaluée. Si $\mathcal{T}_i$ est faisable, l'algorithme se termine et renvoie l'ensemble $\mathcal{T}_i$, sinon il remonte jusqu'au premier noeud dont il n'a pas encore exploré tous les enfants. Pour évaluer la faisabilité d'un sous-ensemble $\mathcal{T}_i$, toutes les allocations de quantum possibles sont considérées. D'une manière analogue aux choix de priorité, un arbre -appelé *arbre-quantum*- reflète les valeurs des quanta choisies.

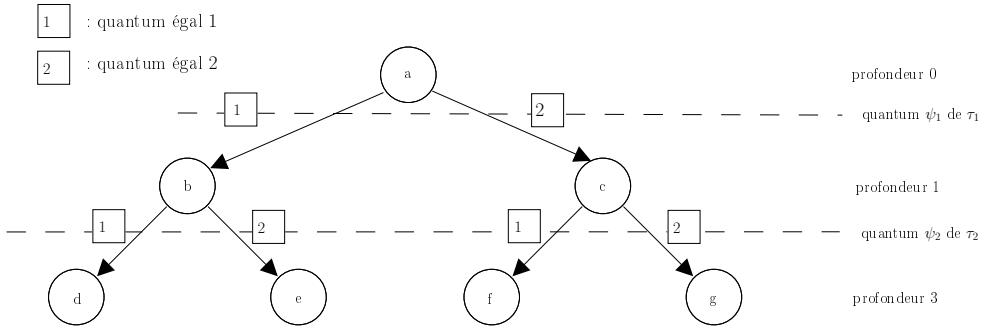


FIG. 6.8 – Arbre construit lors de la recherche d'une allocation de quantum faisable dans $\mathcal{T}_i = \{\tau_1, \tau_2\}$ quand le quantum peut être choisie dans l'intervalle $[1, 2]$ ($\psi_{\min} = 1$ et $\psi_{\max} = 2$). Par exemple, le noeud b modélise l'allocation partielle où le quantum ψ_1 de τ_1 est 1 et la feuille f signifie que $\psi_1 = 2$ et $\psi_2 = 1$.

La recherche d'une allocation de quantum faisable est faite en explorant l'*arbre-quantum*. La figure 6.8

présente l'*arbre-quantum* correspondant à l'ensemble $\mathcal{T}_i = \{\tau_1, \tau_2\}$ quand le quantum peut être choisie dans l'intervalle $[1, 2]$ ($\psi_{\min} = 1$ et $\psi_{\max} = 2$). Un noeud a $\|\psi\|$ enfants où chaque enfant modélise une valeur de quantum différente. Chaque arc est étiqueté avec une valeur. Les arcs entre les noeuds de profondeur k et $k+1$ sont étiquetés avec la valeur correspondante à la valeur du quantum de la $(k+1)^{\text{ème}}$ tâche de $\mathcal{T}_i$. Donc, un noeud de profondeur k modélise les choix effectués pour les k premières tâches de $\mathcal{T}_i$. Par exemple, sur la figure 6.8, le noeud c implique que le quantum de τ_1 est 2. Chaque feuille modélise une allocation de quantum complète pour l'ensemble $\mathcal{T}_i$ considéré. Par exemple, la feuille f correspond à l'allocation $\psi_{_1} = 2$ et $\psi_2 = 1$. Comme dans la recherche en arbre utilisée pour fixer les priorités, une recherche en profondeur d'abord est utilisée pour explorer l'*arbre-quantum*.

6.4.2 Complexité et amélioration

Taille de l'espace de recherche. Nous calculons une borne inférieure sur le nombre de configurations d'ordonnancement différentes avec un ensemble de n tâches, c'est-à-dire la taille de l'espace de recherche. Répartir n tâches dans différentes couches non-vides est similaire à diviser un ensemble de n éléments en sous-ensembles non-vides. Soit k le nombre de couches. Le nombre possible d'allocations est égal, par définition, au nombre de Stirling du second ordre (cf. [78], page 824) :

$$\frac{1}{k!}\sum_{i=0}^{k}(-1)^{(k-i)}\left(\begin{array}{c} k \\ i \end{array}\right) i^n,$$

où $\left(\begin{array}{c} k \\ i \end{array}\right)$ est le coefficient binomial.

La complexité dépend du nombre de tâches à ordonnancer sous RR, car la valeur de leur quantum doit être choisie. Quand il y a k couches, au moins $n-k+1$ tâches sont dans une couche RR (*i.e.*, $n-k+1$ tâches dans une seule couche RR et une tâche dans les $k-1$ couches FPP restantes) et au plus $\max(n, 2(n-k))$ (*i.e.*, les tâches sont réparties le plus uniformément possible dans les couches RR). Du fait que la valeur du quantum peut prendre $\|\psi\| = \psi_{\max} - \psi_{\min} + 1$ différentes valeurs, il y a entre $\|\psi\|^{n-k+1}$ et $\|\psi\|^{\max(n,2(n-k))}$ allocations différentes pour une configuration de k couches.

De plus, un ensemble de n tâches peut être divisé en $k = 1, 2, ..., n$ couches et il existe parmi k couches $k!$ ordres de priorité différents. Donc, une borne inférieure sur la taille de l'espace de recherche du problème d'allocation de priorité, de politiques et de quantum pour un ensemble de n tâches est

$$\sum_{k=1}^{n}\sum_{i=0}^{k}(-1)^{(k-i)} \cdot \left(\begin{array}{c} k \\ i \end{array}\right) \cdot i^n \cdot \|\psi\|^{n-k+1}.$$

Nous pouvons obtenir une borne supérieure en remplaçant $\|\psi\|^{n-k+1}$ par $\|\psi\|^{\max(n,2(n-k))}$.

Comme on peut le voir sur la figure 6.9, l'espace de recherche est composé d'à peu près $2 \cdot 10^{11}$ configurations d'ordonnancement pour un ensemble de 10 tâches. L'espace de recherche croît plus qu'exponentiellement. Donc, en pratique, un parcours exhaustif des solutions n'est pas envisageable dans la plupart des problèmes temps réels.

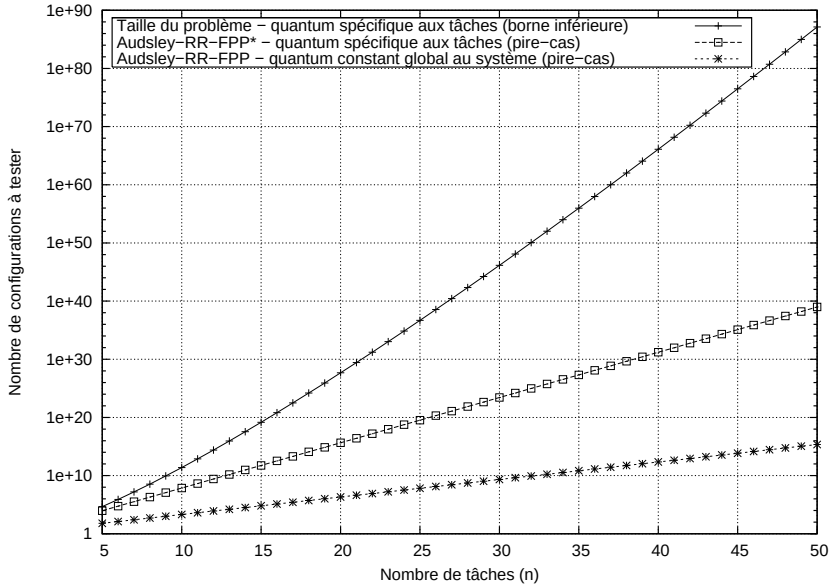


FIG. 6.9 – Complexité du problème pour un nombre de tâches variant de 5 à 50 (échelle log. pour l'axe des ordonnées). Dans le cas d'un quantum spécifique à chaque tâche, la valeur du quantum peut-être choisi dans l'intervalle $[1, 5]$.

Complexité de l'algorithme *Audsley-RR-FPP.** Notre algorithme cherche à chaque niveau de priorité un sous-ensemble $\mathcal{T}_i$ de $\mathcal{R}$ qui soit faisable à la priorité i (ligne 5). Le nombre de tâches appartenant à $\mathcal{R}$ lorsque *Audsley-RR-FPP**considère le niveau de priorité i est inférieur ou égal à i, car au moins une tâche est allouée à chaque niveau de priorité. De plus, on sait qu'il y a $\|\psi\|^k$ différentes allocations de quantum pour un sous-ensemble de k tâches. Donc, l'algorithme, à chaque niveau de priorité i, examine $\sum_{j=1}^{i} \begin{pmatrix} i \\ j \end{pmatrix} \cdot \|\psi\|^j = (\|\psi\| + 1)^i - 1$ allocations dans le pire des cas. Pour les niveaux de priorité allant de 1 à n, l'algorithme considère dans le pire des cas

$$\sum_{i=1}^{n} (\|\psi\| + 1)^i - 1 = \frac{1 - (\|\psi\| + 1)^{n+1}}{1 - (\|\psi\| + 1)} - (n + 1)$$

allocations distinctes. La figure 6.9 présente la complexité en fonction du nombre de tâches. Par exemple, la complexité pour un ensemble de 10 tâches avec $\psi_{min} = 1$ et $\psi_{max} = 5$ est à peu près égal à $72 \cdot 10^6$. Afin de pouvoir comparer, la figure 6.9 présente également la taille de l'espace de recherche et la complexité de *Audsley-RR-FPP* dans le pire des cas, *i.e.*, le cas où le quantum est une constante globale au système. Comme nous l'avons déjà remarqué en section 6.3, la complexité reste exponentielle malgré une réduction significative de l'espace de recherche en comparaison d'une recherche exhaustive,

Réduction de complexité. L'idée est similaire à celle utilisée pour *Audsley-RR-FPP*. Nous voulons réduire le nombre d'ensembles à prendre en compte. Dans cette optique, nous montrons qu'il est possible d'identifier au plus tôt dans l'algorithme de recherche des allocations de priorité et de politiques qui sont non-faisables quelque soit l'allocation de quantum grâce aux propriétés 1 et 2. Il est possible d'identifier et de couper des branches de l'*arbre-prio* qui proposent des sous-ensembles $\mathcal{T}_i$ non-faisables quelque soit

l'allocation de quantum. De plus, grâce à la propriété 1, on peut réduire d'une façon similaire le nombre d'allocations de quantum à considérer dans l'*arbre-quantum* d'un sous-ensemble $\mathcal{T}_i$.

Avec l'algorithme expliqué au §6.4.1, la faisabilité est évaluée aux feuilles quand toutes les tâches se sont vues attribuer une priorité en testant toutes les allocations de quantum possibles. Ici, l'idée est d'évaluer la faisabilité aux noeuds intermédiaires en assignant une priorité plus faible aux tâches pour lesquelles le choix de priorité n'a pas encore été effectué. Avec cette configuration, si une tâche τ_i ayant la priorité i est non-faisable quelque soit l'allocation de quantum au niveau de priorité i, il n'est pas utile de considérer les enfants de ce noeud. En effet, selon la propriété 2, le temps de réponse de cette tâche ne peut pas diminuer, car l'allocation de priorité des enfants de ce noeud ajoute des tâches à l'ensemble des tâches ayant une priorité supérieure ou égale à cette tâche. Tous les enfants de ce noeud correspondent donc à des allocations de priorités non-ordonnançables. Maintenant, il reste à identifier les allocations de priorité et de politiques qui sont non-faisables quelque soit l'allocation de quantum.

Afin d'identifier si pour un ensemble $\mathcal{T}_i$ donné il existe une allocation faisable de quantum, nous explorons l'*arbre-quantum* correspondant à cet ensemble, cf. figure 6.8. Comme pour l'*arbre-prio* nous pouvons couper des branches de l'*arbre-quantum* de l'ensemble $\mathcal{T}_i$. L'idée est encore d'évaluer la faisabilité aux noeuds intermédiaires. Comme une allocation partielle de quantum est définie aux noeuds intermédiaires, nous allouons le plus petit quantum possible aux tâches de $\mathcal{T}_i$ dont la valeur du quantum n'est pas encore fixée. Dans ce cas, il n'est pas utile de considérer les enfants d'un noeud intermédiaire lorsqu'une tâche τ_k, pour laquelle le quantum est déjà défini au niveau de ce noeud, est non-faisable. En effet, grâce à la propriété 1, le temps de réponse de τ_k peut seulement augmenter quand les enfants de ce noeud sont considérés car la valeur du quantum des tâches de $\mathcal{T}_i$ qui n'ont pas encore été considérées ne peut que augmenter.

La procédure de réduction de l'espace de recherche proposée dans ce paragraphe permet de diminuer le nombre moyen de configurations à tester par *Audsley-RR-FPP** d'une manière significative. Par exemple, pour un ensemble de 10 tâches, l'algorithme examine en moyenne seulement 4000 configurations pour obtenir une solution ou pour conclure que l'ensemble de tâches est non-faisable contre $7 \cdot 10^7$ tests à effectuer dans le pire cas. Contrairement au cas d'une valeur globale constante pour le quantum, nous n'avons pas été en mesure de comparer l'algorithme *Audsley-RR-FPP** avec et sans amélioration. En effet, ces tests étaient trop coûteux en puissance et temps de calcul pour des ensembles de tâches supérieurs à 10 tâches.

6.5 Approches heuristiques pour de plus grands ensembles de tâches

La complexité exponentielle de *Audsley-RR-FPP* et *Audsley-RR-FPP** est due au nombre de sous-ensembles $\mathcal{T}_i$ différents auxquels l'on peut assigner la priorité i, cf. §6.3.2. Nous présentons ici une heuristique dans le cas d'une valeur de quantum globale au système qui prend en compte un nombre limité d'ensemble $\mathcal{T}_i$ à chaque niveau de priorité i. Cet algorithme, dénommé *Load-RR-FPP,* a une complexité polynomiale et est mieux adapté aux plus grand ensembles de tâches. Cet algorithme s'exécute en $O(n^3)$ et permet de traiter des ensembles allant jusqu'à 100 tâches.

6.5.1 *Heuristique Load-RR-FPP*

Un algorithme similaire à celui de la section 6.3 est utilisé pour trouver une configuration faisable. Seule la recherche d'un ensemble faisable $\mathcal{T}_i$ à chaque niveau de priorité diffère (ligne 5 de l'algorithme 2). Notre heuristique essaye de placer les tâches avec le plus grand taux d'utilisation au niveau de priorité i considéré. Partager le CPU avec RR permet aux grosses tâches d'être faisables tout en étant ordonnancées aux niveaux de faible priorité et donc, permet de réduire l'interférence causée aux autres tâches.

Soit $\mathcal{L}$ la liste des tâches de $\mathcal{R}$ (*i.e.*, les tâches auxquelles aucune priorité et politique n'ont été attribuées) classée par ordre décroissant du taux d'utilisation (C_k/T_k). Soit $\mathcal{L}_k$ la $k^{\text{ème}}$ tâche de $\mathcal{L}$, *i.e.*, la tâche avec le $k^{\text{ème}}$ plus grand taux d'utilisation.

1. (Créer $\mathcal{L}$, *i.e.*, classer les tâches de $\mathcal{R}$ par ordre décroissant de taux d'utilisation.)
2. (Chercher un ensemble $\mathcal{T}_i$ faisable.) Mettre dans $\mathcal{T}_i$ la tâche avec le plus grand taux d'utilisation, *i.e.*, $\mathcal{L}_1$. Puis, ajouter itérativement la tâche suivante avec le plus grand taux d'utilisation, *i.e.*, $\mathcal{L}_2$,$\mathcal{L}_3$, $\mathcal{L}_4$ etc.
   Pour j de 1 à $\#\mathcal{L}$,
   (a) (Placer les j premières tâches de $\mathcal{L}$ dans $\mathcal{T}_i$.) $\mathcal{T}_i = \{\mathcal{L}_1, ..., \mathcal{L}_j\}$,
   (b) (Tester la faisabilité.)
      i. ($\mathcal{T}_i$ faisable, fin.) Retourner l'ensemble faisable $\mathcal{T}_i$ à la priorité i.
      ii. ($\mathcal{T}_i$ non-faisable, continuer.) Sinon continuer.
3. (Échec.) Pas d'ensemble $\mathcal{T}_i$ faisable trouvé, retourner $\emptyset$.

Cette heuristique peut être encore améliorée : quand l'ensemble $\mathcal{T}_i$ n'est pas faisable à l'étape 2(b)ii, nous enlevons itérativement de $\mathcal{T}_i$ les tâches étant non-faisables. L'étape 2(b)ii devient :

1. ($\mathcal{T}_i$ non-faisable, enlever les tâches non-faisables.) Sinon enlever itérativement de $\mathcal{T}_i$ les tâches non-faisables. Nous enlevons tout d'abord les tâches avec les plus grands taux d'utilisation.
   Tant qu'il y a des tâches non-faisables dans $\mathcal{T}_i$,
   (a) (Enlever la tâche non-faisable avec le plus grand taux d'utilisation.)
   (b) (Tester la faisabilité.)
      i. ($\mathcal{T}_i$ faisable, fin.) Envoyer l'ensemble $\mathcal{T}_i$ faisable à la priorité i.
      ii. ($\mathcal{T}_i$ non-faisable, continuer.) Sinon continuer.

Comme nous le verrons dans la section 6.6 des expérimentations, cette heuristique est très efficace et trouve un ensemble faisable $\mathcal{T}_i$ dans la plupart des cas où un tel ensemble existe.

6.5.2 Complexité dans le pire-cas

A chaque niveau de priorité i, l'algorithme cherche un sous-ensemble $\mathcal{T}_i$ de $\mathcal{R}_i$ (ligne 5 de l'algorithme 2). $\mathcal{R}_i$ est l'ensemble de tâches $\mathcal{R}$ dont la priorité et la politique ne sont pas encore allouées quand le niveau de priorité i est considéré. La complexité est égale à :

$$\sum_{i=1}^{n} \text{comp\_}\mathcal{R}_i \ , \tag{6.7}$$

où comp_$\mathcal{R}_i$ est le nombre d'ensembles testés dans le pire des cas pour un ensemble $\mathcal{R}_i$ donné.

Comme au moins une tâche est affectée à chaque niveau de priorité, le nombre de tâches appartenant à $\mathcal{R}_i$ est inférieur à i, *i.e.*, $\#\mathcal{R}_i \leq i$. Pour un ensemble donné $\mathcal{R}_i$ notre heuristique ajoute itérativement la tâche avec le plus fort taux d'utilisation de $\mathcal{R}_i$ (étape 2), il y a initialement $\#\mathcal{R}_i$ différents ensembles testés. Cependant, quand un ensemble $\mathcal{T}_i$ est non-faisable, nous enlevons les tâches non-faisables une à une (étape 2(b)ii améliorée). Au pire, toutes les tâches sont non-faisables et donc $\#\mathcal{T}_i$ différents ensembles $\mathcal{T}_i$ sont considérés à chaque fois que l'heuristique ajoute une nouvelle tâche. Le nombre d'ensemble $\mathcal{T}_i$ considéré pour un ensemble $\mathcal{R}_i$ donné est :

$$\text{comp\_}\mathcal{R}_i = \sum_{j=1}^{\#\mathcal{R}_i} j = \frac{\#\mathcal{R}_i \cdot (\#\mathcal{R}_i + 1)}{2}, \tag{6.8}$$

donc d'après les équations 6.7 et 6.8, l'algorithme teste :

$$\sum_{i=1}^{n}\sum_{j=1}^{i} j = \sum_{i=1}^{n} \frac{i \cdot (i+1)}{2} \Rightarrow \mathcal{O}(n^3) \ ,$$

allocations dans le pire des cas.

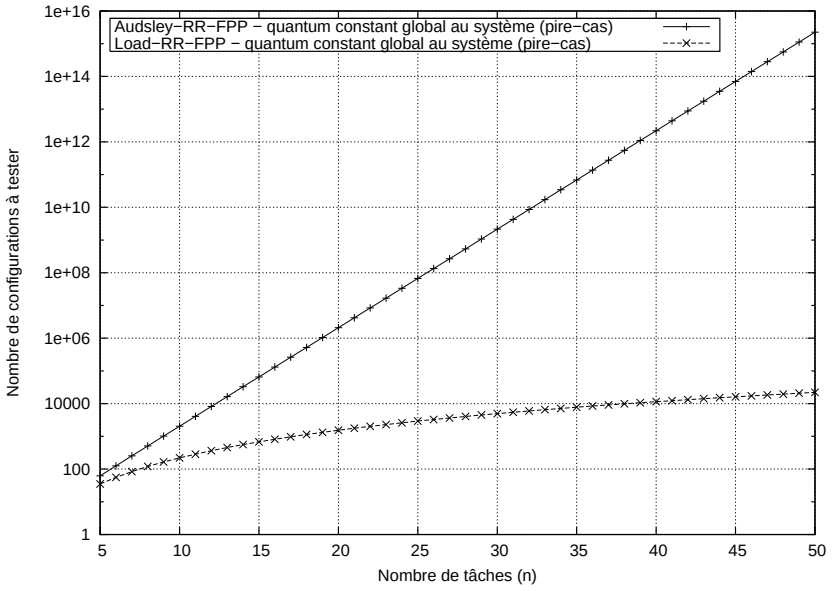


FIG. 6.10 – Nombre de configurations testées dans le pire des cas par *Audsley-RR-FPP* et *Load-RR-FPP* pour un nombre de tâches variant de 5 à 50 quand la valeur du quantum est globale au système (échelle log. pour l'axe des ordonnées).

La figure 6.10 présente la complexité en fonction du nombre de tâches. Par exemple, elle est égale à 220 pour un ensemble de 10 tâches. Pour comparaison, la figure 6.10 montre la complexité d'*Audsley-RR-FPP*. On peut observer une réduction importante vis-à-vis d'*Audsley-RR-FPP* pour des ensembles supérieurs à 10 tâches.

6.6 Expérimentations

Cette section étudie la complexité moyenne des algorithmes proposés : *Audsley-RR-FPP*, *Audsley-RR-FPP** et *Load-RR-FPP* afin d'estimer les plus grands ensembles de tâches qui peuvent être traités en pratique. Ensuite, nous évaluons dans quelle mesure l'utilisation combinée de RR et FPP améliore la faisabilité comparée à FPP seule. Nous considérons l'amélioration apportée dans le cas où la valeur du quantum est globale au système et quand elle est spécifique à chaque tâche. De plus, nous estimons l'efficacité de l'heuristique *Load-RR-FPP*.

6.6.1 Conditions d'expérimentation.

Dans les expérimentations suivantes, nous considérons uniquement les ensembles de tâches non-faisables lorsque seul FPP est utilisée. Comme nous considérons des tâches périodiques à échéance sur requête ($D_i = T_i$), nous utiliserons l'allocation de priorité Rate Monotonic (notée FPP-RM, plus petite la période, plus grande la priorité) optimale dans ce contexte. Selon Liu et Layland [6], un ensemble de n tâches non-concrètes à échéance sur requête est faisable avec FPP-RM si la charge globale U (i.e., $\sum_{i=1}^{n} \frac{C_i}{T_i}$) est inférieure ou égale à $n \cdot (2^{1/n} - 1)$. Nous choisirons donc des valeurs de U nécessairement plus grande que $n \cdot (2^{1/n} - 1)$ afin d'exhiber des ensembles non-faisables sous FPP-RM. Dans la suite, la valeur du quantum est 1 quand la valeur du quantum est globale au système[5] et, peut-être choisi dans l'intervalle $[1, 5]$ quand le quantum est spécifique à chaque tâche. Les ensembles de tâches sont générés avec l'algorithme 4 décrit dans l'annexe A selon le tuple $(n, U, 1, 50, 1, 1, 1000)$ où n est le nombre de tâches et U la charge globale du système. Le temps d'exécution C_i de chaque tâche appartient à l'intervalle $[1, 50]$ et la période T_i est bornée par 1000. Les résultats pour chaque point ont été obtenus avec des ensembles de 200 tâches générés aléatoirement selon les paramètres mentionnés plus hauts.

6.6.2 Évaluation de la complexité

Dans ce paragraphe, nous estimons le nombre de tests utilisés pour obtenir une solution, *i.e.*, trouver une allocation faisable ou conclure que l'ensemble de tâches est non-faisable. Des ensembles de 10 tâches ($n = 10$) sont générés de manière aléatoire avec une charge globale U variant de 0.75 à 0.95. Pour chaque charge de l'ensemble de tâches, la complexité moyenne pour 1) *Audsley-RR-FPP**(solution optimale, quantum spécifique à chaque tâche), 2) *Audsley-RR-FPP* (solution optimale, quantum constant global) et 3) *Load-RR-FPP* (heuristique, quantum constant global) calculée avec 200 ensembles de tâches, est présentée sur la figure 6.11.

Les méthodes proposées dans le §6.3.2 (réduction de complexité quand la valeur du quantum est global au système) et §6.4.2 (réduction de complexité quand la valeur du quantum est spécifique à chaque tâche) sont mises en oeuvre afin d'accélérer la recherche. Par exemple, pour une charge de 0.88, le nombre moyen de configurations testées est environ :

1. 100 fois moindre que le pire des cas correspondant (*i.e.*, $72 \cdot 10^6$) avec *Audsley-RR-FPP**,

[5] Des expérimentations que nous n'avons pas détaillées ici, suggèrent que plus la valeur de quantum est petite, plus il est facile de trouver une allocation de priorité et de politique faisable. Intuitivement, cette tendance s'explique par le fait qu'un petit quantum permet le plus souvent de réduire l'interférence causée par les tâches de même priorité durant une période de temps donné.

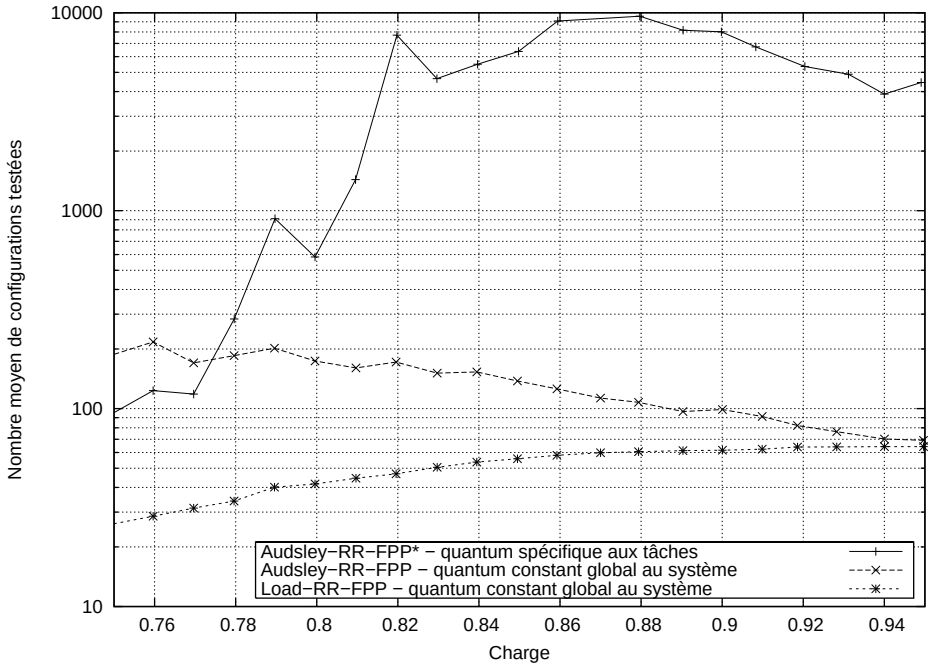


FIG. 6.11 – Nombre moyen de configurations testées par 1) *Audsley-RR-FPP**(quantum spécifique à chaque tâche), 2) *Audsley-RR-FPP* (quantum constant global) et 3) *Load-RR-FPP* (quantum constant global). La charge CPU globale U varie de 0.75 à 0.95. Le nombre de tâches est fixé à 10. Chaque point est la valeur moyenne calculée sur 200 ensembles de tâches. Une échelle log. est utilisée pour l'axe des ordonnées.

2. 10 fois moindre avec *Audsley-RR-FPP* (*i.e.*, 2046).

Nous n'avons pas été en mesure de comparer les résultats de la figure 6.11 avec ceux qui auraient été obtenus sans les procédures de réduction de complexité, car le temps de calcul est trop important quand le quantum est spécifique à chaque tâche.

Il est important de remarquer que, pour un quantum spécifique à chaque tâche, le nombre d'ensembles faisables augmente avec la charge jusqu'à une charge de 86% et diminue ensuite (cf. figure 6.11). Cette tendance peut-être expliquée par deux phénomènes. D'une part, on trouve plus facilement une configuration faisable quand la charge est faible. D'autre part, quand la charge augmente, la procédure pour couper l'espace de recherche du §6.4.2 est plus efficace, car on identifie plus facilement les branches non-faisables de l'arbre de recherche. On peut également observer que la courbe n'est pas régulière. En effet, ces irrégularités sont dues à la grande variabilité des ensembles de tâches testés : dans nos simulations la répartition du nombre de configurations testées a un écart-type supérieur à 1000 et un excès de kurtosis égal à 22. On pourrait remédier à ces irrégularités en augmentant les échantillons considérés.

La tendance décroissante du nombre d'ensembles testés est encore plus accentuée dans le cas où la valeur du quantum est une constante globale (*Audsley-RR-FPP*). En effet, la coupe de l'espace de recherche est encore plus efficace car le fait que le quantum soit fixé réduit les possibilités de trouver un ensemble faisable. Le nombre de configurations considérées par l'heuristique s'approche de sa borne supérieure $O(n^3)$ quand la charge augmente. En effet, l'heuristique n'inclue pas de méthode permettant de couper l'espace de recherche et sa complexité dépend directement du nombre de tâches non-faisables.

L'heuristique devient utile pour des ensembles de tâches plus grand quand on ne peut pas utiliser l'algorithme optimal.

La limite d'*Audsley-RR-FPP* et *Audsley-RR-FPP** est la complexité exponentielle en le nombre de tâches même avec l'utilisation des procédures de réduction de l'espace de recherche. D'autres expérimentations montrent qu'il faut quelques heures pour traiter des ensembles de 30 à 35 tâches avec *Audsley-RR-FPP*. Cependant, *Audsley-RR-FPP** est limité à des ensembles de petites tailles (moins de 20 tâches). Des ensembles de tâches plus grands peuvent être traités en pratique avec l'heuristique *Load-RR-FPP* au dépend de l'optimalité du résultat, cf. section 6.5 et paragraphe 6.6.3 pour des résultats expérimentaux.

6.6.3 Gain en faisabilité en regard de FPP

Navet et Migge [26] ont étudié le même sujet en utilisant des heuristiques allouant les priorités et les politiques. Les auteurs considérent des ensembles de tâches ayant une charge comprise entre 0.75 et 0.90. Ils ont trouvé que plus de 15% des ensembles non-faisables avec FPP le deviennent quand FPP et RR sont combinées. Ici, nous étudions dans quelle mesure l'utilisation d'*Audsley-RR-FPP** et d'*Audsley-RR-FPP*, tous deux optimaux dans leur contexte, améliorent ce résultat. De plus, ces expérimentations nous permettent d'obtenir des éléments plus fins afin d'évaluer l'utilité de RR pour améliorer la faisabilité.

La figure 6.12 montre le pourcentage des ensembles de tâches qui ne sont non-faisables avec FPP et qui le deviennent en utilisant 1) *Audsley-RR-FPP**(solution optimale, quantum spécifique à chaque tâche), 2) *Audsley-RR-FPP* (solution optimale, quantum constant global) et 3) *Load-RR-FPP* (heuristique, quantum constant global). Pour chaque point, ce pourcentage est calculé sur 200 ensembles.

Des ensembles de 10 tâches ($n = 10$) sont générés aléatoirement avec une charge globale U variant de 0.75 à 0.95. On observe que l'usage combiné de RR et FPP permet d'ordonnancer un grand nombre d'ensembles non-faisables avec FPP uniquement : le pourcentage va de presque 100% pour une charge égale à 0.75 à environ 4% pour une charge de 0.95 avec *Audsley-RR-FPP**. La tendance décroissante était attendue, car, en général, les systèmes à forte charge sont plus difficiles à ordonnancer. De toute manière, ces expérimentations montrent avec évidence que l'utilisation combinée de FPP et RR est bénéfique pour la faisabilité, même à charge relativement élevée.

On constate que l'amélioration avec un quantum spécifique à chaque tâche est très importante. Elle est au moins 3 fois supérieure à l'amélioration au cas où le quantum est une valeur globale pour une charge supérieure à 85%. Quand la charge est inférieure à 83%, une solution est trouvée dans la plupart des cas avec un quantum spécifique à chaque tâche ; ce pourcentage reste supérieur à 50% jusqu'à une charge de 89%. Comme prévu, quand la charge devient supérieure, les ensembles faisables deviennent plus rares. La même tendance peut-être observée pour le cas où le quantum est une valeur globale au système : quand la charge est inférieure à 79%, une solution est trouvée pour au moins 80% des cas. Le pourcentage de réussite reste supérieur à 40% jusqu'à une charge de 83%. La figure 6.12 montre également que l'heuristique *Load-RR-FPP* est proche de la solution optimale fournie par *Audsley-RR-FPP* et permet de trouver un ensemble faisable dans la plupart des cas où une configuration faisable existe.

6.6.4 Influence de la taille des ensembles de tâches

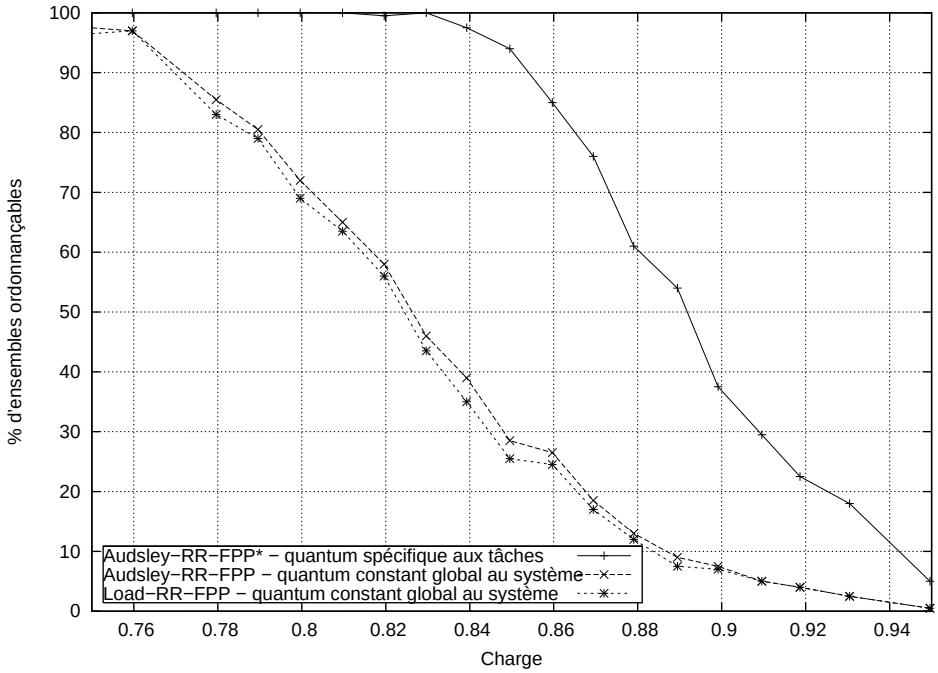


FIG. 6.12 – Pourcentage d'ensembles de tâches non-faisables avec FPP-RM qui deviennent faisables avec 1) *Audsley-RR-FPP**(quantum spécifique à chaque tâche), 2) *Audsley-RR-FPP* (quantum constant global) et 3) *Load-RR-FPP* (quantum constant global). La charge CPU globale U varie de 0.75 à 0.95. Le nombre de tâches est fixé à 10. Chaque point est la valeur moyenne calculée sur 200 ensembles de tâches.

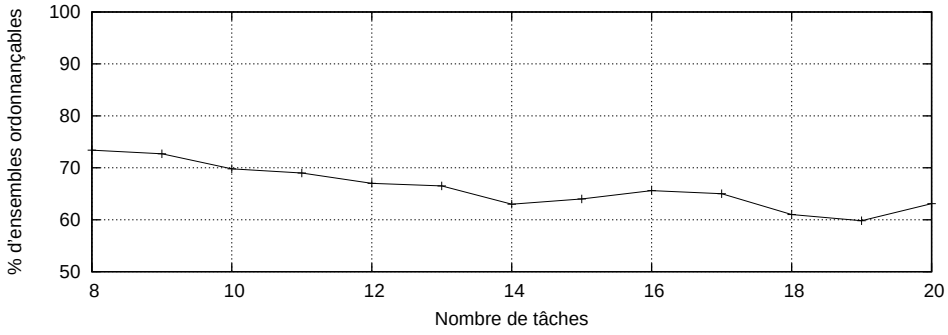


FIG. 6.13 – Pourcentage d'ensembles de tâches non-faisables avec FPP-RM qui deviennent ordonnançables sous Posix en utilisant l'algorithme *Audsley-RR-FPP* . La charge CPU U est égale à 0.8 avec un nombre de tâches compris entre 8 et 20. Chaque point est la valeur moyenne calculée sur 1000 ensembles de tâches.

Des ensembles de n tâches sont générés avec une charge globale U égale à 0.8 avec n variant de 8 à 20. Les résultats présentés dans la figure 6.13 montrent que le nombre d'ensembles faisables diminue sensiblement quand le nombre de tâches augmente. Ce résultat est un peu surprenant, car intuitivement, plus le nombre de tâches est grand, plus les tâches sont petites et plus grand est le degré de liberté pour répartir la charge de travail parmi les différents niveaux de priorités. Ce résultat contre-intuitif est probablement dû au pessimisme du test d'ordonnancement utilisé. En effet, nos expérimentations

montrent qu'en moyenne la borne supérieure sur le temps de réponse est calculée dans 90% des cas avec le premier terme $\left\lceil \frac{s_{i,j}}{\psi_i} \right\rceil \cdot (\overline{\psi}_i - \psi_i) + \widetilde{s}_i(t)$ de l'équation 6.5. Cette valeur dépend de la valeur des quanta des autres tâches de la couche RR et ne prend pas en compte la charge de travail réelle apportée par ces tâches. La borne est donc souvent pessimiste et est pire quand le nombre de tâches dans la couche RR augmente. Dans une certaine mesure, il semble que ce phénomène pessimiste compense le "bénéfice" du plus grand dégré de liberté pour répartir la charge de travail quand le nombre de tâches augmente.

La même tendance peut-être observée avec l'heuristique *Load-RR-FPP* : la figure 6.14 présente le pourcentage d'ensemble de tâches non-faisable avec FPP-RM pour lequel *Load-RR-FPP* trouve une configuration faisable. Des ensembles de n tâches ayant une charge globale égale à 0.84 sont générés avec n variant de 25 à 45. Ces expérimentations insistent sur le fait que l'heuristique est utilisable avec des ensembles plus grands qu'avec *Audsley-RR-FPP** et *Audsley-RR-FPP*.

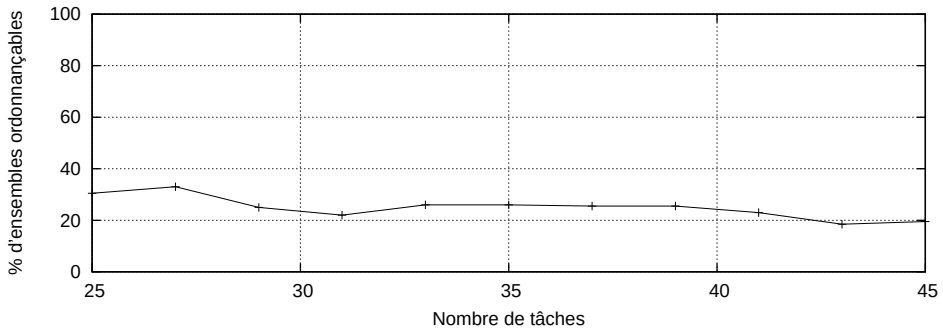


FIG. 6.14 – Pourcentage de tâches non-faisables avec FPP-RM qui deviennent faisables avec l'heuristique *Load-RR-FPP* (quantum constant global). La charge CPU globale U est égale à 0.84 et le nombre de tâches varie de 25 à 45. Chaque point est la valeur moyenne calculée sur 1000 ensembles de tâches.

Nos expérimentations montrent que l'utilisation combinée de RR et FPP avec des quantum spécifiques aux tâches permet d'ordonnancer des ensembles de tâches qui ne sont ni faisables avec FPP ni avec un quantum global au système. Il est important de noter que la surcharge des changements de contexte est négligée alors que RR induit plus de changement de contexte que FPP. Cette hypothèse affecte d'une certaine manière nos conclusions. Un prochain travail serait de trouver une allocation de quantum minimisant le nombre global de changements de contexte.

6.7 Conclusion

Dans ce chapitre, nous proposons des algorithmes d'allocations de priorité, de politiques et de quantum pour des configuration de tâches (processus ou threads ordonnancés globalement) s'exécutant sur les systèmes d'exploitation conformes Posix 1003.1b, en distinguant le cas où le quantum est une constante globale au système et le cas où le quantum est spécifique à chaque tâche. Il a été montré que ces algorithmes sont optimaux dans le sens où s'il existe une configuration faisable qui peut-être identifiée par le test de faisabilité que nous utilisons, alors une configuration faisable est trouvée par nos algorithmes.

Nos algorithmes améliorent significativement la complexité dans le pire des cas comparés à un simple algorithme exhaustif bien que leur complexité reste exponentielle en le nombre de tâches. Par conséquent, ils ne sont pas adaptés pour de grands ensembles de tâches. C'est pourquoi, nous avons proposé l'heuristique *Load-RR-FPP* qui est capable de traiter efficacement des ensembles plus grands. De prochains travaux devront : proposer une heuristique dans le cas où le quantum est spécifique à chaque tâche, évaluer la qualité des heuristiques proposées comparé à l'optimal, prendre en compte les changements de contexte et proposer une allocation de quantum afin de minimiser la surcharge due aux changements de contexte.

Un des résultats des expérimentations est le fait que l'utilisation combinée de FPP et RR avec des quantum spécifiques aux tâches permet d'ordonnancer des ensembles de tâches qui ne sont ni faisables avec FPP ni avec un quantum global au système. Ce résultat est particulièrement intéressant dans le contexte des systèmes embarqués où la pression sur les coûts de production est très important et donc implique l'utilisation maximum du potentiel des ressources.

Troisième partie

Conclusions

Chapitre 7

Conclusions et perspectives

L'objectif que nous nous sommes fixés dans ce travail est la conception d'algorithmes d'ordonnancement temps réel en-ligne faisables optimisant l'utilisation de la plate-forme d'exécution et/ou des critères applicatifs de qualité de service propres à l'application. Nous avons en particulier étudié l'ordonnancement d'activités sur une ressource unique. Deux cas ont été analysés : le cas de tâches indépendantes périodiques s'exécutant sur un processeur et le cas de flux de messages indépendants périodiques sur un réseau de terrain avec accès au médium priorisé. Nos contributions reposent sur le "modèle classique" de l'ordonnancement temps réel où le système est représenté par un ensemble d'activités périodiques indépendantes et deux problématiques ont été abordées :

- **optimisation de l'utilisation de la plate-forme d'exécution** : utiliser au mieux le potentiel de la plate-forme d'exécution tout en garantissant le respect des contraintes temporelles imposées au système ; ceci optimise le nombre de configurations faisables,
- **optimisation des critères applicatifs de qualité de service propres à l'application** (*i.e.*, pris en compte des performances de l'application autre que la faisabilité) : garantir les contraintes de temps tout en optimisant les performances de l'application.

Nous avons donc proposé :

- des **méthodes de configurations** permettant d'optimiser l'utilisation de la plate-forme d'exécution (*i.e.*, maximiser faisabilité) en fixant les paramètres des politiques ou des systèmes considérés d'une manière appropriée. Deux études ont été conduites dans ce cadre :
  - **allocation des "offsets"** dans les systèmes "offset free",
  - **allocation de priorités, de politiques et de quantum** dans les systèmes conformes au standard **Posix 1003.1b**,
- une **nouvelle classe de politiques d'ordonnancement** permettant d'optimiser des critères de qualités de service propres à l'application. De plus, une analyse d'ordonnancement générique pour cette classe a été proposée.

Pour chacun des problèmes étudiés, nous résumons brièvement ci-dessous les contributions et les résultats obtenus.

Optimisation d'une messagerie sur bus priorisé. La première contribution de cette thèse (chapitre 4) est la proposition d'une nouvelle classe de politiques d'ordonnancement appelée politiques Dépendantes

des Dates d'Arrivée (DDA) permettant de "maximiser" l'utilisation de la plate-forme d'exécution. Ces politiques en-ligne à priorité dynamique allouent la ressource à l'activité ayant la plus petite valeur de $A_i + f_i$ où A_i est l'instant d'arrivée et f_i est une valeur constante pour toutes les instances d'une même activité périodique (ex. : tâche, flux). En donnant à f_i une valeur dépendant des caractéristiques de la tâche (ex. : pire temps d'exécution), nous montrons que ces politiques peuvent être utilisées pour l'ordonnancement de trames au niveau MAC sur CAN (avec la méthode de Meschi et al. [52]) et qu'elles fournissent un bon compromis entre la faisabilité et la satisfaction d'autres critères dépendants de l'application tels que la gigue sur le temps de réponse (ex. : réduction de 10.2% de la moyenne de la gigue sur le temps de réponse comparée à NP-EDF selon nos simulations).

Nouvelles politiques à priorité fixe dans les systèmes "offset free". La deuxième contribution (chapitre 5) est la réalisaton de techniques d'allocation d'offsets et de priorités dans les systèmes "offset free" . En effet, afin d'éviter le pic de charge dû au cas où toutes les activités du système sont activées au même instant, nous proposons de nouvelles techniques de définition d'offsets et de priorités, dans le cas préemptif et non-préemptif, afin de mieux répartir la charge et éviter le non-respect des contraintes de temps.

Dans le cas préemptif, nous montrons comment ne considérer qu'un sous-ensemble de tâches dans l'allocation d'offsets en utilisant l'algorithme d'Audsley avant d'effectuer cette allocation. Concrètement, cela permet de réduire significativement l'espace de recherche du problème de l'allocation des offsets (ex. : peut réduire de 50% le nombre d'allocations d'offsets différentes à considérer).

Ensuite, nous proposons des heuristiques pour l'allocation d'offsets proposant des situations asynchrones alternatives. Ces heuristiques permettent d'optimiser l'utilisation de la plate-forme d'exécution (*i.e.*, "maximiser" la faisabilité) en améliorant le nombre de systèmes ordonnançables en regard du cas pessimiste synchrone (ex. : 40.5% des ensembles de tâches non-ordonnançables dans le cas synchrone sont ordonnançables avec l'usage combiné des heuristiques ; les simulations sont réalisées avec des ensembles de 5 à 11 tâches sous une charge moyenne de 0.8). A notre connaissance il n'existe aucune analyse d'ordonnancement des systèmes offset free non-préemptifs distribués sur un réseau priorisé ; aussi, une analyse d'ordonnancement a été réalisée afin de tester l'efficacité de nos heuristiques. Elles se sont avérées très performantes (ex. : réduction moyenne d'un facteur 2.5 des temps de réponse comparé au cas synchrone).

Configuration d'ordonnancement sous Posix 1003.1b. Dans ce travail (chapitre 6), nous analysons dans les systèmes conformes au standard Posix 1003.1b le bénéfice apporté par l'utilisation combinée des deux politiques *sched_rr* (*i.e., Round-Robin* - RR) et *sched_fifo* (*i.e., First-In First-out* - FIFO) disponibles dans ces systèmes. Nos simulations montrent que l'usage combinée de ces deux politiques permet d'obtenir un gain significatif en terme de faisabilité par comparaison à FPP (*Fixed Preemptive Priority*). Nous avons proposé des algorithmes d'allocation de priorités, de politiques d'ordonnancement et de quantum optimaux pour les systèmes Posix 1003.1b (cf. chapitre 6 pour les conditions prècises de l'optimalité). Bien que ces politiques réduisent la complexité, celles-ci restent exponentielles en le nombre de tâches. Nous avons alors réalisé une heuristique peu complexe (*i.e.*, $\mathcal{O}(n^3)$) capable de traiter des problèmes de grandes tailles.

Tous les résultats obtenus dans cette thèse ont été validés, à chaque fois que c'était possible, de manière analytique (ex. : analyse d'ordonnancement générique pour les politiques de la classe DDA). Afin

d'évaluer, dans tous les cas, la qualité de nos propositions, nous nous sommes appliqués à réaliser des simulations. Dans ce but, un générateur aléatoire d'ensemble de tâches et un simulateur d'ordonnancement ont été implementés. L'algorithme du générateur de tâches a été présenté au chapitre 1 et une applet Java du simulateur est disponible (http ://www.loria.fr/~grenier/logiciel/SimApplet.html). Enfin, dans tous ces travaux, le fil conducteur de nos réflexions a été de proposer des approches (i.e., techniques de configuration, politiques d'ordonnancement) génériques ; le terme générique signifie ici que ces approches doivent pouvoir être utilisées sur une classe d'applications et ne pas être liées aux besoins d'une application spécifique ce qui limiterait la portée des contributions.

Limites des propositions. Nous avons essentiellement travaillé en considérant le modèle des tâches indépendantes périodiques ; si celui-ci peut modéliser une large classe d'applications temps réel, il ne couvre pas tous les cas d'intérêt pratique. Par exemple, il peut être nécessaire de modéliser des contraintes de précédence (tâches DAG par exemple) et la suspension temporaire de certaines activités (cf. [79]).

Notre principale métrique de performance a été la faisabilité du système, c'est-à-dire le respect des échéances. C'est naturellement une préoccupation fondamentale pour de nombreuses applications, mais, par exemple, dans les systèmes de contrôle-commande, ne pas respecter certaines échéances n'est pas forcèment synonyme de mauvais fonctionnement (cf. [80]). Parfois, il faudrait donc pouvoir modéliser les contraintes temporelles des tâches d'une manière plus "souple", et l'on peut tout-à-fait envisager l'utilisation de modèles où le dépassement occasionnel de certaines échéances est possible (cf. les systèmes *weakly hard* [81] avec par exemple les contraintes (m, k)-firm).

Au delà du simple critère de faisabilité binaire, l'amélioration de critères de performances propres à l'application a été pris en compte. Nous nous sommes intéressés à un ensemble de critères prédéfinis, comme la gigue, dont la satisfaction est généralement nécessaire pour le bon fonctionnement de l'application. Cependant, ces critères sont "à gros grain" et ne sont pas valables pour toutes les applications. Quand cela est possible, il faut naturellement privilégier la conception conjointe de l'application et de l'ordonnancement (cf., par exemple, [82] pour un aperçu des méthodes existantes), et ainsi mieux prendre en considération les contraintes spécifiques de l'application.

Notre démarche a été de proposer des approches génériques (*i.e.,* techniques de configuration, politiques d'ordonnancement). La généricité de nos algorithmes est garantie dans la classe des systèmes où les activités (tâches, flux de messages) sont périodiques et indépendantes, mais les résultats ne seront souvent pas valables si ces hypothèses ne sont pas remplies. Par exemple, les algorithmes d'allocation proposés dans le cadre des systèmes Posix 1003.1b ne sont plus valables si l'on considère des systèmes concrets (déphasages initiaux fixés) ou *offset free* (déphasages initiaux pouvant être choisis).

Perspectives : modèles génériques pour l'ordonnancement temps réel. Pour optimiser l'ordonnancement deux approches peuvent être envisagées : soit on crée une solution spécifique à un problème donné, soit on propose une méthode générique pour une classe de problèmes, et il faut alors proposer des techniques d'analyse et d'optimisations pour cette classe de problèmes (comme nous l'avons fait pour les politiques DDA). Nous nous intéressons à la proposition d'une solution générique, dans la veine de travaux réalisés par Pop et Eles [83], permettant de pallier aux limites des méthodes existantes. Concrètement, la modélisation du système temps réel devra :

- prendre en compte le support d'exécution. Les ressources de la plate-forme d'exécution (ex. : processeurs, réseaux, mémoires partagées) devront être définies avec les contraintes associées à chacune des ressources (ex. : un seul processus s'exécute sur un processeur à un même instant),
- proposer un modèle d'activité générique définissant les paramètres temporels (ex. : temps d'exécution, période) et les ressources utilisées. De plus, il faudra exprimer les contraintes des activités (ex. : échéance, précédence, *weakly hard*) mais également leur comportement (ex. : que se passe-t-il quand on rate une échéance ou quand des fautes se produisent à l'exécution),
- modéliser la politique d'ordonnancement de chaque ressource, c'est-à-dire définir les protocoles d'accès au réseau et aux ressources partagées, la politique d'ordonnancement utilisée sur un processeur. Par exemple, Demmeler et Giusto [84] propose un modèle de haut niveau de protocoles de communication permettant de définir sans ambiguïté leur fonctionnement.

A l'aide de ce formalisme, le concepteur pourra modéliser son application. En particulier, il faut lui permettre d'exprimer les besoins de performances directement dans le formalisme proposé. Tout le problème étant ici que l'on veut un modèle générique capable d'exprimer les spécificités d'une application particulière.

Une fois le système modélisé, il faut s'intéresser à l'optimisation de sa configuration. Il existe de nombreux problèmes d'optimisation : dimensionnement de la plate-forme (ex. : décider du nombre de processeurs et du débit du réseau), de placement, d'ordonnancement (ex. : dans le cas d'un réseau de type TDMA (*Time Divison Multiple Access*) comment allouer les slots), etc. Dans un premier temps, nous nous intéresserons uniquement au problème d'optimisation de l'ordonnancement dans un contexte où la plate-forme d'exécution, l'application et les métriques de performances à optimiser sont connues. Deux classes de problèmes seront traitées : 1) le cas où l'algorithme d'ordonnancement n'est pas imposé (ex. : possibilité de proposer de nouvelles politiques d'ordonnancement comme les politiques DDA) 2) le cas où l'algorithme est connu mais certains paramètres sont configurables (ex. : systèmes offset free). Il faudra fournir des approches permettant de générer automatiquement des solutions respectant les contraintes de temps et optimisant les critères de performances introduits par le concepteur du système. Bien entendu, des outils et des méthodes devront permettre d'évaluer les performances (ex. : simulation) et le respect des contraintes (ex. : analyses d'ordonnancement).

Il faut remarquer que certains travaux actuels vont dans la même direction. par exemple, Pop et *al.* [83] proposent des approches d'analyse et d'optimisation dans les systèmes temps réel distribués sur des architectures hétérogènes. Leur approche est très prometteuse mais peut certainement être étendue, en particulier au niveau du traitement des problèmes d'exclusion mutuelle, du comportement des activités en présence de fautes, et des techniques d'optimisation.

Bibliographie

[1] Migge (J.), Jean-Marie (A.) et Navet (N.), « Timing analysis of compound scheduling policies : Application to Posix1003.1b », *Journal of Scheduling*, vol. 6, n° 5, 2003, p. 457–482.

[2] Stankovic (J. A.), « Misconceptions about real-time computing : A serious problem for next-generation systems », *IEEE Computer*, vol. 21, n° 10, 1988, p. 10–19.

[3] Elloy (J.-P.), « Le temps réel, rapport établi par le groupe de réflexion temps réel du cnrs », *Technique et Sciences Informatiques TSI*, vol. 7, n° 5, 1988, p. 493–500.

[4] Xu (J.) et Parnas (D.), « Scheduling processes with release times, deadlines, precedence and exclusion relations », *IEEE Transactions on Software Engineering*, vol. 16, n° 3, 1990, p. 360–369.

[5] Baruah (S.), Howell (R.) et Rosier (L.), « Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor », *Journal of Real-Time Systems*, 1990.

[6] Liu (C.) et Layland (J.), « Scheduling algorithms for multiprogramming in hard-real time environnement », *Journal of the ACM*, vol. 20, n° 1, 1973, p. 40–61.

[7] Bini (E.) et Buttazzo (G.), « Biasing effects in schedulability measures », dans *Proceedings of the 16th Euromicro Conference on real-Time systems*, Catania, Italy, 2004.

[8] Jeffay (K.), Stanat (D.) et Martel (C.), « On non-preemptive scheduling of periodic and sporadic tasks », dans *les actes du 12th IEEE Real-time Systems Symposium (RTSS'91)*, 1991.

[9] Dertouzos (M.), « Control robotics : The procedural control of physical processes », dans *les actes du IFIP congress*, p. 807–813, 1974.

[10] George (L.), Rivierre (N.) et Spuri (M.), « Preemptive and non-preemptive real-time uniprocessor scheduling ». Rapport technique n° RR-2966, Institut National de Recherche en Informatique et Automatique (INRIA), 1996. Disponible à `http://www.inria.fr/rrrt/rr-2966.html`.

[11] Mok (A.), *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. Thèse de doctorat, MIT, 1983.

[12] Leung (J.) et Whitehead (J.), « On the complexity of fixed priority scheduling of periodic, real-time tasks », *Performance Evaluation*, vol. 2, 1982, p. 237–250.

[13] Kim (K. H.) et Naghibzadeh (M.), « Prevention of task overruns in real-time non-preemptive multiprogramming systems », *SIGMETRICS Performance Evaluation Review*, vol. 9, n° 2, 1980, p. 267–276.

[14] Audsley (N.), « Optimal priority assignement and feasibility of static priority tasks with arbitrary start times ». Rapport technique n° YO1 5DD, Real-Time Research Group, Department of Computer Science, Université de York, 1991.

[15] George (L.), Muhlethaler (P.) et Rivierre (N.), « Optimality and non-preemptive real-time scheduling revisited ». Rapport technique n° RR-2516, Institut National de Recherche en Informatique et Automatique (INRIA), 1995.

[16] Gerber (R.), Hong (S.) et Saksena (M.), « Guaranteeing end-to-end timing constraints by calibrating intermediate processes. », dans *les actes du 15th IEEE Real-Time Systems Symposium (RTSS'94)*, p. 192–205, décembre 1994.

[17] Bate (I.) et Burns (A.), « An approach to task attribute assignment for uniprocessor systems », dans *les actes de la 11th Euromicro Conference on Real-Time Systems*, p. 46–53, Université de York, Angleterre, 1999.

[18] DiNatale (M.) et Stankovic (J.), « Applicability of simulated annealing methods to real-timescheduling and jitter control », dans *les actes du 16th IEEE Real-Time Systems Symposium (RTSS'95)*, p. 190–199, Pisa, Italy, décembre 1995.

[19] Natale (M. D.) et Stankovic (J.), « Scheduling distributed real-time tasks with minimum jitter », *IEEE Transactions on Computers*, vol. 49, n° 4, 2000, p. 303–316.

[20] Coutinho (F.), Fonseca (J.), Barreiros (J.) et Costa (E.), « Using genetic algorithms to reduce jitter in control variables transmitted over can », dans *les actes de la 7th International CAN Conference (ICC'00)*, octobre 2000.

[21] Baruah (S.), Buttazzo (G.), Gorinsky (S.) et Lipari (G.), « Scheduling periodic task systems to minimize output jitter », dans *les actes de la 6th International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*, décembre 1999.

[22] David (L.), *Contribution à la gestion de la régularité d'exécution des tâches d'une application temps réel à contraintes strictes, dans un contexte d'ordonnancement en ligne.* Thèse de doctorat, LISI/ENSMA, décembre 2002.

[23] Han (C.-C.) et K.-J. (L.), « Scheduling distance-constrained real-time tasks », dans *les actes du 13th Real-Time Systems Symposium (RTSS'92)*, p. 300–308, décembre 1992.

[24] Crespo (A.), Ripoll (I.) et Albertos (P.), « Reducing delays in RT control : the control action interval », dans *les actes du 14th IFAC World Congress*, 1999.

[25] Goossens (J.) et Richard (P.), « Performance optimization for hard real-time fixed priority tasks », dans *les actes de la 12th International Conference on Real-Time Systems (RTS'04)*, 2004.

[26] Navet (N.) et Migge (J.), « Fine tuning the scheduling of tasks through a genetic algorithm : Application to Posix1003.1b compliant OS », *IEEE Proceedings Software*, vol. 150, n° 1, 2003, p. 13–24.

[27] Schrage (L.), « A proof of the optimality of the shortest remaining time discipline », *Operations Research*, vol. 16, 1968, p. 687–690.

[28] Smith (D.), « A new proof of the optimality of the shortest remaining time discipline », *Operations Research*, vol. 26, n° 1, 1976, p. 197–199.

[29] Bunde (D. P.), « SPT is optimally competitive for uniprocessor flow », *Information Processing Letters*, vol. 90, n° 5, 2004, p. 233–238.

[30] Altisen (K.), Goessler (G.), Pnueli (A.) *et al.*, « A framework for scheduler synthesis », dans *les actes du 20th IEEE Real-Time Systems Symphosium (RTSS'99)*, 1999.

[31] Grolleau (E.), Choquet-Cheniet (A.) et Cottet (F.), « Validation de systèmes temps réel à l'aide de réseaux de petri », dans *les actes de la conférence Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'98)*, 1998.

[32] Cervin (A.) et Eker (J.), « Control-scheduling codesign of real-time systems : The control server approach », *Journal of Embedded Computing*, vol. 1, n° 2, 2004, p. 209–224.

[33] Abeni (L.) et Buttazzo (G.), « Integrating multimedia applications in hard real-time systems », dans *les actes du 19th IEEE Real-Time Systems Symposium (RTSS'98)*, 1998.

[34] Cena (G.) et Valenzano (A.), « Achieving round-robin access in controller area networks », *IEEE Transactions on Industrial Electronics*, vol. 49, n° 6, 2002, p. 1202–1213.

[35] Nolte (T.), Nolin (M.) et Hansson (H.), « Real-time server-based communication for CAN », *IEEE Transactions on Industrial Informatics*, vol. 1, n° 3, 2005, p. 192–201.

[36] (ISO/IEC) 9945-1 :2004 et IEEE Std 1003.1, 2004 Edition. « Information technology—portable operating system interface (POSIX®)—part 1 : Base definitions ». IEEE Standards Press, 2004.

[37] Törngren (M.), « Fundamentals of implementing real-time control applications in distributed computer systems », *Real-Time Systems*, vol. 14, n° 3, 1998, p. 219–250.

[38] Grenier (M.) et Navet (N.), « New on-line preemptive scheduling policies for improving real-time behavior », dans *les actes de la 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2005)*, vol. 2, p. 315–322, Catania, Italy, septembre 2005. Disponible à `http://www.loria.fr/~nnavet/publi/etfa2005_mgnn.pdf`.

[39] Grenier (M.) et Navet (N.). « Fine-tuning MAC-level protocols for optimized real-time QoS ». Soumis au Journal IEEE Transactions on Industrial Informatics, 2007.

[40] Grenier (M.), Goossens (J.) et Navet (N.), « Near-optimal fixed priority preemptive scheduling of offset free systems », dans *les actes de la 14th International Conference of Real-Time and Network Systems (RTNS'06)*, p. 35–42, Poitiers, France, 2006.

[41] Grenier (M.) et Navet (N.), « Improvement in the configuration and analysis of Posix 1003.1b scheduling », dans *les actes de la 15th International Conference of Real-Time and Network Systems (RTNS'07)*, p. 141–150, Nancy, France, 2007.

[42] Grenier (M.) et Navet (N.), « Scheduling configuration on posix 1003.1b systems ». Rapport technique n° RR-6209, Institut National de Recherche en Informatique et Automatique (INRIA), juin 2007.

[43] Wittenmark (B.), Nilsson (J.) et Törngren (M.), « Timing problems in real-time control systems », dans *les actes de la American Control Conference*, 1995.

[44] Årzén (K.-E.), Cervin (A.), Eker (J.) et Sha (L.), « An introduction to control and scheduling co-design », dans *les actes de la 39th IEEE Conference on Decision and Control*, 2000.

[45] Bosch (R.). « CAN Specification Version 2.0 », 1991.

[46] ISO. « ISO International Standard 11898 - Road vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communications », 1993.

[47] Navet (N.), Song (Y.-Q.), Simonot-Lion (F.) et Wilwert (C.), « Trends in automotive communication systems », *journal IEEE special issue on Industrial Communications Systems, invited paper*, vol. 96, n° 6, juin 2005, p. 1204–1223.

[48] ISO. « ISO International Standard 11519-3 - Road vehicles - Low-speed serial data communication - Vehicle Area Network (VAN) », 1994.

[49] SAE. « SAE J1850 Class B data communication network interface », 1992.

[50] Pedreiras (P.) et Almeida (L.), « Flexible scheduling on controller area network », dans *les actes de la 10th International Conference on Real-Time Systems (RTS'02)*, 2002.

[51] Zuberi (K.) et Shin (K.), « Non-preemptive scheduling of messages on Controller Area Network for real-time control applications », dans *les actes du Real-Time Technology and Applications Symposium (RTAS'95)*, 1995.

[52] Meschi (A.), Natale (M. D.) et Spuri (M.), « Earliest Deadline message scheduling with limited priority inversion », dans *les actes du 4th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'96)*, 1996.

[53] Natale (M. D.), « Scheduling the CAN bus with Earliest Deadline techniques », dans *les actes du 21st IEEE Real-time Systems Symposium (RTSS'00)*, p. 259–268, 2000.

[54] Natale (M. D.) et Meschi (A.), « Scheduling messages with Earliest Deadline techniques », *Real-Time Systems*, vol. 20, n° 3, 2001, p. 255–285.

[55] Nolte (T.), Hansson (H.) et Norström (C.), « Minimizing CAN response-time jitter by message manipulation », dans *les actes du 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)*, p. 197–206, 2002.

[56] Migge (J.), *Scheduling under Real-Time Constraints : a Trajectory Based Model.* Thèse de doctorat, Université de Nice Sophia-Antipolis, 1999.

[57] Stankovic (J.), Spuri (M.), Ramamritham (K.) et Buttazzo (G.), *Deadline Scheduling for Real-Time Systems EDF and Related Algorithms.* Kluwer Academic Publishers ISBN 0-7923-8269-2, 1998.

[58] Spuri (M.), « Analysis of deadline scheduling in real-time systems ». Rapport technique n° RR-2772, Institut National de Recherche en Informatique et Automatique (INRIA), 1996. Disponible à `http://www.inria.fr/rrrt/rr-2772.html`.

[59] Astrom (K.) et Wittenmark (B.), *Computer-Controlled Systems.* Prentice Hall ISBN 0-13-168600-3, third (édition, 1997.

[60] Baruah (S.), Buttazzo (G.), Gorinsky (S.) et Lipari (G.), « Scheduling periodic task systems to minimize output jitter », dans *les actes de la 6th International Conference on Real-Time Computing Systems and Applications (RTCSA'98)*, 1998.

[61] Ohlin (M.), Henriksson (D.) et Cervin (A.), *TrueTime 1.5 : Reference Manual.* Department of Automatic Control, Lund University, Sweden, janvier 2007.

[62] Åström, Johan (K.) et Hägglund (T.), *Advanced PID Control.* ISA - The Instrumentation, Systems, and Automation Society, Research Triangle Park, North Carolina, 2005.

[63] Tindell (K.) et Clark (J.), « Holistic schedulability analysis for distributed hard real-time systems », *Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems)*, vol. 40, 1994, p. 117–134.

[64] Anderson (J.) et Srinivasan (A.), « Pfair scheduling : Beyond periodic task systems », dans *les actes de la 7th International Conference on Real-time Computing Systems and Applications*, p. 297–306. IEEE Computer Society Press, décembre 2000.

[65] Goossens (J.), « Scheduling of offset free systems », *Real-Time Systems*, vol. 24, n° 2, mars 2003, p. 239–258.

[66] Audsley (N. C.), « On priority assignment in fixed priority scheduling. », *Information Processing Letters*, vol. 79, n° 1, 2001, p. 39–44.

[67] Goossens (J.) et Devillers (R.), « The non-optimality of the monotonic priority assignements for hard real-time systems », *Real-Time Systems*, vol. 13, n° 2, septembre 1997, p. 107–126.

[68] Lehoczky (J.), « Fixed priority scheduling of periodic task sets with arbitrary deadlines », dans *les actes du 11th Real-Time Systems Symposium (RTSS'90)*, p. 201–209, 1990.

[69] Tindell (K.), Burns (A.) et Wellings (A.), « Analyzing real-time communications », *Real-Time Systems*, vol. 9, 1995, p. 147–171.

[70] Choquet-Geniet (A.) et Grolleau (E.), « Minimal schedulability interval for real time systems of periodic tasks with offsets », *Theoretical of Computer Sciences*, vol. 310, 2004, p. 117–134.

[71] Lehoczky (J.), Sha (L.) et Ding (Y.), « The rate monotonic scheduling algorithm : exact characterization and average case behavior », dans *les actes du 10th Real Time Systems Symposium (RTSS'89)*, p. 166–171, 1989.

[72] Tindell (K.), « Adding time-offsets to schedulability analysis ». Rapport technique n° YCS 221, Real-Time Research Group, Department of Computer Science, Université de York, 1994.

[73] Palencia (J.) et Harbour (M. G.), « Schedulability analysis for tasks with static and dynamic offsets », dans *les actes du 19th IEEE Real-Time Systems Symposium (RTSS'98)*, p. 26–37, Madrid, Spain, décembre 1998.

[74] Mäki-Turja (J.) et Nolin (M.), « Fast and tight response-times for tasks with offsets. », dans *les actes de la 17th EUROMICRO Conference on Real-Time Systems*, p. 10, Palma de Mallorca Spain, juillet 2005. IEEE.

[75] Traore (K.), Grolleau (E.) et Cottet (F.), « Schedulability analysis of serial transactions », dans *les actes de la 6th international conference of Real-Time and Network Systems, RTNS'06*, p. 141–149, Poitiers, france, 2006.

[76] Tindell (K.), « An extendible approach for analyzing fixed priority hard real-time tasks ». Rapport technique n° YCS-92-189, Real-Time Research Group, Department of Computer Science, Université de York, 1992.

[77] Brito (R.) et Navet (N.), « Low-power round-robin scheduling », dans *les actes de la 12th international conference on real-time systems (RTS'04)*, 2004.

[78] Abramowitz (M.) et Stegun (I.), *Handbook of Mathematical Functions*. Dover Publications (ISBN 0-486-61272-4), 1970.

[79] Ridouard (F.) et Richard (P.), « Worst-case analysis of feasibility tests for self-suspending tasks », dans *les actes de la 14th Real-Time and Network Systems conference (RTNS'06)*, p. 15–24, Poitiers, 2006.

[80] Cervin (A.), *Integrated Control and Real-Time Scheduling.* Thèse de doctorat, Department of Automatic Control, Lund Institute of Technology, 2003.

[81] Bernat (G.), Burns (A.) et Llamosi (A.), « Weakly hard real-time systems », *IEEE Transactions on Computers*, vol. 50, n° 4, 2001, p. 308–321.

[82] Simon (D.), « Conception conjointe commande/ordonnancement et ordonnancement régulé », dans *4ème Ecole d'été temps réel (ETR'05)*, p. 177–194, Nancy, 2005.

[83] Pop (P.), Eles (P.), Peng (Z.) et Pop (T.), « Analysis and optimization of distributed real-time embedded systems », *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, n° 3, jul 2006, p. 593–625.

[84] Demmeler (T.) et Giusto (P.), « A universal communication model for an automotive system integration platform », dans *les actes du Design Automation and Test in Europe Conference*, p. 47–54, 2001.

Quatrième partie

Annexes

Annexe A

Chapitre 1

A.1 Génération aléatoire des tâches

Nous décrivons ici l'algorithme générique, cf. algorithme 4, utilisé pour générer les ensembles de manière aléatoires. Dans chaque chapitre, des précisions seront données lorsque cela sera nécessaire.

Dans les expérimentations, nous utilisons un tuple $(n, U, c_{\min}, c_{\max}, d_{\min}, d_{\max}, t_{\max})$ pour préciser les paramètres utilisés pour générer aléatoirement les ensembles de tâches. Une charge globale U et un nombre de tâches n sont choisis. Nous générons des ensembles de n tâches jusqu'en trouver un dont la charge est comprise entre $[U - 0.02, U + 0.02]$, cf. algorithme 4 ligne 1. Le taux d'utilisation $\frac{C_i}{T_i}$ de chaque tâche τ_i est aléatoirement choisi selon une loi uniforme dans l'intervalle $\left[\frac{U}{n} \cdot 0.9 \ , \ \frac{U}{n} \cdot 1.1\right]$, cf. algorithme 4 ligne 8. Ensuite, le temps d'exécution C_i de chaque tâche τ_i est uniformément répartie dans l'intervalle $[c_{\min}, c_{\max}]$, cf. algorithme 4 lignes 10 et 11. On peut déduire la période du taux d'utilisation $\frac{C_i}{T_i}$ et du temps d'exécution C_i, *i.e.*, $T_i = \left\lfloor C_i / \frac{C_i}{T_i} \right\rfloor$, cf. algorithme 4 lignes 12 et 13. Ensuite, si la période est inférieure à une valeur $t_{\max}$ donnée, nous ajoutons la tâche (ayant une échéance relative $\overline{D}_i$ (resp. O_i) aléatoirement choisi(e) dans l'intervalle $[d_{\min}, d_{\max}]$ (resp. $[0, T_i]$)), sinon une autre tâche est crée.

Entrée : n : le nombre de tâches voulu
U : la charge voulue est dans $[U - 0.02, U + 0.02]$
$c_{\min}$, $c_{\max}$: le temps d'exécution des tâches appartient à : $[c_{\min}, c_{\max}]$
$d_{\min}$, $d_{\max}$: l'échéance d'une tâches τ_k est dans : $[d_{\min}, d_{\max}]$
$t_{\max}$: la période de la tâche est $\leq t_{\max}$
Sortie : ensemble de tâches $\mathcal{T} = \{\tau_1, \tau_2..., \tau_n\}$ généré
Données : $\mathcal{T}$: ensemble de tâches
n_{current} : le nombre de tâches dans l'ensemble $\mathcal{T}$
$U_{\mathcal{T}}$: la charge courante de l'ensemble $\mathcal{T}$
τ_k : la tâche générée

tant que $U_{\mathcal{T}} < U - 0.02$ *ou* $U_{\mathcal{T}} > U + 0.02$ **faire**
  $U_{\mathcal{T}} = 0$;
  $\mathcal{T} = \emptyset$;
  $n_{\text{current}} = 1$;
  `générer` n `tâches :`
  tant que $n_{current} \leq n$ **faire**
    `générer une tâche` τ_i `:`
    charge $\frac{C_i}{T_i}$ uniformément répartie dans l'intervalle $[0.9 \cdot U_{\min}, 1.1 \cdot U_{\max}]$:
    $\frac{C_i}{T_i} = U_{\min} + \texttt{rand()} \cdot (U_{\max} - U_{\min})$;
    temps d'exécution C_i uniformément répartie dans l'intervalle $[c_{\min}, c_{\max}]$:
    $C_i = c_{\min} + \lfloor \texttt{rand()} \cdot (c_{\max} - c_{\min}) \rfloor$;
    on déduit la période de C_i et $\frac{C_i}{T_i}$:
    $T_i = \lfloor C_i / \frac{C_i}{T_i} \rfloor$;
    si $T_i \leq t_{max}$ **alors**
      échéance relative $\overline{D}_i$ uniformément répartie dans $[d_{\min}, d_{\max}]$:
      $\overline{D}_i = d_{\min} + \lfloor \texttt{rand()} \cdot (d_{\max} - d_{\min}) \rfloor$;
      offset O_i uniformément répartie dans $[0, T_i]$:
      $O_i = \lfloor \texttt{rand()} \cdot T_i \rfloor$;
      $\tau_i = (C_i, T_i, D_i, O_i)$;
      `ajouter la tâche :`
      $\mathcal{T} = \mathcal{T} \cup \tau_i$;
      $n_{\text{current}} = n_{\text{current}} + 1$;
      $U_{\mathcal{T}} = U_{\mathcal{T}} + \frac{C_i}{T_i}$;
    finsi
  fintq
fintq
retourner $\mathcal{T}$

Algorithme 4 : algorithme utilisé pour générer un ensemble de tâches $\mathcal{T}$ selon le tuple $(U, n, c_{\min}, c_{\max}, d_{\min}, d_{\max}, t_{\max})$ où la fonction rand() retourne un nombre aléatoire dans l'intervalle $[0, 1]$.

Annexe B

Chapitre 4

B.1 Échelle logarithmique pour coder les priorités

Cette annexe présente l'échelle logarithmique, introduite par Meschi et *al.* [52] pour le cas des échéances, pour coder les priorités $A_{i,j} + f_i$ des trames $\tau_{i,j}$ dans un nombre limité b de bits pour réduire les erreurs d'inversion de priorité dues au codage en utilisant une résolution plus fine pour les priorités (*i.e.*, $A_{i,j} + f_i$) les plus proches.

Un exemple d'échelle logarithmique est présentée dans la figure B.1. L'origine des temps de l'échelle logarithmique, noté t_{start}, est mise à jour à chaque début de transmission ($t_{start} \stackrel{\text{def}}{=} \max_{i,j}(B_{i,j} \leq t)$, où t est l'instant courant). La ligne de temps (*timeline*) de t_{start} à f_{max} ($f_{max} \stackrel{\text{def}}{=} \max_{\tau_i \in \mathcal{T}}(f_i)$) est divisée en blocs de taille croissante. Chaque bloc est divisé en q slots où q est un entier connu lors de l'étape de conception (cf. équation B.2). La priorité de la trame correspond à l'indice du slot contenant la priorité $A_{i,j} + f_i - t_{start}$ courante. Selon [52], l'indice du slot l est calculé de la manière suivante :

$$l = h \cdot q + \left\lfloor \frac{A_{i,j} + f_i - t_{start} - 2^h f_{min}}{2^h S} \right\rfloor,$$

où S est la taille du premier slot ($S \stackrel{\text{def}}{=} \frac{f_{min}}{q}$ et $f_{min} \stackrel{\text{def}}{=} \min_{\tau_i \in \mathcal{T}}(f_i)$). L'indice du bloc, noté h, où la

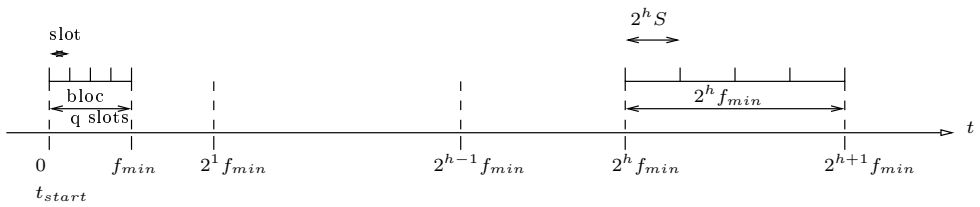


FIG. B.1 – Échelle logarithmique débutant à t_{start}, où f_{min} est la priorité relative minimum de l'ensemble de flux ($f_{min} \stackrel{\text{def}}{=} \min_{\tau_i \in \mathcal{T}}(f_i)$), h est l'indice du bloc calculé selon l'équation B.1 et S est la taille du premier slot de temps (*i.e.*, $\frac{f_{min}}{q}$).

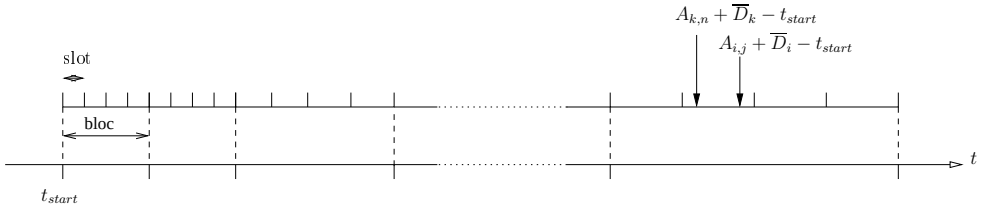


FIG. B.2 – Erreur de codage avec les deux trames $\tau_{k,n}$ et $\tau_{i,j}$ ayant une échéance respectivement égale a $A_{k,n} + \overline{D}_k - t_{start}$ et $A_{i,j} + \overline{D}_i - t_{start}$.

priorité $A_{i,j} + f_i - t_{start}$ courante "tombe" est [52] :

$$h = \begin{cases} \left\lfloor log_2 \frac{A_{i,j}+f_i-t_{start}}{f_{min}} \right\rfloor & \text{if } A_{i,j} + f_i - t_{start} > f_{min} \\ 0 & \text{otherwise} \end{cases} . \quad \text{(B.1)}$$

Le nombre de slot q dans un bloc (cf. annexe A [52] pour les détails du calcul) est donné par :

$$q = \left\lfloor \frac{2^n}{\lfloor log_2 k_{max} \rfloor + \frac{k_{max}}{2^{\lfloor log_2 k_{max} \rfloor}}} \right\rfloor , \quad \text{(B.2)}$$

où $k_{max} \stackrel{\text{def}}{=} \frac{f_{max}}{f_{min}}$. Par définition, la taille du slot où $A_{i,j} + f_i - t_{start}$ "tombe" est égale à $2^h S$.

B.2 Erreur de codage

Quand la priorité est codée dans un nombre limité de bits (par exemple en utilisant l'échelle logarithmique), les trames qui ont des priorités (c'est-à-dire des échéances sous NP-EDF) distinctes peuvent appartenir au même slot et une "inversion de priorité" peut se produire (ex. : la trame ayant la plus petite échéance est moins prioritaire). Dans ce cas, ces erreurs sont appelées erreurs de codage.

La figure B.2 montre un exemple d'erreur de codage sous NP-EDF quand l'échelle logarithmique présentée en annexe B.1 est utilisée avec les deux trames $\tau_{k,n}$ et $\tau_{i,j}$, ayant une échéance respectivement égale à $A_{k,n} + \overline{D}_k - t_{start}$ et $A_{i,j} + \overline{D}_i - t_{start}$ qui tombe dans le même slot. Dans ce cas, la trame $\tau_{k,n}$ avec l'échéance la plus petite peut être moins prioritaire malgré le fait que $A_{k,n} + \overline{D}_k - t_{start} < A_{i,j} + \overline{D}_i - t_{start}$.

B.3 Travail plus prioritaire $W_i(\mathtt{a}, t)$

Cette annexe détaille le calcul du travail plus prioritaire $W_i(\mathtt{a}, t)$ quand des inversions de priorité se produisent du fait du nombre limité de bits disponibles pour coder l'échéance.

Le travail plus prioritaire $W_i(\mathtt{a}, t)$ est par définition, le travail induit par les flux τ_k, différents de τ_i, dans une *deadline busy period* de longueur t. Une deadline busy period est une période d'utilisation sans interruption du réseau durant laquelle seules les trames ayant une priorité plus grande que $\tau_i(\mathtt{a})$ (ex. : échéance plus petite dans le cas NP-EDF) sont exécutées. Dans la suite, nous notons u le début cette deadline busy period. $A_{k,n_0} = \min\{A_{k,n} \mid n \in \mathbb{N}, A_{k,n} \geq u\}$ et $A_{k,n_1} = \min\{A_{k,n} \mid n \in \mathbb{N}, A_{k,n_1} \leq u + t\}$

sont respectivement la première et la dernière trame du flux τ_k dans la deadline busy period. Le nombre de trames m du flux τ_k dans l'intervalle $[u, u+t]$ est $m = n_1 - n_0 + 1$.

L'objectif est de borner, pour chaque flux τ_k, le travail induit par les trames de τ_k dans $[u, u+t]$. Le travail de la dernière trame du flux τ_k arrivée en A_{k,n_1}est pris en compte ssi :

1. f_{k,m_1}arrive avant de l'intervalle $[u, u+t]$: $A_{k,n_1} \leq u + t$.

$$A_{k,n_1} \leq u + t$$
$$A_{k,n_0} + (m-1) \cdot T_i \leq u + t$$
$$\text{Donc : } m = \left\lfloor \frac{t}{T_k} \right\rfloor + 1 \leq \frac{t}{T_k} + 1 \text{ car } m \in \mathbb{N} \text{ et } A_{k,n_0} \geq u$$

2. De plus, la dernière trame τ_{k,n_1} doit avoir une priorité plus importante que $\tau_i(\mathtt{a})$. Nous faisons ici l'hypothèse que toutes les trames ayant une échéance dans le même slot que $A_{i,j} + \overline{D}_i - t_{start}$ ont une priorité plus importante que $\tau_i(\mathtt{a})$. Donc, nous considérons que les trames ayant une échéance inférieure ou égale à $A_{i,j} + \overline{D}_i + S_i(\mathtt{a}, t)$ ont une priorité plus importante où$S_i(\mathtt{a}, t)$ est une borne supérieure sur la taille du slot auquel $A_{i,j} + \overline{D}_i - t_{start}$ appartient, cf. [53] pour les détails du calcul.

$$A_{k,n_1} + \overline{D}_k - t_{start} \leq u + \mathtt{a} + \overline{D}_i + S_i(\mathtt{a}, t) - t_{start}$$
$$\text{Donc : } m = \left\lfloor \frac{\mathtt{a} + \overline{D}_i + S_i(\mathtt{a}, t) - \overline{D}_k}{T_k} \right\rfloor + 1 \leq \frac{\mathtt{a} + \overline{D}_i + S_i(\mathtt{a}, t) - \overline{D}_k}{T_k} + 1 \text{ car } m \in \mathbb{N}$$

Donc, une borne supérieure du nombre de trames induit par un flux τ_k dans une deadline busy period de longueur t est : $\min\left(\left\lfloor \frac{t}{T_k} \right\rfloor + 1, \left\lfloor \frac{\mathtt{a}+\overline{D}_i+S_i(\mathtt{a},t)-\overline{D}_k}{T_k} \right\rfloor + 1\right)$. Une borne sur le travail plus prioritaire $W_i(\mathtt{a}, t)$, dû aux flux différents de τ_i, est donc :

$$W_i(\mathtt{a}, t) = \sum_{k \neq i} \left(\min\left(\left\lfloor \frac{t}{T_k} \right\rfloor + 1, \left\lfloor \frac{\mathtt{a} + \overline{D}_i + S_i(\mathtt{a}, t) - \overline{D}_k}{T_k} \right\rfloor + 1 \right) \right)^+ \cdot C_k \ .$$

Annexe C

Chapitre 5

C.1 NETCAR-Analyzer : logiciel de calcul de temps de réponse et d'allocation d'offsets

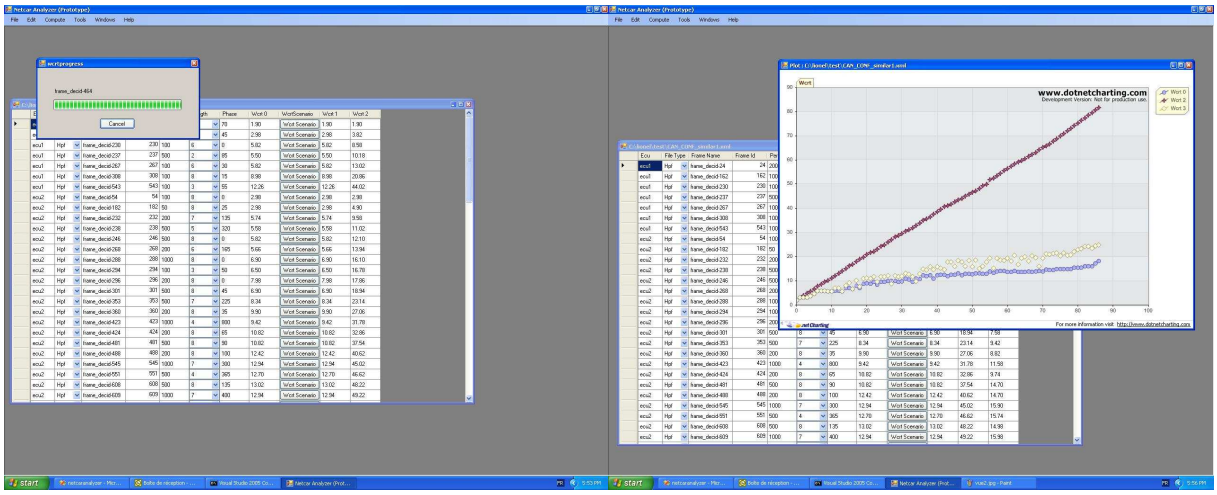
Le logiciel NetCarAnalyzer, développé en C++, enregistré à l'APP sous le numéro IDDN FR 001 300015 000 S P 2007 000 10000, permet d'assigner les offsets des flux selon les deux algorithmes décrits dans le chapitre 5 et de calculer les temps de réponse d'une configuration donnée (qui pourra avoir été obtenue avec d'autres algorithmes de déphasage comme celui développé à la DRIA). Ce programme fonctionne sous WIN32 et existe dans deux versions : une version avec interface graphique et une version en mode ligne de commande. La figure C.1 présente une copie d'écran de l'interface graphique. Les fichiers XML d'entrée et de sortie sont spécifiés à l'aide d'une boîte de dialogue et pourront être visualisés à l'aide d'un navigateur Internet.

Les temps de calcul des algorithmes d'allocation d'offsets ne posent aucun problème en pratique, même pour des messageries de très grande taille. Le calcul des temps de réponse est un problème difficile algorithmiquement (NP-complet) et les temps de calcul varient considérablement (*i.e.*, exponentiellement) en fonction de la taille des messageries, des périodes des trames, mais aussi des choix d'offsets effectués. Sur les messageries réelles testées les temps de calcul sont très raisonnables (moins de 1 seconde pour une messagerie PSA avec 87 trames périodiques) mais, en particulier si le nombre de périodes différentes devait être plus important, le temps de calcul pourrait croître considérablement.

C.2 Illustration du problème sur l'algorithme dissimilar offset.

Cette annexe illustre le problème de l'algorithme dissimilar offset qui alloue des offsets identiques à de nombreux flux quand de multiples couples ont le même pgcd.

Considérons une station qui transmet 5 trames différentes. Le tableau *C.1* donne les valeurs des pgcd des différents couples.



(a) calcul temps de réponse (b) tracés des courbes

FIG. C.1 – Fenêtre principale du logiciel NetCarAnalyzer qui implémente les algorithmes d'allocation des déphasages initiaux des trames et le calcul de bornes sur les temps de réponse.

Flux	τ_1^2	τ_2^2	τ_2^2	τ_4^2	τ_5^2
τ_1^2	X	500	500	500	500
τ_2^2	X	X	100	100	100
τ_2^2	X	X	X	1000	1000
τ_4^2	X	X	X	X	1000
τ_5^2	X	X	X	X	X

TAB. C.1 – pgcd des couples d'une station.

Appliquons l'algorithme d'allocation dissimilar offset sur cette station. La liste des couples classés par valeur décroissante du pgcd est donc : (τ_2^2, τ_4^2), (τ_2^2, τ_5^2), (τ_4^2, τ_5^2), (τ_1^2, τ_2^2), etc. L'algorithme parcourt ensuite successivement la liste des couples afin d'allouer les décalages :

- (τ_2^2, τ_4^2) : O_2^2 et O_4^2 ne sont pas fixés (cf. cas 1 chapitre 5 § 5.2.2) :$O_2^2 = 20$ (au hasard) et $O_4^2 = O_2^2 + \lfloor \frac{1000}{2} \rfloor = 520$,
- (τ_2^2, τ_5^2) : O_2^2 est fixé et O_5^2 non fixé (cas 2 chapitre 5 § 5.2.2) : $O_5^2 = O_2^2 + \lfloor \frac{1000}{2} \rfloor = 520$,
- (τ_4^2, τ_5^2) : O_4^2 fixé et O_5^2 fixé (cas 3 chapitre 5 § 5.2.2) donc on ne fait rien,
- etc.....

On s'aperçoit que les décalages initiaux choisis par l'algorithme pour τ_4^2 et τ_5^2 sont égaux ce qui va induire des arrivées synchronisées, d'autant plus que les périodes de transmission sont égales.

Annexe D

Chapitre 6

D.1 Preuves de la propriété 1

Dans cette annexe, nous prouvons que l'analyse d'ordonnançabilité [1] garantit la propriété 1. Le premier paragraphe est consacré à l'étude de la fin d'exécution $e_{i,j}$ de $\tau_{i,j}$ calculée avec la fonction de calcul de borne supérieure sur le temps de réponse [1] dans les deux configurations $(\mathcal{P}, \Psi_{\mathcal{P}})$ et $(\mathcal{P}, \Psi'_{\mathcal{P}})$ dans lesquelles seule l'allocation des quantums diffère. La preuve donnée utilise les résultats de ce paragraphe.

D.1.1 Borne sur la fin d'exécution : propriétés de bases

Nous comparons les bornes sur la fin d'exécution de τ_i sous la même allocation de politique et de priorité $\mathcal{P}$ avec deux allocations de quantums $\Psi_{\mathcal{P}}$ et $\Psi'_{\mathcal{P}}$ différentes. Soit respectivement $e_{i,j}$ et $e'_{i,j}$ la borne sur la fin d'exécution de $\tau_{i,j}$ sous $(\mathcal{P}, \Psi_{\mathcal{P}})$ et sous $(\mathcal{P}, \Psi'_{\mathcal{P}})$. Comme τ_i est dans une couche RR, $e_{i,j}$ est calculée avec l'équation 6.4 du §6.2.4 :

$$e_{i,j} = \min\{t > 0 \mid \overline{\Psi}_i(t) + s_{i,j} = t\} \ ,$$

où (équation 6.5 du §6.2.4)

$$\overline{\Psi}_i^{\Psi_{\mathcal{P}}}(t) = \min\left(\left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}) + \widetilde{s}_i(t), s_i^*(x)\right) \ ,$$

où $\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}$ est la somme des quantums de toutes les autres tâches de la couche RR. Comme $s_{i,j}$, $\widetilde{s}_i(t)$ et $s_i^*(x)$ sont indépendants de l'allocation de quantums (cf. §6.2.4), il est suffisant de comparer le premier terme du min() pour dire dans quelle configuration la borne sur le temps de réponse est la plus petite. Deux cas peuvent se produire :

1. $\left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}) > \left\lceil \frac{s_{i,j}}{\psi_i^{\Psi'_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi'_{\mathcal{P}}} - \psi_i^{\Psi'_{\mathcal{P}}})$ alors on conclut que $e_{i,j} \geq e'_{i,j}$,
2. sinon :

$$\left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}) \leq \left\lceil \frac{s_{i,j}}{\psi_i^{\Psi'_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi'_{\mathcal{P}}} - \psi_i^{\Psi'_{\mathcal{P}}}) \ ,$$

et $e_{i,j} \leq e'_{i,j}$.

Lorsque dans l'équation 6.5 $s_i^*(x)$ est le minimum, on a $e_{i,j} = e_{i,j}^{'}$.

De ces remarques, nous déduisons que pour une autre allocation de quantums $\Psi_{\mathcal{P}}^{'}$, si les deux conditions suivantes sont garanties :

condition 1 : le quantum $\psi_i^{\Psi_{\mathcal{P}}^{'}}$ de τ_i sous $\Psi_{\mathcal{P}}^{'}$ est inférieur ou égal à son quantum $\psi_i^{\Psi_{\mathcal{P}}}$ sous $\Psi_{\mathcal{P}}$,

condition 2 : la somme des quantums de toutes les autres tâches dans la couche RR $\mathcal{T}_i^{\mathcal{P}}$ sous $\Psi_{\mathcal{P}}^{'}$ est supérieure ou égale à celle sous $\Psi_{\mathcal{P}}$, *i.e.*, $\overline{\psi}_i^{\Psi_{\mathcal{P}}^{'}} - \psi_i^{\Psi_{\mathcal{P}}^{'}} \geq \overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}$où $\overline{\psi}_i^{\Psi_{\mathcal{P}}} = \sum_{\tau_k \in \mathcal{T}_i} \psi_{\tau_k}^{\Psi_{\mathcal{P}}}$ est la somme des quantums de toutes les tâches de la couche RR $\mathcal{T}_i^{\mathcal{P}}$ sous l'allocation de quantums $\Psi_{\mathcal{P}}$,

alors nous avons :

$$\left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_{\mathcal{P}}^{'}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_{\mathcal{P}}^{'}} - \psi_i^{\Psi_{\mathcal{P}}^{'}}) \geq \left\lceil \frac{s_{i,j}}{\psi_i^{\Psi_{\mathcal{P}}}} \right\rceil \cdot (\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}}) \ , \tag{D.1}$$

et donc $\forall \tau_{i,j}$, $e_{i,j} \leq e_{i,j}^{'}$ qui implique que la borne supérieure sur le temps de réponse de τ_i sous $(\mathcal{P}, \Psi_{\mathcal{P}}^{'})$ est supérieure ou égale à la borne sous $(\mathcal{P}, \Psi_{\mathcal{P}})$.

D.1.2 Preuve

Comme les prérequis de la propriété 1 sont exactement les conditions 1 et 2 du §D.1.1, la borne sur le temps de réponse de τ_i n'est pas supérieure sous $(\mathcal{P}, \Psi_{\mathcal{P}}^{'})$ que sous $(\mathcal{P}, \Psi_{\mathcal{P}})$. Comme τ_i n'est pas faisable sous $(\mathcal{P}, \Psi_{\mathcal{P}})$, elle n'est pas faisable sous $(\mathcal{P}, \Psi_{\mathcal{P}}^{'})$.

D.2 Preuve de la propriété 2

Soit $(\mathcal{P}, \Psi_{\mathcal{P}})$ et $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ deux configurations d'ordonnancement telles que, pour une tâche arbitraire τ_i :

1. l'ensemble des tâches avec une priorité supérieure à τ_i sous $\mathcal{Q}$, noté $\mathcal{T}_{hp(p_i)}^{\mathcal{Q}}$, est un sous-ensemble de l'ensemble de tâches ayant une priorité supérieure à τ_i sous $\mathcal{P}$, *i.e.*, $\mathcal{T}_{hp(p_i)}^{\mathcal{Q}} \subseteq \mathcal{T}_{hp(p_i)}^{\mathcal{P}}$,
2. l'ensemble des tâches ayant une priorité égale à τ_i sous $\mathcal{Q}$, noté $\mathcal{T}_{p_i}^{\mathcal{Q}}$, est un sous-ensemble de l'ensemble de tâches ayant une priorité égale à τ_i sous $\mathcal{P}$, *i.e.*, $\mathcal{T}_{p_i}^{\mathcal{Q}} \subseteq \mathcal{T}_{p_i}^{\mathcal{P}}$,
3. et les tâches avec la même priorité que τ_i sous $\mathcal{Q}$, *i.e.*, les tâches de $\mathcal{T}_{p_i}^{\mathcal{Q}}$, ont le même quantum sous $\Psi_{\mathcal{Q}}$ et sous $\Psi_{\mathcal{P}}$, *i.e.*, $\forall \tau_j \in \mathcal{T}_{p_i}^{\mathcal{Q}}$, $\psi_j^{\Psi_{\mathcal{Q}}} = \psi_j^{\Psi_{\mathcal{P}}}$.

Nous montrons ici que la borne supérieure sur le temps de réponse sous $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ est inférieure ou égale à la borne supérieure sous $(\mathcal{P}, \Psi_{\mathcal{P}})$ et donc que la propriété 2 est garantie.

Comme la borne sur le pire temps de réponse de τ_i est exprimée avec l'équation 6.2 du §6.2.4, alors la propriété 2 est garantie si et seulement si :

$$\begin{gathered} j^* = \min\{j \mid (e_{i,j}^{\mathcal{Q}} \leq a_{i,j+1}^{\mathcal{Q}}) \vee (e_{i,j}^{\mathcal{P}} \leq a_{i,j+1}^{\mathcal{P}})\}, \\ \forall j \leq j^*, \ e_{i,j}^{\mathcal{Q}} - a_{i,j+1}^{\mathcal{Q}} \leq e_{i,j}^{\mathcal{P}} - a_{i,j+1}^{\mathcal{P}} \ , \end{gathered} \tag{D.2}$$

où $e_{i,j}^{\mathcal{P}}$ est la valeur de $e_{i,j}$ sous $(\mathcal{P}, \Psi_{\mathcal{P}})$. Comme la date d'arrivée $a_{i,j}$ est indépendante de l'allocation, $a_{i,j}^{\mathcal{P}} = a_{i,j}^{\mathcal{Q}}$ quelque soit i. Donc, l'équation D.2 devient :

$$\begin{aligned} j^* = \min\{j \mid \min(e_{i,j}^{\mathcal{Q}}, e_{i,j}^{\mathcal{P}}) \leq a_{i,j+1})\}, \\ \forall j \leq j^*,\ e_{i,j}^{\mathcal{Q}} \leq e_{i,j}^{\mathcal{P}}\ . \end{aligned} \tag{D.3}$$

Le calcul de $e_{i,j}^{\mathcal{P}}$ sous $(\mathcal{P}, \Psi_{\mathcal{P}})$ est réalisé avec l'équation 6.4 (RR) ou 6.3 (FPP). Dans ces équations, seul $\widetilde{s}_i^{\mathcal{P}}(t)$,$(\overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}})$ et $\overline{s}_i^{\mathcal{P}}(x)$ dépendent de la configuration d'ordonnancement $(\mathcal{P}, \Psi_{\mathcal{P}})$. On doit noter ici que les fonctions $\widetilde{s}_i(t)$, $(\overline{\psi}_i - \psi_i)$ et $\overline{s}_i(x)$ sous $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ sont majorées par ces fonctions sous $(\mathcal{P}, \Psi_{\mathcal{P}})$. En effet, d'après les conditions 1 et 2 :

$$\mathcal{T}_{hp(p_i)}^{\mathcal{Q}} \subseteq \mathcal{T}_{hp(p_i)}^{\mathcal{P}}$$

$$\begin{aligned} \Longrightarrow \sum_{\tau_k \in \mathcal{T}_{hp(p_i)}^{\mathcal{Q}}} s_k(t) \leq \sum_{\tau_k \in \mathcal{T}_{hp(p_i)}^{\mathcal{P}}} s_k(t) \\ \Longrightarrow \widetilde{s}_i^{\mathcal{Q}}(t) \leq \widetilde{s}_i^{\mathcal{P}}(t) \end{aligned} \tag{D.4}$$

Les conditions 2 et 3 sont :

$$\mathcal{T}_{p_i}^{\mathcal{Q}} \subseteq \mathcal{T}_{p_i}^{\mathcal{P}} \text{ et } \forall \tau_j \in \mathcal{T}_{p_i}^{\mathcal{Q}},\ \psi_j^{\Psi_{\mathcal{Q}}} = \psi_j^{\Psi_{\mathcal{P}}}$$

$$\Longrightarrow \overline{\psi}_i^{\Psi_{\mathcal{Q}}} - \psi_i^{\Psi_{\mathcal{Q}}} \leq \overline{\psi}_i^{\Psi_{\mathcal{P}}} - \psi_i^{\Psi_{\mathcal{P}}} \tag{D.5}$$

$$\text{et } \overline{s}_i^{\mathcal{Q}}(u+x) \leq \overline{s}_i^{\mathcal{P}}(u+x) \tag{D.6}$$

Comme $e_{i,j}$ est calculée avec l'équation 6.3 (resp. l'équation 6.4) quand τ_i est dans une couche FPP (resp. RR) sous $\mathcal{P}$ et sous $\mathcal{Q}$, $e_{i,j}^{\mathcal{Q}} \leq e_{i,j}^{\mathcal{P}}$ d'après les inégalités D.4, D.5, et D.6. Donc, l'équation D.3 est vraie et la propriété 2 est garantie. Le cas où τ_i est dans une couche RR sous $\mathcal{P}$ ($e_{i,j}^{\mathcal{P}}$ calculée avec l'équation 6.4) tandis que τ_i est dans une couche FPP sous $\mathcal{Q}$ ($e_{i,j}^{\mathcal{Q}}$ calculée avec l'équation 6.3) n'est pas détaillé ici mais peut-être prouvé d'une manière similaire.

D.3 Preuve d'optimalité de *Audsley-RR-FPP**

Nous montrons que l'algorithme *Audsley-RR-FPP**est optimal dans le sens où s'il existe une allocation de priorités, de politiques et de quantums faisable alors une configuration faisable est trouvée par *Audsley-RR-FPP**[6]. Nous rappelons qu'ici une configuration d'ordonnancement est faisable si, pour chaque tâche, la borne supérieure sur le pire temps de réponse, calculée avec une fonction de calcul vérifiant la propriété 2 décrite au §6.2.5, est inférieure ou égale à son échéance.

Nous rappelons d'abord le théorème suivant prouvé dans [14, 66, 10, 1] dans des contextes variés d'ordonnancement à priorité fixe.

Théorème 2 *[66] Soit $(\mathcal{P}, \Psi_{\mathcal{P}})$ une configuration d'ordonnancement faisable jusqu'à la priorité i,* i.e.*, les*

[6] Comme *Audsley-RR-FPP* est un cas particulier de *Audsley-RR-FPP** où la valeur du quantum est unique, la preuve est également valable pour *Audsley-RR-FPP*.

tâches ayant une priorité comprise entre n et i sont faisables. S'il existe une configuration d'ordonnancement $(\mathcal{A}, \Psi_{\mathcal{A}})$ faisable, alors il existe une configuration $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ faisable ayant la même configuration que $(\mathcal{P}, \Psi_{\mathcal{P}})$ pour les niveaux de priorité de n à i.

Grâce au théorème 2, nous pouvons prouver l'optimalité de *Audsley-RR-FPP**. En effet, s'il s'avère que *Audsley-RR-FPP**échoue au niveau de priorité i, l'allocation de priorité, de politiques d'ordonnancement et de quantums $(\mathcal{P}, \Psi_{\mathcal{P}})$ fournie par *Audsley-RR-FPP**est faisable jusqu'au niveau de priorité $i+1$. Comme *Audsley-RR-FPP**effectue une recherche exhaustive pour allouer le niveau de priorité i, il ne peut exister aucune configuration $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ ayant la même configuration que $(\mathcal{P}, \Psi_{\mathcal{P}})$ pour les priorités de $i+1$ à n. Donc, d'après le théorème 2, il n'existe pas d'ordonnancement faisable.

Nous donnons ici une preuve intuitive du théorème 2, qui est valide sous Posix grâce au lemme 6 et à la propriété 2. Nous faisons remarquer que le théorème 2, et donc le résultat sur l'optimalité de *Audsley-RR-FPP**n'est pas garantie si la propriété 2 n'est pas vérifiée par la fonction de calcul sur la borne supérieure sur le pire temps de réponse du test d'ordonnancement employé.

Le théorème 2 est garanti si une configuration d'ordonnancement faisable $(\mathcal{A}, \Psi_{\mathcal{A}})$ peut être transformée en une configuration d'ordonnancement faisable $(\mathcal{Q}, \Psi_{\mathcal{Q}})$ ayant la même configuration que$(\mathcal{P}, \Psi_{\mathcal{P}})$ pour les priorités de i à n. Cette transformation peut-être réalisée de manière itérative en changeant la configuration de certaines tâches de $(\mathcal{A}, \Psi_{\mathcal{A}})$ en la configuration qu'elles ont dans$(\mathcal{P}, \Psi_{\mathcal{P}})$. La procédure est la suivante : pour un niveau de priorité k de n à i, allouer dans $(\mathcal{A}, \Psi_{\mathcal{A}})$ la priorité $k+n-i$ aux tâches ayant la priorité k dans $(\mathcal{P}, \Psi_{\mathcal{P}})$ (*i.e.*, l'ensemble $\mathcal{T}_k^{\mathcal{P}}$) et fixer leur quantum à la valeur $\psi_i^{\mathcal{P}}$ qu'il a dans $\Psi_{\mathcal{P}}$ ($\forall \tau_j \in \mathcal{T}_k^{\mathcal{P}}$, $p_j^{\mathcal{A}} = p_j^{\mathcal{P}} + n - i$, $sched_j^{\mathcal{A}} = sched_j^{\mathcal{P}}$ et $\psi_j^{\Psi_{\mathcal{A}}} = \psi_j^{\Psi_{\mathcal{P}}}$). Comme à chaque étape, les tâches dans $\mathcal{T}_k^{\mathcal{P}}$ont le même quantum, le même ensemble de tâches de priorité supérieure et le même ensemble de même priorité sous la configuration courante $(\mathcal{A}, \Psi_{\mathcal{A}})$ que sous $(\mathcal{P}, \Psi_{\mathcal{P}})$, elles restent faisables sous la configuration $(\mathcal{A}, \Psi_{\mathcal{A}})$ courante grâce au lemme 6. Selon la propriété 2, les autres tâches $(\mathcal{T} \setminus \mathcal{T}_k^{\mathcal{P}})$ respectent également leur échéance car l'allocation de quantums et l'ensemble de tâches de priorité supérieure ou égale est réduit ou reste inchangé sous la configuration de priorité courante comparée à la configuration initiale $(\mathcal{A}, \Psi_{\mathcal{A}})$**.** On peut remarquer que dans la preuve, les niveaux de priorité ont été artificiellement étendus en ajoutant $n-i$ niveaux de faible priorité dans le but d'éviter le cas où une tâche de plus haute priorité est déplacée dans une couche non-vide car la propriété 2 ne traite pas cette situation.

Résumé

Notre objectif est la conception d'algorithmes d'ordonnancement temps réel en-ligne **faisables** (*i.e.*, garantissant le respect des contraintes temporelles imposées au systèmes) optimisant 1) **l'utilisation de la plate-forme d'exécution** (*i.e.*, utiliser au mieux le potentiel de la plate-forme d'exécution tout en garantissant le respect des contraintes temporelles imposées au système) et/ou 2) **optimisant des critères de performances propres à l'application**. Deux cas ont été analysés : le cas de tâches indépendantes périodiques s'exécutant sur un processeur et le cas de flux de messages indépendants périodiques sur un réseau de terrain avec accès au médium priorisé. Nous avons donc proposé pour traiter les deux problèmatiques abordées :

- des **méthodes de configurations** permettant d'optimiser l'utilisation de la plate-forme d'exécution en fixant d'une manière appropriée les paramètres : des politiques ou des activités du système considéré. Deux études ont été conduites dans ce cadre : l'**allocation des "offsets**" dans les systèmes "offset free" et l'**allocation de priorités, de politiques et de quantum** dans les systèmes conformes au standard **Posix 1003.1b**,
- une **nouvelle classe de politiques d'ordonnancement** permettant d'optimiser des critères de performances propres à l'application.

Mots-clés: ordonnancement, temps réel, multi-tâches, systèmes d'exploitation, priorité fixe, priorité dynamique, round-robin.

Abstract

Our goal is to make feasible (*i.e.*, all required time constraints are met) on-line real-time scheduling algorithms. These algorithms have to optimise 1) the **utilisation of the execution platform** (*i.e.*, meet time constraints and use platform at its fullest potential) and/or 2) **optimise the application dependent performance criteria**. We study two cases : the case of independent periodic tasks scheduled on a processor and the case of independent periodic traffic streams scheduled on a priority bus.

To deal with these two problems, we propose :

- **configuration methods** to allow to optimise the utilisation rate of the execution platform by setting the parameters of the policies or of the activities of the considered system. We perform two studies : the **allocation of offsets** in "offset free" systems (*i.e.*, offsets can be chosen off-line) and the **priorities, policies and quantum allocations** in systems compliant to the standard **Posix 1003.1b**,
- a **new scheduling policies class** to allow to optimise application performance dependent criteria.

Keywords: scheduling, real-time, multi-tasking, operating systems, fixed priority, dynamic priority, round-robin.