Thèse de Doctorat

Université de Bretagne Occidentale

présentée en vue d'obtenir le titre de

Docteur en Sciences

Spécialité informatique

par

Jérôme Legrand

le 1^{er} décembre 2004

Contribution à l'ordonnancement des systèmes temps réel comprenant des tampons

---

JURY

PRESIDENT :

Mr Jacques Tisseau Professeur des universités, ENIB Brest

RAPPORTEURS :

Mme Isabelle Puaut Professeur des universités, IRISA Rennes
Mr Guy Vidal-Naquet Professeur des universités, Supelec Paris

EXAMINATEURS :

Mme Isabelle Demeure Professeur, ENST Paris
Mr Lionel Marcé Professeur des universités, UBO Brest (directeur de thèse)
Mr Frank Singhoff Maître de conférences, UBO Brest

Remerciements

Je tiens tout d'abord à remercier Mr Lionel Marcé qui a accepté d'être mon directeur de thèse. Je le remercie pour son soutient sans lequel je n'aurais pas pu travailler aussi sereinement.

Je remercie Mr Frank Singhoff pour son encadrement, en particulier pour son investissement dans mon travail de thèse, sa disponibilité et son amitié. Mon travail est largement inspiré des idées émises lors de nos réunion et il a su me fournir l'energie nécessaire pour perséverer quand j'en manquais.

Je remercie Mr Laurent Nana sans qui je n'aurais peut être pas choisi le chemin de la recherche, pour son encadrement et pour son soutient sans faille.

Je remercie les deux rapporteurs de ce mémoire, Mme Isabelle Puaut de l'université de Rennes et Mr Guy Vidal-Naquet de Supelec Paris. J'adresse également mes remerciements à Mme Isabelle Demeure de l'ENST et Mr Jacques Tisseau de l'ENIB pour le temps qu'ils ont bien voulu consacrer à cette thèse en tant que membre du jury.

Je remercie Mr Gerardo Rubino pour l'aide qu'il m'a apporté concernant mon travail sur les files d'attente.

Cette thèse a été financée en partie grâce à un contrat région liant l'équipe d'acceuil 2215 et la société TNI. Je tiens donc à remercier la région pour sa participation financière mais aussi Mr François Dupont et Mr Pierre Dissaux de TNI.

De la même façon, je tiens à remercier tous ceux qui ont contribué à ce mémoire par la relecture des versions préliminaires. Merci donc à Maryse Berthou, Gilles Jacovetti, Charlotte Legrand, Valerie-anne Nicolas, et Mr Loic Plassart.

Pour terminer, j'exprime ma reconnaissance à l'ensemble des membres de l'équipe d'acceuil 3883.

Résumé

Cette thèse présente une approche pour l'étude de la faisabilité de systèmes temps réel embarqué et répartis. Nous considérons qu'un tel système est composé de sous-systèmes mono-processeur reliés entre eux par un réseau. Les sous-systèmes comprennent des tâches périodiques indépendantes et des tampons. Les tampons reçoivent des messages par l'intermédiaire du réseau ou d'autre tâches du sous-système. Les messages sont ensuite consommés par une tâche périodique. Nous proposons une approche pour tester la faisabilité de ces sous-systèmes. En particulier, nous proposons des critères maximums et moyens de performance pour l'étude des tampons. Ces critères sont basés sur la théorie des files d'attente et sur des résultats provenant des réseaux ATM.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Le terme "système temps réel" regroupe l'ensemble des systèmes dont les spécifications contiennent des contraintes de nature temporelle. Il peut s'agir des temps de réponse à un événement donné, des rythmes de traitement d'informations. Parmi les domaines concernés, nous trouvons l'aéronautique, le spatial, l'automobile, le ferroviaire, le médical, les télécommunications, la défense ...

Dans les systèmes temps réel, il n'est pas suffisant de considérer uniquement l'exactitude des résultats produits par l'application, c'est à dire, contrôler si la mise en œuvre des fonctionnalités correspondent à leurs spécifications. La difficulté spécifique des systèmes temps réel est de s'assurer de leur faisabilité temporelle. Autrement dit, il faut vérifier le respect effectif de leurs contraintes temporelles. Afin de répondre à ce besoin, différentes approches ont été proposées pour simuler voire vérifier de tels systèmes (par exemple les réseaux de Petri [Pet81], les langages synchrones [GGBM91], ...). Dans le cadre de cette thèse, nous étudions l'une de ces méthodes généralement désignée sous le terme de théorie de l'ordonnancement temps réel.

Dans la terminologie de l'ordonnancement temps réel, une **tâche** représente un traitement ou un ensemble de traitements de l'application. Les tâches sont en concurrence pour l'accés aux ressources, dont le processeur, et il est nécessaire de définir la manière dont elles seront exécutées sur ce dernier. Les **algorithmes d'ordonnancement** de tâches, tels que Rate Monotonic (RM) ou Earliest Deadline First (EDF) [SSNB95, AB90], remplissent ce rôle. Une application est faisable, ou ordonnançable, si toutes les tâches qui la composent respectent leurs contraintes temporelles. Il s'agit de déterminer si les tâches du système modélisé sont exécutées dans les temps. Si c'est le cas, on parle également d'ordonnancement valide de tâches. De nombreux tests de faisabilité sont proposés dans la littérature. L'article fondateur de Liu et Layland [LL73] présente un critère d'ordonnançabilité pour un système de plusieurs tâches indépendantes se partageant un seul processeur. Des travaux ont été menés afin d'étendre les résultats de Liu et Layland afin de les rendre applicables à des contextes de plus en plus réalistes [Riv98, Leb98].

Cette thèse traite de la faisabilité des systèmes temps réel répartis.

Nous considérons qu'un système réparti est constitué de plusieurs processeurs reliés par un réseau. Ce modèle d'application correspond à une pratique industrielle que l'on retrouve, par exemple, dans l'avionique modulaire : le système est constitué de sous-systèmes communiquant par l'intermédiaire d'un réseau [Ari97]. Ces sous-systèmes sont éventuellement fournis par différents partenaires industriels. Le fonctionnement global du système est défini par un intégrateur de système. Nous supposons qu'un sous-système est composé de tâches **périodiques** et de **tampons**. Les tampons collectent des informations délivrées par un autre sous-système au travers du réseau ou périodiquement lors d'une communication entre tâches de ce même sous-système.

Nous proposons d'étendre les résultats de la théorie de l'ordonnancement temps réel à ce type d'architecture. Notre objectif est de préciser la notion de faisabilité pour le modèle d'application étudié.

D'une part, la vérification du respect des contraintes temporelles est accomplie grâce aux méthodes classiques de faisabilité de système de tâches périodiques (borne sur le taux d'utilisation processeur, calcul du temps de réponse [JP86, ABRT93], ...). Nous cherchons ici à vérifier des contraintes temporelles qui sont locales à chaque sous-système.

D'autre part, **il est nécessaire que le nombre de messages présents dans le tampon n'excède pas sa taille**. Pour cela, il faut soit, valider la taille du tampon étudié, soit, déterminer une taille nécessaire et dans la mesure du possible suffisante. Nous proposons des critères de performances permettant d'analyser ces tampons. Ces critères sont de type déterministes (pire cas) ou probabiliste (cas "moyen").

La théorie des files d'attente propose des critères de performance pour dimensionner les tampons. Les tâches qui consomment les messages sont alors soumises à une contrainte de précédence : elles sont activées dès qu'un message arrive dans le tampon. Malheureusement, dans ce cas, la vérification du respect des contraintes temporelles des tâches est plus difficile pour les raisons que nous allons énoncer.

Tout d'abord, les tests de faisabilité disponibles pour des jeux de tâches avec précédence imposent des restrictions sur leurs paramètres. Il faut notamment que les périodes des tâches liées par une contrainte de précédence soient identiques [RCR01].

En outre, les tâches sont activées selon le type d'arrivée des messages. Peu de résultats concernant la faisabilité existent en dehors des modèles de tâches classiques tels que les modèles périodiques et sporadiques. Par exemple, Lehoczky a proposé des tests de faisabilité pour un jeu de tâches activées aléatoirement [Leh96].

Cette thèse explore une voie différente qui consiste à privilégier les jeux de tâches pour lesquels les tests de faisabilité sont simples, peu contraignants et permettent d'obtenir des garanties. Nous considérons donc les jeux de tâches périodiques sans contrainte de précédence. La vérification du respect des contraintes temporelles est alors relativement aisée [Riv98]. Toutefois, la difficulté est reportée sur le dimensionnement des tampons, il n'est plus possible d'appliquer directement les résultats de la théorie des files d'attente.

Contributions de cette thèse

Les contributions faites dans cette thèse sont de trois ordres : des contributions de modélisation, des contributions théoriques et des contributions techniques.

Contribution sur le plan de la modélisation de systèmes temps réel comprenant des tampons

Nous proposons un modèle d'application temps réel avec tampons. Notre application reçoit des données des sous-systèmes. Il n'y a pas de contrainte de précédence entre l'arrivée de ces données dans le tampon et l'activation de la tâche qui va consommer ce message.

Par conséquent, nous pouvons utiliser les tests de faisabilité disponibles dans la littérature pour vérifier le respect des contraintes temporelles des sous-systèmes mono-processeur. Par contre, l'analyse des tampons devient plus difficile.

Ce modèle d'application a été étudié dans le cadre d'une collaboration avec la société TNI [LSM+02].

Les tampons de notre application sont modélisés grâce à la théorie des files d'attente. Dans cette théorie, classiquement, la précédence entre l'arrivée des messages et le consommateur existe. Nous proposons donc une loi de service P afin de prendre en compte le caractère indépendant des tâches périodiques.

Contributions théoriques

Nous étudions deux types de file d'attente : les files M/P/1 et P/P/1. Nous proposons une résolution approchée de la file M/P/1 et une résolution exacte de la file P/P/1.

La file M/P/1 décrit un tampon partagé par des flux d'arrivées aléatoires de messages et une tâche périodique qui consomme ces messages [SLNM04b, LSNM05]. Nous proposons une approximation de la file M/P/1 basée sur les files d'attente classiques telles que la file M/G/1. Il est donc nécessaire d'évaluer le temps de service moyen W_s et sa variance σ_s^2. Grâce à ces deux paramètres, nous pouvons utiliser les critères de performance issus de la théorie des files d'attente. Le temps d'attente et le taux d'occupation mesurés sur les simulations diffèrent, selon les jeux de tâches, de 0% à 10% avec les critères théoriques.

La file P/P/1 décrit un tampon partagé par n tâches périodiques qui produisent des messages et une tâche périodique qui les consomme [LSN+03]. La résolution de la file P/P/1 est basée sur les similitudes qui existent entre cette file et certains résultats issus du monde des réseaux hauts débits. C'est le cas des services de communication utilisés par le transport de la voix dans les réseaux ATM : la couche d'adaptation AAL1 [GK96]. Nous nous sommes basés sur ces résultats afin de proposer des bornes maximums pour le taux d'occupation et le temps d'attente des messages pour la file P/P/1. Des simulations menées sur la file P/P/1 confirment les résultats attendus par la résolution exacte.

Contributions techniques

Cheddar est un logiciel de modélisation et d'analyse d'applications temps réel [SLNM04a]. Nous étendons ses fonctionnalités pour prendre en compte les tampons.

En ce qui concerne la modélisation, nous ne nous sommes pas contentés des caractéristiques des tampons étudiés dans cette thèse. Notamment, il est possible de choisir si le ou les consommateurs sont activés ou non sur réception de messages.

L'analyse des tampons est réalisé grâce à différents critères de performance. Les critères moyens de performance de la théorie des files d'attente classiques, telles que M/M/1,

M/D/1 ou M/G/1, ont été implémentés, ainsi que les critères proposés pour les files M/P/1 et P/P/1. En outre, il est possible de suivre graphiquement l'évolution du taux d'occupation du tampon en fonction du chronogramme d'ordonnancement des tâches.

Finalement, des algorithmes de simulation ont été mise en œuvre à partir des boites à outils disponibles dans Cheddar. Les simulateurs ont pour objectif de confirmer l'exactitude de la résolution de la file P/P/1 et la justesse de la résolution approchée de la file M/P/1.

Plan de ce mémoire de thèse

Nous commençons dans le chapitre 2 par dresser un état de l'art concernant les systèmes temps réel. Nous décrivons plus particulièrement les tests permettant de vérifier les contraintes temporelles de ces systèmes. Ensuite, nous faisons un bref rappel des résultats classiques de la théorie des files d'attente. Enfin, nous présentons les motivations des choix effectués dans le cadre de notre travail.

Dans le chapitre 3, nous détaillons le modèle d'application ciblé ainsi que la modélisation des tampons par des files d'attente. Nous présentons également les problèmes spécifiques à notre modèle d'application.

Les chapitres 4, 5 et 6 sont consacrés à la description de nos résultats.

Tout d'abord, nous présentons nos propositions concernant la résolution approchée de la file M/P/1. Elle consiste en une approximation du temps de service moyen de la loi P et de sa variance.

Ensuite, nous présentons nos propositions concernant la résolution exacte de la file P/P/1. Nous faisons une description des résultats concernant les réseaux ATM, résultats sur lesquels nous nous sommes basés. Nous en déduisons des bornes maximums sur la taille des tampons et sur le temps d'attente des messages.

Finalement, nous menons des simulations sur les files M/P/1 et P/P/1. Les algorithmes de simulation ainsi que leur implémentation sont décrits. Nous faisons, ensuite, une analyse des résultats obtenus.

Dans le chapitre 7, nous donnons un aperçu des fonctionnalités du logiciel Cheddar. Nous présentons les modifications apportées afin de modéliser et d'analyser les applications temps réel comprenant des tampons.

Dans le chapitre 8, nous proposons une première approche de l'analyse temporelle d'un plan PILOT. Le langage PILOT est un langage graphique permettant de concevoir et d'exécuter des missions pour des robots mobiles télé-opérés. Ce langage est développé dans le laboratoire EA3883/UBO [NLSM03].

Enfin, nous concluons cette thèse au chapitre 9. Nous en profitons pour évaluer les perspectives de recherche de nos travaux.

Chapitre 2

Etat de l'art et motivations

Dans ce chapitre, nous faisons un état de l'art des tests de faisabilité des systèmes temps réel et de la théorie des files d'attente.

L'objectif de ce travail est, bien entendu, de faciliter la compréhension des résultats que nous proposons dans les chapitres suivants mais également de mieux appréhender les éléments qui ont motivé notre travail. Nous présentons notamment les raisons qui nous ont poussé à choisir le modèle d'application étudié dans cette thèse.

Ce chapitre est organisé en quatre parties :

- La section 2.1 définit ce qu'est un système temps réel.

- Les sections 2.2, 2.3 et 2.4 donnent les principaux résultats concernant les systèmes temps réels. Premièrement, nous définissons le type de système étudié : les ordonnanceurs, le modèle de tâche,... Ensuite, nous faisons une synthèse des solutions de la littérature qui permettent de valider de manière temporelle les applications temps réel. Ces applications sont exécutées sur des plates-formes mono-processeurs, multiprocesseurs ou réparties. Un rappel des notations utilisées se trouve en annexe de ce document (cf. Annexe C).

- La section 2.5 rappelle quelques résultats classiques de la théorie des files d'attente. Nous présentons les résultats de travaux traitant à la fois de contraintes temps réel et de file d'attente. Nous présentons les différentes approches qui prennent en compte les contraintes temps réel de messages transitant dans des files d'attente (contrôle d'admission de flux), ou encore, les solutions utilisant les résultats présentés dans la section 2.5 pour résoudre l'ordonnancement de tâches temps réel activées aléatoirement.

- Nous détaillons, dans la section 2.6, les motivations de nos travaux sur les tampons

2.1 Systèmes temps réel

Le terme "système temps réel" regroupe l'ensemble des systèmes dont les spécifications contiennent des informations de nature temporelle, telles que des temps de réponse à un événement donné, des rythmes de traitement d'informations, ... Parmi les domaines concernés, nous trouvons l'aéronautique, le spatial, l'automobile, le ferroviaire, le médical, les télécommunications, la défense ...

Il n'est pas suffisant de considérer uniquement l'exactitude des résultats produits par l'application temps réel, c'est à dire, contrôler si la mise en œuvre des fonctionnalités correspond à leurs spécifications. La difficulté spécifique des systèmes temps réel est d'assurer le respect effectif des contraintes temporelles. Afin de répondre à ce besoin, différentes approches ont été proposées pour modéliser de tels systèmes.

Les systèmes que nous étudions sont composés d'un ou plusieurs processeurs éventuellement reliés entre eux. Sur chacun des processeurs est exécuté un ensemble de traitements.

La notion de faisabilité est primordiale dans les systèmes temps réel. Dans la terminologie temps réel, une **tâche** représente un traitement ou un ensemble de traitements. Un processeur ne peut exécuter qu'une seule tâche à la fois. Les tâches sont donc en concurrence et il est nécessaire de définir la manière dont les tâches seront exécutées sur le processeur. Nous verrons par la suite que les **algorithmes d'ordonnancement** de tâches, tels que Rate Monotonic (RM) ou Earliest Deadline First (EDF), remplissent ce rôle. Une application est faisable, ou ordonnançable, si toutes les tâches qui la composent respectent leurs contraintes temporelles. Il s'agit de déterminer si les tâches du système modélisé sont exécutées dans les temps. Le cas échéant, on dit que l'ordonnancement des tâches est valide. De nombreux tests de faisabilité sont proposés dans la littérature. L'article fondateur de Liu et Layland présente des critères d'ordonnançabilité pour un système de plusieurs tâches indépendantes se partageant un seul processeur [LL73]. De nombreux résultats étendent les travaux de Liu et Layland afin de les rendre applicables à des contextes de plus en plus réalistes [Riv98, Leb98, KRP+94].

Les systèmes temps réel sont souvent séparés en deux catégories :

- Les systèmes temps réel **durs** ou **critiques**, dans lesquels le non respect des contraintes temporelles entraîne des conséquences graves d'un point de vue humain ou économique. Par exemple, on ne peut pas se permettre d'incertitude sur un système de contrôle d'un avion ou un système de supervision d'une centrale nucléaire.

- Les systèmes temps réel **mous**, pour lesquels le non respect des contraintes temporelles correspond à un mode dégradé. Par exemple, le retard d'une ou plusieurs images dans une application de visio-conférence ne remet pas en cause le bon déroulement de son exécution.

Certaines dépendances compliquent l'analyse temporelle de l'application. Les tâches peuvent échanger des informations soit par le biais de ressources partagées (zones mémoires du processeur), soit par l'échange de messages circulant sur un réseau. Les traitements peuvent également être soumis à des contraintes de précédence. Ces différentes contraintes ont une influence sur l'exécution des tâches et doivent donc être prises en compte lors de la validation temporelle du système. En outre, ces dépendances rendent plus difficile l'obtention de tests de faisabilité [SSNB95] et invalident les tests classiques [RPGC02].

Avant de présenter les résultats classiques de faisabilité des systèmes temps réel, nous devons tout d'abord décrire le modèle de tâche utilisé et les algorithmes classiques d'ordonnancement. Nous donnons également quelques éléments techniques des applications temps réel.

2.1.1 Modèles de tâches

Nous avons vu qu'une tâche exécute un ou plusieurs traitements de l'application temps réel. La plupart des résultats que l'on trouve dans la littérature sont proposés dans le cadre de traitements cycliques ou événementiels. Une tâche est généralement décrite par un ensemble de paramètres dont la valeur dépend des spécifications de l'application. Comme nous le verrons par la suite, il peut être difficile d'obtenir ces valeurs nécessaires pour valider temporellement une application.

Nous détaillons maintenant chacun des paramètres qui composent le modèle de tâche que nous utiliserons par la suite. Ce modèle est dérivé du modèle classique introduit par Liu et Layland [LL73].

Les tâches peuvent être de type périodique, sporadique ou apériodique.

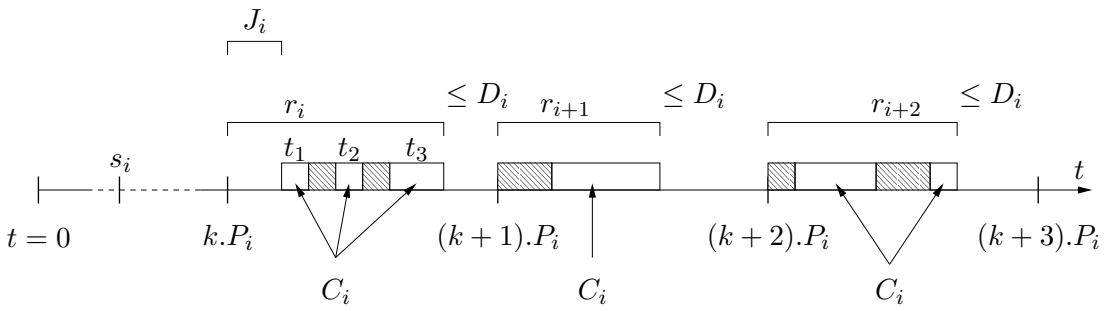

Fig. 2.1 – Le modèle de tâche périodique i

- Une tâche **périodique** modélise un traitement répétitif (cf. figure 2.1). La tâche est activée toutes les P_i unités de temps. Ce délai fixe entre chaque activation est communément appelé période.

- Lorsque le traitement possède un comportement cyclique variable, la période P_i représente la période de temps minimum entre deux réveils successifs de la tâche. On parle alors de tâche **sporadique**.

- Un traitement événementiel est modélisé par une tâche **apériodique** (cf. figure 2.2). La date d'activation, noté s_i, correspond à l'arrivée de la tâche dans le système.

La capacité C_i est une borne sur la durée d'exécution de la tâche i. Plusieurs approches ont été proposées pour estimer cette capacité [CCDP00] :

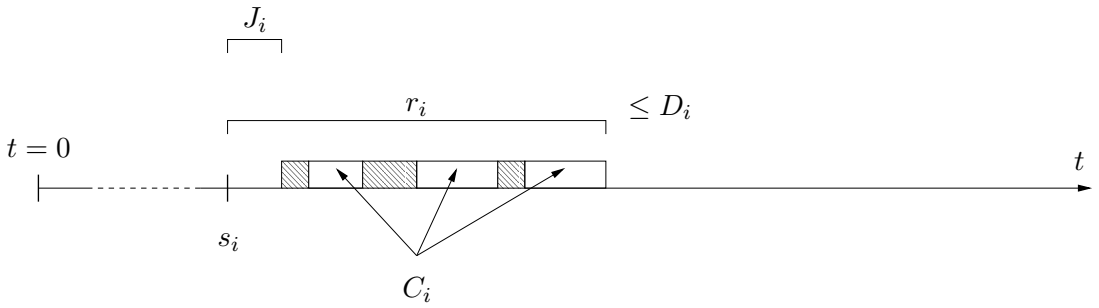

FIG. 2.2 – Le modèle de tâche apériodique i

- L'analyse WCET[1]. La borne sur le temps d'exécution est déterminé à partir de l'étude du code des tâches.

  La présence de mécanismes matériels, tels que le cache, le "pipeline" ou la prédiction de branchement, compliquent l'analyse du WCET. Toutefois, il est possible de prendre en compte certains mécanismes dans le calcul du WCET, tel que le cache [Pua02].

- L'exécution de la tâche. La tâche est exécutée dans des conditions particulières pour obtenir son temps d'exécution maximum sans interférence des autres tâches.

Sur la figure 2.1, la capacité de la tâche i est égale à la somme des intervalles de temps t_1, t_2 et t_3. Ce traitement doit être achevé au plus tard à un instant désigné par le terme "délai critique".

Le délai critique, noté D_i, est défini relativement à l'activation de la tâche i.

Une tâche i peut également être caractérisée par son temps de réponse. Généralement noté r_i, ce paramètre est le délai entre la date de première activation de i et l'instant de sa terminaison. r_i est donc relatif à s_i, l'instant où la tâche arrive dans le système. Les contraintes temporelles d'un système temps réel sont respectées si pour toutes les tâches i, $r_i \leq D_i$.

Les différentes tâches ont une priorité statique ou dynamique. Les priorités sont utilisées pour décider, à chaque instant, quelle tâche est exécutée sur le processeur : la tâche active, ayant la priorité la plus importante et dont la capacité n'est pas épuisée, est choisie pour être exécutée. Une priorité statique reste fixe durant la vie du système tandis qu'une priorité dynamique peut évoluer au cours du temps. Ces priorités sont utilisées pour connaître la tâche exécutée à un instant donné.

[1] Worst Case Execution Time

En pratique, il est difficile de réaliser une activation qui soit strictement périodique. Aussi, le paramètre J_i est une borne sur la latence entre l'activation théorique d'une tâche et son activation réelle due, par exemple, à la précision de l'horloge de l'ordonnanceur (cf. figures 2.1 et 2.2). L'activation d'une tâche est l'instant où elle est prête à être exécutée sur le processeur. Ainsi, la $k^{ème}$ activation d'une tâche périodique intervient dans l'intervalle $[k * P_i,\ J_i + k * P_i]$ qui correspond au délai entre la date théorique d'activation $k * P_i$ et cette date plus J_i.

Finalement, nous retenons les définitions suivantes :

Définition 1 (Le modèle de tâche périodique (ou sporadique)) *Une tâche périodique (ou sporadique) i est définie par l'ensemble des paramètres $\{s_i,\ J_i,\ P_i,\ C_i,\ D_i$ et $r_i\}$*

Définition 2 (Le modèle de tâche apériodique) *Une tâche apériodique i est définie par l'ensemble des paramètres $\{s_i,\ J_i,\ C_i,\ D_i$ et $r_i\}$.*

Il existe beaucoup de résultats permettant de vérifier la faisabilité d'applications temps réel modélisées par des traitements périodiques. Il existe beaucoup moins de résultats lorsque des dépendances dues à des ressources partagées, où des précédences, existent entre les tâches.

En outre, les résultats proposés dans la littérature sont souvent des pires cas. Une partie des ressources disponibles est donc réservée inutilement. En particulier, le modèle de Liu et Layland ne considère que les temps d'exécution pire cas, ce qui peut mener à qualifier un système de non faisable alors qu'un ordonnancement existe.

Ainsi, plusieurs extensions du modèle de tâche classique ont été développées afin d'offrir une modélisation plus proche des applications existantes ainsi qu'une utilisation plus fine des ressources du système.

Le modèle **Multiframe** propose de prendre en compte le fait que les durées d'exécution des différentes instances d'une tâche peuvent varier afin de soumettre une amélioration de l'utilisation du processeur. Cette technique est intéressante lorsque les temps d'exécution des tâches varient suffisamment (exemple : flux vidéo,...) [MC96, MBCG98, Tak98].

Un autre modèle de tâche, le modèle **Elastique**, caractérise une tâche par une capacité, une période, une période minimum, une période maximum et un coefficient. La période est équivalente à un ressort auquel on associe une rigidité : plus le coefficient est grand, plus la tâche est amenée à modifier sa période selon les besoins en qualité de service [BLA98]. Il s'agit aussi d'un modèle utilisé dans le cadre d'applications multimédias.

Dans [KT96], un modèle de tâche orienté média continu (flux vidéo, son, ...) est proposé : le modèle **Q_Thread**[2]. C'est une extension du modèle classique de tâche périodique où

[2]Quality Of Service Thread

la période et la capacité sont variables afin de s'adapter à la qualité de service désirée. Plusieurs politiques de contrôle sont disponibles : les auteurs proposent de conserver la capacité, la période ou d'équilibrer les deux. Le comportement d'un événement peut être modélisé par une tâche sporadique possédant deux périodes. L'une pour un mode de fonctionnement normal d'exécution et l'autre lors de la réception en rafale de messages [Tin92].

Le modèle **LBAP**[3] permet une représentation plus fine d'un trafic réseau et introduit le concept de qualité de session afin d'obtenir un contrôle sur les allocations des ressources [Ham97, YAWM97].

Enfin, le modèle **RBE**[4] caractérise les activations successives d'une tâche. Le taux d'arrivée peut être supérieur au taux attendu sans que les contraintes temporelles de l'application ne soient violées [JG99, JB95].

Par rapport aux résultats issus de l'approche classique, ces extensions améliorent effectivement la faisabilité des applications. Cette amélioration est obtenue grâce à une modélisation plus spécifique ou à l'utilisation d'heuristiques, au détriment de la généralité des résultats.

2.1.2 Ordonnanceurs de tâches

Nous considérons principalement dans cette thèse les algorithmes d'ordonnancement basés sur la priorité des tâches. Il existe d'autres types d'ordonnanceur, dit "hors-ligne", où la séquence d'ordonnancement des tâches est prédéterminée.

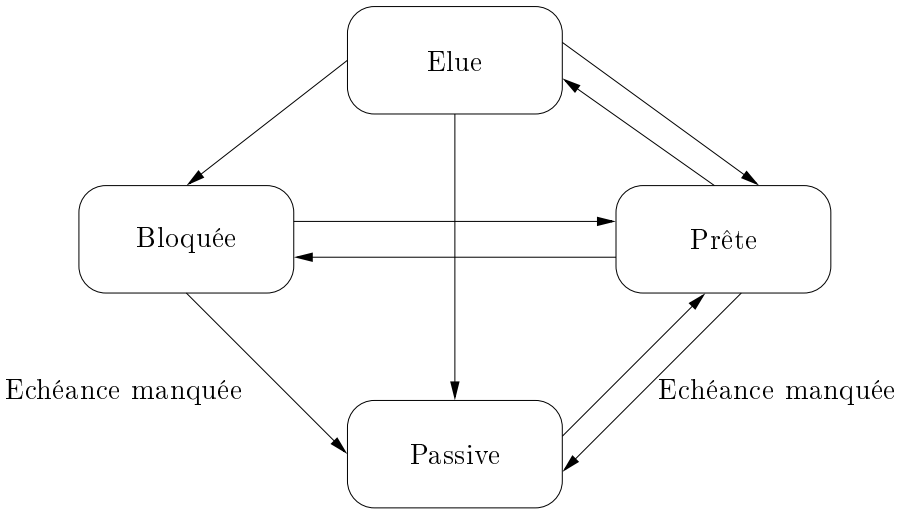

FIG. 2.3 – Les différents états d'une tâche

[3]Linear Bounded Arrival Processes
[4]Rate Based Execution

Une tâche passe par différents états au cours de son cycle de vie avant, pendant et après son exécution sur le processeur (cf. figure 2.3 [CDKM00]) :

- Une tâche est **élue** lorsqu'elle est choisie par l'ordonnanceur pour être exécutée sur un processeur.
- Une tâche est **prête** si elle peut être exécutée.
- Une tâche est **bloquée** lorsqu'elle tente d'accéder à une ressource et que celle-ci est indisponible. Par exemple, elle peut déjà être en cours d'utilisation (mémoire, entrée/sortie,...).
- Une tâche est **passive** dans les autres cas.

Une tâche est instanciée à chacune de ses activations. Cette opération consiste à mettre le code de la tâche en mémoire pour qu'il soit exécuté. On parle d'instance de la tâche.

Les ordonnanceurs ont pour rôle d'élire la tâche qui doit être exécutée sur un processeur à un instant donné. Ce rôle est essentiel car la faisabilité du système temps réel, en terme de respect des contraintes temporelles des tâches, dépend de la manière dont ces tâches sont ordonnancées.

Nous expliquons brièvement le fonctionnement d'un ordonnanceur et nous décrivons certaines de leurs caractéristiques (préemptif ou non, optimal,...) afin de mieux appréhender et classer les tests de faisabilité que nous présentons par la suite.

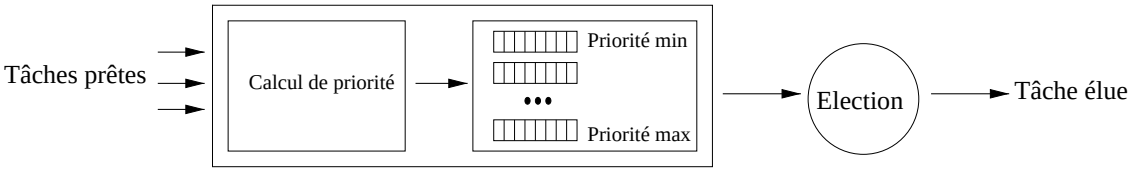

FIG. 2.4 – Fonctionnement d'un ordonnanceur

Il est possible de décrire le fonctionnement d'un ordonnanceur en trois étapes (cf. figure 2.4) :

1. Les priorités des tâches sont calculées en fonction de l'algorithme d'ordonnancement. Les priorités des tâches peuvent être fixées une fois pour toutes ou évoluer à chaque instant pendant la vie du système.

2. Une gestion de file d'attente intervient pour les tâches ayant la même priorité. Différentes politiques peuvent être appliquées telles que FIFO (attente de la terminaison de la tâche courante pour passer à la tâche suivante), Round-Robin (identique à FIFO mais passage à la tâche suivante après un quantum de temps) ou des politiques hybrides (POSIX 1003.b [Gal95], Solaris [Vah96], ...).

3. Durant la phase d'élection, la tâche en tête de la file d'attente de plus forte priorité est élue.

Suivant le fonctionnement des ces trois étapes, il est possible de classer les ordonnanceurs selon les critères suivants :

Définition 3 (Algorithme optimal) *Un algorithme d'ordonnancement A est optimal parmi les algorithmes de la classe C si $A \in C$ et que A propose un ordonnancement pour tous les systèmes de tâches qui sont faisables dans C. Lorsque la classe n'est pas précisée, tous les algorithmes d'ordonnancement sont considérés [CFH+03].*

Définition 4 (Ordonnancement statique et dynamique) *Dans le cas d'un ordonnancement statique, le jeu de tâches et toutes ses caractéristiques (délais critiques, capacités, contraintes de précédences,...) sont parfaitement connus. Au contraire, un ordonnanceur dynamique supporte que de nouvelles tâches soient ajoutées ou supprimées pendant la vie du système [SSNB95].*

Définition 5 (Algorithme d'ordonnancement hors-ligne et en ligne) *Un algorithme d'ordonnancement hors-ligne produit, à partir des paramètres des tâches, les priorités utilisées par l'ordonnanceur avant l'exécution de l'application temps réel (priorités dites statiques). Pour un algorithme en ligne, les priorités évoluent durant l'exécution de l'application (priorités dites dynamiques) [SSNB95].*

Définition 6 (Ordonnanceur préemptif et non préemptif) *Avec un ordonnanceur préemptif (resp. non préemptif), une tâche peut être interrompue par une autre tâche de plus forte priorité (resp. ne peut être interrompue). Les algorithmes préemptifs sont souvent de plus faible complexité que les algorithmes non préemptifs. Il est à noter que, même si l'algorithme d'ordonnancement est préemptif, le partage de ressources dans un système implique la présence de sections critiques qui sont des zones non préemptibles [SSNB95].*

Définition 7 (Algorithme d'ordonnancement non oisif) *Lorsque l'algorithme d'ordonnancement est non oisif, à chaque instant t, le processeur doit exécuter une instance de tâche active, s'il y en a dans le système.*

Définition 8 (Changement de mode) *Le changement de mode correspond à la suppression, au changement ou à l'ajout de tâches dans le système. Chaque mode offre un ensemble de fonctionnalités nécessaires au système à un moment donné. Le changement de mode permet également de prendre en compte les événements anormaux (erreurs, réexécution de tâches,...) [TBW92].*

On peut donc classer les algorithmes d'ordonnancement selon les priorités en trois catégories [CFH+03] :

- Les priorités des tâches sont statiques ou fixes.
- La priorité des tâches est considérée comme dynamique mais chacune de leurs instances possède une priorité statique.
- La priorité des tâches et de leurs instances évolue dans le temps.

En général, la distinction entre les deux dernières catégories n'est pas faite. Elles regroupent les algorithmes d'ordonnancement pour les jeux de tâches à priorité dynamique.

Définition 9 (Surcharge d'un système temps réel) *On dit que le système est en surcharge lorsque plusieurs tâches sont en mesure d'accéder au processeur au même instant.*

Le choix de la tâche qui doit être effectivement exécutée dépend alors de la manière dont l'ordonnanceur est programmé. Dans ces situations, il peut être difficile de déterminer les tâches qui ne respectent pas leur échéance. Ce qui peut ne pas être acceptable pour un système temps réel critique. En outre, des solutions permettent de prendre en compte les ressources partagées et de supporter les tâches apériodiques [AB90].

2.1.2.1 Algorithmes d'ordonnancement à priorités fixes

Les algorithmes à priorités fixes les plus connus sont les suivants : Rate Monotonic (RM), Deadline Monotonic (DM) et High Priority First (HPF) [SSNB95, AB90].

- L'algorithme d'ordonnancement **RM** attribue la priorité la plus forte à la tâche dont la période est la plus petite. C'est un algorithme hors ligne.
- Lorsque l'algorithme d'ordonnancement est **DM**, la tâche périodique dont la valeur du délai critique D_i est la plus petite possède la plus grande priorité.
- Les priorités peuvent également être choisies manuellement par le concepteur de l'application. On parle d'algorithme **HPF**.

Considérons un jeu de deux tâches T_1 et T_2 ordonnancées par RM (exemple tirés du livre [DB99]). Les tâches sont décrites par leur capacité, $C_1 = 6$ et $C_2 = 9$, et leur période, $P_1 = 10$ et $P_2 = 30$. Les délais critiques sont égaux aux périodes.

Les figures 2.5 et 2.6 donnent l'ordonnancement de ce jeu de tâches dans les cas préemptif et non préemptif.

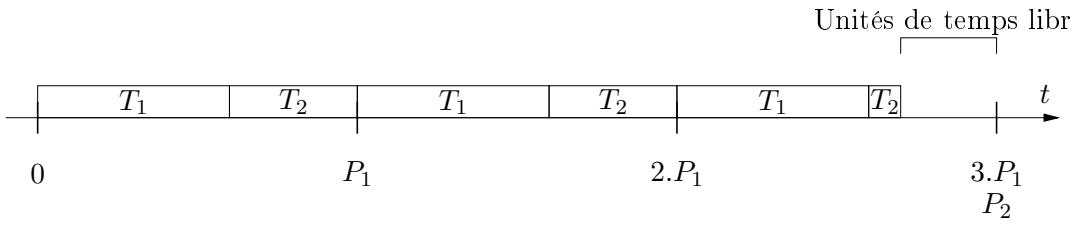

FIG. 2.5 – Ordonnancement RM préemptif

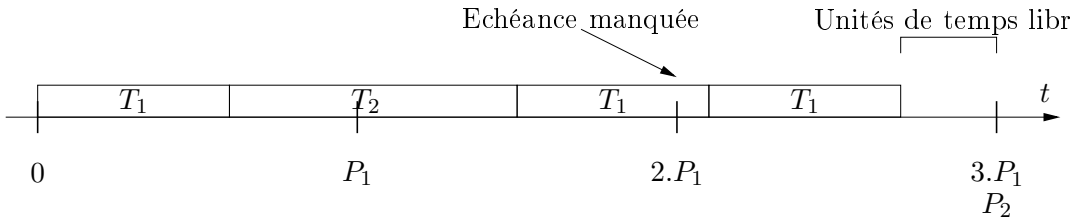

FIG. 2.6 – Ordonnancement RM non-préemptif

RM a été introduit par LIU et LAYLAND en 1973 [LL73] et reste encore très populaire [CFH+03]. En effet, RM possède quelques caractéristiques très avantageuses telles que son optimalité parmi les algorithmes à priorités fixes pour des jeux de tâches périodiques, sa simplicité de mise en œuvre et sa prédictibilité en surcharge.

D'un point de vue plus général, le principal inconvénient des algorithmes d'ordonnancement à priorité fixe est le manque de réactivité face à un environnement dynamique. Le processeur ne peut être utilisé à 100 % sauf configuration "remarquable" du jeu de tâches. En outre, la priorité est basée sur la proximité de l'échéance de la tâche (urgence) et non sur l'importance des traitements qu'elle doit effectuer (DM, RM). Lorsqu'on utilise RM ou DM dans un système temps réel, il faut donc faire un compromis entre ces deux critères, mais également considérer des critères plus spécifiques à l'application tels que l'accès privilégié à certaines ressources pour certaines tâches du système.

POSIX 1003.b regroupe les extensions temps réel du standard ISO/ANSI POSIX définissant une interface portable de système d'exploitation [Gal95]. Un ordonnanceur respectant cette norme comprend une file d'attente par niveau de priorité. La file d'attente est une zone mémoire permettant de stocker, dans ce cas, les tâches jusqu'à ce qu'elle puisse être exécutée. Les tâches de même priorité sont placées dans la file suivant une politique particulière. La tâche qui se trouve en tête de la file de plus haute priorité est élue. Les différentes politiques que l'on trouve dans POSIX 1003.b sont :

- SCHED_FIFO : la tâche quitte la tête de la file si elle a terminé son exécution, si elle est bloquée (accès à une ressource en section critique, attente d'un délai,...), ou si elle est explicitement libérée. Dans les deux derniers cas, la tâche est à nouveau placée dans la file.

- SCHED_RR (Round Robin) : son fonctionnement est quasiment identique à celui de SCHED_FIFO. La seule différence est que la tâche en cours d'exécution quitte la tête de la file lorsqu'une certaine quantité de temps, appelée quantum, est écoulée.

- SCHED_OTHERS : le fonctionnement de cette politique est non normalisé.

Tâches	C_i	s_i	Priorité	Politique
T_1	1	7	1	FIFO
T_2	5	0	4	Round Robin
T_3	3	0	4	Round Robin
T_4	6	4	2	FIFO

TAB. 2.1 – Tâches avec ordonnanceur de type POSIX 1003.b

Le tableau 2.1 présente un exemple de système dont l'ordonnanceur est de type POSIX 1003.b (cet exemple provient du cours d'introduction au temps réel de Mr Frank Singhoff).

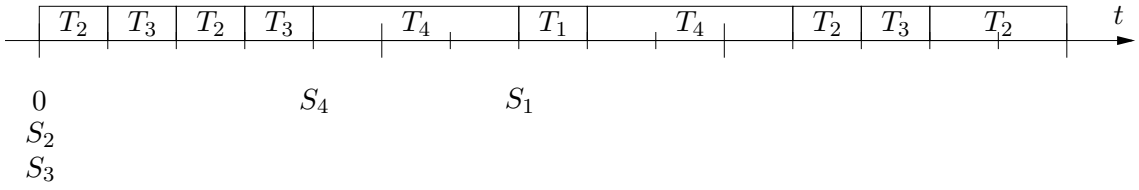

FIG. 2.7 – Exemple d'ordonnancement POSIX.4

On suppose ici que le niveau de priorité le plus fort est le niveau 1. Le quantum de temps pour la politique Round Robin est de une unité de temps. Nous obtenons l'ordonnancement de la figure 2.7).

2.1.2.2 Algorithmes d'ordonnancement à priorités dynamiques

Les algorithmes d'ordonnancement à priorités dynamiques sont en général utilisés pour les systèmes temps réel mou où une échéance manquée n'entraîne pas de dommage grave (applications multimédias, ...). Les algorithmes les plus connus sont les suivants : Earliest Deadline First (EDF) et Least Laxity First (LLF) [SSNB95, AB90].

- L'algorithme **EDF** attribue la priorité la plus forte à la tâche dont l'échéance est la plus proche.

- L'algorithme **LLF** attribue la plus grande priorité à la tâche de plus faible laxité. La laxité d'une tâche est la différence entre le temps restant jusqu'à la prochaine

échéance et la durée d'exécution restante de la tâche. (Ainsi, si elle est égale à 0, la tâche doit être exécutée sans interruption jusqu'à la fin de sa capacité pour ne pas rater son échéance).

Considérons un jeu de deux tâches T_1 et T_2 ordonnancées par EDF (exemple tirés du livre [DB99]). Les tâches sont décrites par leur capacité, $C_1 = 6$ et $C_2 = 9$, et leur période, $P_1 = 10$ et $P_2 = 30$. Les délais critiques sont égaux aux périodes.

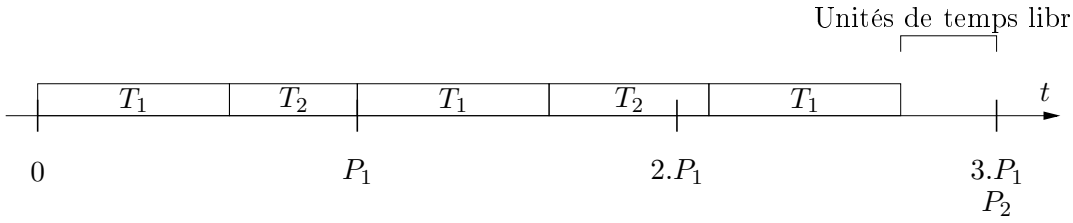

FIG. 2.8 – Ordonnancement EDF préemptif

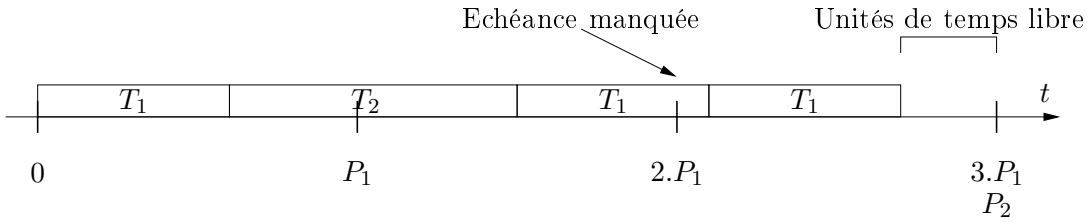

FIG. 2.9 – Ordonnancement EDF non-préemptif

Les figures 2.8 et 2.9 donnent l'ordonnancement de ce jeu de tâches dans le cas préemptif et non préemptif. Nous remarquons qu'à l'instant $t = 20$, les priorités des deux tâches sont identiques. Cet exemple illustre l'instabilité en surcharge de EDF. Le concepteur du système doit alors déterminer quelle tâche sera exécutée. Nous avons choisi de privilégier la tâche déjà en cours d'exécution, soit la tâche T_2.

EDF est optimal si les tâches ne sont pas liées par des contraintes de précédence et ne partagent pas de ressources (tâches indépendantes).

Contrairement aux algorithmes d'ordonnancement à priorités fixes, EDF et LLF ont un comportement moins déterministe en surcharge. En outre, ces algorithmes sont plus difficiles à implémenter que les algorithmes à priorités fixes.

2.1.3 Exécutif temps réel

Les systèmes temps réel sont très souvent construits autour d'un logiciel de base particulier, l'exécutif temps réel (ou noyau temps réel ou RTOS[5]). Cette partie du système

[5]Real Time Operating System

se charge d'allouer le temps processeur à chacune des tâches. Cette allocation doit alors prendre en compte les contraintes de temps et les besoins en ressources matérielles ou logicielles (mémoire, canaux de communication ...) de ces tâches.

La souplesse des mécanismes apportée par un exécutif temps réel entraîne des problèmes supplémentaires : la difficulté de maîtriser le déroulement de l'exécution de l'application, les blocages de tâches par d'autres plus prioritaires (la préemption) ou du fait de ressources non disponibles. Il est alors difficile de s'assurer que les tâches terminent leurs traitements en temps voulu, qu'il n'y a pas de situation **d'étreinte fatale** ou **deadlock** (deux tâches attendant chacune que l'autre ait terminé), ou de **famine** (une tâche bloquée indéfiniment par une ressource non disponible).

2.1.4 Conclusion

Un système temps réel est composé de traitements (ou tâches) exécutés en concurrence sur un ou plusieurs processeurs. Ces tâches sont modélisées par un ensemble de paramètres dont les valeurs sont issues des spécifications de l'application. Un algorithme d'ordonnancement élit la tâche qui accède au processeur à un instant donné.

Les tests de faisabilité permettent de vérifier que les tâches terminent leur exécution dans les temps. Nous présentons maintenant les principaux tests concernant les architectures mono-processeurs, multi-processeurs et réparties.

Il existe bien sûr beaucoup d'autres méthodes, telles que les réseaux de Petri, la programmation linéaire ou les langages rationnels, permettant d'étudier une application temps réel [CKS02, GL01, LM00, Rit97, CEKT02, GS96, RD01]. La modélisation d'un système temps réel grâce aux réseaux de Petri permet de générer l'ensemble des ordonnancements de tâches d'un système. Cette approche est particulièrement utile lorsque les problèmes rencontrés pour vérifier la faisabilité sont NP-durs. Des travaux proposent une analyse hors ligne de systèmes multi-processeurs [GCG01] ou de systèmes de tâches périodiques, liées par des contraintes de précédence et partageant des ressources [GCGC98b, GCG02]. Ces approches formelles sont intéressantes, mais sortent du cadre de notre travail. Elles ne seront donc pas détaillées ici.

2.2 Faisabilité dans les systèmes mono-processeur

Dans cette partie nous donnons les principaux tests de faisabilité d'applications temps réel pour des systèmes comprenant un processeur.

Nous disposons, pour ce genre de systèmes, de tests de faisabilité globalement acceptés par la communauté dans le cadre d'applications simples ne comprenant que des tâches indépendantes et périodiques. Cependant, compte tenu de la complexité des applications réelles, les tâches ne sont généralement pas indépendantes. Ceci est notamment dû à la présence de ressources partagées et de contraintes de précédence.

Nous donnons maintenant les résultats classiques concernant la faisabilité de systèmes temps réels mono-processeurs. Nous présentons ensuite quelques extensions de ces résultats. Enfin, nous concluons.

2.2.1 Tests de faisabilité des systèmes mono-processeur

Nous présentons principalement trois types de tests : la simulation sur la période d'étude, le test sur la charge processeur et le calcul du temps de réponse.

- La **période d'étude** représente une séquence infiniment répétée de l'ordonnancement. Lorsque l'ordonnancement est valide sur cette période d'étude du système, alors les contraintes temporelles des tâches seront respectées durant toute la vie du système [LM80].

- Un deuxième procédé, moins sensible au passage à l'échelle, consiste à s'assurer que la **charge processeur** U des tâches est inférieure ou égale à une borne donnée. La charge processeur d'une tâche i est le pourcentage du temps processeur nécessaire à l'exécution de i, soit $u_i = \frac{C_i}{P_i}$.

- Finalement, on peut vérifier pour chaque tâche que le **temps de réponse** relatif au début d'activation est effectivement inférieur au délai critique. Autrement dit, une tâche i respecte ses contraintes temporelles si $r_i \leq D_i$. Le temps de réponse r_i d'une tâche i représente le délai entre la date de première activation de la tâche (s_i) et sa terminaison.

Dans cette section, nous nous concentrons principalement sur les tests de faisabilité des systèmes composés de tâches périodiques indépendantes et d'un ordonnanceur non oisif. Il existe dans la littérature de nombreuses extensions des résultats présentés par la suite [ABR+93, Lar96, Riv98, Leb98, CDKM00].

2.2.1.1 Test sur la période d'étude

Pour les systèmes considérés, on vérifie que l'ordonnancement du jeu de tâches est bien valide sur la période d'étude suivante :

$$[0; max(\forall i : s_i) + 2 * PPCM(\forall i : P_i)] \tag{2.1}$$

Où $PPCM(\forall i : P_i)$ est la fonction qui calcule le plus petit commun multiple des périodes de toutes les tâches. La simulation est rapidement difficile à exploiter pour les systèmes comprenant un nombre important de tâches avec des périodes qui impliquent un $PPCM$ élevé. En outre, il faut que le comportement du système temps réel soit complètement déterministe.

2.2.1.2 Test sur la charge processeur

Lorsque le test sur la période d'étude ne peut être appliqué, on peut vérifier la charge processeur des tâches. Certains des tests présentés sont assez proches du calcul du temps de réponse que nous verrons ensuite. Néanmoins, pour plus de clarté, nous les classons parmi les tests sur la charge processeur.

Pour un jeu de n tâches, dont les délais critiques sont égaux aux périodes, ordonnancé par un algorithme RM préemptif, Liu et Layland ont démontré que la charge processeur doit suivre la règle suivante [LL73] (cf. annexe B) :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n.(2^{\frac{1}{n}} - 1) \tag{2.2}$$

Où $\frac{C_i}{P_i}$ est la fraction du temps processeur consommée par la tâche i durant la vie du système. Cette condition est suffisante mais non nécessaire.

Pour l'algorithme RM non-préemptif, deux tests, basés sur les travaux concernant les systèmes où les tâches partagent des ressources, ont été proposés [SRL90] :

$$\sum_{j=1}^{i} \frac{C_j}{P_j} + \frac{B_i}{P_i} \leq i.(2^{\frac{1}{i}} - 1) \quad \forall i, 1 \leq i \leq n \tag{2.3}$$

Et

$$\sum_{i=1}^{n} \frac{C_i}{P_i} + \max_{1<i\leq n} \left(\frac{B_i}{P_i}\right) \leq n.(2^{\frac{1}{n}} - 1) \tag{2.4}$$

Où B_i représente la durée maximale de blocage de la tâche i par une tâche de priorité inférieure.

Nous remarquons que ces deux tests considèrent le scénario pire cas où une tâche de priorité inférieure bloque une tâche de priorité supérieure durant toute sa capacité. Ils peuvent donc entraîner une réservation excessive de la ressource processeur.

Pour un jeu de n tâches, dont les délais critiques sont arbitraires, ordonnancé par un algorithme DM préemptif, nous avons à disposition deux tests sur le taux d'utilisation du processeur. Une première condition suffisante est donnée dans [LL73] :

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n.(2^{\frac{1}{n}} - 1) \tag{2.5}$$

Le second test est également une condition suffisante. Pour qu'une tâche i respecte ses contraintes temporelles, il faut que la somme de sa capacité C_i et des interférences des

tâches de plus forte priorité sur l'intervalle de temps où i est exécutée (soit $[O, D_i]$) soit inférieure ou égale au délai critique D_i. Autrement dit, il faut que[6] :

$$C_i + \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{P_j} \right\rceil .C_j \leq D_i \tag{2.6}$$

Avec $\forall i, 1 \leq i \leq n$.

En ce qui concerne l'algorithme DM non préemptif, il ne semble pas exister de test processeur.

Lorsque les priorités d'un jeu de n tâches sont déterminées grâce à EDF ou LLF préemptif, une condition nécessaire et suffisante est que le taux processeur soit inférieur ou égal à :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Lorsque $D_i = P_i$ (tâches à échéance sur requête).

A la différence de RM ou DM, nous constatons que l'utilisation du processeur peut atteindre 100%.

Pour des délais critiques arbitraires, une condition suffisante mais non nécessaire de la faisabilité est [LM80] :

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$$

Lorsque $D_i \leq P_i$.

Etant donné que l'utilisation du processeur ne peut dépasser 100%, la condition suivante reste nécessaire :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Jeffay et al. ont prouvé une condition équivalente pour EDF non préemptif, lorsque pour tout i, $D_i = P_i$ [JSP91]. Les auteurs précisent que ce test n'est pas valable lorsque les tâches sont ordonnancées par LLF non préemptif. La condition proposée est nécessaire, or, LLF n'est pas optimal parmi les algorithmes à priorités dynamiques [Mok83]. Les temps de réponse obtenus par LLF peuvent donc être supérieurs à ceux obtenus par EDF.

[6] L'opérateur $\lceil x \rceil$ appliqué à une valeur décimale renvoie l'entier directement supérieur à x

2.2.1.3 Test sur le temps de réponse

Le test sur le temps de réponse est une autre approche permettant de vérifier pour chaque tâche le respect de ses contraintes temporelles.

Le temps de réponse d'une tâche i est généralement constitué des instants où elle requiert le processeur afin d'effectuer son traitement périodique (pendant C_i unités de temps) ainsi que des instants où elle ne peut être exécutée car d'autres tâches de priorité plus importante sont présentes et actives dans le système.

Nous présentons uniquement les méthodes de calcul du temps de réponse lorsque les priorités des tâches sont statiques. On peut trouver dans [KRP+94], un algorithme de calcul de temps de réponse pour des tâches ordonnancées par EDF.

Joseph et Pandia, puis Audsley et al. ont proposé une équation permettant de calculer le temps de réponse de tâches à priorités statiques dont les délais critiques sont inférieurs ou égaux aux périodes [JP86, ABR+93] :

$$r_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil .C_j \tag{2.7}$$

Où hp(i) est l'ensemble des tâches de plus haute priorité que i. Le temps de réponse r_i est obtenu grâce à une méthode itérative qui consiste à calculer les différents éléments de la suite récursive :

$$\begin{cases} w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i^n}{P_j} \right\rceil .C_j \\ w_i^0 = C_i \end{cases} \tag{2.8}$$

L'arrêt du calcul intervient lorsque :

$$w_i^{n+1} = w_i^n$$

Le temps de réponse est alors égal à cette dernière valeur. Notons que si $w_i^n > P_i$, l'échéance de la tâche i n'est pas respectée.

Soit l'exemple constitué d'un jeu de trois tâches T_1, T_2 et T_3 ordonnancées par RM (exemple tiré du livre [BW97]). Les tâches sont décrites par les capacités $C_1 = 3$, $C_2 = 2$ et $C_3 = 5$, et les périodes $P_1 = 7$, $P_2 = 12$ et $P_3 = 20$. Les délais critiques sont égaux aux périodes.

Grâce à la méthode itérative, nous calculons le temps de réponse de la tâche T_1, puis T_2 et finalement T_3.

Nous commençons par calculer le temps de réponse de T_1. w_1 est initialisé avec la valeur de la capacité C_1. Soit $w_1^0 = 3$. Il n'y a pas de tâche de plus forte priorité que T_1, la valeur de la suite récursive ne va plus évoluer. Le temps de réponse de la tâche T_1 est donc de 3 unités de temps.

Nous procédons de la même manière pour T_2. T_2 peut être préemptée par la tâche de plus forte priorité T_1.

1. $w_2^0 = 2$.
2. $w_2^1 = 2 + \lceil \frac{2}{7} \rceil * 3 = 5$.
3. $w_2^2 = 2 + \lceil \frac{5}{7} \rceil * 3 = 5$.

La valeur de la suite itérative converge, le temps de réponse de la tâche T_2 est donc de 5 unités de temps.

La tâche T_3 subit les interférences de T_1 et T_2. Si nous procédons de la même manière que pour les tâches précédentes, nous obtenons les étapes suivantes :

1. $w_3^0 = 5$.
2. $w_3^1 \lceil \frac{5}{7} \rceil * 3 + \lceil \frac{5}{12} \rceil * 2 = 10$.
3. $w_3^2 \lceil \frac{10}{7} \rceil * 3 + \lceil \frac{10}{12} \rceil * 2 = 13$.
4. $w_3^3 \lceil \frac{13}{7} \rceil * 3 + \lceil \frac{13}{12} \rceil * 2 = 15$.
5. $w_3^4 \lceil \frac{15}{7} \rceil * 3 + \lceil \frac{15}{12} \rceil * 2 = 18$.
6. $w_3^5 \lceil \frac{18}{7} \rceil * 3 + \lceil \frac{18}{12} \rceil * 2 = 18$.

La valeur de la suite itérative converge, le temps de réponse de la tâche T_3 est de 18 unités de temps.

Les temps de réponse des tâches T_1, T_2 et T_3 sont inférieurs à leur période respective. Elles respectent donc leurs échéances.

Quand les délais critiques sont arbitraires, le calcul du temps de réponse est plus complexe. Comme les tâches de plus forte priorité, les activations précédentes peuvent interférer avec l'activation courante. En effet, pour chaque tâche, il faut tenir compte du fait qu'une activation n n'est pas forcément terminée alors que l'activation $n+1$ est prête à démarrer (cf. figure 2.10). Par exemple, lorsque $D_i > P_i$, l'exécution d'une instance d'une tâche peut ne pas être terminée alors que l'instance suivante est activée et donc prête à s'exécuter. En général, afin de simplifier le problème, on fait l'hypothèse que l'activation $n+1$ d'une tâche ne peut commencer avant la terminaison de l'activation n. Dans [Leh90], LEHOCZKY propose un calcul du temps de réponse qui tient compte de ce phénomène de rafale. Le temps de réponse est alors évalué de la façon suivante :

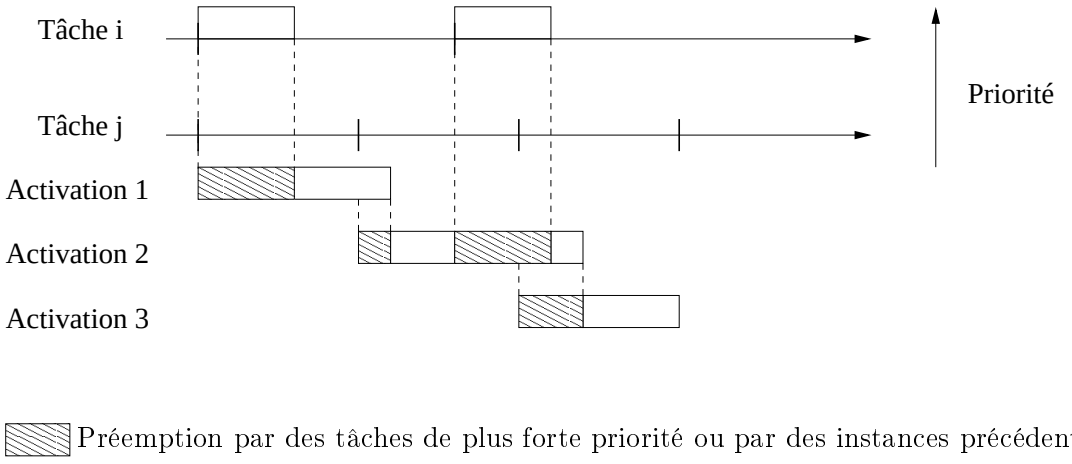

FIG. 2.10 – Interférences des activations précédentes

$$r_i = max_{q=0,1,2,...}(J_i + w_i(q) - q.P_i)$$

Avec

$$w_i(q) = (q+1).C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{J_j + w_i(q)}{P_j} \right\rceil * C_j$$

Tel que

$$\forall q : w_i(q) \geq (q+1).P_i$$

Dans cette partie, nous avons présenté les principaux tests de faisabilité applicables à des algorithmes préemptifs (ou non) à priorités fixes et dynamique en présence de tâches périodiques et indépendantes. Ces tests peuvent être étendus pour prendre en compte les dépendances entre tâches.

2.2.2 Prise en compte des dépendances entre tâches

Jusqu'à présent, nous nous sommes intéressés aux jeux de tâches indépendantes. Cependant, dans de nombreux systèmes temps réel, il existe des dépendances entre les tâches. Ces dépendances prennent la forme de précédence dans l'ordonnancement des tâches, ou encore d'accès conditionné à une zone mémoire partagée entre des tâches.

Les dépendances rendent plus difficile l'évaluation de la faisabilité : les tests concernant les jeux de tâches indépendantes ne peuvent plus être employés [SSNB95, RPGC02]. Néanmoins, des solutions ont été proposées pour RM et EDF pour vérifier le respect des contraintes temporelles de tels jeux de tâches.

2.2.2.1 Contraintes de précédence

Une contrainte de précédence entre les tâches d'un système temps réel définit un ordre partiel sur leur exécution : si les tâches i et j sont liées par une contrainte de précédence, alors $T_i \prec T_j$ signifie que l'exécution de chaque instance de la tâche j doit être précédée par l'exécution d'une instance de la tâche i [RCR01].

Un graphe peut être utilisé pour représenter une contrainte de précédence : les tâches sont les sommets du graphe et les contraintes de précédence les arcs [RCK00].

Nous trouvons deux classes de contraintes de précédence :

- Les contraintes de précédence **simples** : les tâches en relation de précédence ont la même période. Dans le cas contraire, la tâche de plus grande période finit par manquer une échéance.

- Les contraintes de précédence **généralisées** : il n'y a aucune restriction sur la période des tâches appartenant au même graphe de précédence. En revanche, aucune méthode n'a été proposée pour tester la faisabilité d'un jeu de tâches soumises à ce type de précédence [RCK00].

Des anomalies d'ordonnancement apparaissent lorsque des tâches du système sont soumises à des contraintes de précédence. Graham a démontré que lorsque un jeu de tâches est ordonnancé de manière optimale sur plusieurs processeurs avec des priorités assignées selon un algorithme donné, un nombre fixe de processeurs, des durées d'exécutions constantes et des contraintes de précédence, alors le fait d'augmenter le nombre de processeurs, de réduire les capacités ou d'assouplir les contraintes de précédence peut augmenter la durée de l'ordonnancement. Par conséquent, une tâche i peut ne plus respecter ses contraintes temporelles si l'on augmente le nombre de processeurs ou que l'on relâche des contraintes de précédence.

Nous exposons maintenant quelques solutions pour prendre en compte les contraintes de précédence dans les tests de faisabilité.

Chetto et Blazewicz ont proposé des méthodes basées sur la modification de certains paramètres des tâches afin de garantir le respect des dépendances pour des jeux de tâches ordonnancées par RM ou EDF []. Etant donné qu'aucun mécanisme particulier n'intervient, les tests de faisabilité classiques peuvent être utilisés. Pour les algorithmes d'ordonnancement HPF, les priorités des tâches sont modifiées en appliquant la relation suivante :

$$\forall i, j | i \prec j : p_i > p_j \tag{2.9}$$

La priorité des tâches précédentes doit être plus importante que celle des tâches suivantes.

Pour un jeu de tâches ordonnancées suivant EDF, les priorités des tâches sont modifiées par l'intermédiaire de leur délai critique.

$$D_i^* = \min(D_i, \min(\forall j | i \prec j : D_j^* - C_j)) \tag{2.10}$$

On attribue un nouveau délai critique à chaque tâche i en fonction des échéances des tâches suivantes. En effet, si l'on veut assurer la précédence entre une tâche i et celles qui la suivent, il faut que cette tâche soit plus prioritaire. Par conséquent, il faut que le délai critique de la tâche i soit inférieur aux délais critiques des tâches suivantes tout en leur laissant suffisamment de temps pour leur exécution.

Une troisième méthode a été introduite pour vérifier la faisabilité d'un jeu de tâches avec contraintes de précédence. Tindell propose un paramètre J_i (Jitter) afin de modéliser les contraintes de précédence [TC94]. J_i est égal au délai entre l'activation théorique d'une tâche et son activation réelle. Afin de modéliser la contrainte de précédence, on affecte à J_i le délai entre la date d'activation de la tâche i et la date où toutes les tâches précédentes à i ont terminé leur exécution. Tindell propose une extension de l'équation de calcul du temps de réponse [TC94] :

$$\begin{cases} r_i = w_i + J_i \\ w_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i + J_j}{P_j} \right\rceil . C_j \end{cases} \tag{2.11}$$

Comme pour l'équation du calcul classique du temps de réponse 2.7, cette équation est résolue grâce à un calcul récursif. Le temps de réponse d'une tâche partageant une ressource est un pire cas éventuellement plus grand que le temps de réponse réel.

D'autres solutions existent. Il est par exemple possible de réaliser ces contraintes en modifiant les conditions initiales (S_i). Des heuristiques d'ordonnancement ont également été proposées par Xu et Parnas [XP90].

2.2.2.2 Ressources partagées

Dans les systèmes temps réel, des zones mémoires sont utilisées pour l'échange de données entre les tâches. On parle alors de ressources partagées. Les opérations de lecture et écriture de la ressource doivent être atomiques. Il peut être dangereux pour l'intégrité des données d'autoriser, par exemple, deux opérations d'écriture à la fois. On associe donc à la ressource un sémaphore. Ce sémaphore protège son accès, on peut le voir comme un jeton : une tâche qui désire accéder à la ressource prend le jeton. Tant que cette tâche possède le jeton, elle est la seule à pouvoir lire ou écrire. Une fois ces opérations terminées, elle libère le jeton. Une tâche désirant effectuer une lecture ou une écriture doit éventuellement attendre la libération de la ressource : elle est **bloquée** sur la ressource. La zone comprise entre la prise et la libération de la ressource est appelée **section critique**.

D'une part, la vérification du respect des contraintes temporelles de tâches périodiques utilisant des sémaphores est NP-Dure [AB90, AD90, SSNB95]. D'autre part, les sémaphores peuvent dans certains cas générer une anomalie d'ordonnancement : l'inversion de priorité. L'inversion de priorité se produit quand une tâche, possédant une ressource, bloque une tâche de plus forte priorité en attente d'accès à cette ressource.

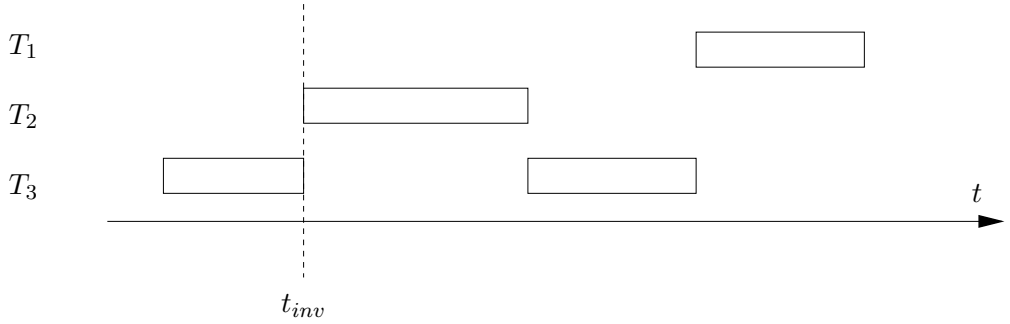

FIG. 2.11 – Exemple d'inversion de priorité

Sur la figure 2.11, nous avons un jeu de tâches T_1, T_2 et T_3 dont la priorité est inversement proportionnelle à leur indice. Les tâches T_1 et T_3 partagent une ressource. Nous observons à l'instant t_{inv} une inversion de priorité : alors que la tâche T_1 attend que la tâche T_3 libère la ressource pour reprendre son exécution, la tâche de priorité intermédiaire T_2 préempte T_3.

La vérification des contraintes temporelles est possible à condition que l'attente de la libération d'une ressource soit bornée et que l'on soit en mesure d'évaluer les temps de blocage.

Plusieurs protocoles de gestion d'accès à une ressource partagée ont été proposés tels que PIP, PCP, SRP... [SRL90, SSNB95]. Ces protocoles permettent de borner le temps de blocage noté B_i. Ils sont caractérisés par le nombre de ressources accessibles, leur complexité d'implantation, la possibilité d'interblocage (deux tâches attendent chacune que l'autre ait terminé son exécution),...

Nous présentons les deux protocoles les plus répandus :

- Le protocole **PIP**[7] : une tâche qui bloque une tâche plus prioritaire, est exécutée durant sa section critique en héritant de la priorité de la tâche bloquée. Etant donné que PIP n'évite pas les interblocages, il est préférable que les tâches n'accèdent qu'à une seule ressource. La borne maximum sur le temps de blocage B_i d'une tâche i est la somme des durées des accès en sections critiques des tâches moins prioritaires que i, soit :

[7] Priority Inheritance Protocol ou protocole à héritage simple

$$B_i = \sum_{j \in lp(i)} SC_j \tag{2.12}$$

Où $lp(i)$ représente l'ensemble des tâches moins prioritaires que i et SC_j la durée de la section critique d'une tâche j.

- Le protocole **PCP**[8] : le plus haut niveau de priorité de toutes les sections critiques acquises à un instant donné est stocké dans une variable "plafond". L'accès à la ressource est bloquant si la priorité de la tâche réclamant cet accès est inférieure ou égal au plafond. Grâce à ce protocole, une tâche peut accéder à plusieurs ressources sans risquer un interblocage. Le temps de blocage B_i est égal à la durée de la plus grande section critique des tâches partageant une ressource :

  $$B_i = max_{j \in rp(i)}(SC_j) \tag{2.13}$$

  Avec $rp(i)$, l'ensemble des tâches partageant la même ressource que i et SC_j, la durée de la section critique d'une tâche j.

Grâce à ces protocoles, il est alors possible de tester la faisabilité d'un jeu de tâches partageant des ressources. Par exemple, pour un jeu de n tâches ordonnancées selon l'algorithme à priorités fixes RM préemptif, le test sur le taux processeur devient :

$$\sum_{k=1}^{i-1} \frac{C_k}{P_k} + \frac{C_i + B_i}{P_i} \leq i.(2^{\frac{1}{i}} - 1) \quad \forall i : 1 \leq i \leq n. \tag{2.14}$$

Et le temps de réponse peut être obtenu par la méthode itérative suivante :

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i^n}{P_j} \right\rceil .C_j \tag{2.15}$$

Dans ce cas, le test sur le temps de réponse devient une condition suffisante mais non nécessaire.

D'autres solutions existent pour vérifier la faisabilité des tâches partageant des ressources. En général, ces solutions proposent des mécanismes pour éviter l'inversion de priorité ou les interblocages. On peut, par exemple, interdire la préemption d'une tâche en section critique ou l'accès d'une tâche à une ressource si l'on sait que cet accès va provoquer une inversion de priorité [AB90].

[8] Priority Ceiling Protocol ou protocole à héritage "plafond"

2.2.3 Conclusion

Nous avons présenté les tests de faisabilité classiques des systèmes composés d'un processeur. Trois types de tests sont disponibles afin de vérifier l'ordonnançabilité d'un jeu de tâches : le test sur la période d'étude, le test sur le taux d'utilisation processeur et le test du temps de réponse.

La présence de dépendance invalide la plupart de ces résultats. Malgré tout, il existe des solutions permettant de prendre en compte la présence de ressources partagées et/ou de contraintes de précédence dans un système temps réel.

L'approche classique de Liu et Layland dans le cadre de systèmes mono-processeur reste intéressante de par sa simplicité de mise en œuvre et de par la généralité de ces résultats. En dehors de ces cas simples, les résultats sont moins généraux, mais les approches proposées permettent de caractériser plus finement l'application concernée.

2.3 Faisabilité dans les systèmes multi-processeurs

Dans cette partie nous abordons la faisabilité dans les systèmes temps réel multi-processeurs.

Nous considérons qu'un système multi-processeurs est constitué de plusieurs processeurs et d'un mémoire partagée. Les tâches communiquent par l'intermédiaire de cette mémoire. Les temps de communication sont considérés comme négligeables. Généralement, la base de temps des processeurs est commune. Ceci facilite la gestion et l'étude de l'ordonnancement des tâches.

Les systèmes multi-processeurs peuvent être classés en plusieurs catégories.

- Un système multi-processeurs est **homogène** lorsqu'il est composé de processeurs dont la puissance est identique. Autrement dit, la durée d'exécution d'une tâche est la même sur tous les processeurs.

- Lorsque différents processeurs possèdent leur propre puissance d'exécution, on parle de système **hétérogène**. Dans [GBF02], deux types de systèmes hétérogènes sont distingués : les systèmes uniformes où la puissance de calcul varie linéairement selon les tâches et les systèmes non-linéaires où chaque couple tâche/processeur possède une vitesse d'exécution particulière.

Nous verrons que peu de résultats concernant les systèmes hétérogènes existent même si les systèmes uniformes possèdent certains avantages [GBF02]. Il est évidemment possible d'utiliser des processeurs de différentes puissances et de telles plates-formes se trouvent déjà dans le commerce. Pour améliorer l'efficacité de tels systèmes, il suffit de remplacer uniquement certains processeurs ou d'ajouter un ou plusieurs processeurs plus puissants.

Pour un système homogène, il aurait fallu remplacer tous les processeurs ou ajouter des processeurs identiques.

Nous présentons, dans un premier temps, les impacts, puis les problèmes liés à la présence de plusieurs processeurs sur les algorithmes d'ordonnancement. Nous décrivons ensuite certaines solutions proposées dans la littérature. Finalement, nous concluons.

2.3.1 Algorithmes d'ordonnancement pour systèmes multi-processeurs

Dans un système multi-processeurs, l'ordonnanceur doit non seulement élire la tâche qui doit être exécutée mais également choisir le processeur qui doit accueillir cette tâche.

Dans le cas multi-processeurs, les algorithmes d'ordonnancement sont classés selon le type de priorité des tâches : statique, dynamique mais statique pour les instances des tâches et complètement dynamique, mais également selon le degré de migration des tâches [CFH+03] :

- Pour une première catégorie d'algorithmes, les tâches et leurs instances (dans le cadre de tâches périodiques ou sporadiques) appartiennent à un processeur : elles ne migrent pas d'un processeur à l'autre. Ce sont les algorithmes d'ordonnancement par partitionnement.

- La deuxième catégorie décrit les systèmes où la migration des tâches est autorisée mais pas leurs instances.

- Dans la dernière catégorie, il n'y a aucune restriction. Non seulement les tâches peuvent migrer sur un autre processeur, mais leurs instances également.

Les deux dernières catégories regroupent les algorithmes d'ordonnancement globaux. Dans ces algorithmes, les tâches actives sont stockées dans une file unique.

On peut penser de manière intuitive que les jeux de tâches correctement ordonnancés grâce à un algorithme d'ordonnancement appliquant un partitionnement aux tâches, le seront également par un second moins restrictif au niveau de la migration. Or, il est prouvé que ces algorithmes ne sont pas nécessairement comparables.

Définition 10 (Comparaison des algorithmes d'ordonnancement) *A et B, deux algorithmes d'ordonnancement, sont incomparables lorsqu'il existe des jeux de tâches pouvant être ordonnancés par A et non par B et vice versa.*

Il existe donc des jeux de tâches qui ne sont faisables que pour un certain niveau de migration.

Par exemple, dans [LW82], les auteurs démontrent que les approches globales et par partitionnement d'ordonnancement de jeux de tâches à priorité statique pour des systèmes homogènes sont incomparables. Dans [BF03], les auteurs proposent une preuve équivalente pour les systèmes uniformes ordonnancés par EDF.

Une synthèse sur la comparaison en terme de faisabilité d'algorithmes d'ordonnancement pour les systèmes multi-processeurs classés selon le type de priorités des tâches et le degré de migration est proposée dans [CFH+03].

Nous rappelons uniquement dans ce chapitre les résultats concernant les algorithmes non oisifs. La définition donnée dans la partie traitant des résultats mono-processeurs doit être légèrement modifiée pour prendre en compte les spécificités du cas multi-processeurs.

Définition 11 (Algorithme d'ordonnancement non oisif) *Lorsque l'algorithme d'ordonnancement est non oisif, non seulement à chaque instant t, les processeurs doivent exécuter une instance de tâche active, s'il y en a dans le système, mais dans le cas où il y aurait moins d'instances actives que de processeurs, les instances les plus prioritaires sont exécutées sur les processeurs les plus puissants [GBF02].*

2.3.2 Problèmes spécifiques du cas multi-processeurs

L'ordonnancement de tâches périodiques à priorité statique ou dynamique utilisant une heuristique d'assignation de tâche (ordonnancement partitionné) sur un système multi-processeurs est un problème NP-dur [LW82]. En effet, assigner des tâches périodiques à des processeurs est équivalent aux problèmes de "bin packing" [BF03]. Le problème d'assignation des tâches à un processeur revient à se poser la question suivante : si nous disposons d'une collection de n objets de taille $p_1, p_2, ..., p_n$, est-ce que ces objets peuvent être placés dans des contenants de taille $b_1, b_2, ..., b_n$.

Nous détaillons trois types de problèmes : les problèmes de complexité des algorithmes d'ordonnancement, les problèmes de stabilité et de performance du système et les anomalies d'ordonnancement.

Il n'existe pas d'algorithme d'ordonnancement de tâches en ligne optimal pour les systèmes multi-processeurs. D'après Mok , dans certains cas particuliers, avec une totale connaissance des échéances, des temps d'exécution et des dates de début d'exécution des tâches, on peut envisager un tel algorithme [Mok83].

En ce qui concerne les systèmes où l'ordonnancement est non-préemptif, la plupart des problèmes sont, en général, NP-complets [SSNB95]. Le recours à des heuristiques est donc souvent rencontré.

La migration d'une tâche implique le transfert du contexte d'exécution associé à cette tâche, ce qui entraîne une surcharge au niveau des processeurs. Traditionnellement, la migration des tâches a été interdite dans les systèmes temps réel [CFH+03]. Généralement, le

coût engendré par le transfert du contexte d'exécution peut être important et jusqu'à récemment, il existait peu de résultats théoriques pour l'analyse des systèmes où la migration des tâches est permise.

Les résultats concernant les systèmes dont l'algorithme d'ordonnancement est global considèrent souvent que les migrations n'impliquent aucun coût. D'après [GBF02], il est possible de borner le nombre maximum de ces migrations et d'inclure la charge produite dans les durées d'exécution des tâches.

Mais, la migration des tâches pose d'autres problèmes [AB90]. Le système peut devenir instable. En effet, si l'ordonnancement est effectué en ligne, la tâche peut continuer à migrer indéfiniment. En outre, si trop de tâches migrent dans le système, celui-ci ne gère alors que ces migrations. Finalement, si les tâches communiquent entre elles, le re-routage des messages des tâches qui ont migré pose un problème supplémentaire.

Les systèmes multi-processeurs présentent des cas d'anomalie d'ordonnancement. L'anomalie de Graham en est un exemple [Gra01]. Cette anomalie n'est pas spécifique au cas multi-processeurs. Nous l'avons déjà expliquée dans la section 2.2.2.1. Toutefois, il nous semble important de montrer que cette anomalie peut être due à l'interaction de tâches se trouvant sur des processeurs différents.

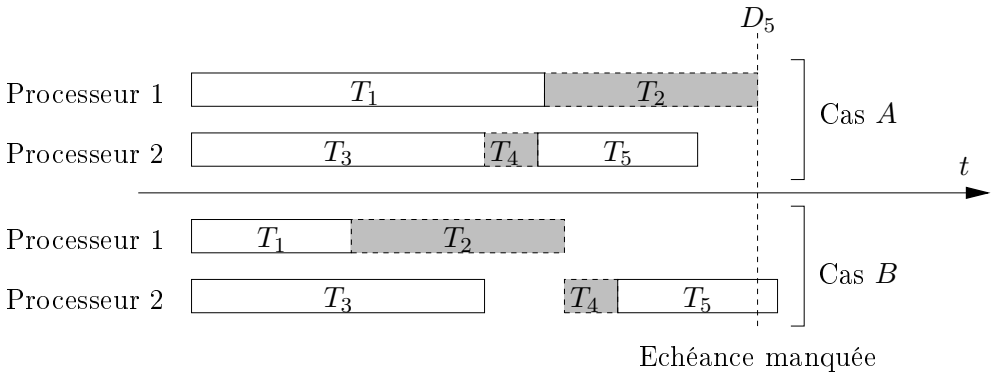

Fig. 2.12 – L'anomalie de Graham

Dans l'exemple de la figure 2.12, le système est composé des tâches T_1, T_2, T_3, T_4 et T_5 dont les priorités sont inversement proportionnelles à leur indice (T_1 étant la plus prioritaire). Les tâches T_1 et T_2 appartiennent au processeur 1, tandis que les tâches T_3, T_4 et T_5 appartiennent au processeur 2. Les tâches T_2 et T_4 partagent une ressource. Cette ressource est accédée par une tâche à la fois durant toute sa capacité (section critique). Les tâches T_4 et T_5 sont liées par une contrainte de précédence : l'exécution de T_5 ne débute que lorsque T_4 est terminée. Dans le premier ordonnancement, les échéances des tâches sont respectées (cas A). On réduit la capacité de la tâche T_1 (cas B). Nous observons sur la figure 2.12 que, dans ce cas, la tâche T_4 ne peut s'exécuter sur le processeur 2 tant que la ressource

n'est pas libérée par T_2. Le retard entraîné par cet accès a pour conséquence le non respect de l'échéance de T_5.

2.3.3 Quelques solutions pour étudier la faisabilité des systèmes multi-processeurs

Dans le cadre des systèmes multi-processeurs, il faut non seulement être capable de garantir le respect des contraintes temporelles des tâches mais également essayer de minimiser le nombre de processeurs requis. Nous présentons maintenant différentes solutions à ce problème.

Nous commençons par les résultats proposés dans le cadre d'algorithmes d'ordonnancement globaux. Puis, nous abordons le cas des algorithmes d'ordonnancement des systèmes où la migration des tâches est interdite. Nous détaillons notamment quelques heuristiques de placement proposées dans la littérature. Finalement, nous abordons d'autres approches pour étudier les systèmes multi-processeurs.

2.3.3.1 Tests de faisabilité des algorithmes d'ordonnancement globaux

Lorsque l'ordonnancement des tâches est global, la plupart des tests de faisabilité proposés dans la littérature sont des bornes maximums sur le taux d'utilisation processeur. De telles bornes existent pour des systèmes multi-processeurs uniformes dont les tâches sont périodiques et ordonnancées par Rate Monotonic [GBF02] ou EDF [FGB01]. Ces bornes sont également disponibles pour les systèmes multi-processeurs homogènes dont les tâches périodiques sont ordonnancées par Rate Monotonic [ABJ01].

D'après Dhall [DL78], lorsque l'on emploie un algorithme d'ordonnancement tel que RM ou EDF, de manière globale, il en résulte une utilisation arbitrairement basse du processeur. En outre, un algorithme d'ordonnancement global considère que le transfert du contexte d'exécution de la tâche est négligeable en temps. Si ce n'est pas le cas, on préfère alors un algorithme d'ordonnancement partitionné [BF03]. La principale difficulté de ces algorithmes est de placer les tâches sur le processeur tout en minimisant leur nombre.

2.3.3.2 Heuristiques de placement

Nous avons vu que les algorithmes d'ordonnancement ne sont pas optimaux pour les systèmes où la migration est interdite. Par conséquent, beaucoup d'heuristiques de placement de tâches ont été proposées dans la littérature. Un algorithme de placement attribue à chaque tâche un processeur hôte de manière à ce que les contraintes temporelles des tâches soient respectées. En général, les tâches sont assignées de manière permanente sur chaque processeur. De telles heuristiques sont jugées sur le nombre de processeurs supplémentaires nécessaires par rapport au nombre de processeurs optimal.

Etant donnée la complexité de cette problématique, la plupart des heuristiques de la littérature concerne des systèmes homogènes exécutant des jeux de tâches indépendantes à

priorité fixe. En effet, minimiser le nombre de processeurs requis afin que les contraintes temporelles d'un jeu de tâches soient respectées est un problème NP-complet [Loc86],

Dans la plupart des cas, l'algorithme d'ordonnancement choisi est RM [OS93]. En effet, RM est optimal parmi les algorithmes d'ordonnancement concernant les jeux de tâches à priorité fixe. En outre, beaucoup de résultats existent pour cet algorithme d'ordonnancement (cf. section 2.2). De la même manière, si la priorité des tâches est dynamique, EDF sera préféré car il est optimal parmi tous les algorithmes d'ordonnancement.

Le test sur la charge processeur associé à RM est suffisant mais non nécessaire (cf. équation 2.2). Utiliser une telle condition peut entraîner une sur-réservation de la ressource processeur et par conséquent augmenter le nombre de processeurs nécessaires [Mok83, LL73]. Or, l'objectif des heuristiques de placement est de diminuer ce nombre. Des travaux ont donc été menés afin d'améliorer ce test [DL78].

[DL78] est la première étude de l'ordonnancement de jeux de tâches périodiques à priorité fixe sur un système multi-processeurs utilisant une heuristique d'assignation de tâches. Les auteurs proposent les heuristiques d'ordonnancement Rate-Monotonic-First-Fit (RM-FF) et Rate-Monotonic-Next-Fit (RM-NF). La plupart des algorithmes proposés dans la littérature sont basés sur le fonctionnement de ces deux heuristiques.

Ces heuristiques fonctionnent selon le schéma suivant : tout d'abord, les tâches sont éventuellement groupées puis ordonnées selon un paramètre donné (période, charge processeur, ...). Ensuite, elles sont assignées au processeur courant jusqu'à ce que la charge processeur maximum théorique soit dépassée. Cette charge dépend du ou des algorithmes d'ordonnancement du système. Si une tâche ne peut être placée sur le processeur courant, un autre processeur est choisi. On termine lorsque toutes les tâches sont attribuées aux processeurs.

Lorsque ces heuristiques fonctionnent en ligne, le jeu de tâches peut évoluer (ajout ou suppression de tâches). Dans ce cas, la complexité de l'heuristique doit être suffisamment faible et le nombre de processeurs requis risque d'être plus important que pour une heuristique hors-ligne.

Chacune de ces étapes peut être améliorée afin de minimiser le nombre de processeurs nécessaires. On peut travailler sur de meilleures heuristiques pour ordonner ou grouper les tâches. Il est également intéressant d'obtenir des conditions d'ordonnançabilité plus précises afin d'augmenter l'utilisation des processeurs.

RM-NF fonctionne de la manière suivante :

1. Les tâches sont ordonnées dans l'ordre croissant de leur période.

2. On assigne une tâche i à un processeur j si la condition de faisabilité est respectée. Dans le cas contraire, on assigne la tâche au processeur $j+1$. Finalement on passe à la tâche suivante ($i = i+1$) et on réitère cette étape.

3. On arrête l'algorithme lorsqu'il n'y a plus de tâche à ordonnancer, j représente alors le nombre de processeurs nécessaires pour exécuter le jeu de tâches.

Le fonctionnement de RM-FF est similaire à celui de RM-NF mais la deuxième étape est légèrement différente. Lorsque l'on passe à la tâche suivante on revient au premier processeur (j est réinitialisé à 1 avant de réitérer la deuxième étape).

De nombreuses heuristiques basées sur Rate Monotonic ont été proposées dans la littérature.

Dans [LMM98], une heuristique de placement est décrite. Son fonctionnement est le suivant : un ensemble de tâches S est modifié de manière à obtenir un ensemble S' harmonique. Ensuite, les tâches sont placées de manière virtuelle en utilisant RM-FF et la condition de faisabilité R-BOUND proposée par les auteurs. Si S' est ordonnançable, S l'est aussi. Dans le cas contraire, on ne peut rien dire sur l'ordonnançabilité de S.

Dans [OS95], une heuristique de placement appelée RM-FFDU est proposée. Elle concerne les jeux de tâches périodiques et indépendantes appartenant à un système homogène. Cet algorithme permet d'obtenir un rapport nombre de processeurs nécessaires sur nombre de processeurs optimal égal à $\frac{5}{3}$ dans le pire cas.

Les heuristiques de placement RM-ST, RM-GT et RM-GT/M sont proposées dans [BLOS94]. Pour chacun de ces trois algorithmes une condition de faisabilité est établie.

RM-ST est un algorithme hors-ligne qui s'applique de préférence sur les ensembles de tâches dont la charge processeur U est inférieure à $0,5$.

Dans le cas contraire ($U > 0,5$), on utilise un autre algorithme hors-ligne : RMGT. Il consiste à appliquer RM-ST à un premier groupe constitué des tâches dont la charge est inférieure ou égale à $\frac{1}{3}$, puis, à appliquer RM-FF aux autres tâches.

L'algorithme en-ligne RM-GT/M permet d'ajouter ou de supprimer dynamiquement des tâches dans le système. Si l'on veut ajouter une tâche, on procède de la même manière que pour un placement de tâche hors-ligne : si la condition de faisabilité d'ordonnancement est respectée, alors la tâche est attribuée au processeur courant. Dans le cas contraire, un processeur vide est choisi. Pour supprimer une tâche, les auteurs proposent une solution pour les deux cas de figure suivants : si la tâche concernée appartient au processeur courant, elle est abandonnée. Sinon, toutes les tâches du processeur concerné sont déplacées et le processeur est marqué vide.

Dans [OS93], une heuristique également basée sur RM est proposée : Rate-Monotonic-Best-Fit (RM-BF). Les performances de RM-BF sont proches de celles de RM-FF [OS93]. Le fonctionnement est similaire à celui de RM-FF mais tous les processeurs sont inspectés dans un ordre spécifique.

Enfin, dans [BF03], une heuristique permet d'assigner des tâches dont la priorité est dynamique : EDF-FFD (First Fit Decreasing). Cette proposition est la suite du travail mené par Funk dans le cadre des systèmes multi-processeurs uniformes [FGB01].

Certaines heuristiques d'ordonnancement, employées lorsque la migration des tâches est interdite, ont une complexité importante. Ainsi, si l'ordonnancement est effectué en ligne, les heuristiques RM-FF et RM-BF leur sont préférées [CFH+03].

2.3.3.3 Autres approches

D'autres approches existent pour analyser le comportement temporel d'un système multi-processeurs. Par exemple, dans [RD01], les auteurs proposent une modélisation de l'exécution de tâches sur un système multi-processeurs sous la forme de réseau de PETRI T-temporels : le RDPTO. A partir du RDPTO, le calcul du graphe des classes d'états permet de mettre en évidence le respect ou non des contraintes temporelles des tâches. D'autre part, le problème du placement des tâches sur un système multi-processeurs peut également être étudié grâce à l'utilisation de la théorie des graphes [BD98].

2.3.4 Conclusion

Nous avons abordé la faisabilité des systèmes temps réel multi-processeurs. Les plates-formes peuvent être homogènes (processeurs identiques) ou hétérogènes (processeurs différents).

Le délai de communication entre les tâches situées sur des processeurs différents est considéré comme nul et les tâches peuvent éventuellement migrer d'un processeur à l'autre.

Nous retrouvons des anomalies d'ordonnancement dans les systèmes multi-processeurs telle que l'anomalie de Graham.

L'ensemble des tests de faisabilité est essentiellement composé de bornes sur le taux d'utilisation processeur et d'heuristiques que l'on peut classer selon le degré de migration des tâches.

En particulier, lorsque la migration n'est pas autorisée, il n'existe pas d'algorithme d'ordonnancement optimal. Des heuristiques de placement des tâches sont proposées. Ces heuristiques assignent les tâches aux processeurs de manière à ce que leur contrainte temporelle soit respectée.

2.4 Faisabilité dans les systèmes répartis

Dans cette partie, nous abordons la faisabilité des systèmes temps réel répartis.

Nous considérons qu'un système réparti est constitué de plusieurs processeurs reliés par un ou plusieurs bus. Les tâches communiquent par l'intermédiaire des bus. Contrairement au cas multi-processeurs, les temps de communication ne sont plus négligeables.

D'après [Wil96], il n'est pas envisageable d'obtenir les mêmes garanties en terme de déterminisme dans le cas réparti que dans le cas centralisé, les réseaux n'étant fondamentalement pas complètement déterministes.

D'autre part, l'auteur estime que si l'on peut aujourd'hui considérer que la conception d'applications temps réel est convenablement maîtrisée dans le cas centralisé, ce n'est pas vrai dans le cas réparti.

Les problèmes spécifiques aux systèmes répartis sont les suivants [AD90, RCR01] :

- Nous avons vu que la migration des tâches pose un certain nombre de problèmes : le coût de transfert du contexte d'exécution associé à cette tâche, le re-routage des messages et la migration perpétuelle de tâches. Les approches présentées dans le cas multi-processeurs considèrent souvent que ces migrations n'impliquent aucun coût. Ce n'est plus vrai dans les système répartis, principalement du fait de la présence d'un réseau.

- Les précédences entre tâches disposées sur des sites/processeurs distants impliquent de vérifier le respect des contraintes temporelles des "traitement répartis". Pour cela, il est nécessaire de déterminer le temps mis par un message pour atteindre la tâche réceptrice. Ce délai dépend, entre autres, de la manière dont les messages transitent sur le bus.

- Les tâches peuvent accéder à des ressources partagées se trouvant sur un autre processeur.

Nous supposons que dans les systèmes étudiés, les tâches ne migrent pas d'un site à l'autre. En outre, nous considérons qu'aucun message n'est perdu lors de la communication (ce problème est étudié dans [WP00]).

Dans un premier temps, nous définissons le temps de communication des messages sur un réseau. Ensuite, nous décrivons quelques éléments de compréhension concernant les réseaux. Puis, nous abordons les traitements répartis. Enfin nous concluons.

2.4.1 Délai de communication des messages sur le réseau

Comme nous l'avons vu précédemment, la connaissance et l'analyse du réseau constituent une part importante de la résolution du problème de validation d'un système temps réel réparti.

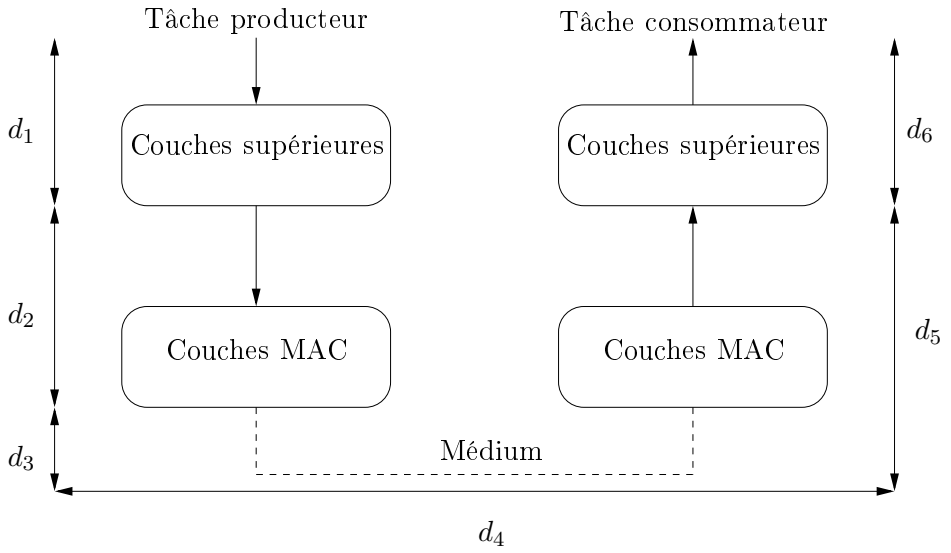

FIG. 2.13 – Décomposition du délai de communication

Nous observons sur la figure 2.13 que le délai de communication entre deux tâches se trouvant sur des sites différents peut être décomposé de la manière suivante [CDKM00] :

1. Temps de traversée des couches logicielles (d_1 et d_2) : un message émis par une tâche traverse les couches supérieures de l'application.

2. Temps d'accès au médium et temps de transmission (d_3) : le message accède au réseau.

3. Temps de propagation sur le réseau (d_4) : le message parcours le médium. Le médium peut être, par exemple, une paire torsadée ou de la fibre optique.

4. Temps de traversée des couches logicielles (d_5 et d_6) : le message franchit les couches supérieures de l'application distante et finalement est délivré à la tâche réceptrice.

Le délai de communication comprend donc un délai de traversée des couches logicielles, un temps de propagation sur le réseau, un temps de transmission et enfin, un temps d'accès au médium.

Afin d'évaluer le délai de communication, il est nécessaire de connaître la valeur des différents éléments qui le composent :

- Le temps de traversée des couches logicielles est généralement simple à déterminer. Il est parfois donné par l'éditeur du système d'exploitation utilisé.

- Le délai de propagation sur le réseau est proportionnel à la distance entre émetteur et récepteur.

- Le temps de transmission constitue le délai nécessaire à un site pour écrire les données sur le réseau. Ce délai dépend, entre autres, du type de réseau choisi (CAN, Arinc 629, ...).

- Le temps d'accès au médium est le paramètre le plus difficile à obtenir. C'est la durée que doit attendre un message avant de pouvoir utiliser le médium. Le temps d'accès au bus dépend du protocole d'arbitrage du réseau.

Si nous considérons le réseau comme un processeur, les messages/données comme des tâches et le protocole d'accès comme un algorithme d'ordonnancement non préemptif (car le transfert d'une tâche ne peut être interrompu), une borne sur le temps d'accès peut être déterminée en utilisant les équations de temps de réponse mono-processeur. Evidemment cette borne existe si l'ordonnancement des messages sur le réseau est déterministe.

Finalement, pour obtenir le délai de communication entre deux tâches, il suffit d'additionner l'ensemble des délais précédemment décrits.

Des études du comportement temporel ont été proposées pour la majorité de bus de terrain. Ces études se basent sur l'application de techniques basées sur l'approche de TINDELL que nous allons décrire dans la partie suivante [TBW95].

2.4.2 Bus de communication et protocole d'arbitrage

Les bus de communication sont utilisés pour l'échange de données entre les différentes stations du système réparti. Une station est un système mono ou multi-processeurs. En général, les échanges de données entre stations "esclaves" sont contrôlés par une station "maître". Un protocole d'arbitrage est associé au bus. Il gère la manière dont les messages accèdent au réseau.

Dans un premier temps, nous décrivons quelques protocoles d'arbitrage. Ensuite, nous détaillons des bus de communication avioniques et industriels qui utilisent ces protocoles.

Le protocole **TDMA** utilise un contrôleur de bus pour synchroniser l'émission de messages pour chaque site. Ce contrôleur émet périodiquement une trame de synchronisation sur le réseau. Chaque station connaît l'instant d'émission de ces messages vis-à-vis de cette trame de synchronisation. Le contrôleur peut être fixe ou dynamique (élection en cas de panne).

Les protocoles **CSMA** (CD, CA et autres) se basent sur l'écoute du réseau. Lorsqu'il est libre, la transmission débute. Dans le cas où plusieurs messages sont émis simultanément, la résolution des collisions s'effectue différemment selon le type de CSMA utilisé. Pour

CSMA/CD, la transmission du message est stoppée et le message est réémis après un intervalle de temps aléatoire. Pour CSMA/CA, la station de plus forte priorité continue l'envoi de son message et les autres stoppent leur transmission. Lorsque le bus est libre, les stations transmettent un identificateur bit à bit puis écoutent la porteuse. Un bit de valeur 1 est masqué par un bit de valeur 0. Une station (coupleur) qui lit un bit différent de celui qui vient d'être transmis perd le droit d'émettre et passe en réception. La station ayant envoyé l'identificateur de plus haute priorité gagne le droit d'émettre ses données. Les autres stations tentent ensuite automatiquement un nouvel accès au bus (arbitrage non destructif).

La **scrutation** (ou **polling**) consiste à désigner un site maître qui contrôle l'envoi de messages des sites esclaves. Ce protocole reste très utilisé car il est simple et déterministe.

D'autres protocoles existent tels que **token ring** ou **token bus** [KU94].

Ces protocoles sont utilisés pour gérer l'accès au réseau. Nous présentons des réseaux dont le service de communication est déterministe. On les trouve dans les domaines tels que l'avionique, l'automobile, ...

Decker décrit les principales caractéristiques des bus avioniques [Dec96]. Hormis la communication, ils offrent des outils d'interopérabilité (avionique modulaire) et, éventuellement, la mise en place d'horloges communes. Les bus les plus connus sont le bus "Arinc 429", le bus "Arinc 629" et le bus "1553".

Le bus **ARINC 429** est le plus ancien et le plus utilisé des bus en avionique civile. Le bus est constitué d'un seul maître et d'au plus 20 esclaves. La communication est monodirectionnelle : un seul maître existe dans le système et seule cette station est autorisée à émettre des données sur le bus.

Le bus **ARINC 629** a été proposé par Boeing, puis, normalisé par l'ARINC en 1989. Un bus 629 connecte plusieurs stations qui peuvent être maîtres et/ou esclaves. On peut donc mettre en place des communications bidirectionnelles. Le protocole d'accès au bus utilisé est DATAC[9]. C'est un protocole de type TDMA. Il n'a pas de diffusion périodique d'une trame de synchronisation servant "d'horloge" aux autres stations. Ce qui bien évidemment est intéressant du point de vue de la tolérance aux pannes. Une des difficultés consiste à initialiser le bus : [BGJ98] propose une étude des collisions lors de l'introduction de stations sur un bus ARINC 629 CP vide.

Le Bus **1553** est utilisé depuis 1978 dans presque tous les systèmes militaires et une grande partie des systèmes avioniques civils. Le bus 1553 peut contenir plusieurs esclaves/maîtres. Il est donc bidirectionnel. Le protocole d'arbitrage est de type scrutation. Chaque ordre émis par la station maître est acquitté par la station destinataire (l'ordre contient l'adresse du destinataire). Le bus 1553 est donc un bus dont le fonctionnement est plutôt asynchrone.

[9] Digital Autonomous Terminal Access Control

Le bus **CAN** à été créé par Bosch et Intel pour les systèmes embarqués dans les automobiles [TBW95]. Les principales qualités de ce type de bus sont leur fiabilité et leur faible coût. Le bus CAN est un réseau multi-maîtres de type producteur/consommateur. Les stations sont toutes reliées au bus CAN (topologie en bus). La transmission des données se fait par diffusion. Le protocole d'accès au bus est de type CSMA/CA. Il procure notamment des services de sûreté de fonctionnement très évolués grâce à des fonctions matérielles telles que l'acquittement en réception ou la détection d'erreur de transmission. CAN est particulièrement adapté aux systèmes temps réel répartis de petite taille (longueur maximum de 40 m pour un débit de 1MBit/s) et dont les contraintes de fiabilité sont élevées.

Le bus **FIP** (ou **WorldFIP** dans sa version internationale) est un réseau orienté productique. La topologie d'un réseau de station basé sur FIP peut être en bus ou en étoile. La transmission des informations se fait par diffusion sur le médium (paire torsadée ou fibre optique). Le contrôle d'accès au bus est réalisé par une station maître. Les autres stations sont définies comme des producteurs ou des consommateurs de données. La station maître organise les échanges de données grâce à une table de scrutation (polling). FIP gère la diffusion de données périodiques ou apériodiques. Les contraintes temporelles du trafic périodique sont garanties. FIP offre donc un service de communication déterministe. La taille du réseau peut atteindre une longueur de 4000 m pour un débit de 1 MBit/s.

Finalement, le bus **Profibus** est développé par Bosch, Klöckner-Moller et Siemens. Plusieurs variantes à Profibus existent : FMS [10], PA[11] (systèmes en zone explosible) et DP[12]. En général, la topologie d'un réseau de stations basé sur Profibus est en bus. Le contrôle d'accès au réseau est multi-maître : l'élection de la station maître est réalisée grâce à un jeton (la station maître qui possède le jeton est active). La station garde le jeton pendant un laps de temps défini à la configuration. Ensuite, le protocole d'accès au bus est de type scrutation maître/esclave. Les données cycliques sont transmises puis les données acycliques. Comme pour les réseaux FIP, les contraintes temporelles des données périodiques sont garanties. La taille du réseau peut atteindre une longueur de 400 m pour un débit de 1,5 MBit/s (FMS). Ce type de bus est assez coûteux et complexe.

De nombreux réseaux temps réel existent tels que **TTP** [PEP99], l'**Ethernet commuté** [KS02] ou **FDDI** [LS95].

2.4.3 Faisabilité des systèmes temps réel répartis

Dans la plupart des travaux sur les systèmes répartis, la migration de tâches n'est pas autorisée. Certaines solutions abordées dans la partie multi-processeurs restent valables. En particulier, les heuristiques de placement peuvent être adaptées aux systèmes répartis. Nous discutons donc des problèmes spécifiques au cas réparti.

[10] Field Message Specification
[11] Process Automation
[12] Decentralized Periphery

La vérification des contraintes temporelles des tâches d'un système temps réel réparti peut être de type :

- Locale : on étudie les contraintes temporelles locales aux processeurs du système. Pour cela, les résultats décrits dans la partie mono-processeur sont utilisés.

- Globale : **on vérifie le respect des contraintes temporelles des traitements répartis**.

Un traitement réparti est une séquence de tâches exécutées sur des processeurs différents. Ces tâches communiquent par l'intermédiaire du bus. Nous supposons que les tâches sont activées lors de la réception d'un message.

Les contraintes temporelles de ces tâches sont vérifiées grâce à des tests basés sur le calcul du temps de réponse (cf. équation 2.7). Il ne s'agit plus de vérifier le respect d'une contrainte qui est locale à un processeur donné, mais de vérifier une contrainte qui met en relation plusieurs processeurs reliés par un réseau : on cherche à vérifier, par exemple, que les échéances des tâches du traitement réparti sont bien respectées (soit $\forall i,\ r_i \leq D_i$).

L'approche la plus fréquemment utilisée dans la littérature est le calcul **Holistique** [TC94]. Tindell propose d'utiliser le paramètre J_i pour modéliser les contraintes de précédence du traitement réparti : on affecte au paramètre J_i, i étant une tâche ou un message, le temps de réponse de la tâche ou du message précédent dans la chaîne de traitements. Le temps de réponse d'un message i est le délai entre la date d'envoi et la date de réception de ce message. Son application est simple et tient compte des dépendances cycliques entre les tâches situées sur de sites différents.

$$r_i = J_i + M_{ji} \tag{2.16}$$

Le temps de réponse r_i des tâches, resp. des messages, est évalué grâce à l'équation 2.11, resp. 2.16.

On considère que le délai de communication M_{ij} de la tâche j vers la tâche i est connu (cf. section 2.4.1).

Le calcul holistique fonctionne de la façon suivante :

1. Les r_i et les J_i des tâches et des messages sont initialisés à 0.

2. On calcule le temps de réponse des tâches et des messages grâce aux équations 2.11 et 2.16.

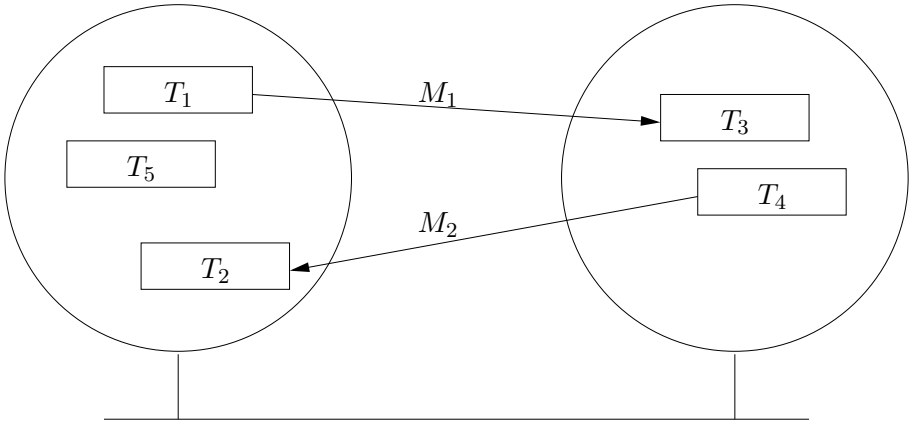

FIG. 2.14 – Exemple de traitements répartis

3. On affecte aux J_i le temps de réponse de la tâche ou du message précédant la tâche/message i.

4. On retourne à l'étape n^o 2 tant qu'au moins un des r_i évolue.

Soit l'exemple constitué des tâches T_1, T_2, T_3, T_4 et T_5 (cf. figure 2.14, cet exemple provient du cours d'introduction au temps réel de Mr Frank Singhoff). Les tâches T_1, T_2 et T_5 sont placées sur le processeur a et les tâches T_3 et T_4 placées sur le processeur b. Un premier traitement réparti est composé des tâches T_1 et T_3, et un deuxième des tâches T_2 et T_4. On remarque qu'une dépendance cyclique existe entre ces deux traitements. Les messages M_1 et M_2 sont émis périodiquement.

Messages	Période (en ms)	Délai de communication (en ms)
M_1	100	6
M_2	60	1

TAB. 2.2 – Délai de communication

On suppose que les délais de communication incluent les temps de propagation, d'accès, de transmission et de traversée des couches.

A la première étape de l'algorithme du calcul holistique, les J_i et r_i sont initialisés à 0.

Nous passons à la deuxième étape de l'algorithme. Nous calculons les temps de réponse des tâches et des messages. Les valeurs sont données dans le tableau 2.4. Nous remarquons

Tâche	Période	Capacité	Priorité	Processeur
T_1	100	4	1	a
T_2	60	5	2	a
T_3	100	3	2	b
T_4	60	2	1	b
T_5	90	3	3	a

TAB. 2.3 – Paramètres des tâches

Message/Tâche	M_1	M_2	T_1	T_2	T_3	T_4	T_5
r_i	6	1	4	9	5	2	12
J_i	0	0	0	0	0	0	0

TAB. 2.4 – Première itération du calcul holistique

que les temps de réponse r_1, r_4 et r_5 n'évolueront plus. En effet, J_1, J_4 et J_5 restent nuls car il n'y a pas de précédence.

Dans la troisième étape de l'algorithme, on affecte aux J_i la valeur du temps de réponse de la tâche ou du message précédent selon les relations suivantes : $J_{M_1} = r_{T_1}$, $J_{M_2} = r_{T_4}$, $J_{T_3} = r_{M_1}$ et $J_{T_2} = r_{M_2}$. Nous revenons à la deuxième étape et calculons les r_i en utilisant les nouvelles valeurs des J_i. Les résultats sont donnés dans le tableau 2.5. Les paramètres nécessaires au calcul des temps de réponse r_{M_1} et r_{M_2} sont fixés, r_{M_1} et r_{M_2} ne seront donc plus modifiés.

Message/Tâche	M_1	M_2	T_1	T_2	T_3	T_4	T_5
r_i	**10**	**3**	4	**10**	**11**	2	12
J_i	**4**	**2**	0	**1**	**6**	0	0

TAB. 2.5 – Deuxième itération du calcul holistique

On applique l'algorithme jusqu'à ce que les valeurs des J_i et des r_i ne soient plus modifiées. Le résultat final est donné dans le tableau 2.6.

Cette technique a fait l'objet de nombreuses extensions. En particulier, RICHARD propose une solution afin de prendre en compte les systèmes avec contrainte de précédence ou possédant des tâches apériodiques [RCR01]. D'autres extensions existent permettant d'analyser des systèmes comportant des tâches gérées selon une politique d'ordonnancement de type POSIX 1003.b (politique SCHED_FIFO ou SCHED_RR) [CP00]. Enfin, KIM propose une amélioration du calcul grâce à la prise en compte des temps de réponse dans le cas le plus favorable [KLSC00].

Message/Tâche	M_1	M_2	T_1	T_2	T_3	T_4	T_5
r_i	10	3	4	**12**	**15**	2	12
J_i	4	2	0	**3**	**10**	0	0

TAB. 2.6 – Dernière itération du calcul holistique

Le problème des ressources partagées distantes est abordé dans [Sun97]. A chaque ressource est associée une tâche qui permettra aux tâches locales ou distantes d'accéder à cette ressource. Ainsi, une tâche voulant accéder à une ressource partagée sera considérée comme la dernière tâche dans la chaîne d'un traitement réparti.

Le calcul Holistique offre une solution intéressante pour déterminer des délais de bout en bout. Toutefois, le temps de réponse obtenu est un pire cas. En outre, la vérification de la faisabilité dans un contexte réparti reste peu utilisée.

Il existe néanmoins d'autres approches pour vérifier la faisabilité d'un tel système réparti. Elles sont basées sur des heuristiques ou sur des extensions du modèle classique de tâche.

Dans [Sun97], le modèle de tâche considéré est le suivant : une tâche est un ensemble de sous-tâches. Chaque sous-tâche possède un processeur hôte, un temps d'exécution et une priorité. Plusieurs heuristiques sont décrites. Elles sont basées sur l'étude de la période pendant laquelle des tâches de priorité plus forte ou égale sont exécutées.

Les extensions du modèle classique peuvent également être utilisées dans un contexte réparti. Par exemple, dans [CMB00], une tâche peut être exécutée sur plusieurs nœuds hétérogènes (processeurs différents). Pour répondre à ce besoin, les auteurs proposent le modèle de tâche DGMF qui est une généralisation de l'approche Multiframe.

2.4.4 Conclusion

Les systèmes répartis étudiés sont constitués d'un ensemble de processeurs reliés par un bus. Le délai de communication entre les tâches situées sur des processeurs différents n'est plus considéré comme négligeable et seuls les systèmes n'autorisant pas la migration de tâches sont considérés.

La faisabilité de ces systèmes consiste à vérifier, d'une part, le respect des contraintes temporelles locales à chaque processeur. Les résultats classiques des systèmes mono-processeur sont alors utilisés. D'autre part, l'approche holistique permet de vérifier le respect des contraintes temporelles globales du système.

Toutefois, cette méthode implique souvent des résultats pire cas trop grands par rapport à la réalité : en pratique il se peut qu'un ordonnancement soit effectivement valide alors que les temps de réponse calculés sont plus grands que les échéances.

Malgré ses inconvénients, il s'agit de la méthode la plus générale qui ait été proposée à ce jour. Les autres approches apportent parfois de meilleurs résultats mais semblent trop spécifiques à des applications et/ou modèles de tâches donnés.

2.5 Théorie des files d'attente

Il existe de nombreux domaines dans lesquels la théorie des files d'attente permet de dimensionner les systèmes : les moyens de transport, les moyens de communication (analyse de réseau), ... D'une manière générale, la modélisation d'un problème par des files d'attente permet de comprendre la formation et la dissipation de suites de clients dans le système étudié.

Un client peut être une voiture arrivant à un carrefour, une personne dans une salle d'attente ou encore des données circulant dans un réseau.

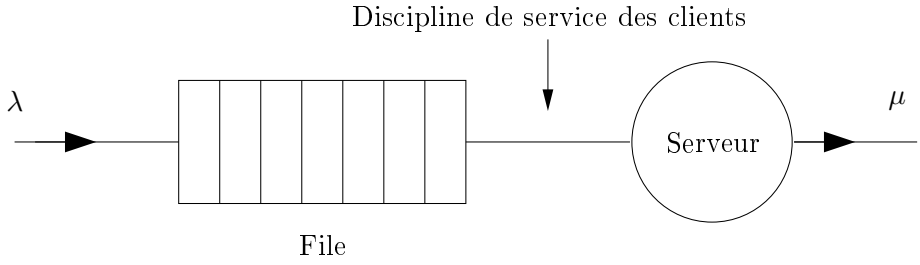

FIG. 2.15 – Description d'une file d'attente

Une file d'attente est composée d'une file et d'un serveur (cf. figure 2.15). Le fonctionnement d'une file d'attente est le suivant :

1. Les clients arrivent dans le système à une certaine fréquence.

2. Le serveur traite les requêtes des clients aussitôt qu'elles lui sont présentées. Si des clients arrivent alors que le serveur est occupé, ils restent en attente dans la file. Dès que le serveur a terminé son exécution, il s'occupe du client suivant.

   En général, les clients sont servis suivant un protocole FIFO[13]. D'autres disciplines de service peuvent être choisies telles que LIFO[14] ou des politiques basées sur la priorité des clients.

3. Si la taille de la file est fixée, différentes politiques peuvent être appliquées alors qu'un client arrive et que la file est pleine.

Au moins trois paramètres permettent de caractériser une file d'attente. Il s'agit :

- Du **taux d'arrivée moyen** des clients dans la file : λ.

[13] Le premier client arrivé dans la file est le premier servi
[14] le premier client arrivé dans la file est le dernier servi

- Du temps moyen passé par un client à être servi : $\frac{1}{\mu}$. Cette durée est aussi appelée **temps de service**.

  Définition 12 (Temps de service) *Le temps de service S_i d'un client est le temps passé par celui ci à être traité par le serveur [Kle75b]. S_i est le temps entre l'activation du serveur et sa terminaison. Le serveur est activé sur réception d'un client dans la file :*

  - *Lorsque la file d'attente est vide, le client est immédiatement pris en charge par le serveur. S_i est le délai séparant l'arrivée du client et la terminaison du service.*

  - *Dans les autres cas, le client attend dans la file la fin du service du client précédent. S_i est le délai séparant l'entrée du client dans le serveur, et la terminaison du service.*

- De **taux d'occupation** du serveur, ou **intensité du trafic** : ρ. ρ représente le pourcentage du temps pendant lequel le serveur est occupé. Sa valeur est obtenue grâce à la relation $\rho = \frac{\lambda}{\mu}$.

  Définition 13 (Loi de conservation du débit) *Pour garantir que le tampon ait une taille bornée, il faut que $\rho < 1$.*

  La loi de conservation du débit certifie que le nombre de consommations est plus important que le nombre de productions (cf. définition 13). Nous considérons que cette condition est toujours respectée. Dans le cas contraire, le nombre de clients présents dans la file d'attente croît vers l'infini.

Nous verrons par la suite que d'autres paramètres entrent en jeu tels que le nombre de serveurs, la taille de la file, ...

2.5.1 Notation de Kendall

Classiquement, la notation employée pour décrire une file d'attente est celle de Kendall ([Kle75b], [Rob90]). Une file d'attente est définie grâce à quatre paramètres : $A|B|C|D$.

- A représente le taux d'arrivée des clients (ou encore la distribution du temps entre deux arrivées consécutives).

- B décrit la distribution du temps nécessaire pour s'occuper d'un client (ou le taux de service).

- C est le nombre de serveurs.

- D donne la taille maximum de la file. Si aucune valeur n'est indiquée, on considère que la file possède une capacité infini.

Les distributions d'arrivée et de départ peuvent être de type D, M ou G.

- Une distribution D ou **déterministe** décrit un taux d'arrivée λ (ou le délai entre 2 arrivées $\frac{1}{\lambda}$) de clients constant ou un temps de service $\frac{1}{\mu}$ (ou taux de service μ) constant. La variance associée à ces délais est nulle.

- Une distribution **markovienne** ou M est décrite par un processus de Poisson de paramètre λ ou μ selon qu'elle représente le taux d'arrivée ou de service d'une file. La variance est, selon les cas, égale à $\frac{1}{\lambda}$ ou $\frac{1}{\mu}$. Notons qu'un processus de poisson permet de modéliser les phénomènes aléatoires.

  En ce qui concerne la distribution markovienne, la probabilité d'avoir n événements (arrivées ou services) de fréquence ε dans l'intervalle de temps t est de :

  $$P_n(t) = \frac{(\varepsilon t)^n}{n!} e^{-\varepsilon t}$$

  La probabilité qu'une durée T (intervalle entre deux arrivées ou temps de service) soit comprise entre $t1$ et $t2$ est de :

  $$P(t1 \leq T \leq t2) = e^{-\varepsilon t1} - e^{-\varepsilon t2}$$

  Avec $t1 \leq t2$ et ε, le taux moyen d'arrivée ($\varepsilon = \lambda$) ou de service ($\varepsilon = \mu$).

  En outre, cette distribution exhibe la **propriété markovienne** : les événements consécutifs sont sans mémoire, l'état futur dépend uniquement de l'état présent. Un système est dit Markovien, si ses distributions d'arrivée et de service sont markoviennes. Lorsqu'une seule des distributions est markovienne, le système est alors semi-markovien.

- Finalement, la distribution **générale** ou G, est décrite par un taux moyen (λ pour les arrivées ou μ) associé à une variance σ^2.

2.5.2 Files d'attente classiques

Nous présentons dans cette section les files d'attente classiques M/M/1, M/D/1, M/G/1 et D/D/1 [Kle75b, Rob90]. Ces quatre files d'attente ont une taille infinie et ne possèdent qu'un seul serveur.

La file **M/M/1** représente un système markovien. La fréquence d'arrivée et les temps de service sont markoviens. Les critères de performance de M/M/1 sont obtenus grâce à la résolution du processus de naissance et de mort (cf. figure 2.16).

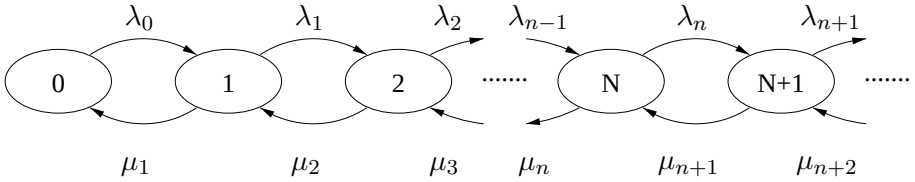

FIG. 2.16 – Processus de naissance et de mort

La file d'attente **M/G/1** est un système semi-markovien. Les arrivées sont décrites par un processus de Poisson. La loi de service est générale. L'obtention des critères de performance nécessite l'utilisation de chaînes de MARKOV dites immergées.

Contrairement à la file M/M/1, la file **M/D/1** possède un temps de service constant. Les critères de performance sont obtenus à partir de résultats de M/G/1 en appliquant une variance nulle.

La file **D/D/1** est complètement déterministe : le temps entre 2 arrivées et le temps de service sont constants. La durée entre 2 arrivées successives n'excède pas la durée de service. Dans le cas contraire, l'augmentation du nombre de clients dans la file serait permanente. Ainsi, le serveur est toujours prêt lorsqu'un client arrive dans le système. L'occupation maximum de la file est donc de 1 client. Les résultats, tels que le nombre maximum de clients dans la file ou le plus grand temps d'attente, peuvent être déduits graphiquement pour ce type de file d'attente.

2.5.3 Critères de performance classiques

Nous présentons maintenant différents critères de performance issus de la théorie des files d'attente pour les files M/M/1, M/D/1 et M/G/1. L'ensemble de ces critères est résumé sous la forme d'un tableau à la fin de cette section.

Ces critères sont obtenus lorsque la file a atteint son état d'équilibre (régime stationnaire). Lorsque les arrivées de messages sont markoviennes, il faut étudier la file sur une période de temps infinie pour obtenir cet état d'équilibre. Ce point est particulièrement gênant lorsque l'on effectue des simulations (cf. chapitre 6).

Les critères classiques de la théorie des files d'attente sont les suivants : W, L et P_n.

- W est le **délai moyen d'attente** d'un client dans le système. W est égal à W_q, le temps moyen passé par un client dans la file, plus W_s, le temps de service de ce client :

$$W = W_q + W_s \tag{2.17}$$

- L est le **nombre moyen** de clients dans une file d'attente L est égal à L_q, le nombre moyen de clients dans la file, plus L_s, le nombre moyen de clients dans le serveur :

$$L = L_q + L_s \tag{2.18}$$

  W et L sont liés par la loi de Little ([Kle75b]) :

  Théorème 1 (La loi de Little) *Le nombre moyen de clients dans la file, resp. système, est égal au nombre moyen de clients arrivant dans la file pendant le temps d'attente moyen d'un client dans la file, resp. système. Soit,*

$$L = \lambda W$$

  Et

$$L_q = \lambda W_q$$

- La probabilité qu'il y ait n clients dans le système est donnée par P_n. Il existe des cas particuliers intéressants tels que P_W, P_N et P_0 qui sont respectivement la probabilité qu'un client attende son service, la probabilité que la file d'attente soit pleine lorsqu'un client arrive et la probabilité d'avoir 0 client dans la file.

File	L	L_q	W	W_q	P_n
M/M/1	$\frac{\lambda W_s}{1-\rho}$	$\frac{\lambda^2 W_s^2}{1-\rho}$	$\frac{W_s}{1-\rho}$	$\frac{\lambda W_s^2}{1-\rho}$	$(1-\rho)\rho^n$
M/G/1	$\lambda W_s + \frac{\lambda^2(W_s^2+\sigma^2)}{2(1-\rho)}$	$\frac{\lambda^2(W_s^2+\sigma^2)}{2(1-\rho)}$	$W_s + \frac{\lambda(W_s^2+\sigma^2)}{2(1-\rho)}$	$\frac{\lambda(W_s^2+\sigma^2)}{2(1-\rho)}$	
M/D/1	$\lambda W_s + \frac{\lambda^2 W_s^2}{2(1-\rho)}$	$\frac{\lambda^2 W_s^2}{2(1-\rho)}$	$W_s + \frac{\lambda W_s^2}{2(1-\rho)}$	$\frac{\lambda W_s^2}{2(1-\rho)}$	

Tab. 2.7 – Critères de performance des files M/M/1, M/D/1 et M/G/1

Le tableau 2.7 résume les principaux critères de performance pour les files d'attente M/M/1, M/D/1 et M/G/1 ($W_s = \frac{1}{\mu}$ et $\rho = \lambda W_s = L_s$). Les résultats de M/D/1, resp. M/M/1, peuvent être retrouvés à partir des résultats de M/G/1 en appliquant une variance σ^2 nulle, resp. égale à $\frac{1}{\mu}$.

2.5.4 Files d'attente dans les systèmes temps réel

Dans cette section, nous présentons des travaux basés sur l'utilisation de la théorie des files d'attente dans les systèmes temps réel.

La présence de tampons dans les systèmes temps réel est commune. pourtant, à notre connaissance, peu de résultats ont été proposés dans le cadre de systèmes temps réel composés de tâches périodiques ordonnancées par un algorithme à priorités fixes.

Des travaux ont été menés sur des systèmes multi-processeurs où les communications entre tâche sont effectués par l'intermédiaire de tampon [TZ99]. Dans ces travaux, le tampon est considéré comme une ressource partagée proposant la dernière valeur d'une donnée. Dans le pire cas, l'occupation du tampon est la somme du nombre de consommateur et de producteurs plus un. Les systèmes que nous étudions sont différents. D'une part, l'ensemble des tâches producteurs et consommateurs sont situées sur un même processeur. D'autres part, toutes les données produites doivent être consommées et traitées par la tâche consommateur contrairement aux travaux précédent où seule la dernière valeur produite est considérée (éventuellement certaine valeur ne sont pas pris en compte).

Nous nous sommes donc intéressés aux travaux menés sur la théorie des files d'attente. Néanmoins, les résultats ne conviennent pas au système que nous étudions. Leur objectif est de vérifier des contraintes temporelles associées aux clients, or nous nous intéressons à la taille des zones de stockage de clients n'ayant pas d'échéance particulière.

La théorie des files d'attente propose des concepts qui peuvent sembler pouvoir résoudre notre problème :

- Le serveur périodique. Une file d'attente proposant un serveur périodique est en fait une file d'attente composée de plusieurs files servies cycliquement par un seul serveur [SLF92]. A part le comportement périodique du serveur, une telle file d'attente ne permet pas de prendre en compte dans la distribution du temps de service la variabilité du temps de réponse des tâches due à l'ordonnanceur temps réel.

- Les files d'attente à priorités[15]. Dans ces files, il est possible d'attribuer une priorité aux messages [AVS02, Sta92]. Par exemple, les clients des files HOL[16] ont une priorité fixe [Kle75a].

Chen propose des techniques de mesure du temps d'attente moyen de trafic temps réel [CD96]. Un trafic est dit temps réel lorsqu'il est soumis à des contraintes temporelles, autrement dit, les messages doivent respecter une échéance donnée. Ce travail est basé sur une file M/G/1 non-préemptive et non-oisive. Chaque trafic temps réel est un client, au sens file d'attente, dont la priorité est donnée par l'algorithme d'ordonnancement EDF.

[15] ou "priority queueing"
[16] Head Of the Line

LEHOCZKY a utilisé les files à priorités afin de vérifier les contraintes temporelles de tâches activées de manière aléatoire [Leh96]. Pour cela, il a développé la théorie des files d'attente temps réel[17]. Il propose un critère d'ordonnançabilité valable lorsque le taux d'activation moyen des tâches est élevé (heavy traffic).

D'autres travaux ont été menés dans la communauté temps réel. En particulier, beaucoup de lois de service ont été étudiées dans le monde des réseaux [ZF94]. Ces lois ont généralement pour but de fournir une meilleure bande passante ou de garantir des délais de bout en bout statistiques ou déterministes pour les données transmises. Mais la plupart de ces lois de service sont dépendantes de l'état du tampon et utilisent des mécanismes matériels pour obtenir les garanties voulues, mécanismes qui n'existent pas dans les systèmes que nous étudions.

2.5.5 Conclusion

La théorie des files d'attente permet de dimensionner les systèmes. Une file d'attente est définie par quatre paramètres : son taux moyen d'arrivée, son temps de service, le nombre de serveurs et éventuellement la capacité maximale de la file.

Nous décrivons quelques files d'attente classiques, telles que les files M/M/1 et M/G/1, ainsi que certains de leurs critères de performance. Il est, par exemple, possible de déterminer le temps moyen d'attente d'un client, le taux moyen d'occupation ou la probabilité que n clients se trouvent dans la file à un instant donné.

Nous présentons également des exemples de systèmes temps réel dimensionnés à l'aide de la théorie des files d'attente.

[17] ou "real time queueing theory"

2.6 Motivations de nos travaux

Cette thèse traite de la faisabilité des systèmes temps réel répartis.

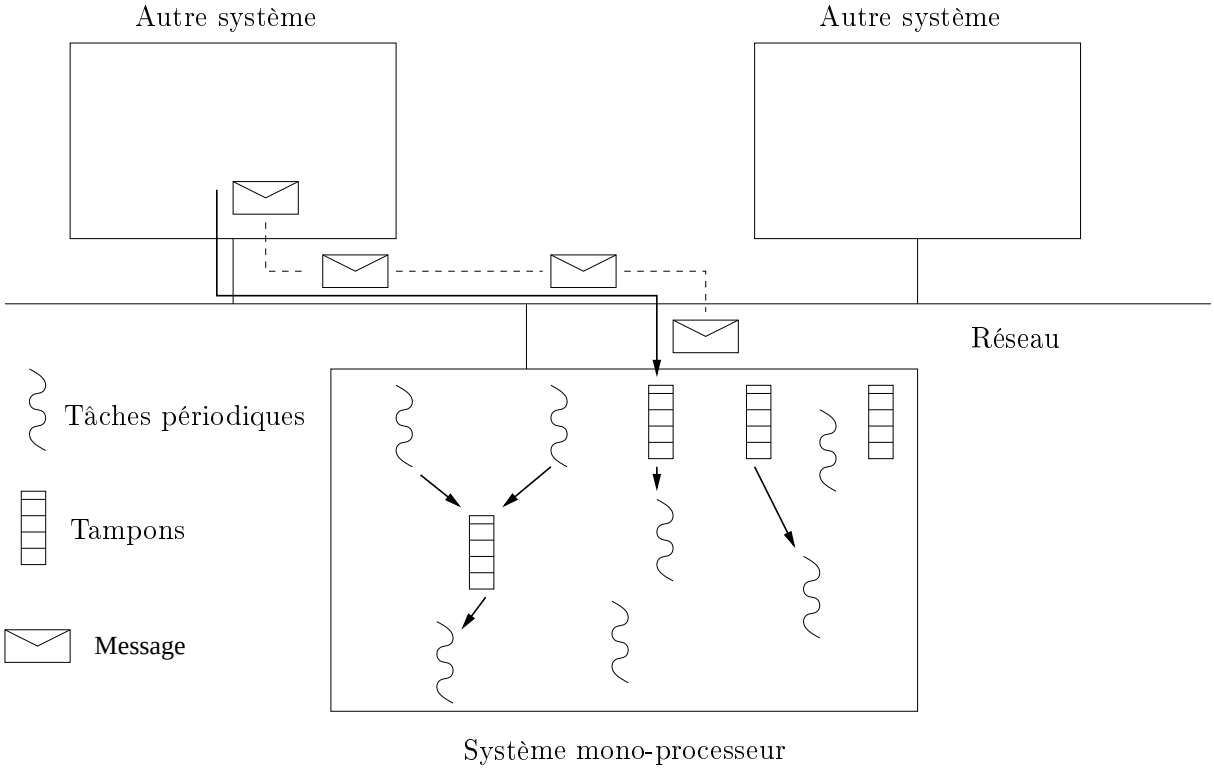

FIG. 2.17 – Proposition d'architecture répartie

Nous considérons qu'un système réparti est constitué de plusieurs sous-systèmes mono-processeur reliés par un réseau (cf. figure 2.17).

Un sous-système est composé de tâches et de **tampons**. Nous supposons que les tâches du système sont **périodiques** et ordonnancées, par exemple, selon un algorithme à priorités fixes. La migration des tâches entre les différents sous-systèmes est interdite. Les tampons collectent des informations délivrées par un autre sous-système au travers du réseau ou périodiquement lors d'une communication entre tâches de ce même sous-système.

Les messages arrivent dans le tampon à une cadence donnée. Dans certains systèmes, cette cadence peut être difficile à définir. C'est la cas des applications de supervision que l'on trouve dans les systèmes avioniques [Bur00]. Un système avionique est composé de nombreux équipements logiciels ou matériels. Par conséquent, la probabilité de panne d'un de ces équipements peut être non négligeable. Il est donc nécessaire d'avoir recours à des périodes de maintenance régulières. Afin de réduire les coûts de maintenance, un service

de supervision collecte, analyse et stocke les données relatives aux pannes. Finalement, les résultats sont présentés durant les périodes de maintenance.

Les instants d'occurrence des pannes peuvent être difficiles à déterminer précisément. Une cadence d'arrivée connue est caractérisée par un délai inter-arrivée minimum. Dans le cas contraire, nous considérons que les messages arrivent de manière aléatoire avec un certain taux d'arrivée moyen.

Il est intéressant d'étudier ce modèle d'application car il correspond à une pratique industrielle que l'on retrouve, par exemple, dans l'avionique modulaire : le système est constitué de sous-systèmes communiquant par l'intermédiaire d'un réseau [Ari97]. Ces sous-systèmes sont éventuellement fournis par différents partenaires industriels. Le fonctionnement global du système est défini par un intégrateur de système.

La présence de tampons apporte une certaine souplesse dans le fonctionnement d'une application temps réel dont les tâches communiquent. Les tâches sont activées à leur rythme mais il est nécessaire que la taille des tampons soit suffisante pour qu'il n'y ait pas de situation de débordement. La modularité induite par les tampons facilite le travail de l'intégrateur de système.

Notre objectif est de vérifier, d'une part, que les tâches respectent leurs contraintes temporelles et, d'autre part, que la taille des tampons est suffisante pour qu'il n'y ait pas de débordement. Nous ne nous soucions pas des contraintes temporelles des messages qui transitent sur le réseau ou dans les tampons. Pour répondre à ce problème, nous utilisons les résultats classiques et éprouvés de l'ordonnancement temps réel et de la théorie des files d'attente.

Afin de remplir ces objectifs, nous considérons que les tâches ne sont pas liées par des contraintes de précédence. Les activations des tâches ne dépendent donc, ni des arrivées de messages dans un tampon, ni de la terminaison d'une autre tâche.

L'étude du respect des contraintes temporelles locales à un sous-système est donc relativement simple. Nous utilisons pour cela les tests du cas mono-processeur pour des jeux de tâches périodiques et indépendantes (cf. section 2.2).

En revanche, la difficulté est reportée sur le dimensionnement des tampons. L'absence de précédence entre l'arrivée des messages et l'activation de la tâche consommateur ne correspond pas au fonctionnement classique d'une file d'attente. Comme nous le verrons par la suite, nous ne pouvons pas assimiler complètement le serveur d'une file d'attente et une tâche qui consomme les messages dans un tampon.

Les critères que propose la théorie des files d'attente sont uniquement des critères moyens. Par conséquent, si le respect des contraintes temporelles des tâches est important pour

les applications ciblées, il peut être intéressant de privilégier cet aspect pour obtenir des garanties sur la faisabilité de l'ordonnancement.

Dans la section 2.5.4, nous avons vu que les travaux sur les tampons dans les systèmes temps réel ne correspondent pas à ce que nous cherchons. Nos résultats concernant l'analyse de performance des tampons dans un système temps réel sont détaillés dans les chapitres 3, 4, 5 et 6. Le cas des tampons possédant n flux d'arrivées de message et une tâche consommateur est étudié dans le chapitre 4. Dans le chapitre 5, les résultats concernent les tampons partagés par n tâches producteurs et une tâche consommateur. Nous proposons pour le premier, resp. le deuxième, type de tampon des critères moyens, resp. maximums, de performance.

Chapitre 3

Analyse de performance de tampons partagés par des tâches périodiques

Dans le chapitre 2, nous avons présenté les résultats classiques des systèmes temps réels mono-processeurs, multi-processeurs et répartis. Ces systèmes sont constitués d'un ensemble de traitements (ou tâches) soumis à des contraintes temporelles. Beaucoup de travaux ont été menés afin de vérifier qu'un ensemble de tâches est effectivement ordonnançable compte tenu des spécifications de l'application. Nous introduisons dans ce chapitre nos propositions concernant l'analyse de performances des tampons d'un système temps réel.

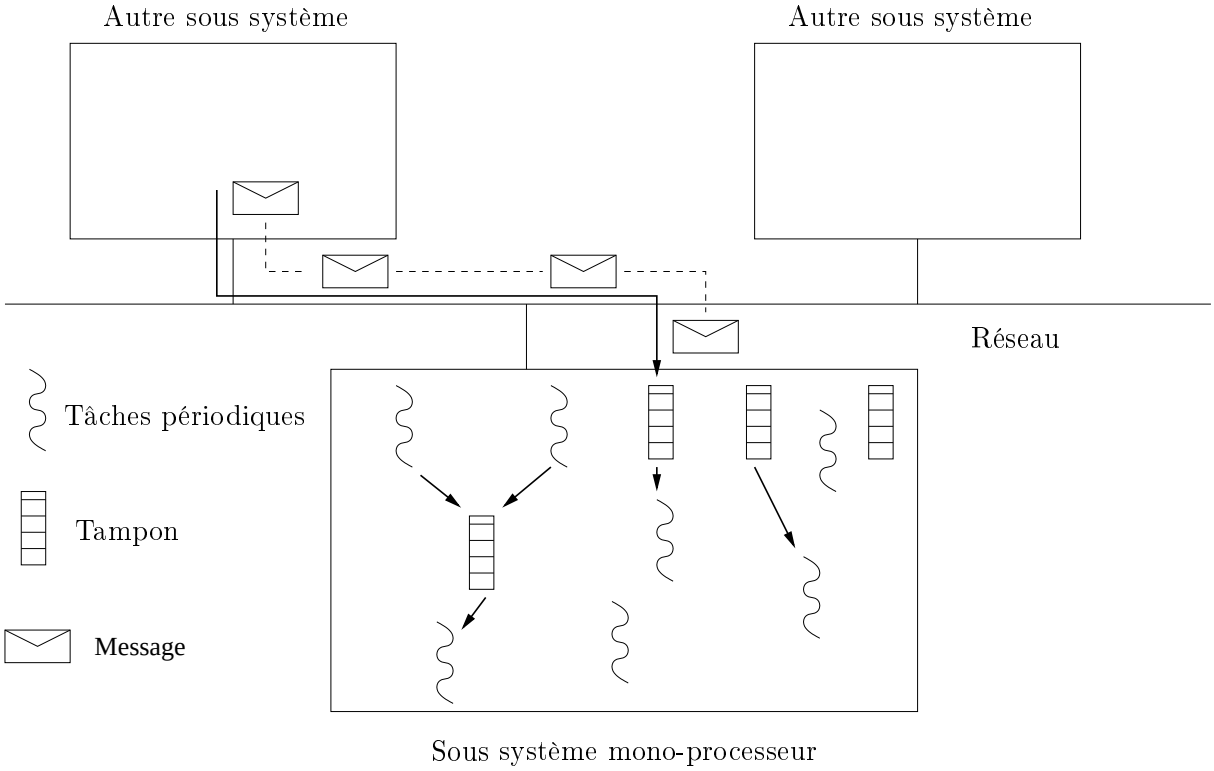

FIG. 3.1 – Architecture répartie étudiée

Nous étudions des systèmes temps réel constitués de composants mono-processeur connectés entre eux par l'intermédiaire d'un réseau (cf. figure 3.1). Chaque processeur héberge une application formée d'un ensemble de tâches. En particulier, nous vérifier la faisabilité des sous-systèmes mono-processeur. Nous ne supposons pas d'algorithmes d'ordonnancement particulier.

Des tampons sont utilisés pour qu'un ensemble de tâches, situées sur le même processeur ou non, envoient des données à un d'autres tâches d'un sous-systèmes mono-processeur. La tâche qui envoie, resp. réceptionne, un message est appelée "tâche producteur", resp. "tâche consommateur".

L'accès au tampon n'est pas soumis à un protocole particulier comme PIP ou PCP. L'utilisation d'un tel protocole aurait pour conséquence de modifier éventuellement l'ordre de production ou de consommation des messages. Or, cet ordre n'a pas d'importance pour nous : nous considérons toujours le pire cas où la production de message est prioritaire sur la consommation.

Nous considérons qu'**il n'y a pas de précédence entre l'arrivée des messages dans le tampon et l'activation de la tâche consommateur**.

Dans la suite de la thèse, nous étudions deux types de configuration :

- Dans le chapitre 4, nous présentons nos résultats concernant des tampons recevant des données du réseau. Ces données sont consommées par une tâche situées sur le sous-système comprenant le tampon.

- Dans le chapitre 5, nous présentons nos résultats concernant des tampons partagés par n tâches producteurs et une tâche consommateur. L'ensemble des tâches est situé sur le sous-système comprenant le tampon.

Le modèle de tâche utilisé est celui décrit dans le chapitre 2 : une tâche T_i est décrite par l'ensemble de paramètres $\{S_i, J_i, P_i, C_i, D_i$ et $r_i\}$. Nous supposons que les tâches :

- Sont périodiques.

- Sont ordonnancées par un algorithme préemptif à priorité fixe.

- Ne sont pas soumises à des contraintes de précédence.

- Partagent des tampons.

La vérification de la faisabilité de telles applications est effectuée en deux étapes :

- La vérification du respect des contraintes temporelles est accomplie grâce aux méthodes classiques de faisabilité de tâches périodiques partageant des ressources (borne sur le taux d'utilisation processeur [LL73], calcul du temps de réponse [JP86, ABRT93], ...).

- Ensuite, **il est nécessaire que le nombre de messages présents dans le tampon n'excède pas sa taille**. Nous présentons, dans les chapitres 4 et 5, les critères de performances permettant d'analyser ces tampons. Ces résultats sont établis en supposant que toutes les tâches respectent leur échéance ($\forall i : r_i \leq D_i$). Nous considérons que l'analyse des tampons n'a pas d'influence sur la vérification de la faisabilité de l'ordonnancement.

Nous rappelons que la théorie des files d'attente permet d'analyser les performances de systèmes composés de serveurs, de clients et de zone d'attente : des personnes dans la salle d'attente d'un docteur, un switch réseau aiguillant les données... Si de nouveaux clients arrivent dans le système alors que le serveur est occupé, leur requête est placée dans une file. La connaissance du taux d'arrivée moyen et du nombre moyen de requêtes que le serveur peut traiter, permet de prédire, entre autres, le nombre moyen, le temps d'attente moyen et la probabilité d'avoir un nombre donné de clients dans la file.

Une file d'attente est décrite par au moins 3 caractéristiques (notation de KENDALL) : $A|B|c$. A représente la fréquence d'arrivée des clients. B décrit la distribution du temps de service. Finalement, c est le nombre de serveurs. Le taux d'arrivée et le temps de service ont des distributions qui peuvent être de type Déterministe D, Markovien M ou Générale G.

Les tampons des systèmes que nous étudions sont modélisés par des files d'attente où les clients sont les messages servant aux communications entre tâches. Une première approche aurait été de considérer que la tâche consommateur correspond au serveur et le tampon à la file. Or, dans notre cas, il n'y a pas de précédence entre l'arrivée des messages dans le tampon et l'activation de la tâche consommateur. Il y a donc une différence de fonctionnement entre nos tampons et les files d'attente. Nous posons donc des hypothèses pour contourner ce problème. L'objectif est d'obtenir une file d'attente dont le comportement soit identique à celui de notre tampon.

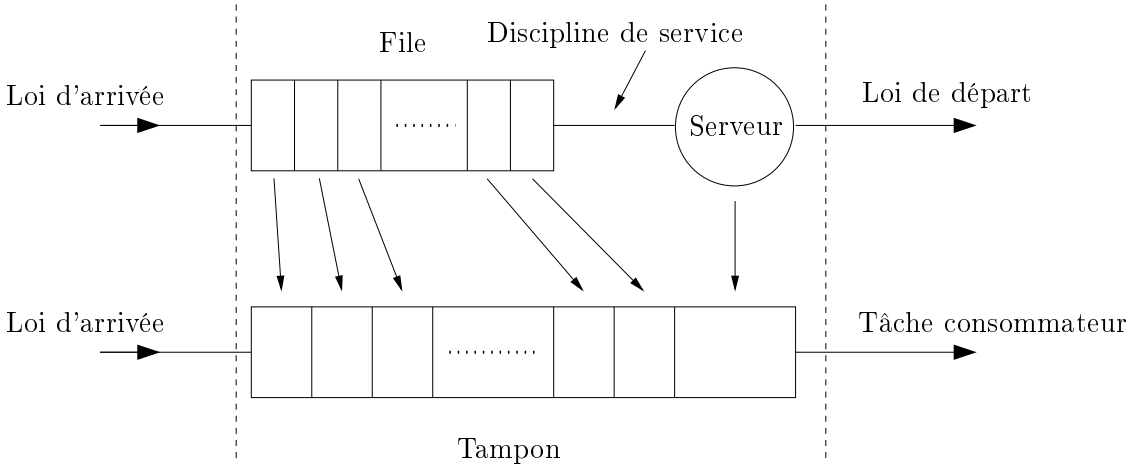

FIG. 3.2 – Modélisation du tampon par une file d'attente

Nous considérons que les éléments constituant une file d'attente, c'est à dire, la file et le serveur, correspondent au tampon (cf. figure 3.2). Les dates de sortie des messages de la file d'attente sont égales aux dates de consommation de la tâche consommateur. Le temps passé dans le tampon est donc égal au temps passé dans la file et le serveur. Nous pouvons faire la même observation pour le taux d'occupation.

Finalement, nous proposons une loi P pour modéliser le comportement de notre tampon grâce à une file d'attente (absence de précédence, périodicité des tâches,...). Cette loi est décrite par un délai moyen (temps entre 2 arrivées ou temps passé par un message dans le serveur) et une variance sur ce délai. **Ces deux paramètres sont nécessaires pour obtenir les critères de performances de la théorie des files d'attente**.

Dans un premier temps, nous énonçons les caractéristiques de la loi P. Ensuite, nous abordons le cas de la loi de service, puis de la loi d'arrivée P. Finalement, nous concluons.

3.1 Modélisation du comportement du tampon : la loi P

Tout d'abord, nous décrivons les caractéristiques de la loi P. Ces éléments nous servent à déterminer les paramètres nécessaires à l'utilisation de critères de performance de la théorie des files d'attente.

La loi P décrit le comportement temps réel et périodique des tâches producteurs ou consommateurs ordonnancées selon un algorithme préemptif à priorité fixe.

Afin de simplifier l'expression des critères de performances et des preuves fournies par la suite, nous considérons les hypothèses suivantes :

- Un seul message est produit ou consommé au maximum lors de l'activation périodique d'une tâche.

- Un message est consommé ou produit e_j^i unités de temps après l'activation de la $i^{ème}$ instance de la tâche j. e_j^i appartient à l'intervalle $[0, r_j]$ où r_j est le temps de réponse de la tâche j. Par la suite, nous considérons qu'un message est consommé à la fin du traitement de celui-ci par le serveur. Ce qui implique que e_j^i est égal à r_j^i, le temps de réponse de la $i^{ème}$ instance de la tâche j.

3.2 Loi de service P

Nous énonçons maintenant les caractéristiques propres à la loi de **service** P. La loi de service P est caractérisée par son temps de service moyen W_s. Nous rappelons la définition générale du temps de service d'une file d'attente :

Définition 14 (Temps de service) *Le temps de service S_i est le temps passé par un client à être traité par un serveur. [Kle75b]. S_i est donc le temps entre l'activation du serveur et la fin du traitement du message.*

- *Si le système est vide, le serveur est activé dès qu'un nouveau client arrive.*

- *Si un ou plusieurs clients sont en attente dans la file, le serveur est activé immédiatement après la fin du traitement du client précédent.*

Nous transposons cette définition aux tampons étudiés. Nous avons vu que le serveur ne correspond pas à la tâche consommateur. Les dates de sortie des messages de la file d'attente sont obtenues grâce aux dates de consommation de la tâche consommateur. Nous obtenons une nouvelle définition du temps de service :

Définition 15 (Temps de service de la loi P) *Dans notre modèle, le temps de service est égal au délai entre la date d'entrée du message dans le serveur et la date de consommation du message par la tâche consommateur.*

Dans notre application, les activations de la tâche consommateur sont indépendantes des arrivées de message. Les tâches produisent ou consomment à leur rythme : la tâche consommateur peut, d'une part, ne pas être prête alors que des messages sont présents dans le tampon, et d'autre part, s'activer alors qu'aucun message n'est présent dans le tampon. Dans ce cas, la tâche consommateur termine normalement son exécution.

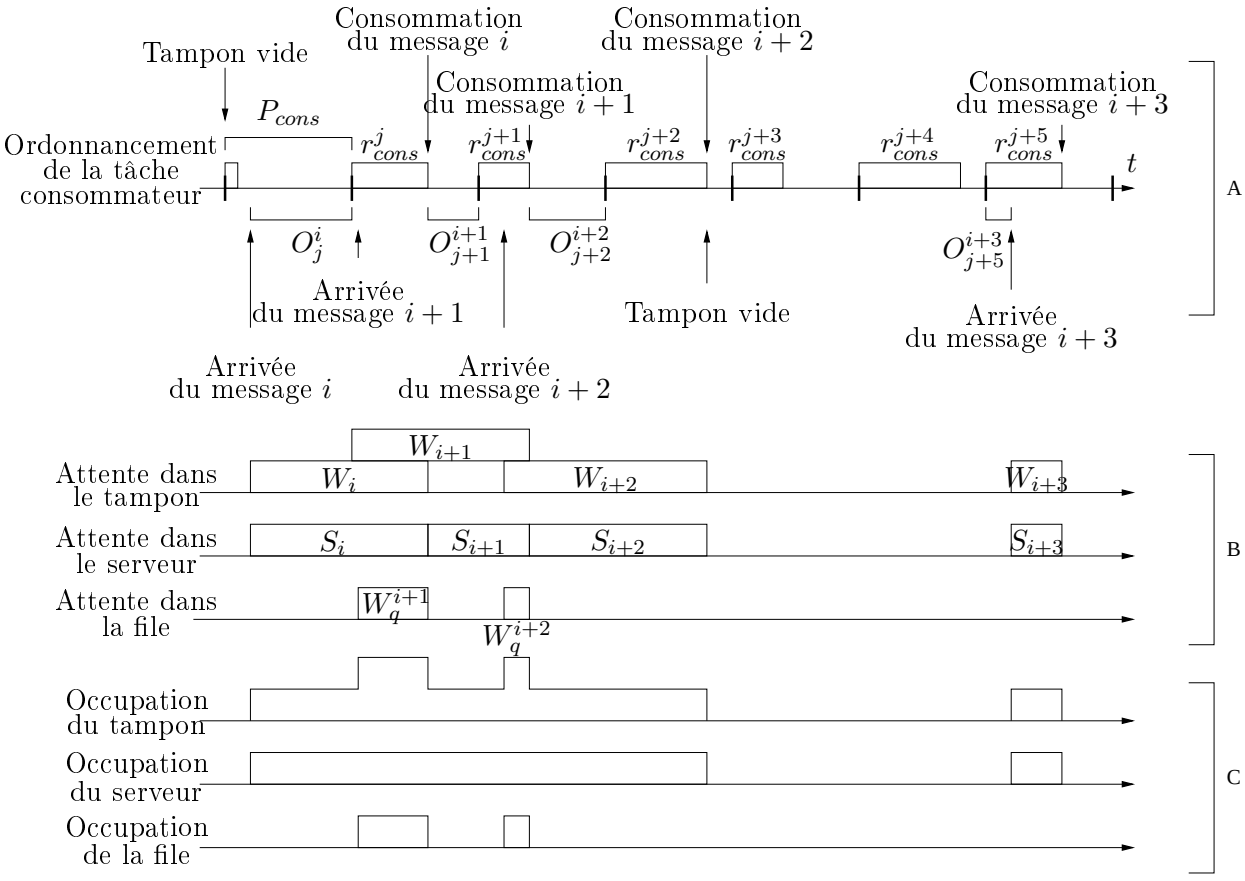

FIG. 3.3 – Temps de service de la loi P

Nous trouvons sur la figure 3.3 trois groupes de chronogrammes :

- Le chronogramme A illustre l'ordonnancement de la tâche consommateur *cons*. P_{cons} est la période de la tâche consommateur et r^j_{cons} est le temps de réponse de la $j^{ème}$ instance de la tâche consommateur.

- Les chronogrammes B représentent le temps d'attente dans le tampon W_i, le temps d'attente dans la file W^i_q et le temps de service S_i des messages i correspondant à l'ordonnancement de la tâche consommateur.

  D'après la définition 15, le temps de service S_i du message i est égal au délai entre la date d'entrée du message dans le serveur et la date de consommation du message par la tâche consommateur. Nous connaissons r^j_{cons}, le délai entre la $j^{ème}$ activation de la tâche consommateur et la date de consommation de i. Il nous manque donc le délai séparant la date d'entrée de i dans le serveur et la date d'activation de l'instance j qui va consommer ce message. Nous notons O^i_j ce délai. Par conséquent, S_i est égal à O^i_j plus r^j_{cons}.

  Nous remarquons que W_i est bien égal à $W^i_q + S_i$. Le temps d'attente dans la file W^i_q et W^{i+3}_q des message i et $i+3$ sont nuls. En effet, pour ces messages, le serveur est libre à leur arrivée. Ils ne restent donc pas dans la file.

- Les chronogrammes C donnent le taux d'occupation du tampon, de la file et du serveur. Comme pour le temps d'attente, nous observons que le taux d'occupation du tampon est bien égal au nombre de messages dans la file plus le nombre de messages dans le serveur.

En se basant sur l'exemple de la figure 3.3, nous proposons de décrire le comportement du tampon pour les messages i et $i+1$:

1. Initialement, la file d'attente est vide. Lors de son arrivée, le premier message i est donc immédiatement traité par le serveur. Le temps d'attente dans la file W^i_q est nul. Le nombre de message dans le serveur est donc de 1 tandis que la file reste vide.

2. Contrairement au cas du message i, lorsque le message $i+1$ arrive dans la file, le serveur est occupé. $i+1$ est donc stocké dans la file pendant W^{i+1}_q unité de temps. W^{i+1}_q représente l'attente de la fin du traitement courant du serveur. Le nombre de messages dans le serveur est toujours de 1, par contre l'occupation de la file est maintenant de 1 message.

3. Lorsque le message i est consommé, $i+1$ entre dans le serveur. Le nombre de messages dans le serveur ne change pas (il ne s'agit plus du même message) et la file est à nouveau vide.

4. ...

Finalement, le temps de service de la loi P est donné par le théorème 2.

Théorème 2 (La loi p : temps de service) *Lorsque le serveur est occupé, le $i^{ème}$ temps de service de la loi P est égal à :*

$$S_i^{'} = r_{cons}^{j} + P_{cons} - r_{cons}^{j-1}$$

Dans le cas contraire, S_i est égal à :

$$S_i^{''} = r_{cons}^{j} + O_j^i$$

O_j^i appartient à l'intervalle $[-r_{cons}^{j}; P_{cons} - r_{cons}^{j-1}]$.

Eléments de preuve :

Nous savons que le temps de service est l'addition du délai O_j^i séparant la date d'entrée du message i dans le serveur et la date d'activation de l'instance j qui va consommer ce message et du délai r_{cons}^{j} entre cette activation et la date de consommation effective.

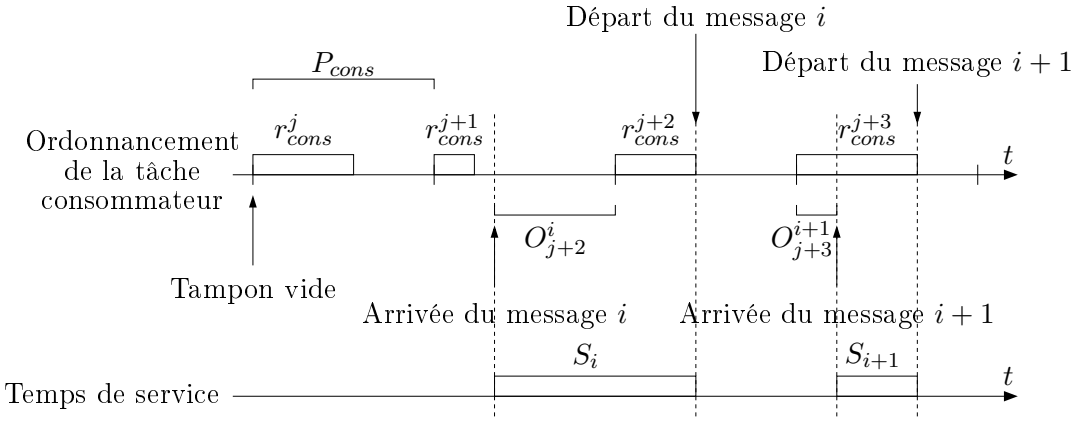

FIG. 3.4 – Temps de service lorsque le serveur est libre

Quand le tampon est vide, le message i est pris en charge par le serveur dès son arrivée. Nous considérons que i arrive entre la fin d'exécution d'une instance $j-1$ et la date de consommation de l'instance j. Sur la figure 3.4, nous observons alors plusieurs cas de figure :

- Lorsque le message i arrive avant la date d'activation de l'instance j, O_j^i est le délai entre l'arrivée du message dans l'instance $j-1$ et la date d'activation de la $j^{ème}$ instance de la tâche consommateur. O_j^i varie dans l'intervalle $[0; P_{cons} - r_{cons}^{j-1}]$.

- Lorsque le message i arrive après la date d'activation de l'instance j, O_j^i est le délai entre l'arrivée du message dans l'instance j et la date d'activation de la $j^{ème}$ instance de la tâche consommateur. O_j^i varie dans l'intervalle $[-r_{cons}^j; 0]$.

Quand le système contient un nombre de messages en attente supérieur à 1, dès que le serveur a terminé son exécution, il traite immédiatement le message suivant (cf. définition 14). D'après la définition 15, la date d'entrée du message dans le serveur correspond à la date de consommation du message précédent.

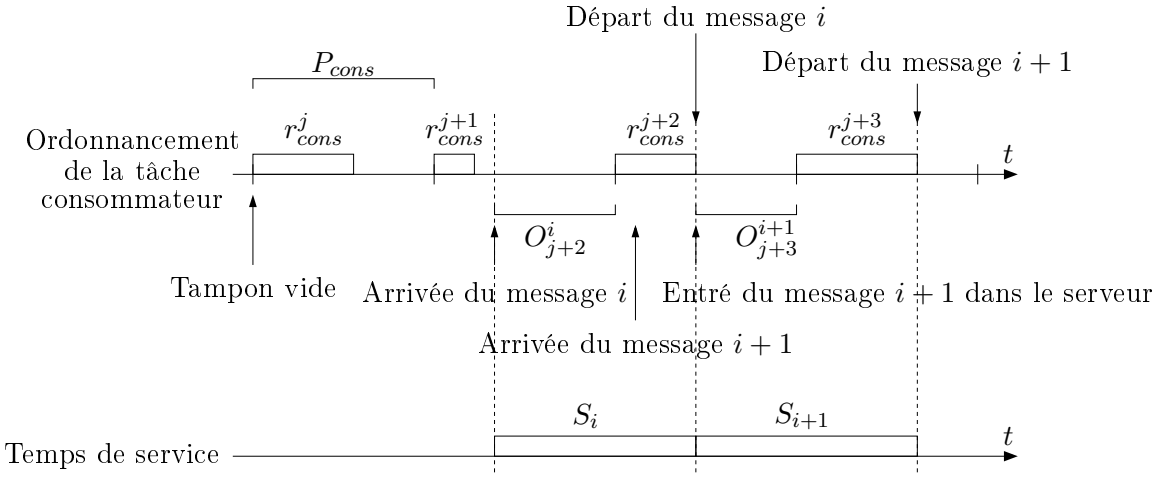

FIG. 3.5 – Temps de service lorsque le serveur est occupé

Nous observons sur la figure 3.5 que dans ce cas, O_j^i est égal à $P_{cons} - r_{cons}^{j-1}$. Le temps de service est donc égal à $P_{cons} - r_{cons}^{j-1}$ plus r_{cons}^j. □

3.3 Loi d'arrivée P

Occupons nous maintenant de la loi d'**arrivée** P. La loi d'arrivée P est caractérisée par le temps moyen entre deux arrivées A.

Nous considérons qu'un message est produit à chaque activation j de la tâche producteur *prod*, e_{prod}^j unités de temps après la $j^{ème}$ activation de *prod*.

Théorème 3 (La loi p : délai entre deux arrivées) *Le délai entre 2 arrivées consécutives selon la loi P est donné par la relation :*

$$A_i = e_{prod}^j + P_{prod} - e_{prod}^{j-1}$$

La preuve est identique à celle donnée pour la loi de service P en considérant qu'un message est systématiquement consommé par activation. Le serveur est dans ce cas toujours occupé.

3.4 Temps moyen entre 2 arrivées de messages A et temps de service moyen W_s

Nous proposons, ci-dessous, les expressions mathématiques permettant de calculer les valeurs moyennes de l'ensemble des délais S_i et A_i. Pour cela, nous utilisons le calcul classique de moyenne et de variance d'un échantillon de valeurs.

Définition 16 *Le temps de service moyen et la variance de ce temps de service sont égaux à :*

$$W_s = \frac{1}{n}\sum_{i=1}^{n} S_i \tag{3.1}$$

Et

$$\sigma_s^2 = \frac{1}{n}\sum_{i=1}^{n} S_i^2 - W_s^2 \tag{3.2}$$

Où n est le nombre de temps de service.

Définition 17 *Le délai moyen entre deux arrivées et la variance de ce délai sont égaux à :*

$$A = \frac{1}{n}\sum_{i=1}^{n} A_i \tag{3.3}$$

Et

$$\sigma_a^2 = \frac{1}{n}\sum_{i=1}^{n} A_i^2 - A^2 \tag{3.4}$$

Où n est le nombre de délais inter-arrivées.

3.5 Conclusion

Dans ce chapitre, nous présentons les éléments sur lesquels se basent nos solutions. Nous travaillons sur des applications mono-processeur reliées à d'autres systèmes par l'intermédiaire d'un réseau. Ces applications sont composées de tâches périodiques et indépendantes, de tampons et d'un algorithme d'ordonnancement à priorités fixes, préemptif et non oisif.

La validité de l'ordonnancement des tâches est vérifiée grâce aux résultats classiques de la littérature.

Nous modélisons les tampons de notre application par des files d'attente. Par la suite, les tampons sont nommés conformément à la notation de Kendall. Les tampons sont classés en deux catégories.

Des tampons reçoivent des messages aléatoirement par l'intermédiaire du réseau. Nous utilisons alors les critères moyen de performance issus de la théorie des files d'attente. Pour cela il est nécessaire d'évaluer les paramètres de la loi de service P.

Une des principales difficultés est d'obtenir le temps de service moyen W_s. En effet, nous avons vu que son expression est différente selon l'état du serveur.

Une autre difficulté est de déterminer la valeur de O_j^i lorsque le message i arrive et que le serveur est libre. Ce délai dépend alors de la date aléatoire d'arrivée du message.

En outre, la connaissance des dates de consommation ou de production de l'ensemble des instances des tâches est nécessaire. Or, il n'existe pas systématiquement de méthode de calcul pour un système temps réel donné.

Des tampons reçoivent des messages en provenance d'une ou plusieurs tâches périodiques. Dans ce cas, nous n'utilisons pas les critères de la théorie des files d'attente mais nous proposons des critères maximums. Ce travail est basé sur les recherches menées dans le domaine des réseaux ATM.

Chapitre 4

Les critères moyens de performance de la file M/P/1

Dans ce chapitre, nous supposons que les tampons de notre application temps réel réceptionnent des messages de manière aléatoire.

Selon la notation de KENDALL, un tampon partagé par n flux d'arrivée aléatoire de messages et une tâche périodique qui consomme ces messages correspond à une file M/P/1 [Kle75b, Rob90]. La discipline de service de la file est la politique FIFO : le premier message placé dans le tampon est aussi le premier à en être extrait. Dans la suite de ce chapitre, nous nous intéresserons uniquement à l'étude des systèmes possédant un unique serveur.

Nous proposons une approximation de la file M/P/1. Il est nécessaire d'évaluer le temps de service moyen W_s et la variance σ_s^2 du temps de service. Grâce à ces deux paramètres, nous pouvons utiliser les critères de performance de la théorie des files d'attente (cf. tableau 2.7).

Ce chapitre est organisé en trois parties :

- Nous présentons quelques définitions préliminaires dans la section 4.1.
- Nous proposons une résolution approchée de la file M/P/1 dans la section 4.2.
- Finalement nous concluons.

4.1 Notions préliminaires

Préalablement à l'étude du temps de service et de sa variance pour la file M/P/1, nous présentons quelques définitions/résultats.

La tâche consommateur est une tâche périodique indépendante ordonnancée suivant un algorithme d'ordonnancement temps réel. Le consommateur est donc activé indépendamment des arrivées de messages dans le tampon. Dans ce cas, une consommation peut être effective ou non.

Définition 18 (Consommation effective) *Une consommation est dite :*

- *Effective si la tâche consommateur est activée et au moins un message est présent dans le tampon.*
- *Non-effective lorsque la tâche consommateur est activée et qu'il n'y a pas de message dans le tampon.*

Nous pouvons observer sur la figure 4.1, qu'aux instants de consommation des instances j et $j+2$ de la tâche consommateur, le taux d'occupation du tampon est nul. Ce sont donc des consommations non-effectives. Par contre, aux instants de consommation des instances

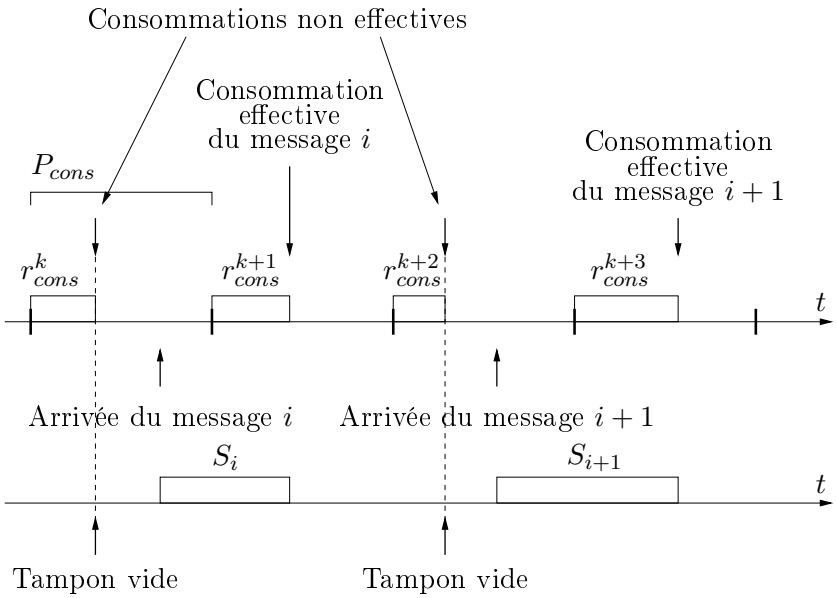

FIG. 4.1 – Consommation effective et non effective

$j+1$ et $j+2$, il y a 1 message dans le tampon. Les consommations sont, dans ce cas, effectives.

De la définition 18, nous pouvons déduire un résultat important pour déterminer le temps de service moyen : le taux de consommation effective.

Théorème 4 *Le taux de consommations effectives U_c de la file M/P/1 est égal à :*

$$U_c = 1 - P_0 = \rho$$

Où ρ est le taux d'utilisation de la file et P_0, la probabilité que le tampon soit vide.
Le nombre moyen de messages présents dans le tampon à l'instant t est donc de :

$$n(t) = \lambda t - \rho t = \lambda t(1 - W_s)$$

Où λt est le nombre moyen d'arrivées sur t unités de temps et ρt le nombre moyen de consommations effectives sur t unités de temps.

Eléments de preuve :

En effet, une consommation est effective si le tampon est non vide. Si P_0 représente la probabilité de n'avoir aucun message dans le tampon, le taux de consommation effective U_c est alors de $1 - P_0$. Or, pour une file G/G/1, $\rho = 1 - P_0$ [Kle75b]. La file M/P/1 étant un cas particulier de la file G/G/1, U_c est égale à ρ.

Le nombre moyen de messages présents dans le tampon à l'instant t est déduit du taux de consommations effectives. □

Maintenant que nous avons défini la notion de consommation effective, étudions le temps de service S_i. L'équation 3.1 permet de calculer le temps de service moyen. Dans cette équation, il est nécessaire de connaître l'ensemble des valeurs des temps de service S_i. Cet ensemble est difficile à déterminer pour deux raisons :

- A un temps de service S_i correspond une consommation effective. Etant donné que les messages arrivent dans le tampons de manière aléatoire, il est difficile de savoir si pour une activation donnée, la consommation associée est effective ou non. Autrement dit, il est difficile de prédire qu'une activation de la tâche consommateur implique une requête au serveur de la file M/P/1 et donc un nouvel S_i.

- La file M/P/1 doit être analysée sur un intervalle de temps infini. Ce qui implique que l'ensemble des temps de service S_i est infini ($n \to \infty$ dans l'équation 3.1). Or, nous avons besoin d'un ensemble fini de S_i pour calculer le temps de service moyen.

4.2 Proposition de résolution approchée de la file M/P/1

La résolution de la file M/P/1 consiste à trouver le temps de service moyen et la variance sur ce temps de service.

Pour cela, nous devons connaître le taux de consommation effectif de message à chaques instants t. Or ce taux semble difficile à obtenir. En première approche, nous nous contentons d'étudier un nombre limité de points :

1. Tout d'abord, nous résolvons le cas où le serveur est requis à chaque activation de la tâche consommateur. Dans ce cas, ρ (ou U_c) tend vers 1 et nous considérons que toutes les consommations sont effectives.

2. Ensuite, nous proposons une approximation lorsque ρ tend vers 0 (U_c tend vers 0). Dans ce cas, seules quelques rares activations de la tâche consommateur impliquent une requête au serveur. Le nombre de consommations effectives tend également vers 0.

3. Les deux cas précédents sont des extrêmes (ρ est compris entre 0 et 1). Grâce à une régression linéaire, nous obtenons donc l'approximation du temps de service moyen de la file M/P/1 et de sa variance valide quelle que soit la valeur de ρ.

4.2.1 Cas où ρ tend vers 1

Dans cette partie, nous étudions la file lorsque ρ tend vers 1. Dans ce cas, le comportement de la file M/P/1 devient principalement déterministe.

Théorème 5 *Lorsque ρ tend vers 1, le temps de service est de :*

$$S'_i = r^i_{cons} + P_{cons} - r^{i-1}_{cons}$$

Où cons est la tâche consommateur.

Le temps de service moyen et sa variance sont égaux à :

$$W_s = P_{cons}$$

$$\sigma_s^2 = \frac{1}{n}\sum_{i=1}^{n} S_i'^2 - W_s^2$$

Eléments de preuve :

En effet, lorsque ρ tend vers 1, toutes les consommations sont effectives (cf. Théorème 4). Comme nous l'avons vu, un des problèmes à résoudre est que si nous étudions la file sur une durée infinie, l'ensemble des S_i à considérer dans le calcul de W_s est infini. Il faut donc pouvoir réduire la période d'étude infinie de la file à une durée finie.

Grâce à la nature temps réel périodique des tâches, les séquences de temps de réponse de leurs instances sont cycliques. La taille commune de toutes ces séquences est la période d'étude [1] du jeu de tâches.

Nous considérons donc un intervalle de temps de nc fois la période d'étude du jeux de tâches.

- $nc * S'_1 = nc * (r^1_{cons} + P_{cons} - r^n_{cons})$
- $nc * S'_2 = nc * (r^2_{cons} + P_{cons} - r^1_{cons})$
- $nc * S'_3 = nc * (r^3_{cons} + P_{cons} - r^2_{cons})$
- ...

[1] La période d'étude (ou hyperpériode) représente une séquence infiniment répétée de l'ordonnancement [LM80]. Eventuellement, il est possible de ne considérer que la tâche consommateur et les tâches de plus forte priorité afin de réduire la taille de la séquence à considérer.

- $nc * S'_{n-1} = nc * (r^{n-1}_{cons} + P_{cons} - r^{n-2}_{cons})$
- $nc * S'_{n} = nc * (r^{n}_{cons} + P_{cons} - r^{n-1}_{cons})$

Finalement, la moyenne de l'ensemble de ces temps de service est égal à :

$$W_s = \frac{1}{nc.n} nc.n.P_{cons} = P_{cons}$$

En ce qui concerne la variance, nous effectuons le calcul sur le même échantillon des n temps de service. La variance σ_s^2 est donc égal à :

$$\sigma_s^2 = \frac{1}{n}\sum_{i=1}^{n} S_i'^2 - W_s^2$$

□

4.2.2 Cas où ρ tend vers 0

Nous étudions maintenant la file lorsque ρ tend vers 0. Dans ce cas, le comportement de la file dépend des instants d'arrivée des messages.

Théorème 6 *Lorsque ρ tend vers 0, le temps de service moyen est de :*

$$S''_i = \frac{S'_i}{2}$$

Le temps de service moyen et sa variance sont égaux à :

$$W_s = \frac{P_{cons}}{2}$$

$$\sigma_s^2 = \frac{P_{cons}^2}{12}$$

Eléments de preuve :

Nous avons vu que lorsque ρ tend vers 1, toutes les consommations sont effectives. Il est alors aisé de calculer le temps de service moyen. Par contre, lorsque ρ tend vers 0, les consommations effectives sont rares (cf. Théorème 4). Il est alors difficile d'obtenir une valeur exacte de W_s. Nous proposons donc une approximation de celui-ci.

Désormais, la probabilité que le serveur soit activé sur réception de message est très importante. Le temps de service est donc de la forme $S''_i = r^j_{cons} + O^i_j$ (cf. théorème 2). Un premier problème consiste à trouver la valeur de O^i_j. En outre, étant donné que le nombre

de requêtes au serveur (ou S_i) est inférieur au nombre d'activation du consommateur, l'ensemble des S_i entrant en compte dans le calcul de W_s est difficile à déterminer.

Pour résoudre ces problèmes, nous étudions la probabilité d'avoir un délai particulier entre 2 arrivées successives suivant un processus de poisson. Cette probabilité est représentée par la fonction suivante [Rob90, Kle75b] :

$$f(\lambda, t) = 1 - e^{-\lambda t} \tag{4.1}$$

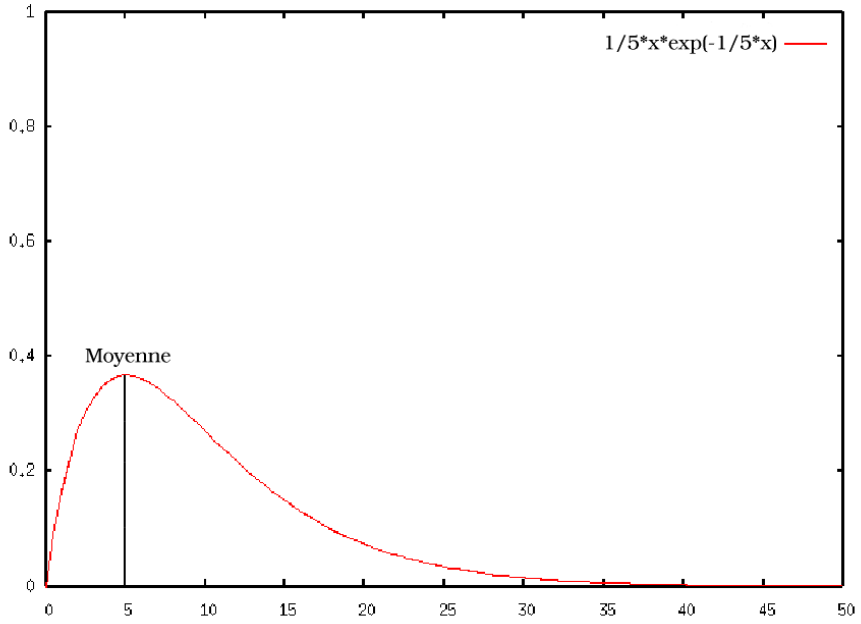

FIG. 4.2 – Processus de poisson où $\lambda = 1/5$

Sur la figure 4.2, nous avons la courbe de la fonction 4.1 pour une valeur de λ égale à $\frac{1}{5}$.

Lorsque ρ tend vers 0, la période de la tâche consommateur étant fixée, λ tend vers 0. Dans ce cas, l'intervalle de temps moyen entre 2 arrivées successives tend vers l'infini et la fonction $f(0, t)$ tend vers 0 quelle que soit la valeur de t.

Nous observons sur la figure 4.3 que dans ce cas la pente de la courbe est quasiment horizontale. La probabilité d'avoir un délai particulier entre 2 arrivées consécutives est la même quel que soit ce délai. Par conséquent, la date d'arrivée intervenant dans l'intervalle[2] $[0, S_i']$ avec une probabilité uniformément répartie implique que :

[2]Le serveur sera nécessairement sollicité durant l'intervalle $[0, S_i']$.

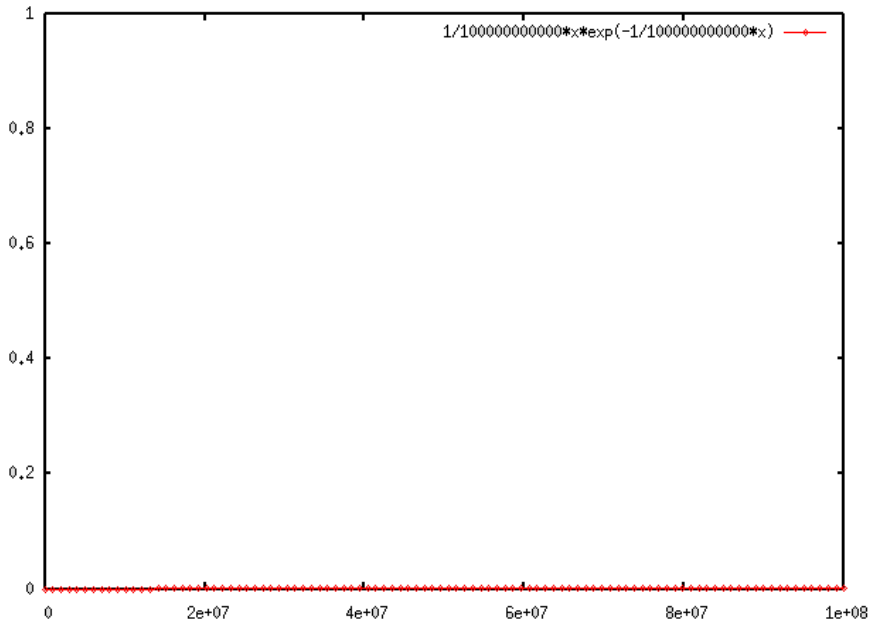

FIG. 4.3 – Processus de poisson où λ tend vers 0

$$S_i'' = \frac{S_i' + 0}{2} = \frac{S_i'}{2}$$

En outre, si les dates d'arrivées sont uniformément réparties sur une durée infinie, le nombre d'occurrences de temps de service nc est le même pour tous les S_i. Le temps de service moyen est donc de :

$$W_s = \frac{1}{n * nc} \sum_{i=1}^{n} \frac{nc * S_i'}{2}$$

Soit,

$$W_s = \frac{1}{2n} \sum_{i=1}^{n} S_i'$$

Finalement, nous avons :

$$W_s = \frac{P_{cons}}{2}$$

La variance d'un délai uniformément réparti sur un intervalle $[a, b]$ est de :

$$\sigma^2 = \frac{(a+b)^2}{12}$$

Nous cherchons la variance du temps de service moyen sur l'intervalle $[0, P_{cons}]$, soit :

$$\sigma_s^2 = \frac{P_{cons}^2}{12}$$

□

4.2.3 Temps de service et variance approchée quel que soit ρ

Nous appliquons une régression linéaire aux deux cas précédemment étudiés afin d'obtenir une expression du temps de service moyen et de sa variance pour tout ρ.

Théorème 7 *Le temps de service de la file M/P/1 est de :*

$$S_i = \rho S_i^{'} + (1-\rho).S_i^{''} = (1+\rho).S_i^{'}$$

Le temps de service moyen est alors égale à :

$$W_s = \frac{P_{cons}}{2}(1+\rho) = \frac{P_{cons}}{2(1-\lambda\frac{P_{cons}}{2})}$$

Et la variance de ce temps de service moyen est de :

$$\sigma_s^2 = \rho.\left(\frac{1}{n}\sum_{i=1}^{n} S_i^{'} - W_s^2\right) + (1-\rho).\frac{P_{cons}^2}{12}$$

La démonstration de ce théorème est basée sur le théorème 8.

Théorème 8 *Les poids associés aux temps de service $S_i^{'}$ et $S_i^{''}$ sont respectivement ρ et $1-\rho$.*

Eléments de preuve :

En effet, la probabilité qu'un message arrive dans la file alors que le serveur est occupé, resp. libre, est égale à ρ, resp. $1-\rho$. Or, lorsque le serveur est occupé, resp. libre, le temps de service du message i est égal à $S_i^{'}$, resp. $S_i^{''}$.

□

Maintenant que nous connaissons le poids associé aux deux formes du temps de service, nous pouvons démontrer le théorème 7.

Eléments de preuve :

D'après le théorème 8, le temps de service devient :

$$(1-\rho)\frac{S_i'}{2}+\rho S_i'$$

Ainsi, nous pouvons réécrire le temps de service moyen sous la forme suivante :

$$W_s=\frac{1}{n}\sum_{i=1}^{n}\left((1-\rho)\frac{S_i'}{2}+\rho S_i'\right)$$

Soit,

$$W_s=\left(\frac{1-\rho}{2}+\rho\right)\frac{1}{n}\sum_{i=1}^{n}S_i'$$

Or nous savons que :

$$\sum_{i=1}^{n}S_i'=P_{cons}$$

D'où,

$$W_s=\frac{P_{cons}}{2}(1+\rho)$$

Mais, ρ dépend du temps de service par la relation $\rho=\lambda W_s$. Si nous effectuons un changement de variable, nous obtenons :

$$W_s=\frac{P_{cons}}{2}(1+\lambda W_s)$$

Finalement,

$$W_s=\frac{P_{cons}}{2(1-\lambda\frac{P_{cons}}{2})}$$

Nous appliquons la même régression linéaire pour obtenir la variance. Nous savons que lorsque ρ tend vers 0, σ_s^2 est égale à :

$$\sigma_s^2=\frac{P_{cons}^2}{12}$$

Et quand ρ tend vers 1, σ_s^2 est égale à :

$$\sigma_s^2=\frac{1}{n}\sum_{i=1}^{n}S_i'-W_s^2$$

Par conséquent,

$$\sigma_s^2 = \rho.\left(\frac{1}{n}\sum_{i=1}^{n} S_i' - W_s^2\right) + (1-\rho).\frac{P_{cons}^2}{12}$$

□

4.2.4 Dates de consommation des messages

Dans cette section, nous nous intéressons à r_{cons}^j, le délai exacte entre l'activation j de la tâche consommateur et la date de consommation du message.

Nos résultats, et plus particulièrement la variance du temps de service, sont basés sur les dates de consommations. Un des avantages de ce choix est qu'il est possible de prendre en compte, dans ces dates, les variations suivantes :

- La variation du temps de réponse de la tâche due aux interférences des tâches de plus forte priorité.
- La variation de la durée réelle d'exécution de la tâche. La capacité de la tâche n'est qu'une borne sur cette durée.
- La variation du temps de réponse de la tâche due aux accès à des ressources partagées.

Nous proposons trois méthodes pour obtenir la valeur des r_{cons}^j pour toutes les instances j :

- Une méthode basée sur l'étude statistique d'une application : on mesure les r_{cons}^j lors de l'exécution de l'application. Cette méthode peut être appliquée lorsque la période d'étude du jeux de tâche est très grande ou qu'on ne la connaît pas précisément. Dans ce dernier cas, se pose le problème de la durée de l'étude statistique.
- Une méthode consistant à simuler l'ordonnancement du jeux de tâche de l'application sur la période d'étude (cf. section 2.2.1.1). Cette méthode ne peut être choisie que si la période d'étude des tâches est de taille raisonnable (il faut qu'un ordinateur soit capable d'exécuter cette simulation dans un temps acceptable).
- L'application des méthodes hors lignes. Il n'en existe pas pour des jeux de tâches dont la priorité est fixe. Pour avoir un intérêt, ces méthodes doivent posséder une complexité inférieure aux deux solutions précédentes.

Nous proposons ici une solution pour calculer les dates de consommation de messages qui s'inspire de la méthode du calcul du temps de réponse classique. La complexité de notre méthode est inférieure à celle d'une simulation de l'ordonnancement lorsque la priorité de la tâche considérée est suffisamment élevée.

Notre méthode utilise la notion de temps creux cyclique [GCGC98a, GCG01].

Définition 19 (Temps creux) *Un temps creux correspond à une unité de temps où aucune tâche n'est exécutée. Le taux de temps creux sur la période d'étude des tâches est égal à*

$$1 - U$$

U étant la charge processeur du système.
Il existe au moins deux types de temps creux :

- *Les temps creux **cycliques** sont identiques à chaque période d'étude du jeu de tâches. Les temps creux cycliques peuvent être modélisés par une tâche T, appelée tâche creuse, où $C_T = P(1-U)$, $D_T = P_T = P$ et $S_T = 0$ (P est la période d'étude du jeux de tâches).*

- *Les temps creux **acycliques** se rencontrent lorsque les tâches ne sont pas synchrones au démarrage.*

Nous cherchons, pour une tâche donnée i, à déterminer sur un intervalle de temps donné le nombre d'unités de temps creux cyclique et le nombre d'unités de temps occupé par l'ensemble $lp(i)$ des tâches de plus faible priorité que i. Nous limitons le jeu de tâche étudié aux tâches de priorité supérieure ou égale à i afin de ne considérer que les unités de temps creux. En effet, dans ce cas, les unités de temps occupées par les tâches de l'ensemble $lp(i)$ deviennent des unités de temps creux.

```
Algorithme Temps_Creux(t,i)
    si la charge processeur U est égale à 1 alors
        retourner 0
    sinon
        temps_creux = 0
        capacité_restante = 0
        pour k de 0 à t faire
            capacité_restante = max(0, capacité_restante - 1)
                + capacité des tâches j activées à l'instant k (avec priorité j ≥ i)
            Si capacité_restante = 0 alors
    fin si
```

FIG. 4.4 – Algorithme pour déterminer les temps creux cycliques

L'algorithme **Temps_Creux(t,i)** permet de calculer, pour une tâche i, le nombre d'unité de temps creux sur l'intervalle t (cf. figure 4.4). Il fonctionne de la manière suivante : d'après la définition 19, lorsque $U = 1$, il n'y a pas de temps creux. Dans le cas contraire, on détermine à chaque instant t, la charge processeur des tâches. Cette charge représente la capacité requise par ces tâches à l'instant t. Si elle est nulle, aucune tâche ne sera exécutée à l'unité de temps suivante. On incrémente alors le nombre d'unités de temps creux.

Le théorème 9 offre une méthode pour calculer le temps de réponse r_i^j des instances j d'une tâche i.

Théorème 9 *Pour un jeu de tâches où $\forall i : r_i \leq D_i$ et $D_i \leq P_i$, le temps de réponse de l'instance j d'une tâche i est égal à :*

$$r_i^j = r_v - (j-1).P_i$$

Où r_v est calculé grâce à l'équation 2.7 du pire temps de réponse d'une tâche virtuelle de capacité :

$$C_v = j.C_i + Temps\_Creux((j-1).P_i, i)$$

Eléments de preuve :

Le théorème repose sur le principe suivant : on crée une tâche virtuelle v de capacité C_v dont la fin d'exécution correspond à la date, relativement à 0, de terminaison de l'instance j de la tâche i considérée. Nous nous servons du temps de réponse r_v de la tâche v pour obtenir les temps de réponse des j instances de la tâche i.

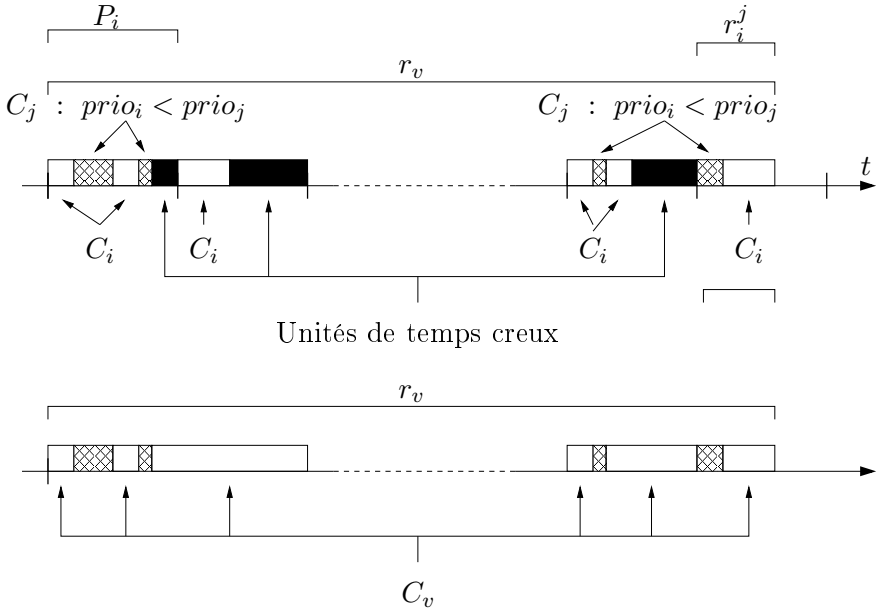

FIG. 4.5 – Temps de réponse d'une instance de la tâche

r_v est obtenu en appliquant la méthode classique du temps de réponse (cf. équation 2.7). On soustrait ensuite l'ensemble des périodes précédant l'instance j pour avoir le temps de

réponse $r_{i,j}$ de l'instance j relativement à son début d'activation (cf. figure 4.5). Il nous reste à obtenir la capacité C_v (la période de cette tâche n'est pas nécessaire pour le calcul).

Par définition [Riv98, Leb98, CDKM00], le temps de réponse d'une tâche est égal à la capacité de cette tâche plus les interférences des tâches de plus forte priorité. La capacité C_v est par conséquent au moins égale à j fois la capacité C_i. L'équation classique du calcul du temps de réponse suppose qu'une tâche active et possédant la priorité la plus importante est exécutée. Dans notre cas, les $j.C_i$ unités de temps de la capacité ne sont pas disponibles à tout instant. En effet, une fois la capacité d'une instance j épuisée, il faut attendre la prochaine activation pour continuer l'exécution de la tâche v. Le temps de réponse de notre tâche v comporte donc des unités de temps creux. Finalement, nous prenons en compte ces unités de temps dans la capacité de la tâche v grâce à l'algorithme TempsCreux(t,i). Soit, $C_v = j.C_i + TempsCreux((j-1).P_i, i)$.
□

La complexité de cet algorithme est inférieure à la complexité O(n.T) d'une simulation du jeux de tâche où n est le nombre de tâches du système et T la période d'étude.

Propriété 1 *La complexité de l'algorithme de calcul du temps de réponse d'une instance d'une tâche est de O(k.t.z).*

Eléments de preuve :

En effet, l'algorithme de calcul du temps creux d'une tâche i dépend du temps t et du nombre k de tâches de priorité supérieure à i. Sa complexité est donc en O(j.t).

La complexité du calcul du temps de réponse dépend du nombre k d'interférences des tâches de plus forte priorité que i. Elle dépend également d'un coefficient z représentant la vitesse de convergence des itérations de l'équation de calcul du temps de réponse 2.8. Finalement, la complexité du temps de réponse d'une instance d'une tâche est en 0(k.t.z).
□

La complexité du calcul de l'ensemble des dates de consommation est donc en O(k^2.t.z). Nous limitons le jeu de tâches aux tâches de plus forte priorité que i. Le nombre de tâches k considéré est donc inférieur au cas de la simulation de l'ordonnancement ($k \leq n$), et la période d'étude t de ce jeu de tâches est inférieure à la période d'étude T du jeu de tâche complet. Le coefficient z a peu d'incidence sur la complexité car l'équation 2.8 converge rapidement. Par conséquent, si la tâche i possède une priorité suffisamment élevée, O(k^2.t.z) sera bien inférieur à O(n.T), la complexité de la simulation de l'ordonnancement.

4.2.5 Probabilité de non débordement

Nous avons vu qu'il est possible d'utiliser les équations du temps moyen d'attente et du taux moyen d'occupation issues de la théorie des files d'attente.

Un critère supplémentaire permet d'évaluer la probabilité que le nombre maximum de messages présents dans le tampon soit inférieur ou égal à la taille du tampon. Nous appelons ce critère "probabilité de non débordement".

Cette probabilité peut être employée de deux manières :

- On peut évidemment obtenir une probabilité de non débordement à partir d'une taille spécifiée.

- On peut obtenir la taille du tampon à partir d'une probabilité de non débordement désirée.

La probabilité de non débordement d'un tampon de taille T dépend de la probabilité d'avoir i messages dans le tampon.

Définition 20 (La probabilité de non débordement) *La probabilité de non débordement associée à un tampon de taille T est :*

$$\prod_{i=0}^{T} P_i$$

Où P_i est la probabilité d'avoir i messages dans le tampon.

Pour certaines files d'attente la probabilité P_i est difficile à déterminer, en particulier pour la file M/G/1. Or la file M/P/1 est une instance de la file M/G/1. Par conséquent, nous ne disposons pas de la probabilité d'état de la file P_i. Nous proposons en annexe un début d'étude de cette probabilité pour la file M/P/1 (cf. annexe A).

4.3 Conclusion

Nous proposons une résolution approchée de la loi de service P. Cette résolution consiste à déterminer le temps de service moyen W_s et la variance de ce temps de service σ_s^2. Grâce à ces deux paramètres, nous pouvons utiliser les équations du temps moyen d'attente et du taux moyen d'occupation des files d'attente classiques telles que M/G/1 (cf. tableau 2.7).

Nous proposons également les premiers éléments permettant de calculer la probabilité de non débordement d'un tampon. Cette probabilité est nécessaire s'il on désire obtenir une taille de tampon suffisante pour avoir une perte minimum de messages.

Nous nous sommes limité à des lois d'arrivées markoviennes. Nos travaux peuvent être étendus aux cas où les arrivées sont générales, soit un intervalle de temps séparant 2 arrivées consécutives décrit par un temps moyen et une variance. Mais il ne semble pas exister pour l'instant de résultats suffisamment avancés pour les appliquer à notre cas.

Chapitre 5

Les critères maximums de performance de la file P/P/1

Dans ce chapitre, nous étudions les systèmes où des tâches périodiques produisent des messages dans un tampon.

Selon la notation de KENDALL, un tampon partagé par n tâches producteurs et m tâches consommateurs correspond à une file P/P/m. La discipline de service des messages est toujours la politique FIFO : le premier message arrivé est aussi le premier servi.

Il existe des similitudes entre notre système producteur/consommateur et certains systèmes issus du monde des réseaux hauts débits. C'est le cas des services de communication utilisés par le transport de la voix dans les réseaux ATM : la couche d'adaptation AAL1 [GK96].

Nous nous sommes donc basés sur les résultats obtenus dans le cadre des réseaux ATM afin de proposer des bornes maximum pour le taux d'occupation et le temps d'attente des messages lorsque les arrivées de messages sont déterministes.

Ce chapitre est organisé en quatre parties :

- Dans la section 5.1, nous décrivons le fonctionnement des couches AAL servant au transport des données à un débit fixe.
- En section 5.2, nous appliquons les résultats des travaux menés sur les réseaux ATM aux tampons étudiés. Notamment, nous donnons les définitions et propositions qui nous permettent d'établir des bornes maximums sur la taille des tampons.
- La section 5.3 propose les bornes maximums sur la taille des tampons.
- Finalement, nous concluons.

5.1 Taille des tampons dans la couche de transport à débit constant des réseaux ATM

Regardons les systèmes constitués d'un processeur émetteur et d'un processeur récepteur connectés par un réseau ATM (cf. figure 5.1).

ATM a été développé afin de fusionner les réseaux destinés au transport de la voix, de la vidéo et des données. Le protocole ATM est orienté connexion, ce qui signifie que deux machines qui veulent communiquer commencent par établir une connexion avant d'envoyer leurs données, ce qui permet notamment d'avoir une garantie sur le temps maximal qu'un paquet passera dans un commutateur.

L'AAL[1] est une interface entre les couches logicielles élevées et le réseau. Différents types d'AAL sont utilisés en fonction de la qualité de service désirée.

[1] ATM Adaptation Layer

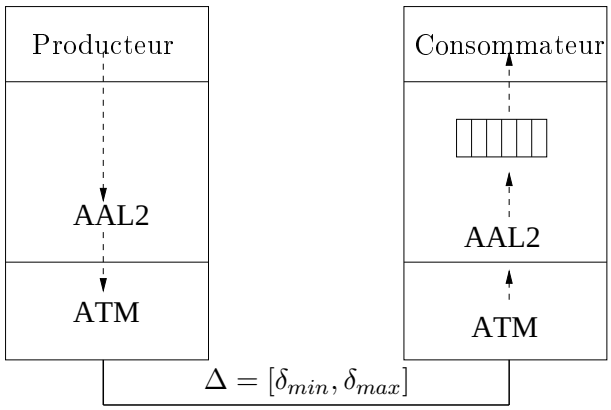

FIG. 5.1 – La couche de transport à débit fixe d'ATM

L'émetteur transmet des données audio à une cadence fixe. Nous noterons ce débit d'émission d. Le débit est exprimé en cellule, l'unité de transfert dans un réseau ATM.

Le temps de transport de ces cellules de l'émetteur au récepteur est variable et est compris entre δ_{min} et δ_{max}, respectivement le délai minimum et maximum de transport. Cette variation du délai de transmission, ou *gigue*, est dû à la traversée des différents noeuds (commutateurs ou brasseurs) situés entre l'émetteur et le récepteur. Cette gigue est notée $\triangle = \delta_{max} - \delta_{min}$.

Se pose alors le problème de la livraison des données par le récepteur aux couches logicielles de plus haut niveau. En effet, il serait difficile de suivre une conversation téléphonique si des distorsions, voire des coupures, de la voie de son interlocuteur existaient. Il est donc nécessaire de restituer le flux de données audio à la cadence de l'émetteur.

La solution consiste à mémoriser les informations audio dans un tampon afin de compenser la gigue produite par le réseau de transmission.

Une des difficultés consiste alors à déterminer la taille de ce tampon pour qu'il n'y ait pas de débordement. Nous notons L_{max} et W_{max} les bornes maximums sur, respectivement, la taille des tampons et le temps d'attente maximum d'un message dans le tampon (cf. équations 2.18 et 2.17). Dans [GK96], il est démontré que L_{max} est nécessaire pour éviter un débordement du tampon mais aussi suffisante car un tampon de plus grande taille est inutile.

Théorème 10 (Taille maximum des tampons ATM) *La taille maximum des tampons régulant le trafic de la couche de transport à débit fixe d'ATM est de :*

$$L_{max} = \left\lceil \frac{W_{max}}{d} \right\rceil \tag{5.1}$$

où $W_{max} = 2.\triangle$ *et* d *est la cadence fixe d'émission des données audio.* $2.\triangle$ *constitue* ***le plus grand délai pendant lequel les informations peuvent s'accumuler dans le tampon****.*

La borne est calculée en dénombrant le nombre de messages produits pendant le temps d'attente de la première cellule dans le tampon. Dans ATM, ce délai est aussi le plus grand délai sans consommation. L'opérateur plafond est utilisé afin que la valeur de la taille du tampon soit entière.

L'équation 5.1 peut être interprétée comme une illustration de la loi de Little dans le cas maximum (cf théorème 1). Autrement dit, la taille maximum du tampon est égale à un taux de production maximum pendant le délai d'attente maximum d'un message :

$$L_{max} = \lambda_{max} W_{max} \tag{5.2}$$

Avec, dans le cas des réseaux ATM, $\lambda_{max} = \frac{1}{d}$.

5.2 Application des résultats de la couche AAL1 à notre tampon

Nous étudions un système composé d'un ensemble de tâches périodiques qui consomment ou produisent des messages dans un tampon (multi-producteurs/multi-consommateurs). Ces tâches accèdent à un tampon et un seul. Nous rappelons qu'une tâche produit ou consomme au maximum **un** message par activation.

Un certain nombre de points communs existent entre les résultats présentés ci-dessus et les systèmes que nous étudions. Comme pour la couche AAL1 d'ATM, les producteurs/consommateurs sont périodiques. Par contre, ces tâches peuvent être exécutées sur le même processeur. Les producteurs et consommateurs interfèrent donc éventuellement les uns par rapport aux autres de par cet accès concurrent.

Dans le cas d'ATM, la borne est déterminée grâce au plus grand délai d'accumulation qui correspond aussi au pire délai d'attente d'une cellule ou encore au plus grand délai sans consommation. Dans les systèmes que nous étudions, nous cherchons toujours le plus grand délai d'accumulation associé à un tampon. Toutefois, contrairement à la couche AAL1, ce délai n'est plus équivalent au plus grand délai sans consommation.

Rechercher une borne sur la taille d'un tampon consiste donc à déterminer la contribution de chaque producteur pendant le délai d'accumulation tout en tenant compte des consommations effectuées. Pour ce faire, nous déterminons une borne sur le délai d'attente des messages dans le tampon.

Avant de donner les différentes bornes maximums sur la taille des tampons, nous présentons quelques définitions et propositions nécessaires à la compréhension de ces bornes.

Tout d'abord, nous devons garantir que le taux de production soit inférieur ou égal au taux de consommation. Cette condition est nécessaire pour avoir une borne maximum sur la taille des tampons.

Définition 21 (Loi de conservation du débit) *La condition nécessaire au non débordement d'un tampon est la suivante :*

$$\sum_{prod \in PROD} \frac{1}{P_{prod}} \leq \sum_{cons \in CONS} \frac{1}{P_{cons}} \tag{5.3}$$

Avec PROD, resp. CONS, l'ensemble des tâches qui produisent, resp. consomment, des messages dans le tampon.

Cette condition est le pendant de la loi de conservation des débits de la théorie des files d'attente (cf. définition 13). Nous remarquons que dans le cas de nos tampons, il n'est pas obligatoire d'avoir une inégalité stricte.

Pour appliquer la technique utilisée dans la couche AAL1, il est primordial de déterminer le délai maximum d'attente d'un message dans le tampon. Nous noterons W_{max} ce délai.

Nous rappelons que le délai d'attente est le délai entre la date de production d'un message et l'instant de sa consommation. Dans les plates-formes mono-processeur considérées, il y a interférence entre les producteurs et les consommateurs.

Ainsi, selon la configuration du jeu de tâches et compte tenu du fait que le tampon fonctionne de façon FIFO, entre l'instant de production d'un message par un producteur i et l'instant de sa consommation, y activations du (ou des) consommateur(s) sont nécessaires pour consommer les messages déjà présents dans le tampon. Nous ne cherchons pas, pour l'instant, à quantifier précisément y.

Définition 22 (Délai d'attente maximum) *Le délai d'attente maximum d'un message est de*

$$W_{max} = (y+1).P_{cons} + D_{cons} \tag{5.4}$$

Où y est le nombre de messages pouvant être déjà présents dans le tampon au moment où le message i arrive.

Nous pouvons observer sur le schéma 5.2 le pire cas suivant : la production du message est réalisée immédiatement après l'activation du consommateur et la consommation de ce message est effectuée en fin d'activation du consommateur. Le délai maximum d'attente est donc au pire cas de $P_{cons} + y.P_{cons} + D_{cons}$.

Nous avons fait l'hypothèse qu'une tâche produit un seul message par activation. Le taux maximum de production est donc de un message par période.

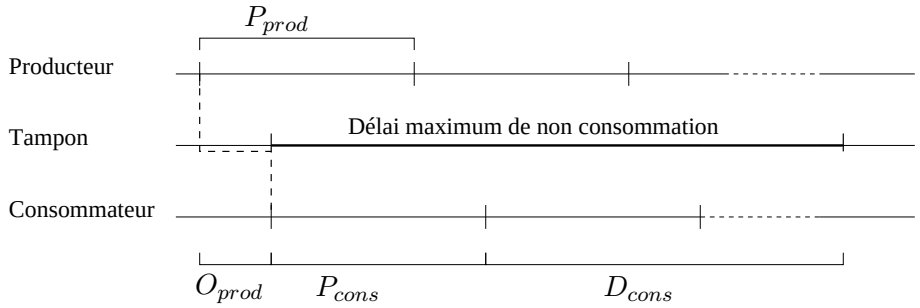

FIG. 5.2 – Désynchronisation des producteurs et consommateurs

Définition 23 *Le taux de production maximum est de :*

$$\lambda_{max} = \frac{1}{P_{prod}} \tag{5.5}$$

A partir des ces différentes définitions, nous proposons une expression de la borne maximum de la taille des tampons.

Dans ATM/AAL1, le délai maximum d'attente est évalué à partir de l'envoi de la première cellule du flux. Or, dans notre cas, le début du délai maximum d'attente peut correspondre à l'envoi de n'importe quelle cellule. Il existe donc une certaine désynchronisation entre les réveils des producteurs et des consommateurs. La borne maximale sur cette désynchronisation est notée O_{prod}. O_{prod} représente le pire délai entre un réveil de la tâche consommateur correspondant au début du délai d'attente étudié, et un réveil de la tâche producteur *prod* qui va émettre un message pendant ce délai (cf. figure 5.2). Durant cette désynchronisation, des messages peuvent être produits. Il est donc nécessaire de calculer la production maximum de messages sur l'intervalle de temps constitué du délai maximum d'attente W_{max} et du délai de désynchronisation O_{prod}.

Proposition 1 *La taille maximum L_{max} d'un tampon P/P/1 est de :*

$$L_{max} = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{W_{max} + O_{prod}}{P_{prod}} \right\rceil - y \right) \tag{5.6}$$

La borne maximum sur la taille du tampon est obtenue par construction à partir du délai maximum d'accumulation des messages et du débit maximum. La valeur de y n'étant pas connue, nous prenons le maximum de l'expression. Notons que contrairement à la solution utilisée dans ATM, y messages sont retirés de la borne dans l'équation 5.6. En effet, la borne des tampons de la couche AAL1 est construite grâce au plus grand délai d'accumulation des messages dans le tampon qui est égal au plus grand délai sans consommation. Nous avons vu que dans notre cas, ces deux délais ne sont plus équivalents. Pendant le délai

d'attente (cf. équation 2.17), y messages sont consommés. Ils doivent donc être retirés afin d'obtenir le nombre de messages accumulés pendant le plus grand délai d'accumulation.

Dans un premier temps, nous étudions le cas où les délais critiques sont inférieurs ou égaux aux périodes ($\forall i : D_i \leq P_i$). Dans ce cas, le délai maximum d'attente d'un message devient $W_{max} = (y+2).P_{cons}$. Le débit maximum reste de un message par activation de la tâche.

Nous traitons les cas où le tampon est partagé par n producteurs et un consommateur, puis par n producteurs et m consommateurs.

Proposition 2 *La taille maximum d'un tampon partagé par un consommateur et n producteurs est de :*

$$L_{max} = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{(y+2).P_{cons}}{P_{prod}} \right\rceil + n - y \right) \tag{5.7}$$

Lorsque $\forall i : D_i \leq P_i$.

Eléments de preuve :

Pour n producteurs et un consommateur, les réveils sont désynchronisés. Au pire cas, la durée de cette désynchronisation est $O_{prod} = P_{prod}$. Nous avons toujours $W_{max} = (y+2).P_{cons}$. En substituant à l'équation 5.6, nous obtenons :

$$L_{max} = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{(y+2).P_{cons} + P_{prod}}{P_{prod}} \right\rceil - y \right)$$

Soit,

$$L_{max} = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{(y+2).P_{cons}}{P_{prod}} \right\rceil + n - y \right)$$

□

Proposition 3 *La taille maximum d'un tampon partagé par m consommateurs et n producteurs est de :*

$$L_{max} = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{(y+2).P_{cons}^{\max}}{P_{prod}} \right\rceil + n - y \right) \tag{5.8}$$

Eléments de preuve :

Enfin, avec m consommateurs et n producteurs, le délai d'attente maximum est déterminé à partir de la plus grande période des consommateurs. Notons $P_{cons}^{\max}$ la plus grande période des consommateurs. Nous avons, $W_{max} = (y+2).P_{cons}^{\max}$. Etant donné que $O_{prod} = P_{prod}$, nous obtenons :

$$L_{max} = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{(y+2).P_{cons}^{\max}}{P_{prod}} \right\rceil + n - y \right)$$

□

Lorsque les délais critiques D_i sont plus grands que les périodes, selon l'ordonnancement des tâches, les différentes activations d'un producteur peuvent être retardées et provoquer une rafale de messages (cf. fin de la section 2.2.1.3 et la figure 5.3).

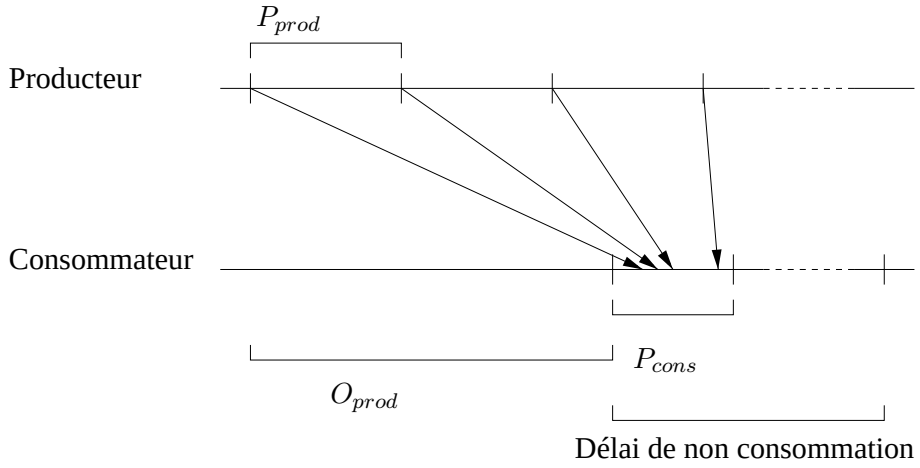

FIG. 5.3 – Rafale de messages

Proposition 4 *La taille maximum d'un tampon partagé par m consommateur et n producteurs dont les délais critiques sont arbitraires est de :*

$$L_{max} = \max_{y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{\forall cons \in CONS : \max((y+1).P_{cons} + D_{cons}) + D_{prod}}{P_{prod}} \right\rceil - y \right)$$

Eléments de preuve :

Précédemment, la désynchronisation maximale était bornée par la période des producteurs. Cette désynchronisation maximale représentait la production au plus tard de message pendant une activation et une seule. Avec un délai critique éventuellement supérieur à la période, au pire cas, cette désynchronisation est bornée par $O_{prod} = D_{prod}$.

Le plus grand délai d'attente devient :

$$W_{max} = \forall cons \in CONS : \max((y+1).P_{cons} + D_{cons})$$

Finalement la borne maximum sur la taille du tampon est de :

$$L_{max} = \max_{y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{\forall cons \in CONS : \max((y+1).P_{cons} + D_{cons}) + D_{prod}}{P_{prod}} \right\rceil - y \right)$$

□

Dans quasiment toutes ces équations, une variable ne dépend pas des spécifications du jeu de tâches mais de l'ordonnancement. Il s'agit de y. Nous ne disposons pas de moyen pour calculer sa valeur. Dans certain cas, il est possible de calculer les valeurs de L_{max} et W_{max} grâce à une étude des limites de ces équations.

5.3 Bornes maximums sur le temps d'attente et le taux d'occupation des tampons

Les équations 5.7 et 5.8 comprennent un terme y non défini. Malgré cette inconnue, des bornes peuvent être déduites de ces expressions.

Dans un premier temps, nous travaillons sur des tampons partagés par un consommateur et un producteur. Puis, nous étudions le cas où le tampon possède plusieurs producteurs et un consommateur.

5.3.1 Système "un consommateur et un producteur"

Le système considéré comprend un consommateur et un producteur périodique. Les bornes maximums sur le taux d'occupation et le temps d'attente de ce tampon sont données par le théorème suivant :

Théorème 11 *Pour un tampon partagé par un consommateur et un producteur périodique, les bornes maximums sur le taux d'occupation et sur le temps d'attente sont de :*

$$L_{max} = 2 \tag{5.9}$$

$$W_{max} = 2.P_{cons} \tag{5.10}$$

Eléments de preuve :

Au pire cas, lorsque le tampon est partagé par un producteur et un consommateur uniquement, la contrainte de débit (cf. définition 21) implique que $P_{cons} = P_{prod}$. En outre, comme producteur et consommateur ont une période identique, nous avons $O_{prod} = 0$. Si l'on substitue ces informations à l'équation 5.6, nous obtenons :

$$L_{max} = \left\lceil \frac{(y+2).P_{cons}}{P_{cons}} \right\rceil - y = y + 2 - y = 2$$

D'après Little, nous avons (cf. théorème 1) :

$$L = \lambda W$$

De plus,

$$W_{max} = (y+2).P_{cons}$$

Or,

$$\lambda = \frac{1}{P_{cons}}$$

Par conséquent,

$$L_{max} = 2 = y + 2$$

Soit,

$$y = 0 \quad et \quad W_{max} = 2.P_{cons}$$

□

5.3.2 Système "un consommateur et n producteurs"

Regardons plus en détail le cas de l'équation 5.7. Le système considéré comprend un consommateur et n producteurs. Ce paradigme est classique dans les systèmes puisqu'il correspond, par exemple, au multiplexage des requêtes de plusieurs clients (les producteurs) vers un serveur (le consommateur). Il se trouve que dans les applications que nous ciblons, ce paradigme est aussi fréquemment présent. Dans ce paragraphe, nous discutons d'une propriété remarquable de la proposition (5.7), qui est la suivante :

Théorème 12 *Pour un tampon P/P/1 partagé par un jeu de tâches harmoniques*[2] *dont n consommateurs avec $D_i \leq P_i$, les bornes maximums sur le taux d'occupation et sur le temps d'attente sont de :*

$$\begin{aligned} L_{max} &= 2.n \\ W_{max} &= 2.n.P_{cons} \end{aligned} \tag{5.11}$$

Dans le cas d'un jeu de tâches non harmoniques, les bornes maximums sur le taux d'occupation et sur le temps d'attente sont de :

$$\begin{aligned} L_{max} &= 2.n + 1 \\ W_{max} &= (2.n + 1).P_{cons} \end{aligned} \tag{5.12}$$

Eléments de preuve :

Nous commençons par étudier le cas où les tâches ne sont pas harmoniques. Nous montrons grâce aux équations 5.3 et 5.6, qu'au pire cas, un seul producteur peut accumuler 3 messages et les $n-1$ producteurs restants accumulent au plus 2 messages.

Nous montrons d'abord qu'un producteur peut produire au plus $y+3$ messages pendant le délai d'attente. Pour un jeu de tâches composé de 1 consommateur et n producteurs, nous savons que $P_{prod} > P_{cons}$. Le débit maximum d'un producteur i est obtenu lorsque $P_i \to P_{cons}$. Soit une production de $\left\lceil \frac{(y+2).P_{cons}+O_i}{P_i} \right\rceil = \left\lceil \frac{(y+2).P_{cons}+P_{cons}}{P_{cons}} \right\rceil = y+3$ messages maximum pendant W_{max}.

Nous cherchons maintenant à déterminer l'intervalle I des valeurs de P_i pour lequel la production reste de $y+3$ messages. Lorsque $P_i \to P_{cons}$, le premier (ou le dernier) des $y+3$ messages est produit dans l'intervalle $]0, P_{cons}[$. Ce délai peut être uniformément réparti aux $y+1$ instances du producteur afin de maximiser sa période tout en conservant une production de $y+3$ messages. Ainsi, la plus grande période du producteur i, telle que

[2] Un ensemble de tâches est harmonique si toutes les tâches possèdent des périodes multiples entre elles.

$y+3$ messages soient produits, est donc $P_i = \frac{P_{cons}}{y+1} + P_{cons} = \frac{(y+2).P_{cons}}{y+1}$. La production est de $y+3$ messages pour un producteur lorsque sa période est comprise dans l'intervalle $I =]P_{cons}, \frac{(y+2).P_{cons}}{y+1}[$.

Nous montrons maintenant que si la période d'un producteur tend vers $\frac{(y+2).P_{cons}}{y+1}$, alors les $n-1$ producteurs restants ont une période qui tend vers $(y+2).P^+_{cons}$. En effet, la loi de conservation du débit (cf. définition 21) peut être exprimée de la façon suivante :

$$\frac{1}{x} + \sum_{prod \in PROD^*} \frac{1}{P_{prod}} \leq \frac{1}{P_{cons}}$$

ou encore :

$$\sum_{prod \in PROD^*} \frac{1}{P_{prod}} \leq \frac{x - P_{cons}}{x.P_{cons}}$$

Où $PROD^*$ est l'ensemble des producteurs $PROD$ auquel nous avons retiré le producteur dont la période est x. Ce producteur est celui qui produit $y+3$ messages.

Posons :

$$f(x) = \frac{x - P_{cons}}{x.P_{cons}}$$

et étudions les limites de cette fonction lorsque $x \to \frac{(y+2).P_{cons}}{y+1}$ et lorsque $x \to P_{cons}$. On obtient :

- $\lim_{x \to \frac{(y+2).P_{cons}}{y+1}} f(x) = \frac{1}{(y+2).P_{cons}}$

  Or $\sum_{prod \in PROD^*} \frac{1}{P_{prod}} \leq \frac{1}{(y+2).P_{cons}}$; ce qui implique, au pire cas : $\forall prod \in PROD^* : P_{prod} \to (y+2).P^+_{cons}$.

- $\lim_{x \to P_{cons}} f(x) = 0$

  Or $\sum_{prod \in PROD^*} \frac{1}{P_{prod}} \leq 0$; ce qui implique, au pire cas : $\forall prod \in PROD^* : P_{prod} \to \infty$.

L'intervalle des périodes des producteurs appartenant à $PROD^*$ est donc $J =](y+2).P_{cons}, \infty[$. Nous obtenons donc une production maximum de :

$$\sum_{prod \in PROD^*} \left\lceil \frac{W_{max} + O_{prod}}{P_{prod}} \right\rceil$$

ou encore :

$$\sum_{prod \in PROD^*} \left\lceil \frac{(y+2)P_{cons} + (y+2).P^+_{cons}}{(y+2).P^+_{cons}} \right\rceil = 2.n - 2$$

Finalement, d'après l'équation 5.6, la borne est de :

$$L_{max} = y + 3 + \sum_{prod \in PROD^*} \left\lceil \frac{(y+2)P_{cons} + (y+2).P^+_{cons}}{(y+2).P^+_{cons}} \right\rceil - y$$

et donc

$$L_{max} = y + 3 + 2.n - 2 - y = 2.n + 1$$

Dans la preuve du théorème 11, nous montrons que $L_{max} = y + 2$, soit :

$$W_{max} = (2.n + 1).P_{cons}$$

Dans le cas d'un jeu de tâches harmoniques, nous appliquons la méthode décrite ci-dessus. Cette fois-ci, un producteur émet au pire cas $y+2$ messages pendant le délai d'attente. En effet, dans le cas harmonique, le décalage entre les activations de deux tâches ayant la même période est nul. Un producteur i émet donc au pire cas $\left\lceil \frac{(y+2).P_{cons}}{P_{cons}} \right\rceil = y + 2$ messages. L'intervalle I des valeurs possibles de P_i devient $[P_{cons}, \frac{(y+2).P_{cons}}{y+1}]$. Si l'on substitue à l'équation 5.6 ces informations comme pour le cas non harmonique, nous obtenons :

$$L_{max} = y + 2 + \sum_{prod \in PROD^*} \left\lceil \frac{(y+2)P_{cons} + (y+2).P_{cons}}{(y+2).P_{cons}} \right\rceil - y$$

Soit,

$$L_{max} = y + 2 + 2.n - 2 - y = 2.n$$

Et,

$$W_{max} = 2.n.P_{cons}$$

□

5.4 Conclusion

Nous faisons une description des réseaux ATM et des résultats concernant la couche AAL1 pour dimensionner les tampons.

A partir des travaux menés sur la couche AAL1, nous avons proposé un certain nombre de bornes maximums sur la taille des tampons partagés par un ensemble de tâches périodiques.

En particulier, trois résultats nous semblent intéressants :

- L'occupation maximum d'un tampon partagé par un consommateur et un producteur est de :

$$L_{max} = 2$$

- L'occupation maximum d'un tampon partagé par un consommateur et n producteurs, lorsque ces tâches sont harmoniques, est de :

$$L_{max} = 2.n$$

- L'occupation maximum d'un tampon partagé par un consommateur et n producteurs, pour des tâches quelconques, est de :

$$L_{max} = 2.n + 1$$

Ces bornes ne supposent pas l'utilisation d'un algorithme d'ordonnancement particulier. Elles peuvent donc être affinées si des hypothèses sont faites sur l'ordre d'exécution des différentes tâches. Pour l'instant, nous ne nous sommes pas encore intéressé aux conséquences de l'utilisation d'algorithme d'ordonnancement particulier sur les critères de performance de la file P/P/1.

Nous travaillons actuellement sur l'élaboration des bornes maximums sur le taux d'occupation et le temps d'attente pour des tampons possédant n producteurs et m consommateurs.

Chapitre 6

Simulation des files M/P/1 et P/P/1

Dans les chapitres 3, 4 et 5, nous proposons des critères d'analyse de performance de tampon. Afin de vérifier la validité de nos résultats, nous avons développé des simulateurs pour les files M/P/1 et P/P/1.

Nous nous sommes donc intéressé aux techniques de simulation à événement discret. Le rôle de nos simulateurs est de faire évoluer l'état du système. L'état du système correspond au nombre de messages présents dans le tampon à un instant t. Pour des raisons de performance, nous nous occupons uniquement des instants de consommation et de production des messages. Ainsi, une simulation consiste à générer les dates d'arrivées et de départs des messages, puis à mesurer les critères qui nous intéressent.

Les simulateurs ont été développés en langage Ada. Ce langage a été choisi pour des raisons de portabilité. En effet, les opérations arithmétiques, les types, le générateur aléatoire,... sont normalisés au travers du standard ISO/IEC 8652 :1995. Les résultats produits par nos simulateurs sont donc identiques quelques soit la plateforme ou le système d'exploitation. Evidement, il faut que cette plateforme ou ce système d'exploitation possède un compilateur Ada certifié ISO/IEC 8652 :1995.

Ce chapitre est organisé en trois parties :

- Dans la section 6.1, nous décrivons le simulateur P/P/1. Ensuite, nous détaillons les conditions dans lesquelles se sont effectuées les simulations. Enfin, nous analysons les résultats obtenus.

- La section 6.2 est organisée de la même manière que la section précédente mais concerne la file M/P/1.

- Finalement nous concluons.

6.1 Simulation de la file P/P/1

L'objectif des simulations de la file P/P/1 est de confirmer que le nombre de messages présents dans le tampon n'est jamais strictement supérieur aux bornes maximums du théorème 12 : $2.N$ et $2.N + 1$.

Nous commençons par décrire l' algorithme du simulateur de file P/P/1 que nous avons utilisé. Les interfaces des procédures implémentant ces algorithmes sont également présentées. Puis, nous analysons les résultats de simulation obtenus.

6.1.1 Description du simulateur P/P/1

Dans cette section, nous décrivons les algorithmes utilisés pour concevoir notre simulateur de file P/P/1, ainsi que les interface des fonctions/procédures qui mettent en œuvre ces algorithmes.

```
Algorithme Simulation (nb_producteur, nb_simulation)
Début
    – Initialisation des simulations
    –
    Initialisation ()
    Pour n allant de 1 à nb_simulation faire
        – Initialisation de la n^ième simulation
        –
        Initialisation_n ()
        – on exécute la n^ième simulation
        –
        Simulation_n ()
        – on affiche les résultats de la n^ième simulation
        –
        Résultat () ;
    Fin pour ;
    – fin des simulations
Fin ;
```

FIG. 6.1 – Algorithme du simulateur de file P/P/1

L'algorithme **Simulation** permet d'effectuer et d'analyser *nb_simulation* simulations d'une file P/P/1 possédant *nb_producteur* producteurs (cf. figure 6.1). Cet algorithme est subdivisé en quatre parties :

1. Une phase d'initialisation.

2. Une phase d'initialisation spécifique à une simulation n donnée.

3. Une phase d'exécution de la simulation n.

4. Une phase d'affichage des résultats de la simulation n.

Lorsque toutes les simulations ont été effectuées, soit $n = nb\_simulation$, l'algorithme se termine.

Nous détaillons maintenant chacune de ces phases.

```
Algorithme Initialisation ()
Début
    tableau_tache ← génération des périodes des tâches
    borne_maximum ← borne maximum théorique sur l'occupation du tampon
Fin ;
Algorithme Initialisation_n ()
Début
    occupation_tampon ← 0
    occupation_max_tampon ← 0
    tableau_temps_reponse ← {-1, -1, ...,-1}
Fin ;
```

FIG. 6.2 – Les procédures d'initialisation du simulateur P/P/1

Les algorithmes **Initialisation** et **Initialisation_n** sont utilisés pour initialiser les variables de la simulation (cf. figure 6.2). Dans le premier, on génère les valeurs des périodes des $nb\_producteur$ tâches producteurs et de la tâche consommateur. Ces valeurs respectent la loi de conservation du débit (cf. définition 21). A partir de ces valeurs, on calcul la borne maximum sur l'occupation du tampon (cf. théorème 12). Le deuxième algorithme remet à 0 les variables nécessaires à une simulation de la file P/P/1 générée.

L'algorithme **Simulation_n** exécute une simulation de la file P/P/1 (cf. figure 6.3).

Cette simulation est effectuée sur la période d'étude du jeu de tâches (cf. section 2.2.1.1. L'indice j de la tâche i appartient donc à l'intervalle $[1, \frac{PPCM}{P_i}]$.

A chaque pas de simulation, l'état du système évolue. Selon que le prochain événement soit une production ou une consommation, le nombre de messages présents dans le tampon augmente ou diminue. La simulation consiste donc à déterminer l'ordre d'apparition des ces événements.

Le prochain événement possède le temps de réponse le plus proche de l'instant courant. Nous considérons dans cet algorithme que la production et la consommation interviennent à la fin du temps de réponse. Les temps de réponse, relativement à 0, de l'instance j des tâches producteurs ou consommateur i sont générés aléatoirement dans l'intervalle $[0, j.P_i + D_i[$. Ces temps de réponse sont stockés dans la variable $tableau\_temps\_reponse$.

La procédure $Temps\_Reponse\_Minimum$ renvoie l'indice de la tâche correspondant à l'événement le plus proche de l'instant courant. Si des dates de production et de consommation sont identiques, l'indice de la tâche producteur est retourné en priorité. Il suffit ensuite de trouver le plus petit élément du tableau $tableau\_temps\_reponse$ et d'ajouter ou retirer un message du tampon. Dès qu'un évènement a été traité, son temps de réponse est effacé afin de générer un nouveau temps de réponse.

Lorsque toutes les instances des tâches ont été traitées, la simulation est stoppée.

```
Algorithme Simulation_n ()
Début
    Tant qu'il reste des activations de tâche à traiter faire
        Pour toutes les tâches i faire
            – on génère si nécessaire le temps de réponse de l'activation courante
            –
            Si tableau_temps_reponse(i) = -1 alors
                tableau_temps_reponse(i) ← temps de réponse uniformément
                        généré dans dans l'intervalle [0 ;période de la tâche] ;
            Fin si ;
        Fin pour ;
        – On cherche la prochaine date de consommation/production de message
        –
        indice_tache ← temps_reponse_minimum(tableau_temps_reponse) ;
        tache ← tableau_tache(indice_tache) ;
        – On met à jour la valeur de l'occupation courante/maximum du tampon
        –
        Si tache est un consommateur alors
            Si occupation_tampon > 0 alors
                occupation_tampon ← occupation_tampon - 1 ;
            Fin si ;
        Sinon si tache est un producteur alors
            occupation_tampon ← occupation_tampon - 1 ;
            – on affecte l'occupation maximum du tampon
            –
            Si occupation_tampon > occupation_max_tampon alors
                occupation_max_tampon ← occupation_tampon ;
            Fin si ;
        Fin si ;
        tableau_temps_reponse(indice_tache) ← -1 ;
    Fin tant que ;
Fin ;
```

FIG. 6.3 – Simulation n de la file P/P/1

```
Algorithme Résultat ()
Début
    Si occupation_max_tampon > borne_maximum alors
        Afficher("la borne maximum théorique est fausse") ;
    Sinon
        Afficher("Simulation" & n & "Occupation maximum : "
                & occupation_max_tampon) ;
    Fin si ;
Fin ;
```

FIG. 6.4 – Calcul des résultats de la simulation P/P/1

L'algorithme **Résultat** est exécuté à la fin de chaque simulation (cf. figure 6.4). Il permet de vérifier que l'occupation maximum observée est bien inférieur à la borne maximum donnée en paramètre.

Nous décrivons maintenant les interfaces des différentes procédures développées en Ada pour la mise en œuvre du simulateur de file P/P/1.

```
package Queueing_System.Pp1 is
    ...
    Procedure Simulate_Pp1 (
        nb_simulation : Natural ;
        nb_producer   : Natural ;
        sys           : System ) ;
    ...
end Queueing_System ;
```

FIG. 6.5 – Interface de la procédure *Simulate_Pp1*

La procédure **Simulate_Pp1** est l'implémentation de l'algorithme *Simulation* (cf. figure 6.5). La variable *sys* de type *System* contient l'ensemble des informations relatives à l'application temps réel modélisée (informations sur les processeurs, les tâches,...).

La procédure **Random_Project_1_N** est utilisée pour générer une file P/P/1 possédant *nb_producer* producteurs et un consommateur (cf. figure 6.6). Le type $Buffer\_Roles\_Range$ est l'intervalle de valeur autorisé pour le nombre de producteur ou de consommateur. La contrainte de débit est respectée (cf. définition 13).

La valeur de la période de la tâche consommateur est affectée par l'intermédiaire du paramètre *consumer_period*. Le paramètre $period\_producer_max$ indique la valeur maximum que peuvent avoir les périodes des tâches producteurs. *deadline_min* et *deadline_max*

```
package Queueing_System is
   ...
   Procedure Random_Project_1_N (
      nb_producer            : in Buffer_Roles_Range;
      producer_period_max    : in Natural;
      consumer_period        : in Natural;
      deadline_max           : in Natural;
      deadline_min           : in Natural;
      base_period_max        : in Natural;
      sys                    : out System);

   Procedure Random_Project_1_N_Harmonic (
      nb_producer            : in Buffer_Roles_Range;
      producer_period_max    : in Natural;
      consumer_period        : in Natural;
      deadline_max           : in Natural;
      deadline_min           : in Natural;
      sys                    : out System);
   ...
end Queueing_System;
```

FIG. 6.6 – Interface de la procédure de génération de tampons

permettent de contrôler les valeurs maximums et minimums des délais critiques. Elles sont obtenues en multipliant ces deux paramètres par la valeur de la période de la tâche concernée. (les délais critiques sont égaux aux périodes si $deadline\_max = deadline\_min = 1$). Le paramètre $base\_period\_max$ donne la taille maximum de la période d'étude des tâches. Plus la valeur est petite, plus la simulation du système est rapide.

La procédure **Random_Project_1_N_Harmonic** est identique à la procédure précédente mais ne génère que des files P/P/1 dont les périodes des tâches producteurs et consommateurs sont harmoniques (cf. figure 6.6).

6.1.2 Résultats et analyse des simulations de la file P/P/1

Dans cette section, nous présentons et analysons les résultats des simulations effectuées sur la file P/P/1.

Le simulateur génère 1000 tampons et pour chacun de ces tampons, 1000 simulations sont exécutées. Les résultats concernent :

- Les jeux de tâches harmoniques avec deux, trois et quatre producteurs sur une durée de 6 fois la période d'étude ($6.PPCM$). Les bornes maximums sur le taux d'occupation testées sont respectivement égales à 4, 6 et 8 (cf. théorème 12).

- Les jeux de tâches quelconques avec deux, trois et quatre producteurs. sur une durée égale à 20 fois la période d'étude. Les bornes maximums sur le taux d'occupation testées sont respectivement égales à 5, 7 et 9 (cf. théorème 12).

Nous observons sur les graphiques 6.7, 6.8, 6.9, 6.10, 6.11 et 6.12, le taux d'occupation du tampon lorsque les périodes des tâches sont harmoniques ou quelconques. Les ordonnées représentent l'occupation maximum du tampon observée pendant les simulations et sur l'axe des abscisses sont positionnés les indices des tampons. La borne maximum théorique sur l'occupation, calculée grâce au théorème 12, est précisée par une droite. Chaque point correspond à une simulation de la file P/P/1.

Dans chacune des simulations, le taux d'occupation ne dépasse jamais les bornes maximums proposées dans le théorème 12. Dans certain cas, le taux d'occupation est même égale à la borne maximum. Selon ce résultat, les bornes maximums théoriques que nous proposons sont non seulement nécessaires mais également suffisantes.

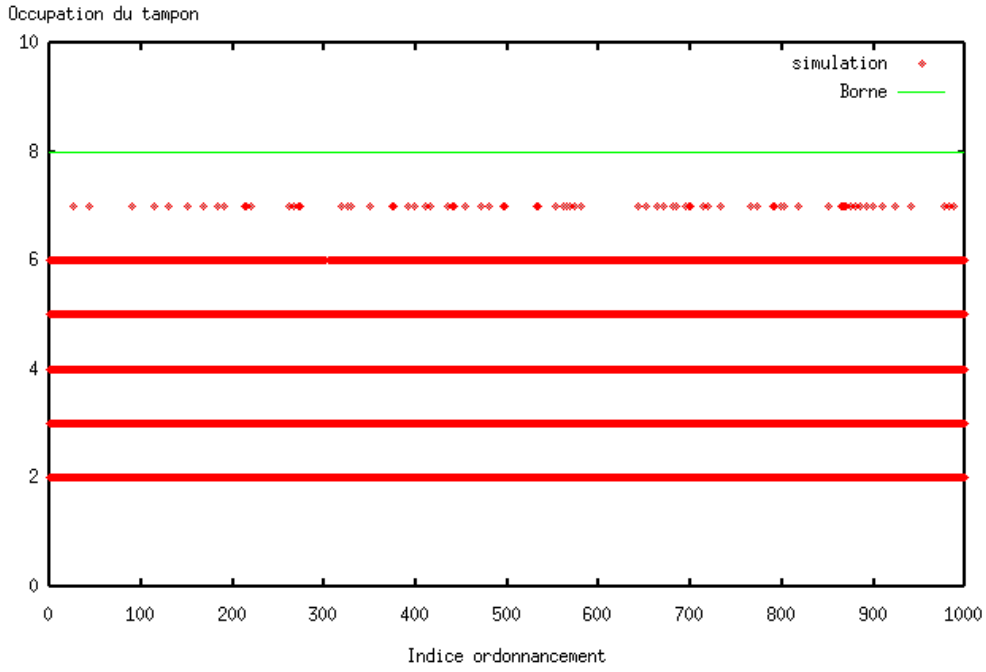

FIG. 6.7 – Système harmonique 4 producteurs/1 consommateur

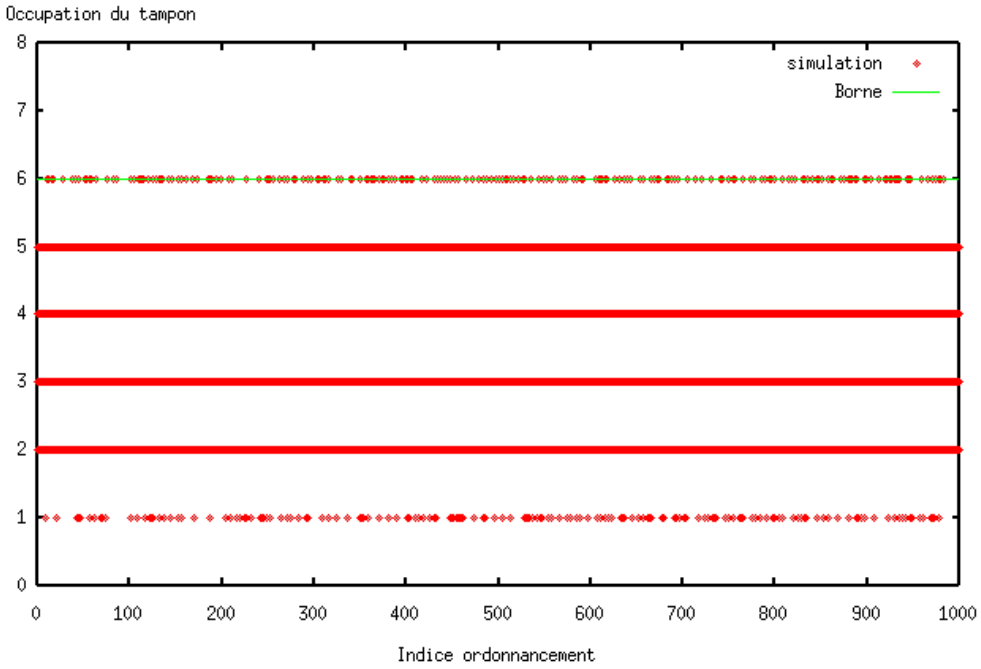

FIG. 6.8 – Système harmonique 3 producteurs/1 consommateur

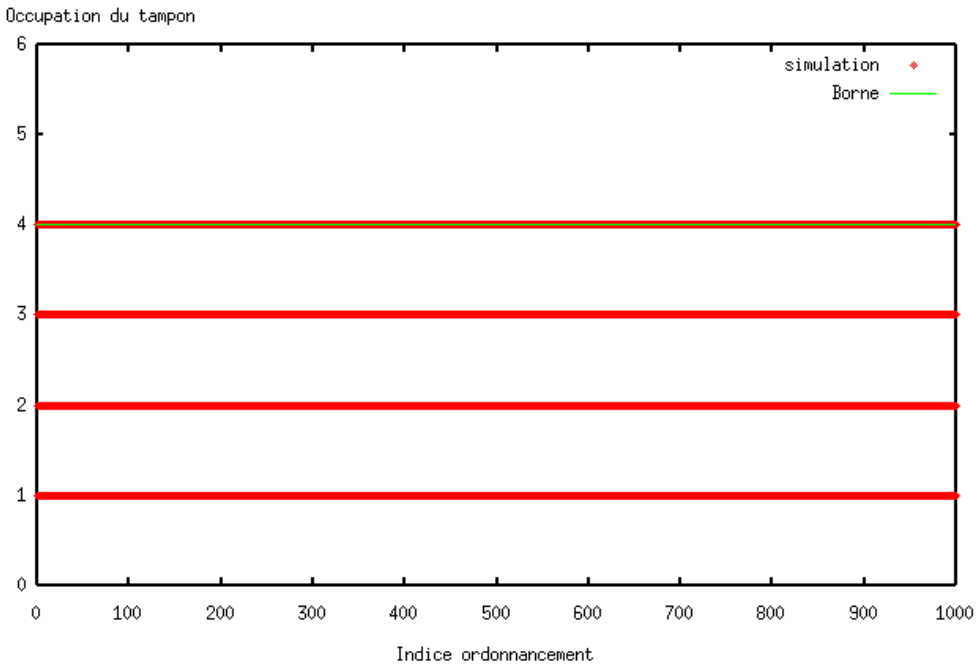

FIG. 6.9 – Système harmonique 2 producteurs/1 consommateur

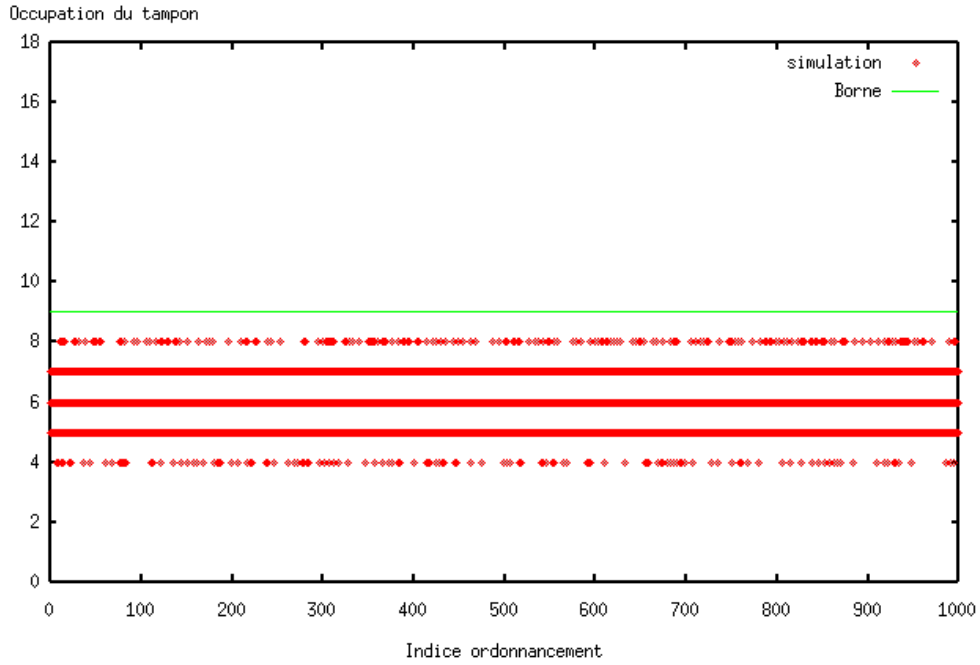

FIG. 6.10 – Système non harmonique 4 producteurs/1 consommateur

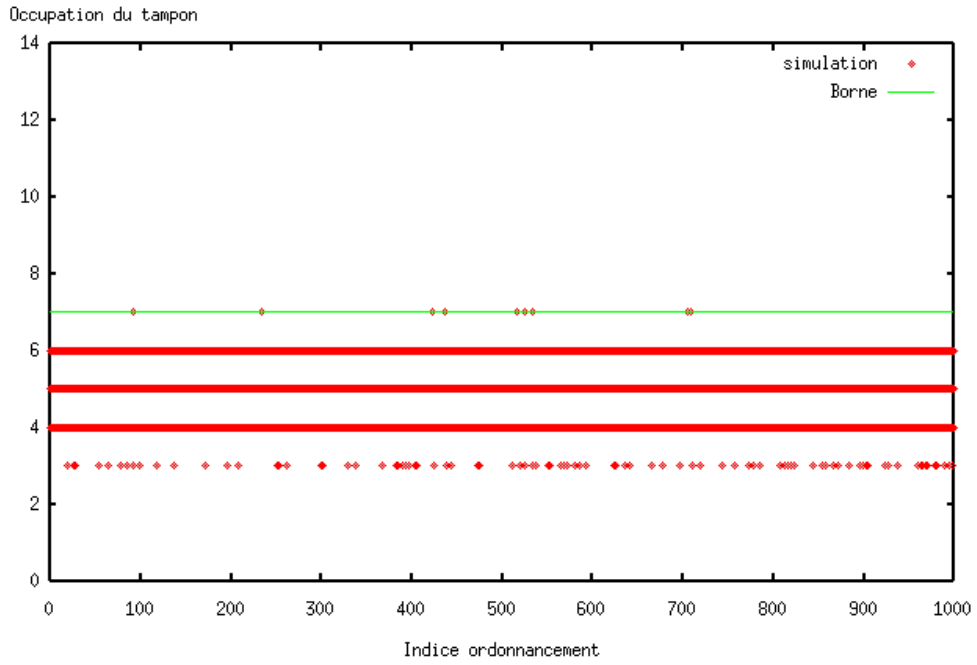

FIG. 6.11 – Système non harmonique 3 producteurs/1 consommateur

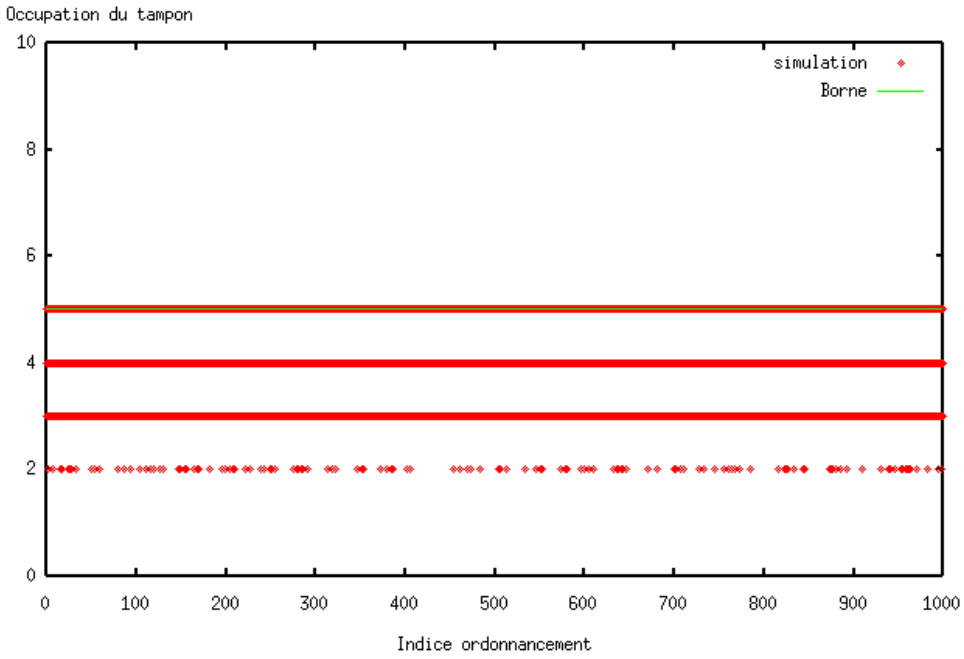

FIG. 6.12 – Système non harmonique 2 producteurs/1 consommateur

6.2 Simulation de la file M/P/1

L'objectif des simulations de la file M/P/1 est de vérifier que le temps d'attente moyen, le taux d'occupation moyen, le temps de service moyen et la variance de ce temps de service sont équivalents aux critères théoriques proposées dans le théorème 7 (cf. tableau 2.7).

Nous commençons par décrire l' algorithme du simulateur de file M/P/1 que nous avons utilisé. Les interfaces des procédures implémentant ces algorithmes sont également présentées. Puis, nous analysons les résultats de simulation obtenus.

6.2.1 Description du simulateur M/P/1

Dans cette section, nous décrivons les algorithmes utilisés pour concevoir notre simulateur de file M/P/1, ainsi que les interface des fonctions/procédures qui mettent en œuvre ces algorithmes.

Algorithme Simulation (nb_simulation, temps_simulation)
Début
– Initialisation des simulations
Initialisation ()
Pour n **allant de** 1 **à** nb_simulation **faire**
– on intialise la $n^{ième}$ simulation
Initialisation_n ()
– on exécute la $n^{ième}$ simulation
Tant que Min(date_arrivee, date_depart) $<$ temps_simulation **faire**
Si date_arrivee $<$ date_depart **alors**
Arrivée_Message () ;
Sinon
Départ_Message ()
Fin si ;
Fin tant que ;
– on affiche les résultats de la $n^{ième}$ simulation
Résultat ()
Fin pour ;
– fin des simulations
Fin ;

FIG. 6.13 – Algorithme du simulateur de file M/P/1

L'algorithme **Simulation** permet d'effectuer et d'analyser *nb_simulation* simulations d'une file M/P/1 sur une durée de *temps_simulation* unités de temps (cf. figure 6.13). L'algorithme *Simulation* est subdivisé en quatre parties :

1. Une phase d'initialisation.

2. Une phase d'initialisation spécifique à une simulation n donnée.

3. Une phase d'exécution de la simulation n.

4. Une phase d'affichage des résultats de la simulation n.

Lorsque toutes les simulations on été effectuées, soit $n = nb\_simulation$, l'algorithme se termine.

Nous détaillons maintenant chacune de ces phases.

```
Algorithme Initialisation ()
Début
    génération du taux d'arrivée λ et de la période du consommateur
    génération des r^j_cons de la tâche consommateur
Fin ;
Algorithme Initialisation_n ()
Début
    date_arrivee ← date_prochaine_arrivee(λ) ;
    date_depart ← date_arrivee + calcul_temps_de_service(μ) ;
    date_precedent ← date_arrivee ;
    occupation_moyenne ← 0 ;
    temps_attente_moyen ← 0 ;
    nb_temps_attente ← 0 ;
    nb_message ← 0 ;
    temps_de_service_moyen ← 0 ;
    Ajouter(liste_date_arrivee, date_arrivee) ;
Fin ;
```

FIG. 6.14 – Les procédures d'initialisation des simulateurs M/P/1

Les algorithmes **Initialisation** et **Initialisation_n** sont utilisés pour initialiser les variables de la simulation (cf. figure 6.14). Dans le premier, on génère les valeurs des taux d'arrivée λ et de la période de la tâche consommateur. Ces valeurs respectent la loi de conservation du débit (cf. définition 13). En effet, $\frac{1}{\mu} < P_{cons}$, soit $\frac{\lambda}{\mu} < \lambda.P_{cons}$. Or, pour chaque tampon, nous avons $\lambda.P_{cons} < 1$. En outre, nous ne considérons que les jeux de tâches dont les délais critiques sont égaux aux périodes.

Le deuxième algorithme remet à 0 les variables nécessaires à une simulation de la file M/P/1 générée. On attribue aux variables *date_arrivee* et *date_depart* les valeurs des

fonctions *Date_Prochaine_Arrivee*(λ) et *Calcul_Temps_De_Service*(μ). Ces fonctions renvoient la date de production d'un message dans le tampon et le temps du service d'un message.

Durant la simulation, nous mesurons le temps de service, la variance de ce temps de service, le temps d'attente moyen et maximum, et finalement l'occupation moyenne et maximum du tampon.

Le fonctionnement du simulateur M/P/1 est relativement proche du simulateur P/P/1. A chaque pas de simulation, l'état du système évolue : le nombre de messages augmente ou diminue selon que le prochain événement soit une arrivée aléatoire de message ou une consommation.

La production et la consommation interviennent lorsque les tâches ont terminé leur exécution. Les temps de réponse r^j_{cons} des instances j de la tâches consommateur *cons* sont générées dans l'intervalle $[0, P_{cons}]$. Les dates de départ sont calculées à partir de ces valeurs. Les délais inter-arrivées sont générées selon une loi exponentielle.

Les variables *date_arrivee* et *date_depart* sont comparées pour déterminer quel sera le prochain événement. Dés qu'un évènement a été traité, on calcul une nouvelle date d'arrivée ou de départ d'un message.

```
Algorithme Arrivée_Message ()
Début
    – calcul du taux d'occupation moyen
    –
    occupation_moyenne ← occupation_moyenne +
        nb_message * (date_arrivee - date_precedent) ;
    nb_message ← nb_message + 1 ;
    date_precedent ← date_arrivee ;
    date_arrivee ← Date_Prochaine_Arrivee(λ) ;
    – ajout de la date d'arrivée dans une liste
    –
    Ajouter(liste_date_arrivee, date_arrivee) ;
Fin ;
```

FIG. 6.15 – Arrivée d'un message dans le tampon M/P/1

L'algorithme **Arrivée_Message** est appelé lorsqu'un message arrive dans le tampon (cf. figure 6.15). Le nombre de messages présents dans le tampon est incrémenté de une unité. Des calculs intermédiaires sont effectués pour le taux d'occupation et le temps d'attente. Finalement, une nouvelle date d'arrivée de message est déterminée.

```
Algorithme Départ_Message ()
Début
    – calcul du taux d'occupation moyen
    –
    occupation_moyenne ← occupation_moyenne +
        nb_message * (date_arrivee - date_precedent) ;
    nb_message ← nb_message - 1 ;
    date_precedent ← date_depart ;

    – calcul du temps de service moyen et variance
    –
    temps_de_service ← Calcul_Temps_De_Service(μ) ;
    temps_de_service_moyen ← temps_de_service_moyen + temps_de_service ;
    variance_temps_de_service ←
        variance_temps_de_service + temps_de_service * temps_de_service ;
    nb_temps_de_service ← nb_temps_de_service + 1 ;

    – calcul temps d'attente
    –
    temps_attente ← Retirer(liste_date_arrivee, date_arrivee) ;

    – calcul de la nouvelle date de départ
    –
    Si nb_message > 0 alors
        date_depart ← date_depart + temps_de_service ;
    Sinon
        date_depart ← date_arrivee + temps_de_service ;
    Fin si ;
    – calcul du temps d'attente moyen
    –
    temps_attente ← date_depart - temps_attente ;
    nb_temps_attente ← nb_temps_attente + 1 ;
    temps_attente_moyen ← temps_attente_moyen + temps_attente ;
Fin ;
```

FIG. 6.16 – Départ d'un message du tampon M/P/1

L'algorithme **Départ_Message** est appelé lorsque le service d'un message est terminé (cf. figure 6.16). Le nombre de messages présents dans le tampon est décrémenté de une unité. Des calculs intermédiaires sont effectués pour le taux d'occupation, le temps d'attente, le temps de service et la variance sur ce temps de service. Finalement, une nouvelle date de départ de message est déterminée.

La simulation n est stoppée lorsque la date du prochain événement est supérieure à *temps_simulation*. On affiche les valeurs des critères mesurées durant la simulation et la valeur des critères théoriques (cf. théorème 7).

```
Algorithme Résultat ()
Début
    – derniers calculs
    –
    temps_service_moyen ← temps_service_moyen / nb_temps_de_service ;
    variance_temps_de_service ← variance_temps_de_service / nb_temps_de_service -
        temps_service_moyen * temps_service_moyen ;
    temps_attente_moyen ← temps_attente_moyen / nb_temps_attente ;
    occupation_moyenne ← occupation_moyenne +
        nb_message * (temps_simulation - date_precedent) ;
    occupation_moyenne ← occupation_moyenne / temps_simulation ;

    – affichage des résultats
    –
    Afficher("Temps de service moyen : " & temps_service_moyen) ;
    Afficher("Variance du temps de service : " & variance_temps_de_service) ;
    Afficher("Temps d'attente moyen : " & temps_attente_moyen) ;
    Afficher("Taux d'occupation moyen : " & occupation_moyenne) ;
Fin ;
```

FIG. 6.17 – Calcul des résultats de la simulation M/P/1

L'algorithme **Résultat** est exécuté après la fin de chaque simulation (cf. figure 6.17). Il affiche les valeurs des critères de performance que nous mesurons durant la simulation. Ces valeurs ne sont valables que lorsque le régime est permanent. Dans l'idéal, pour atteindre ce régime permanent, il faudrait que la durée de simulation soit infinie. Évidemment, appliquer une telle durée n'est pas envisageable. Nous nous contentons donc d'utiliser des valeurs suffisamment grandes pour lesquelles les critères mesurés n'évoluent plus.

Nous décrivons maintenant les interfaces des différentes procédures développées en Ada pour les simulation de la file M/P/1.

```
package Queueing_System is
   ...
   Procedure Simulate (
      a_queueing_system  : in out Generic_Queueing_System'Class;
      simulation_time    : in Double;
      nb_simulation      : in Natural);

   Procedure Random_Qs_Project (
      a_queueing_system  : in out Generic_Queueing_System'Class;
      utilization        : in Double;
      seed               : in Generator;
      mu_max             : in Natural := 4000);

   Function Get_Rand_Parameter (
      min   : in Double;
      max   : in Double;
      seed  : in Generator )
    return Double;

   Function Get_Exponential_Time (
      average_rate  : in Double;
      seed          : in Generator )
    return Double;
   ...
end Queueing_System;
```

FIG. 6.18 – Interface des procédures/fonctions nécessaires au simulateur

La classe **Queueing_System** fournit des fonctions/procédures nécessaires pour simuler les files d'attente M/P/1.

Le procédure **Simulate** implémente l'algorithme de simulation de file d'attente (cf. figure 6.18). Nous nous en servons pour notre file M/P/1.

La procédure **Random_Qs_Project** génère les valeurs λ et μ d'une file d'attente (cf. figure 6.18). Le rapport $\frac{\lambda}{\mu}$ est égal à la valeur du paramètre *utilization* et $\frac{1}{\mu}$ est inférieur à la valeur de mu_max.

La fonction **Get_Rand_Parameter** retourne un nombre aléatoire, flottant, compris entre les valeurs en entrée *min* et *max* (cf. figure 6.18). Les valeurs rendues par cette fonction sont uniformément répartis dans l'intervalle $[min, max]$. La variable *seed* sert à initialiser le générateur de nombre aléatoire disponible dans le compilateur Gnat/Ada.

La fonction **Get_Exponential_Time** est utilisée pour générer un délai inter-arrivée ($average\_rate = \frac{1}{\lambda}$) ou un temps de service exponentiel ($average\_rate = \frac{1}{\mu}$) (cf. figure 6.18).

```
package body Queueing_System.Mm1 is
   ...
   Procedure Next_Customer_Arrival (
      a_queueing_system  : in out Mm1_Queueing_System;
      new_arrival        : in out Double;
      seed               : in Generator;
      old_arrival        : in Double ) is
   begin
      new_arrival := old_arrival +
         Get_Exponential_Time(a_queueing_system.
               arrival_rate, seed);
   end Next_Customer_Arrival;

   Procedure Next_Customer_Departure (
      a_queueing_system  : in out Mm1_Queueing_System;
      new_departure      : in out Double;
      seed               : in Generator;
      current_time       : in Double ) is
   begin
      new_departure := current_time +
         Get_Exponential_Time(a_queueing_system.
               service_rate, seed);
   end Next_Customer_Departure;
   ...
end Queueing_System.Mm1;
```

FIG. 6.19 – Procédure *Next_Customer_Arrival* et *Next_Customer_Departure*

Finalement, pour simuler une file d'attente, il faut redéfinir les procédures **Next_Customer_Arrival** et **Next_Customer_Departure**. Ces procédures retournent la date de la prochaine arrivée et du prochain départ d'un message de la file d'attente. Nous donnons un exemple d'implémentation de ces procédures pour la file M/M/1 sur la figure 6.19.

6.2.2 Résultats et analyse des simulations de la file M/P/1

Dans cette section, nous présentons et analysons les résultats des simulations effectuées sur la file M/P/1.

Le simulateur génère 900 tampons partagés par n flux d'arrivée de messages et une tâche consommateur périodique. Le taux d'utilisation de la file varie de 0.01 à 0.90. Les durées de simulation sont de 20 000 000 000 unités de temps. La période des tâches est inférieure à 1000 unités de temps.

Les résultats de simulation concernent les jeux de tâches où :

- La tâche consommateur possède une priorité moyenne ou faible. Une séquence de 20 temps de réponses uniformément générés dans l'intervalle $[0, P_{cons}]$ est construite. Ils permettent de simuler les interférences des tâches de plus forte priorité.

- La tâche consommateur possède la priorité la plus forte. Il n'y a aucune interférence des autre tâches. Dans ce cas, le temps de réponse des instances de la tâche consommateur est égal à C_{cons}.

L'objectif des courbes présentées dans ce chapitre est d'évaluer les résultats proposés dans cette thèse.

Nous présentons ci-après les courbes concernant le taux d'occupation, le temps d'attente, le temps de service et la variance sur ce temps de service. L'axe des abscisses correspond au taux d'utilisation de la file d'attente M/P/1. L'axe des ordonnées, quant à lui, correspond à la différence entre la valeur du critère théorique étudié et la valeur de ce même critère mesurée au cours de la simulation pour la file M/P/1.

Les courbes des figures 6.20 et 6.21 représentent la différence en pourcentage entre le temps de service moyen, resp. la variance sur le temps de service moyen, proposé dans le théorème 7 et les valeurs mesurées au cours de la simulation de la file M/P/1.

Sur la première figure, la tâche consommateur possède une forte priorité. Nous observons que le temps de service théorique est relativement proche des résultats de simulation. La différence varie entre 0 et 5%. Par contre, la variance théorique donne de bons résultats uniquement lorsque le taux d'utilisation du serveur est faible. Plus ce taux est important, moins notre variance théorique est proche de la variance de la file M/P/1 simulée. La différence varie entre 0 et 60%.

Sur la deuxième figure, la tâche consommateur possède une priorité faible. Dans ce cas, le temps de service théorique est légèrement moins bon que précédemment. La différence varie entre 0 et 10%. Quant à la variance théorique, elle est proche des résultats de simulation lorsque le taux d'utilisation du serveur de la file M/P/1 tend vers 1. Lorsque ce taux tend vers 0, les résultats se dégradent. La différence varie entre 0 et 43%.

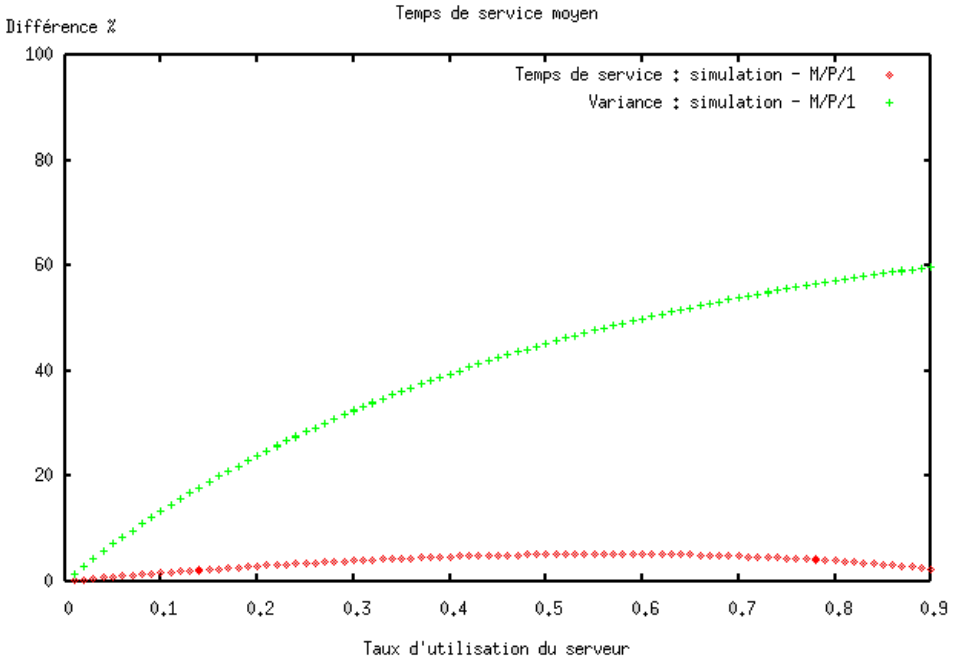

FIG. 6.20 – Temps de service et variance (consommateur à forte priorité)

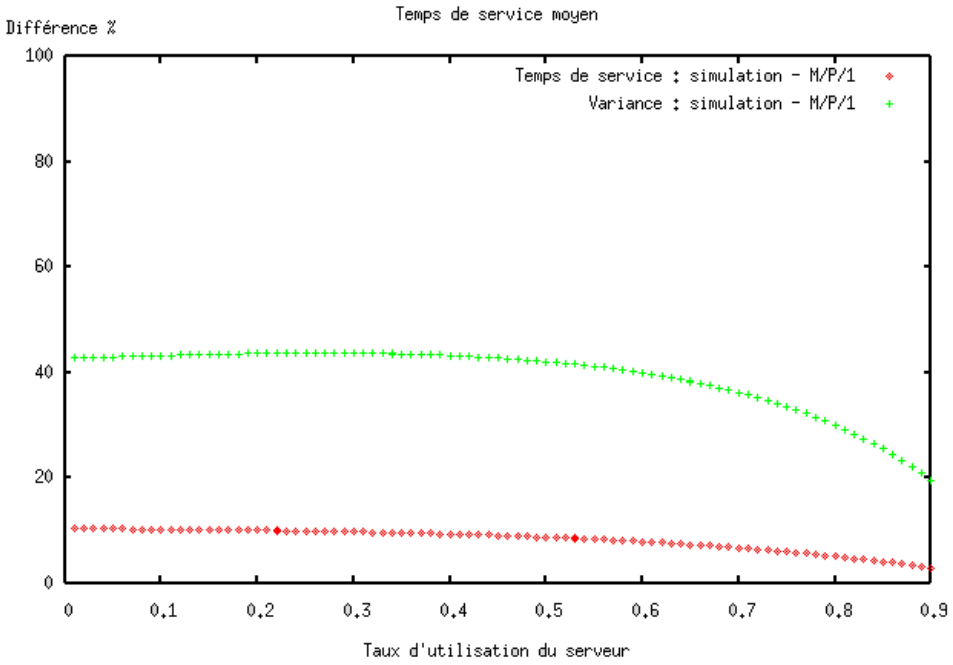

FIG. 6.21 – Temps de service et variance (consommateur à faible priorité)

D'autre part, nous proposons de comparer les valeurs du taux d'occupation et du temps d'attente mesurées sur la simulation de la file M/P/1 avec les critères théoriques des files M/G/1, M/M/1 et M/D/1. Ces critères nécessitent les valeurs du temps de service moyen et éventuellement de la variance de ce temps de service moyen.

Afin de calibrer notre simulateur, nous affichons la différence entre la valeur de critères mesurés au cours de la simulation et la valeur des critères que nous désignons par le terme "semi-théorique". La valeur de ces critères est obtenue en appliquant aux critères de la file d'attente M/G/1 le temps de service moyen et la variance sur ce temps de service issus de la simulation. La qualité de la simulation est inversement proportionnelle à cette différence.

Nous indiquons systématiquement la différence entre la valeur de critères mesurés au cours de la simulation et la valeur des critères de la file M/G/1 auxquels nous appliquons les équations de W_s et σ_s^2 proposées dans le théorème 7. La courbe qui en résulte permet d'évaluer nos propositions.

Nous affichons également la différence entre la valeur de critères mesurés au cours de la simulation et la valeur des critères des files M/M/1 et M/D/1 auxquels nous appliquons différentes valeurs du temps de service moyen (ces files ne nécessitent que ce paramètre) :

- Sur les figures 6.22, 6.23, 6.28 et 6.29, le temps de service moyen est égale à la période de la tâche consommateur.

- Sur les figures 6.24, 6.25, 6.30 et 6.31, le temps de service moyen est égale au temps de service mesuré sur la simulation de la file M/P/1.

- Sur les figures 6.26, 6.27, 6.32 et 6.33, le temps de service moyen est égale à un temps de service proposé dans le théorème 7.

Nous proposons les courbes des temps d'attente et du taux d'occupation, dans un premier temps, lorsque tâche consommateur a une la priorité la plus forte, puis lorsque tâche consommateur a une forte moins importante.

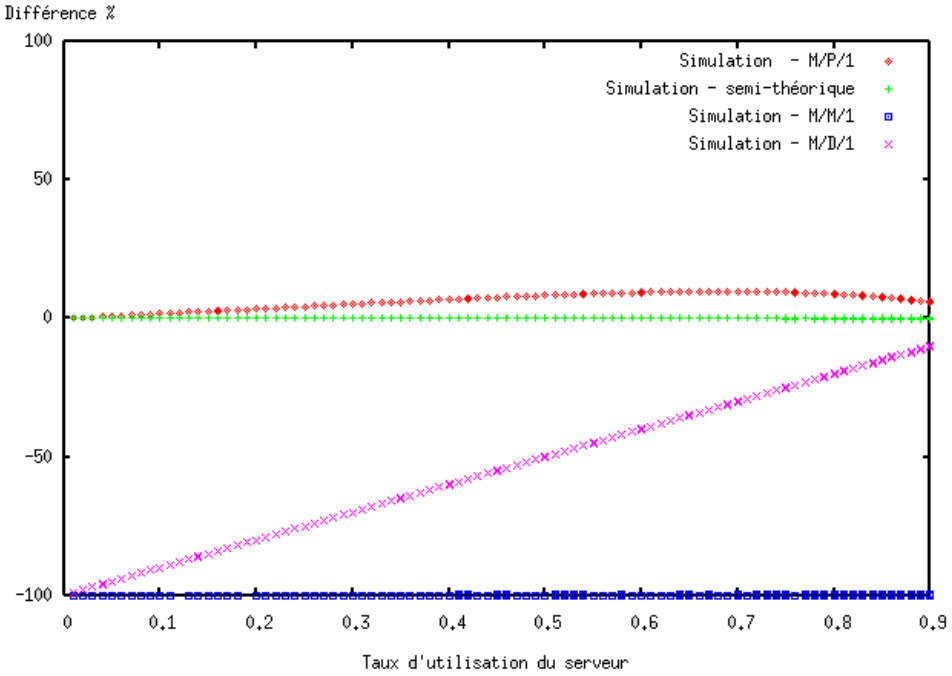

FIG. 6.22 – Taux d'occupation lorsque $W_s = P_{cons}$ et consommateur à forte priorité

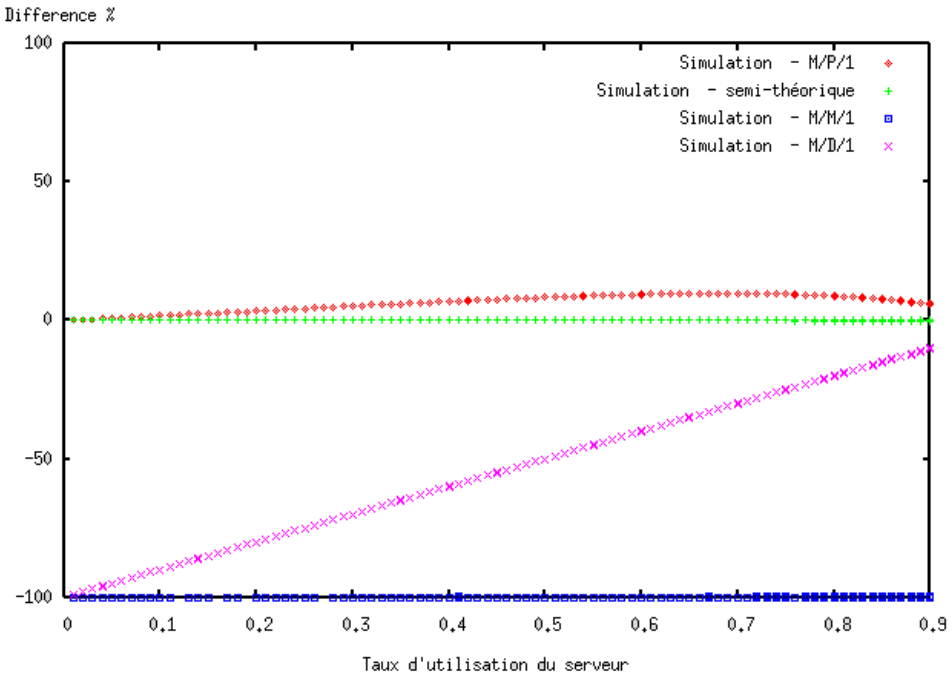

FIG. 6.23 – Temps d'attente lorsque $W_s = P_{cons}$ et consommateur à forte priorité

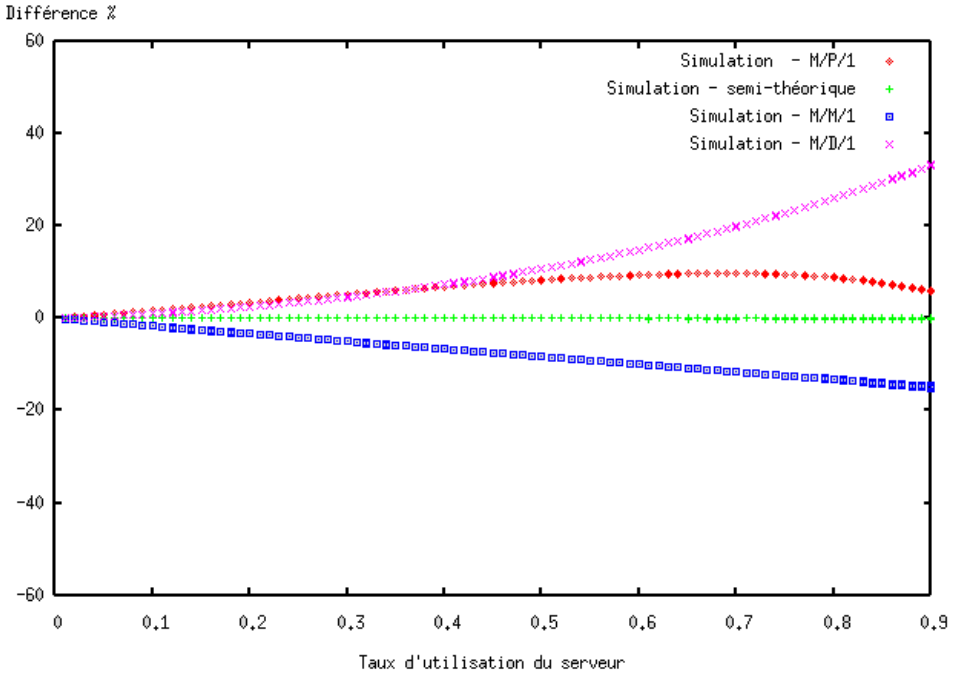

FIG. 6.24 – Taux d'occupation pour W_s de simulation et consommateur à forte priorité

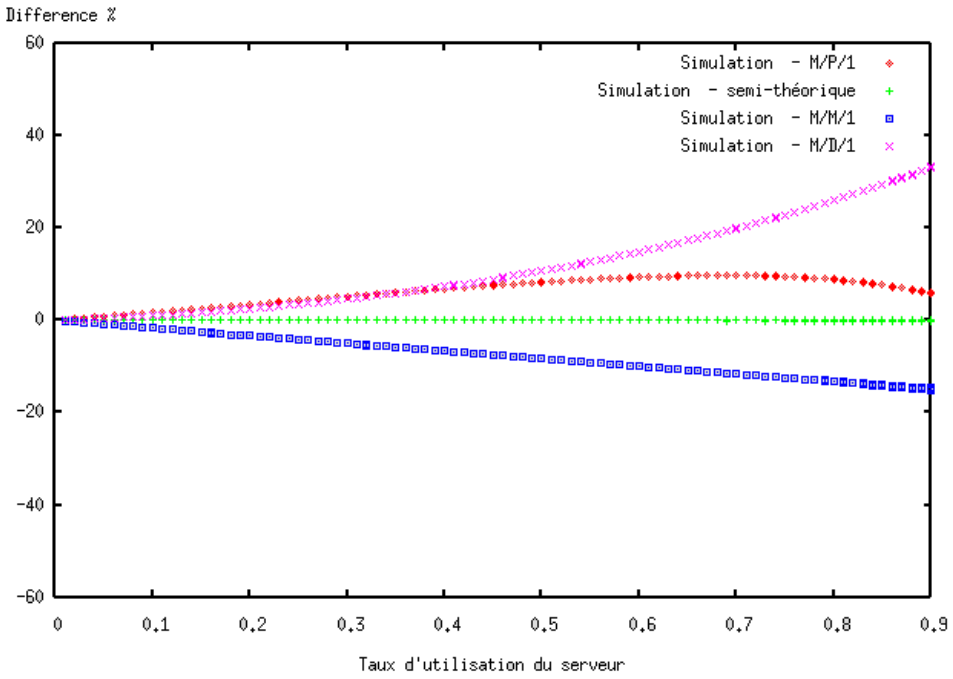

FIG. 6.25 – Temps d'attente pour W_s de simulation et consommateur à forte priorité

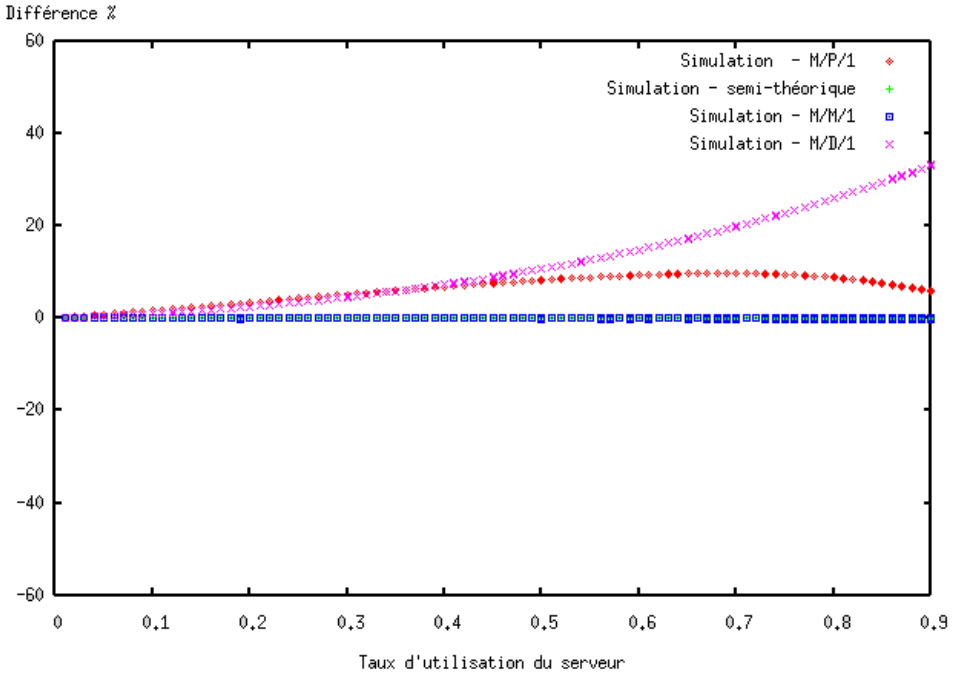

FIG. 6.26 – Taux d'occupation pour W_s théorique et consommateur à forte priorité

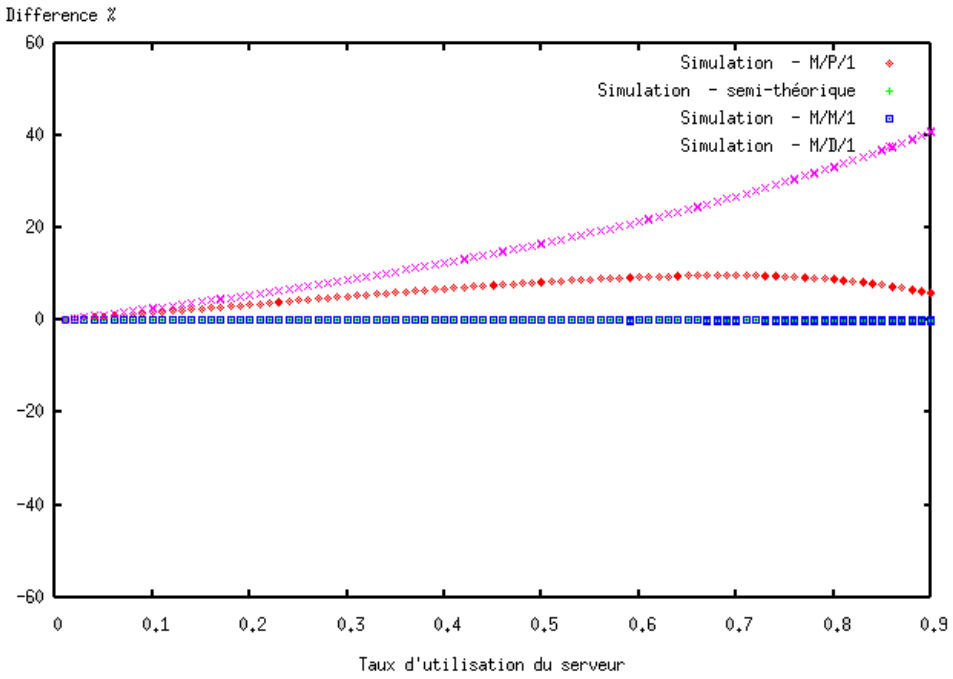

FIG. 6.27 – Temps d'attente pour W_s théorique et consommateur à forte priorité

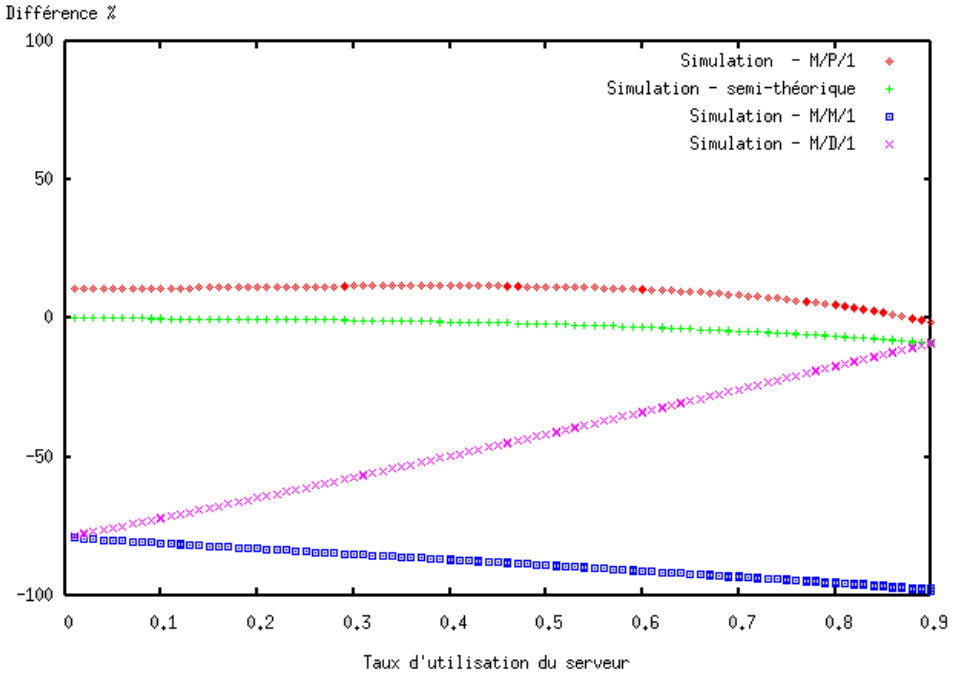

FIG. 6.28 – Taux d'occupation lorsque $W_s = P_{cons}$ et consommateur à priorité faible

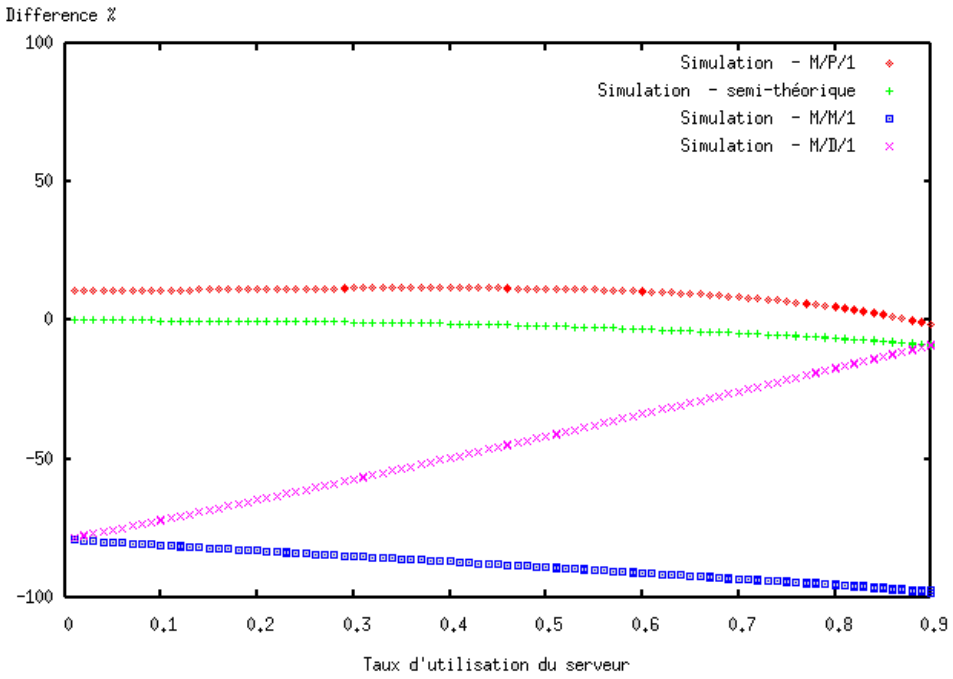

FIG. 6.29 – Temps d'attente lorsque $W_s = P_{cons}$ et consommateur à priorité faible

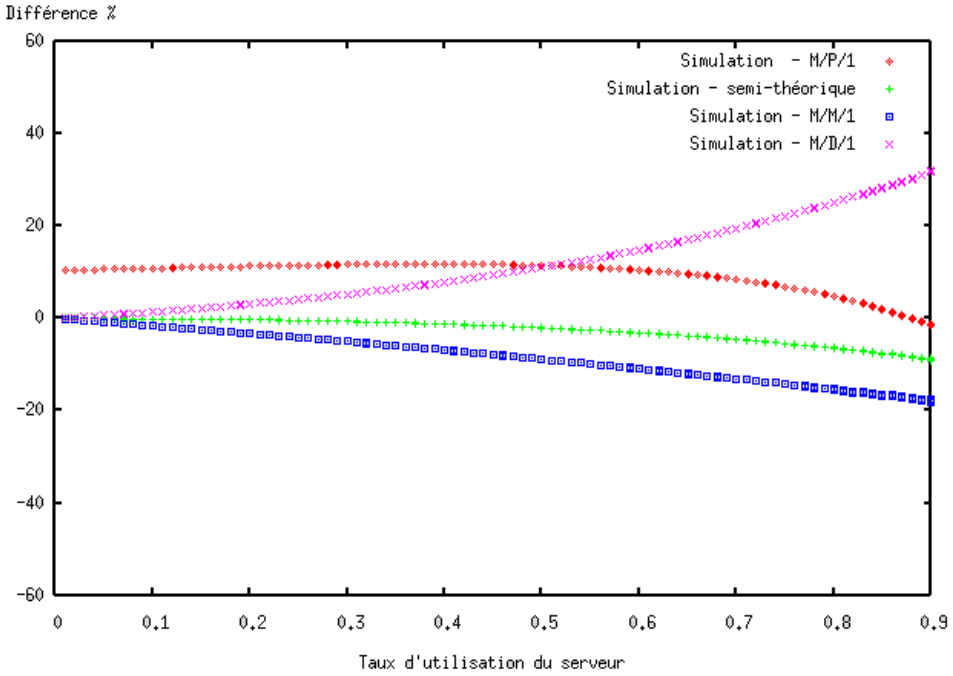

FIG. 6.30 – Taux d'occupation pour W_s de simulation et consommateur à priorité faible

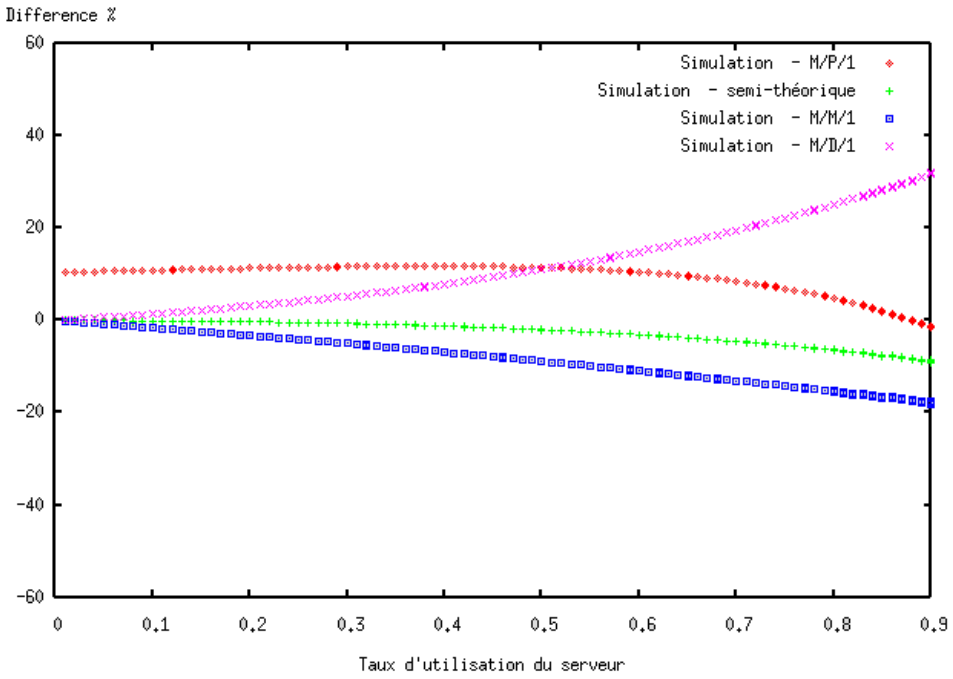

FIG. 6.31 – Temps d'attente pour W_s de simulation et consommateur à priorité faible

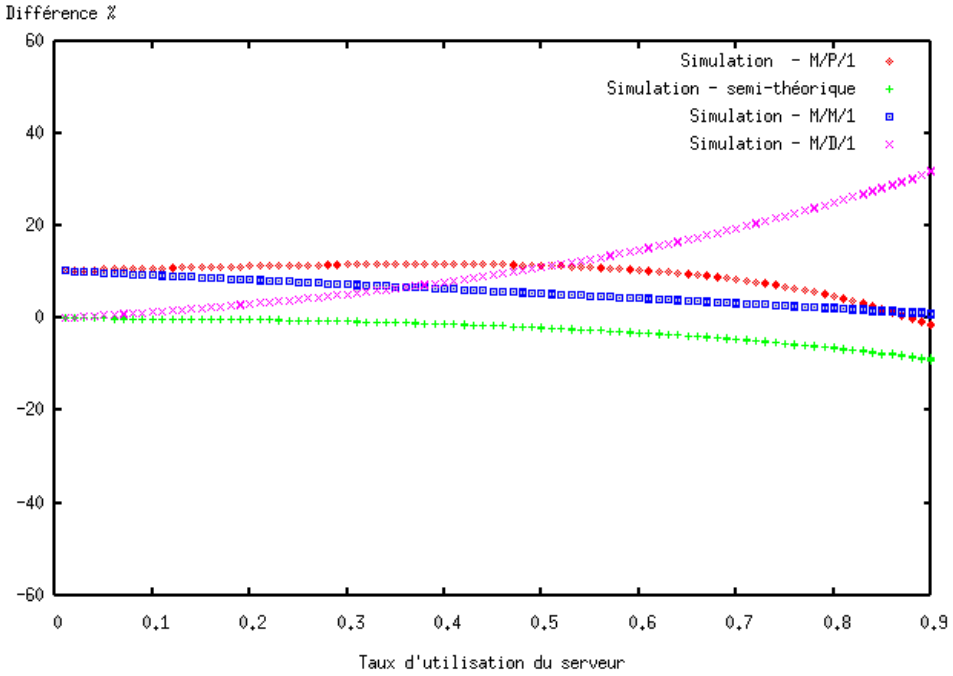

FIG. 6.32 – Taux d'occupation pour W_s théorique et consommateur à priorité faible

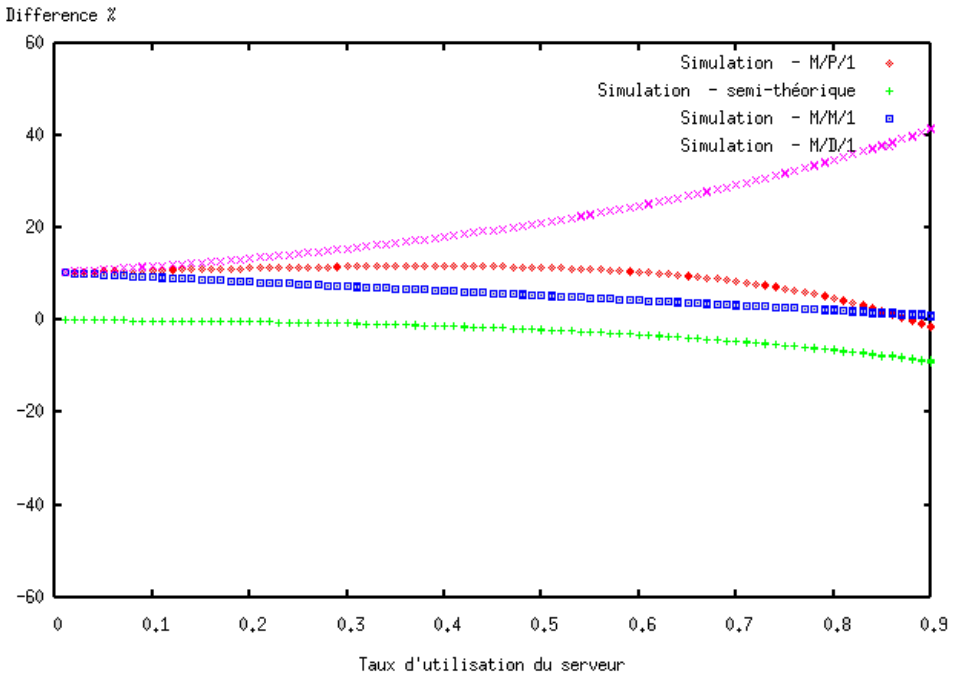

FIG. 6.33 – Temps d'attente pour W_s théorique et consommateur à priorité faible

Etant donné que le taux d'occupation et le temps d'attente moyen sont liés par la loi de LITTLE (cf. théorème 1), les observations que nous faisons sont valables, dans des conditions données, quelque soit le critère considéré.

Lorsque le consommateur possède une forte priorité, sur l'ensemble des figures, les critères semi-théoriques de la file M/G/1[1] sont quasiment identiques aux résultats de simulation de la file M/P/1. Cela signifie que, sur la durée de simulation choisie, la file M/P/1 a bien atteint sont état stable.

Lorsque le consommateur possède une faible priorité, sur l'ensemble des figures, les critères semi-théoriques de la file M/G/1 ne pas tout à fait identiques aux résultats de simulation de la file M/P/1. En particulier, lorsque le taux d'utilisation du serveur de la file M/P/1 tend vers 1, la différence est de l'ordre de 10%. La file M/P/1, n'a pas atteint son état stable. Malgré une augmentation de la durée de simulation, nous ne sommes pas parvenu à atteindre cet état.

En dépit des résultats de la variance, les critères théoriques de la file M/P/1 sont proches des résultats de simulation. La différence varie entre 0 et 10% quelque soit la priorité du consommateur. Ces critères théoriques sont en fait les critères de la file M/G/1 auxquels sont appliqués les W_s et σ_s^2 proposés dans le théorème 7.

W / L	Forte priorité	Priorité moyenne/faible
M/P/1	0 à 10%	0 à 10%
M/M/1 avec $W_s = P_{cons}$	100%	75 à 100%
M/M/1 avec W_s simulation	0 à 19%	0 à 19%
M/M/1 avec W_s théorique	0%	0 à 10%
M/D/1 avec $W_s = P_{cons}$	100 à 5%	75 à 5%
M/D/1 avec W_s simulation	0 à 37%	0 à 37%
M/D/1 avec W_s théorique	0 à 37%	0 à 37%

TAB. 6.1 – Récapitulatif des résultats de simulation

Une synthèse des résultats de simulation de la file M/P/1 est donnée dans le tableau 6.1. Dans chaque case se trouve la différence maximum, en pourcentage, entre la valeur issue de la simulation et la valeur théorique d'un critère de performance donné.

Ces résultats permettent d'évaluer les différents critères théoriques pour l'analyse d'une file d'attente M/P/1. Tout d'abords, seuls les critères basés sur notre temps de service et sur la variance de ce temps de service renvoient des valeurs proches des résultats de simulation. On ne peut se contenter d'affecter au temps de service la période de la tâche consommateur.

[1] on applique le temps de service et la variance sur ce temps de service mesurés sur la simulation

D'après les résultats de simulation, les critères de performance de la file M/D/1 ne sont pas adaptés à la file M/P/1. En effet, ces critères sont les critères de la file M/G/1 auxquels on affecte une variance nulle. Or, nous observons sur les figures 6.24 et 6.25 que lorsque le taux d'utilisation de la file tend vers 1, la variance a un impact de plus en plus important.

Les meilleurs résultats sont donnés par les critères théoriques des files d'attente M/G/1 et M/M/1. Etant donné que la file M/P/1 est un cas particulier de la file M/G/1, il est naturel que les critères de cette dernière sont bien adaptés. Il est plus étonnant que les critères de la file M/M/1 renvoient des valeurs aussi bonne voire meilleurs que les critères de la file M/G/1. Les résultats sont même quasiment parfait lorsque la tâche consommateur possède une forte priorité.

Nous émettons quelques hypothèses sur les raisons de l'efficacité des critères de la file M/M/1.

Nous savons que le temps de service dépend des arrivées poissonniennes. Nous considérons dans le calcul du temps de service et de la variance que l'influence des dates d'arrivées diminue lorsque le taux d'utilisation du serveur tend vers 1.

D'après les figures 6.20 et 6.20, nous constatons qu'en ce qui concerne la variance, il existe des conditions dans lesquelles cette considération n'est pas avérée : lorsque ρ_{mp1} tend vers 1 et que la tâche consommateur a une forte priorité.

Pourtant, la variance théorique du temps de service de la file M/M/1, soit $\frac{1}{\mu^2}$, est différente de la variance provenant de la simulation : la variance théorique est toujours largement supérieure. Par conséquent, à partir d'une certaine valeur ou dans un intervalle de valeur la précision de la variance aurait moins d'influence sur la valeur du taux d'occupation et du temps d'attente.

6.3 Conclusion

L'objectif de ce chapitre est de vérifier la qualité de nos résultats. Pour cela, nous avons développé des simulateurs basés sur les techniques de simulation à événement discret. Nous en faisons une description détaillé.

Nous présentons les conditions dans lesquelles se sont déroulées les simulations des files d'attente P/P/1 et M/P/1. Pour la file P/P/1, le taux d'occupation mesuré lors des simulations n'excède jamais les bornes maximums théoriques proposées. En ce qui concerne la file d'attente M/P/1, les critères des files d'attente M/G/1 et M/M/1 peuvent être utilisés. La différence entre les critères théoriques et les valeurs mesurées est de l'ordre de 0% à 10% selon la priorité de la tâche consommateur et le taux d'arrivée des messages. L'emploi de la file M/M/1 permet en outre de calculer aisément la probabilité de non débordement de la file M/P/1.

Dans le cas de la file M/P/1, les résultats devrait pouvoir être améliorés de différentes manières. Tout d'abords, nous pouvons affiner les approximations des équations du temps de service moyen et de la variance sur ce temps de service. De plus, il serait intéressant d'appliquer, à ces mêmes équations, une régression linéaire d'un degré supérieur, voire d'employer une régression non-linéaire.

Chapitre 7

Contributions à l'outil Cheddar

Le projet cheddar a été mené afin de fournir un outil d'étude des systèmes temps réel ouvert et libre. Cheddar a été développé pour remplir les objectifs suivants :

- Etre utilisé dans un cadre pédagogique, notamment pour des travaux pratiques.
- Servir d'outil de prototypage d'applications temps réel.
- Aider aux travaux de recherche sur l'ordonnancement temps réel. Pour cela, Cheddar implémente la plupart des résultats présentés dans le Chapitre 2.

Cheddar intègre la plupart des résultats classiques de l'ordonnancement temps réel des systèmes mono-processeurs et répartis (cf. chapitre 2). Les tests de faisabilité proposés concernent les ordonnanceurs et les types de tâches les plus courant. En outre, l'utilisateur peut modifier la manière dont l'ordonnanceur fonctionne et dont les tâches sont activées afin d'analyser et de simuler des applications plus spécifiques.

Cheddar fournit un ensemble d'outils pour l'oronnancment temps réel. Ces outils sont développés en Ada et peuvent être aisément connectés à d'autres programmes. (des simulateurs, des services de supervision de systèmes d'exploitation, des "CASE tools", ...). L'interface graphique de Cheddar est développée en GtkAda. Cheddar est disponible pour les systèmes d'exploitations Solaris, Linux et Windows mais il possible de le porter sur toutes les plates-formes supportant Ada et GtkAda. Cheddar est distribué sous la licence GNU (General Public License).

Nous proposons d'étendre les fonctionnalités de Cheddar aux applications temps réel comprenant des tampons. Des fonctions d'édition de telles application ont donc été ajoutées. En outre, le moteur de simulation a été modifié pour prendre en compte l'influence éventuelle des tampons sur l'ordonnancement des tâches. Finalement, des outils d'analyse des tampons ont été implémentés.

Ce chapitre est organisé en trois parties :

- La section 7.1 présente l'ensemble des fonctionnalités disponibles dans Cheddar telles que la description d'une application temps réel, l'analyse de la faisabilité de ces applications ou les fonctionnalités avancées qui permettent de paramétrer le moteur de simulation de Cheddar (l'ordonnanceur, les règle d'activation des tâches,...).
- Dans la section 7.2, nous décrivons nos contributions à Cheddar.
- Finalement, nous concluons.

7.1 Description des fonctionnalités de Cheddar

Nous décrivons dans cette section les différentes fonctionnalités qu'il propose allant de l'édition du système temps réel à son analyse en passant par les outils de simulation.

7.1.1 Modèle d'un système temps réel dans cheddar

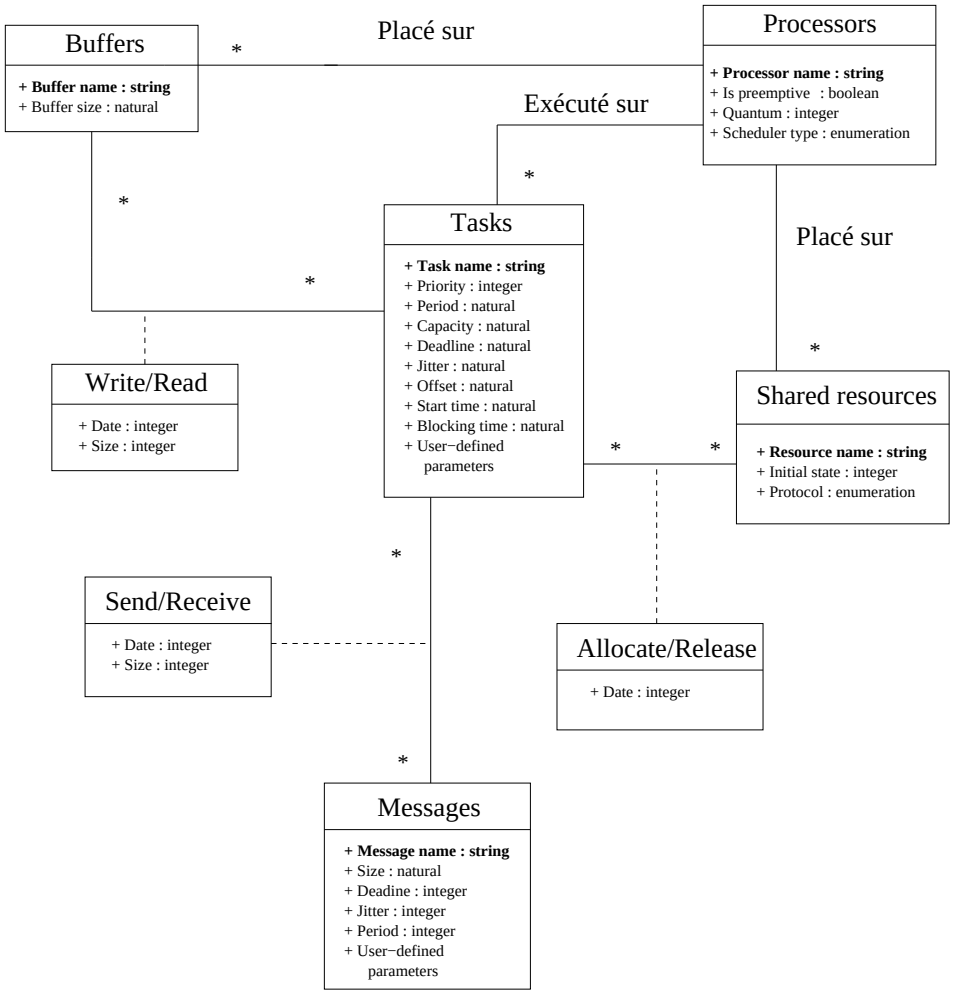

FIG. 7.1 – Diagramme UML d'une application modélisée avec Cheddar

Le diagramme UML[1] de la figure 7.1 résume les différents éléments qui constituent un système temps réel dans Cheddar ainsi que les relations entre ces éléments.

Une application temps réel est au moins composée d'un ou plusieurs processeurs et d'un ensemble de tâches. Des dépendances entre tâches, telles que le partage de ressource, peuvent également être définies. L'ensemble de ces informations constitue un projet Cheddar. Le projet peut être est sauvegardé dans un fichier au format XML[2].

Un processeur est défini par les paramètres suivants :

[1] Unified Modeling Language
[2] Extensible Markup Language

- Le nom du processeur.

- Un type d'ordonnanceur choisi parmi les ordonnanceurs suivants : Earliest Deadline First, Least Laxity First, Rate Monotonic, Deadline Monotonic, POSIX 1003.b (Les politiques SCHED_RR, SCHED_FIFO and SCHED_OTHERS sont supportées) ou les ordonnanceurs définis par l'utilisateur.

- Une variable indiquant si l'ordonnanceur est préemptif ou non (par défaut, l'ordonnanceur est préemptif).

- Le quantum de temps associé à l'ordonnanceur. Cette information est utile lorsque des tâches du système ont la même priorité (fixe ou dynamique). En effet, l'ordonnanceur doit choisir la manières de partager le processeur entre ces tâches. Le quantum est une borne sur le l'intervalle de temps pendant lequel une tâche est exécuté sur le processeur. Si la valeur du quantum est 0, il n'y a pas de borne. La valeur de ce paramètre est également utilisée dans la politique "round robin" (ou SCHED_RR) d'un ordonnanceur POSIX 1003.b et pour les ordonnanceurs paramétrables.

- Un nom de fichier. Ce fichier contient le code d'un ordonnanceur éventuellement défini par un utilisateur.

Une tâche est définie par les paramètres suivants :

- Une tâche est au moins caractérisée par un nom unique, une capacité et un processeur sur lequel elle est exécutée. Les autres paramètres, tels que la date de première activation, la gigue ou le délai critique, sont optionnels mais sont requis pour certains ordonnanceurs .

- Son type. Il décrit la manière dont la tâche est activée au cours du temps. Différents types de tâches sont disponibles dans Cheddar : le type apériodique (la tâche est activée une seule fois), le type périodique (la tâche est activée cycliquement, le délai entre deux activations successives est constant), le type poisson (la tâche est activée cycliquement, le délai entre deux activations successives suit une loi exponentielle) et le type paramétrable (les activations de la tâche sont définies par l'utilisateur).

- Une période. Elle représente le délai constant, resp. moyen, entre deux activations d'un tâche périodique, resp. poisson.

- Une priorité et une politique. Ces paramètres sont utilisés pour l'ordonnanceur Highiest Priority First/POSIX 1003.b. Les politiques disponibles sont SCHED_RR, SCHED_FIFO ou SCHED_OTHERS. Elles décrivent la manière dont les tâches de même niveau de priorité sont gérées.

- Un temps de blocage sur une ressource partagée. La valeur peut être affectée par l'utilisateur ou calculée automatiquement par Cheddar. Le calcul automatique nécessite d'indiquer le protocole d'accès à cette ressource.

- Une règle d'activation qui définie la manière dont la tâche est activée (uniquement utilisé pour les tâches paramétrables).

- Une graine pour initialiser le générateur aléatoire de date d'activation d'une tâche poisson.

Cheddar permet également de définir des dépendances entre les tâches. Les tâches peuvent être liées par des contraintes de précédence ou partager des ressources (cf. section 2.2.2).

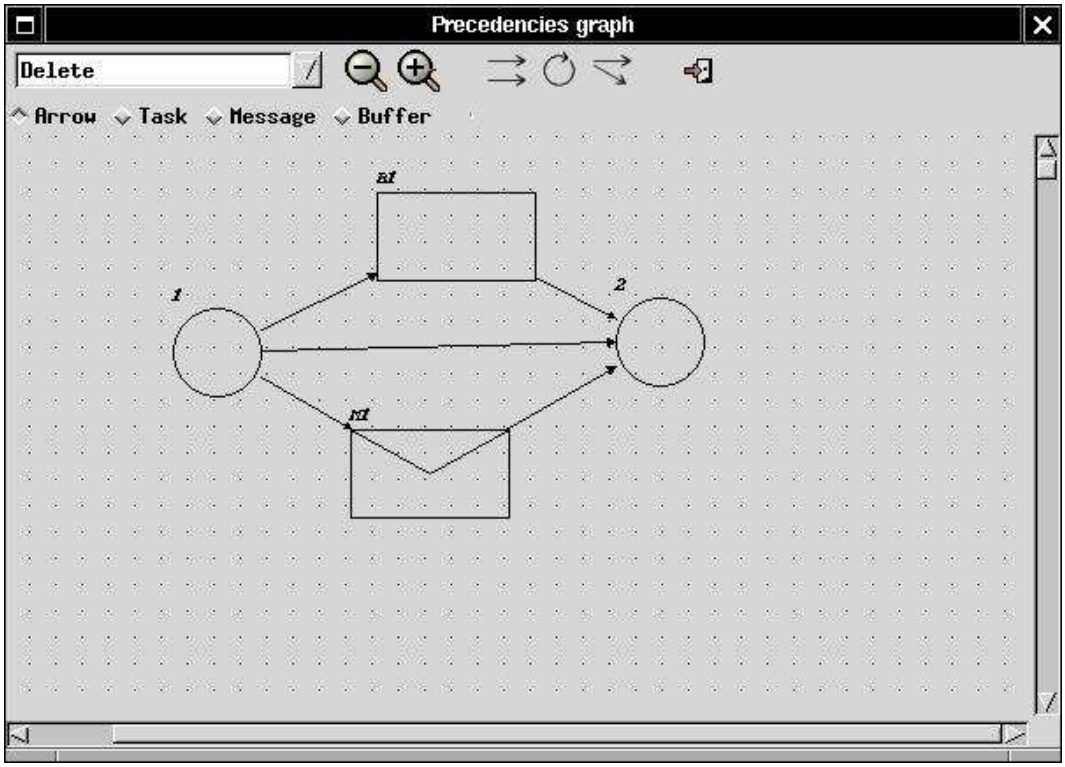

FIG. 7.2 – Définition d'une contrainte de précédence entre tâches

Une contrainte de précédence définit un ordre sur l'exécution des tâches d'un système temps réel. Un outil graphique permet d'éditer ces dépendances (cf. figure 7.2) : les cercles correspondent aux tâches, les rectangles aux messages et les enveloppes aux tampons. Lorsque la fenêtre est vide, il n'y a aucune contrainte de précédence.

Il est également possible de créer des ressources partagées par plusieurs tâches. Ces ressources peuvent être vues comme des sémaphores. Les tâches qui tentent d'accéder à un sémaphore déjà alloué sont bloquées. Les ressources partagées sont définies par les paramètres suivants :

- Un nom unique.
- Une valeur initiale (équivalent à la valeur initiale d'un sémaphore). Si cette valeur est égale à 0, resp. 1, la ressource est utilisée, resp. libre.
- Un protocole d'accès. Il est possible de choisir parmi les protocoles PCP et PIP. Dans ce cas, les priorités des tâches peuvent évoluer dans le temps. On peut également indiqué qu'aucun protocole ne gère l'accès à la ressource.
- Le nom du processeur hôte.
- Finalement, il faut préciser le nom des tâches qui partagent la ressource ainsi que les dates de début et de fin, relative à la date d'activation, de la section critique.

7.1.2 Outils d'analyse et de simulation

Lorsque les différents éléments de l'application ont été édités dans Cheddar, nous pouvons appliquer les outils d'analyse et de simulation. Trois types d'outils sont disponibles dans Cheddar : les outils de simulation, les outils de vérification de la faisabilité et les outils paramétrables.

7.1.2.1 Les outils de simulation

Les outils de simulation servent à extraire des informations à partir d'une simulation du système temps réel étudié. Les informations recueillies permettent, par exemple, de vérifier que les contraintes temporelles des tâches sont bien respectées.

Il est également possible de prendre en compte les dépendances entre tâches telles que les contraintes de précédence ou le partage de ressources.

Evidement, le résultat de la vérification n'est valable que pour l'ordonnancement généré. En outre, la durée de la génération dépends des caractéristiques du jeux de tâches et peut se révéler relativement longue.

7.1.2.2 Les outils de vérification de la faisabilité

Les outils de vérification de la faisabilité traitent les informations données à l'édition de l'application sans ordonnancer le jeu de tâches. Ils sont basés sur les résultats classiques de l'ordonnancement temps réel (cf. chapitre 2).

La faisabilité d'un jeu de tâches peut être vérifiée avec des tests tels que le test sur la période d'étude, le test sur le taux d'utilisation processeur ou encore le test sur le temps de réponse. On peut automatiquement attribuer aux tâches des priorités selon des algorithmes d'ordonnancement classiques comme Rate Monotonic ou Deadline Monotonic.

Lorsque des tâches partagent une ressource, on peut calculer le temps de blocage maximum sur celle-ci. L'évaluation de ce temps de blocage n'est disponible que lorsque la politique d'accès à ces ressources est PCP ou PIP.

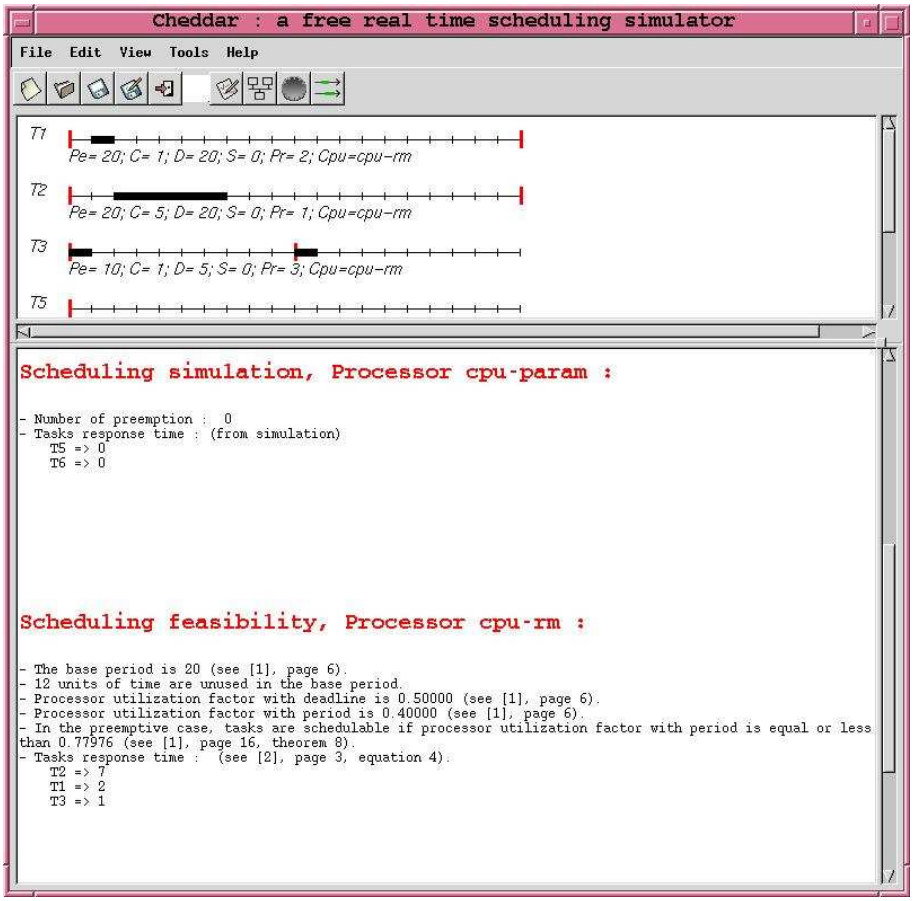

FIG. 7.3 – Exemple d'analyse d'une application temps réel

Cheddar propose des services pour ordonnancer et vérifier la faisabilité d'un jeu de tâche avec certains types de contrainte précédence. Des heuristiques basées sur les travaux de CHETTO and BLAZEWICZ permettent de prendre en compte les contraintes de précédence entre tâches en modifiant la valeur des échéances ou des priorités. L'analyse holistique permet de calculer le temps de réponse des tâches dont le début d'exécution dépends

de la fin d'exécution d'une tâche située sur un processeur éventuellement différent (cf. traitements répartis dans la section 2.4).

Un exemple d'utilisation des outils de vérification est proposé figure 7.3. Sur les chronogrammes d'ordonnancement, les trais verticaux rouges et les rectangles noirs représentent respectivement les dates d'activation et les instants d'exécution des tâches. On peut observer dans la partie haute de la fenêtre principale de Cheddar, les chronogrammes de l'ordonnancement des tâches, tandis que les informations relatives à la faisabilité sont affichées dans la partie basse de la fenêtre.

7.1.2.3 Les outils paramétrables

Les tests de faisabilité sont limités à quelques algorithmes d'ordonnancement et modèles de tâches (principalement le modèle périodique). Or de nombreuses applications temps réel sont développées autour de modèles de tâche et d'ordonnanceurs spécifiques. Il n'est donc plus possible d'utiliser les outils de vérification de la faisabilité présentés précédemment.

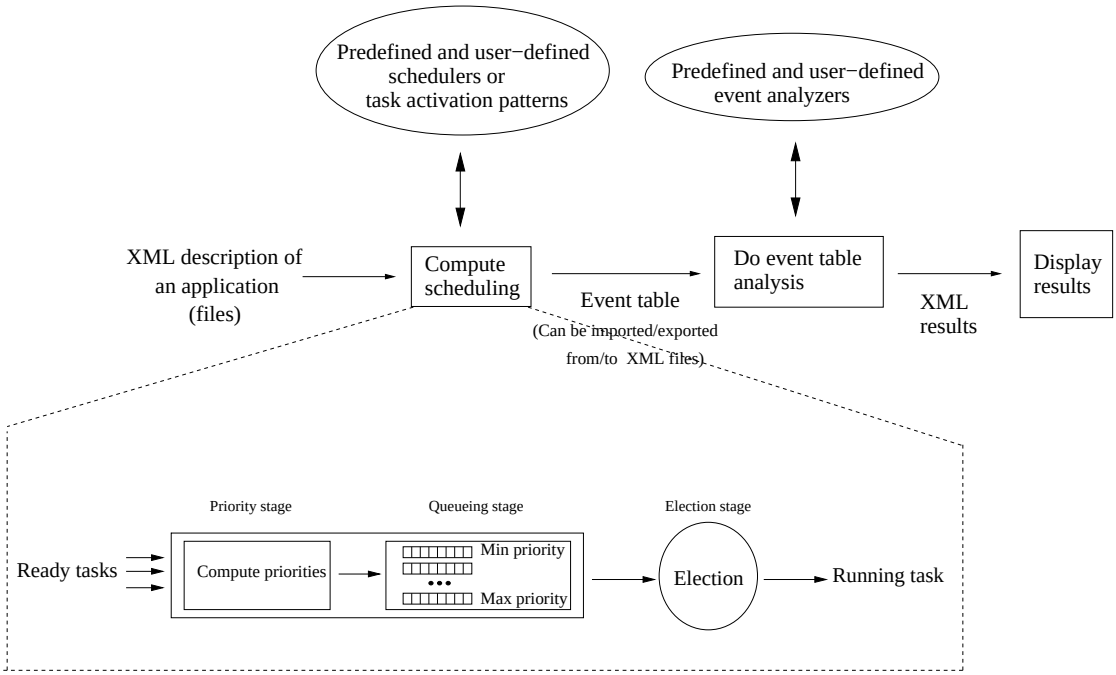

FIG. 7.4 – Fonctionnement d'un ordonnanceur dans Cheddar

Cheddar offre des fonctionnalités permettant de prendre en compte les spécificités d'un systèmes temps réel. L'ordonnanceur ainsi que la manière dont les tâches sont activées peuvent être spécifiés par l'utilisateur. L'analyse du système est effectuée grâce aux outils

de simulation. L'utilisateur peut également définir des mesures particulières sur l'ordonnancement généré.

Les ordonnanceurs ont pour rôle de choisir la tâche qui doit être exécuté sur un processeur à un instant donné. Cheddar intègre un ordonnanceur qui procède en trois étapes (similaire à l'ordonnanceur POSIX 1003.b, cf. figure 7.4) :

- La phase de calcul des priorités : seules les tâches prêtes sont concernées.
- La phase de gestion des files : les tâches prêtes sont insérées dans différentes files d'attente. Une file contient toutes les tâches ayant la même priorité.
- La phase d'élection : l'ordonnanceur cherche la file non vide de plus au niveau de priorité et attribue le processeur à la tâche qui se trouve en première position. La tâche élue garde le processeur durant une unité de temps si l'ordonnanceur est préemptif ou toute sa capacité dans le cas contraire.

Créer un nouvel ordonnanceur consiste à redéfinir les phases que nous venons de décrire. Pour cela, un langage a été développé. Le code produit par l'utilisateur n'est pas compilé mais interprété durant la simulation, il n'est donc pas nécessaire de connaître le fonctionnement interne du simulateur.

Un fichier contenant le code des différentes phases de l'ordonnanceur est organisé en plusieurs parties :

- La phase d'initialisation est désignée par le mot clef "*start_section*".

  Dans cette phase, on peut déclarer les variables nécessaires à l'ordonnanceur. Elles peuvent être de type scalaire (entier, booléen ou flottant), ou de type tableau (d'entiers, de booléens ou de flottants).

  L'utilisateur dispose de variables prédéfinies dans Cheddar.

  Les variables prédéfinies statiques représentent les paramètres des tâches données par l'utilisateur (période, délai critique, capacité,...). Elles sont initialisées à la création des tâches dans Cheddar. Leur valeur ne change pas durant la simulation.

  Les variables prédéfinies dynamiques représentent l'état d'une tâche, du processeur ou d'autres éléments de l'application durant la période de simulation. Leur valeur évolue dans le temps.

- La phase de calcul de priorité est désignée par le mot clef "*priority_section*".

  Le code donné dans cette phase est exécuté à chaque fois qu'une décision concernant l'ordonnancement doit être prise : toutes les unités de temps pour un ordonnanceur préemptif et à la fin de l'exécution des tâches pour un ordonnanceur non-préemptif.

- La phase d'élection est désignée par le mot clef "*election_section*". On y retourne l'index de la tâche qui accède au processeur.

- La manière dont sont activées les tâches dans la partie "*task_activation_section*".

Le langage comprends deux types d'instructions : les instructions haut-niveau et bas-niveau.

- Les instructions haut-niveau cachent le fonctionnement interne du simulateur. Par exemple, l'utilisateur n'a pas besoin de demander à l'ordonnanceur de parcourir le tableau contenant les tâches de l'application. L'écriture d'un ordonnanceur en est donc facilité.

- Les instructions bas-niveau peuvent être utilisées à condition d'avoir une bonne connaissance du fonctionnement du simulateur.

```
start_section :
    dynamic_priority : array (tasks_range) of integer ;
priority_section :
   dynamic_priority := tasks.start_time
      + ((tasks.activation_number-1)*tasks.period)
      + tasks.deadline ;
election_section :
  return min_to_index(dynamic_priority) ;
```

FIG. 7.5 – Ordonnanceur EDF défini par un utilisateur

Un exemple illustre la manière dont on peut créer un ordonnanceur dans Cheddar (cf. figure 7.5). Le comportement de cet ordonnanceur est équivalent à celui de EDF : à chaque unité de temps t, la tâche dont l'échéance est la plus proche de t est élue.

L'échéance de chacune des tâches est calculée dans la section *priority_section*. Ces échéances sont stockées dans la variable vectorielle *dynamic_priority* définie dans la section *start_section*. Les variables prédéfinies statiques *start_time*, *period* et *deadline* correspondent respectivement aux valeurs de la date de première activation, de la période

et du délai critique de la tâche. La variables prédéfinie dynamique *activation_number* représente l'activation courante de la tâche. Cette donnée est mise à jour par le simulateur. En outre, il est possible, grâce à l'éditeur graphique, d'affecter aux tâches de l'application des paramètres définis par l'utilisateur et exploitable dans l'ordonnanceur paramétrable.

La phase d'élection contient une instruction qui indique au simulateur la prochaine tâche qui doit être exécutée. Cette instruction renvoie l'indice de la tâche ayant l'échéance la plus proche à un instant t.

Les instructions des phases de calcul de priorité *priority_section* et d'élection de la tâche *election_section* sont interprétées par le simulateur à chaque fois que les tâches doivent être réordonnancées : à chaque unité de temps pour les algorithmes préemptifs et à chaque fois qu'une tâche libère le processeur dans le cas non-préemptif.

Dans Cheddar, trois types de tâches existent : les tâches apériodiques, périodiques et poissons. Pour une application composée de tâches n'appartenant pas à un de ces trois types, il est possible de préciser la manière dont elles sont activées. Grâce à l'instruction *Set*, on indique le lien entre le nom d'un type d'activation et une expression. Cette expression représente le temps séparant deux activations successives d'une tâche.

Cheddar permet d'effectuer des mesures spécifiques sur une simulation. Ces mesures sont obtenues grâce à une analyseur d'événement. Un événement peut être produit lorsque, par exemple, une tâche devient prête à être exécutée ou qu'elle libère une ressource partagée. Un langage interprété est utilisé pour redéfinir le fonctionnement des différentes parties de l'analyseur :

- La phase d'initialisation contient la déclaration des variables.
- La phase de récupération de données comprend les opérations d'enregistrement des informations relatives aux événements.
- La phase d'affichage à pour objectif d'analyser les données stockées et d'afficher le résultat dans la fenêtre principale de Cheddar.

7.2 Les tampons dans Cheddar

Afin d'étudier les systèmes temps réel comprenant des tampons, nous avons apporté un certain nombre de modifications à Cheddar. Ces modifications concernent la modélisation des tampons et la mise à disposition de nouvelles fonctionnalités dans les outils de simulation et d'analyse.

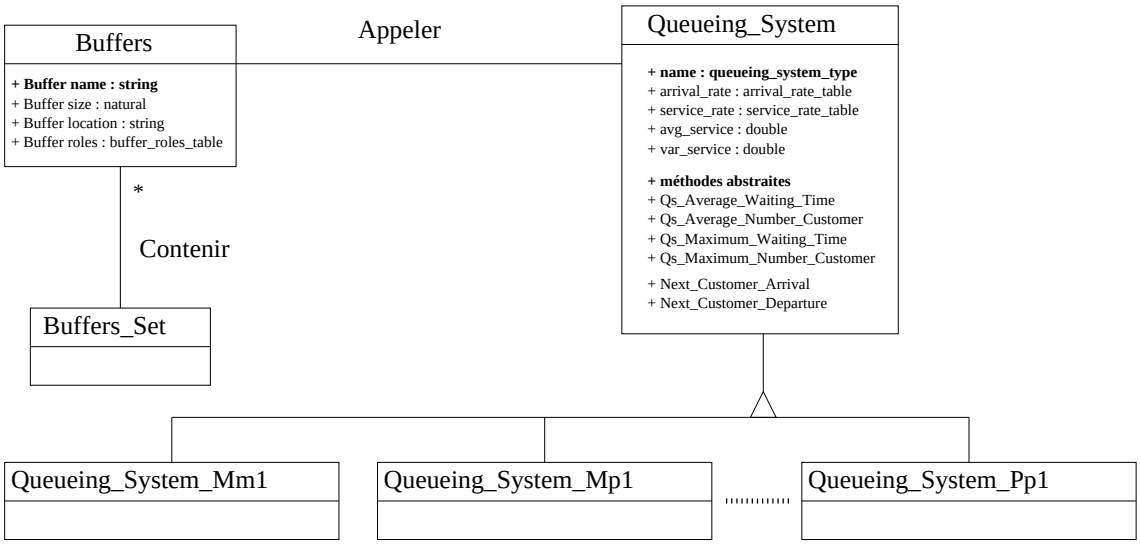

FIG. 7.6 – Diagramme UML des classes concernant les tampons dans Cheddar

Plusieurs paquetages Ada ont été développés pour élaborer ces nouvelles fonctionnalités (cf. figure 7.6). Les classes contenues dans ces paquetages regroupent les outils pour analyser les tampons d'une application temps réel ainsi que les fonctions/procédures de simulation présentées dans le chapitre précédent.

Dans un premier temps, nous présentons les apports concernant l'édition de projet avec tampon. Puis, nous détaillons les classes développées pour l'analyse des files d'attente. Finalement, nous discutons des outils de simulation et d'analyse des tampons

7.2.1 Edition d'un projet Cheddar avec tampon

Tout d'abord, il est nécessaire de pouvoir ajouter/modifier/supprimer un tampon dans Cheddar. Nous avons choisi de les décrire dans la classe **Buffers** grâce aux attributs suivants :

- **name** est une chaîne de caractères contenant le nom du tampon. Ce nom doit être unique.
- **location** donne le nom du processeur sur lequel se trouve le tampon.
- **size** est un entier naturel indiquant la taille du tampon. Elle représente la borne maximum sur le taux d'occupation du tampon.

- Finalement, **roles** comprends l'ensemble des tâches qui accèdent au tampon. Chaque élément de ce tableau contient le type de tâche (producteur ou consommateur) et la taille des données échangées entre la tâche et le tampon. On suppose que les tâches tentent de lire ou écrire des données de taille fixe à chacune de leurs activations.

L'ensemble des tampons d'une application est contenu dans une instance de la classe **Buffer_Set**. Cette classe exporte les procédures qui permettent d'ajouter, de supprimer ou de chercher un tampon dans un projet Cheddar.

Nous ne nous sommes pas limité aux tampons étudiés dans cette thèse. Il est possible, par l'intermédiaire de l'outil graphique spécifiant les contraintes de précédence (cf. figure 7.2), de créer des tampons où la tâche consommateur est activée sur réception de message.

7.2.2 Développement de classes pour l'analyse des files d'attente

Afin de développer des outils d'analyse des tampons, des paquetages Ada implémentent des résultats issus de la théorie des files d'attente et de nos travaux sur M/P/1 et P/P/1. Ces résultats concernent essentiellement les critères théoriques tels que le temps moyen d'attente ou le taux moyen d'occupation d'une file d'attente.

La classe **Queueing_System** et ses dérivées (*Queueing_System.Mm*1, *Queueing_System.Mg*1, ...), contenues dans ces paquetages, possèdent également également les fonctions/procédures nécessaires aux algorithmes de simulation décrits dans le chapitre 6.

```
package Queueing_System is
   ...
   type queueing_system_type is
      (Qs_Mm1, Qs_Md1, Qs_Mp1, Qs_Mg1, Qs_Pp1, Qs_Mms,
      Qs_Mds,Qs_Mps, Qs_Mgs, Qs_Mm1n, Qs_Md1n, Qs_Mp1n,
      Qs_Mg1n, Qs_Mmsn, Qs_Mdsn, Qs_Mpsn, Qs_Mgsn);

   Procedure Set_Qs_Arrival_Rate (
      a_queueing_system  : in out Generic_Queueing_System'Class;
      value              : in Double);

   Function Get_Qs_Arrival_Rate (
      a_queueing_system  : in Generic_Queueing_System'Class;
      place              : in Arrival_Rate_Range)
    return Double;
   ...
end Queueing_System;
```

FIG. 7.7 – Type et procédures/fonctions d'accès aux attributs d'une file

Une file d'attente possède entre autres les attributs suivants :

- L'attribut **name** contient le nom de la file. Ce nom doit appartenir au type énuméré **Queueing_System_Type** (cf. figure 7.7). Les files d'attente considérée sont, entre autres, les files M/M/1, M/G/1, M/D/1, P/P/1 et M/P/1.

- Les attributs **arrival_rate** et **service_rate** sont des tableaux contenant respectivement le taux moyen d'arrivée et de départ des nb_flow_in flux d'entrée et des nb_flow_out flux de sortie de la file d'attente.

- Finalement, les attributs **average_service_time** et **variance_service_time** sont respectivement les valeurs du temps de service moyen et de la variance sur ce temps de service obtenues après simulation.

Ces attributs sont privés. Seules les fonctions/procédures de la classe auquel appartiennent ces attributs peuvent lire ou modifier leur valeur.

Néanmoins des fonctions/procédures permettent à d'autres objets d'accéder à ces attributs (par exemple, pour initialiser les taux d'arrivée et de service d'une file). L'interface de deux de ces fonctions/procédures est décrite sur la figure 7.7.

```
package Queueing_System is
   ...
   Function Qs_Average_Waiting_Time (
      a_queueing_system  : in out Generic_Queueing_System)
    return Double;

   Function Qs_Average_Number_Customer (
      a_queueing_system  : in out Generic_Queueing_System)
    return Double;

   Function Qs_Maximum_Waiting_Time (
      a_queueing_system  : in out Generic_Queueing_System)
    return Double;

   Function Qs_Maximum_Number_Customer (
      a_queueing_system  : in out Generic_Queueing_System)
    return Double;

   Procedure Theoric_Results (
      A_Queueing_System : in out Generic_Queueing_System'Class);
   ...
end Queueing_System;
```

FIG. 7.8 – Interface des fonctions/procédures de calcul des critères de performance

La classe *Queueing_System* est conçu de manière à ce que la création de nouvelles files d'attente soit aisée. Pour cela, nous utilisons des mécanismes d'héritage (cf. figure 7.6). Elle regroupe l'ensemble du code commun à toutes les files d'attente. Le code spécifique doit être redéfini pour chaque file d'attente.

Nous avons vu dans le chapitre 6 qu'il fallait redéfinir le calcul du temps inter-arrivée et du temps de service pour simuler une file d'attente donnée. De la même manière, chaque file possède ses propres équations pour calculer les critères de performance.

Les procédures **Qs_Average_Waiting_Time** et **Qs_Average_Number_Customer** permettent d'obtenir le temps moyen d'attente et le taux moyen d'occupation théorique (cf. figure 7.8). Ces critères de performance sont spécifiques à chacune des files d'attente. Par conséquent, le code de ces deux procédures se trouve dans une classe dérivée de *Queueing_System*.

```
package body Queueing_System.Mm1 is
   ...
   Function Qs_Average_Waiting_Time (
      a_queueing_system : in out Mm1_Queueing_System)
      return Double is
   begin
      Result := 1.0 / (Service_Rate * (1.0 - Arrival_Rate / Service_Rate));
      return Result;
   end Qs_Average_Waiting_Time;

   Function Qs_Average_Number_Customer (
      A_Queueing_System : in out Mm1_Queueing_System)
      return Double is
         Rau : Double := Arrival_Rate / Service_Rate;
   begin
      Result := Rau / (1.0 - Rau);
      return Result;
   end Qs_Average_Number_Customer;
   ...
end Queueing_System.Mm1;
```

FIG. 7.9 – Exemple de fonctions de calcul de critères moyens d'une file M/M/1

Un exemple d'implémentation de ces procédures pour une file M/M/1 est donné figure 7.9.

Les procédures **Qs_Maximum_Waiting_Time** et **Qs_Maximum_Number_Customer** permettent d'obtenir le temps d'attente et le taux d'occupation maximum théorique (cf. figure 7.8).

```
package body Queueing_System.Pp1 is
   ...
   Function Qs_Maximum_Number_Customer (
      a_queueing_system : in Pp1_Queueing_System)
      return Double is
   begin
      If (harmonic = true) then
         return 2*nb_flow_in;
      Else
         return 2*nb_flow_in + 1.0;
      End if;
   end Qs_Maximum_Number_Customer;
   ...
end Queueing_System.Pp1;
```

FIG. 7.10 – Exemple de fonction de critère maximum d'une file P/P/1

Un exemple d'implémentation de la procédure *Qs_Maximum_Number_Customer* pour une file P/P/1 est donné figure 7.10.

Il existe d'autres fonctions pour évaluer des critères théoriques tels que la probabilité d'être dans un état donné de la file (*Get_Probability_Of_State*) ou la probabilité que la file soit pleine (*Get_Probability_Of_Full_Buffer*). Il faut noter que ces critères ne sont pas disponibles pour toutes les files d'attente.

L'ensemble des valeurs des critères théoriques disponibles pour une file d'attente donnée peut être affiché grâce à la procédure **Theoric_Results** (cf. figure 7.7).

7.2.3 Outils d'analyse et de simulation des tampons

Dans cette section, nous traitons des outils d'analyse et de simulation des tampons. Ces outils se servent, entre autres, des différents paquetages développés pour les files d'attente.

Pour la partie simulation, le générateur d'ordonnancement des tâches a été modifié afin de prendre en compte la contrainte de précédence entre l'arrivée des messages et l'activation de la tâche consommateur. En outre, l'utilisateur peut afficher, à partir d'une simulation, un histogramme montrant l'évolution du taux d'occupation du tampon en fonction du temps (cf. figure 7.11).

Les outils de vérification de la faisabilité permettent d'obtenir, selon les caractéristiques du tampon, les bornes maximums et les valeurs moyennes théoriques de critères de performance.

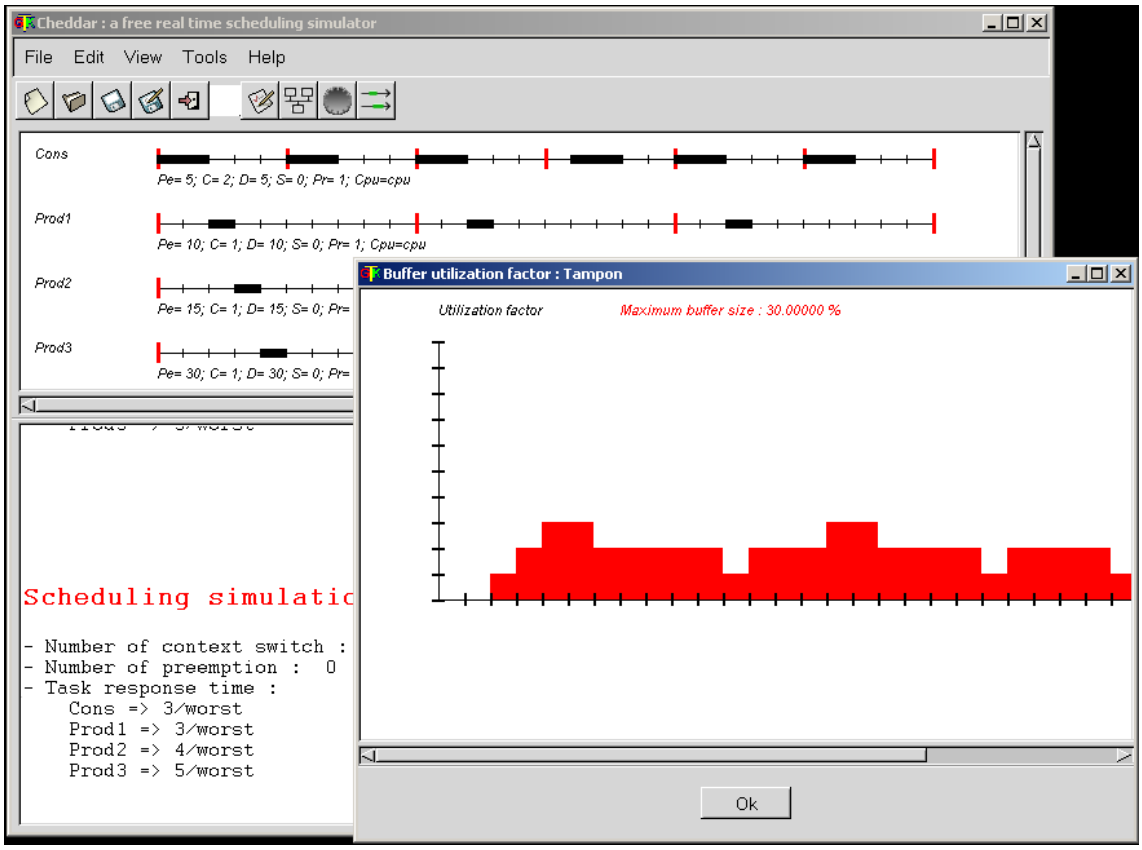

FIG. 7.11 – Histogramme du taux d'occupation d'un tampon

La classe $Buffers$ contient l'ensemble des procédures/fonctions appelées par cet outil.

Les fonctions **Get_Max_Occupation** et **Get_Max_Waiting_Time** calculent respectivement la valeur de l'occupation maximum du tampon et du temps d'attente maximum d'un message dans le tampon (cf. figure 7.12). Elles font appel aux fonctions $Qs\_Maximum\_Waiting\_Time$ et $Qs\_Maximum\_Number\_Customer$ de la classe $Queueing\_System$.

Par exemple, pour les tampons P/P/1, la valeur retournée est $2.N$ lorsque ces tâches sont harmoniques et $2.N + 1$ dans les autres cas (cf. théorème 12).

Les fonctions **Get_Avg_Occupation** et **Get_Avg_Waiting_Time**, calculent respectivement la valeur de l'occupation moyenne du tampon et du temps d'attente moyen d'un message dans le tampon (cf. figure 7.12). Elles font appel aux fonctions $Qs\_Average\_Waiting\_Time$ et $Qs\_Average\_Number\_Customer$ de la classe $Queueing\_System$.

```
package Buffers is
        ...
        Function Get_Max_Occupation (
           sys : System)
         return Double;

        Function Get_Max_Waiting_Time (
           sys : System)
         return Double;

        Function Get_Avg_Occupation (
           sys : System)
         return Double;

        Function Get_Avg_Waiting_Time (
           sys : System)
         return Double;
        ...
end Buffers;
```

FIG. 7.12 – Interface des fonctions de calcul de critères pour les tampons

Par exemple, pour les tampons M/P/1, ces procédures implémentent les critères théoriques de la file M/G/1 (cf. tableau 2.7) dont les paramètres sont donnés dans le théorème 7.

Les procédures de calcul de critère de performance nécessitent de connaître certaines informations concernant les tampons étudiés.

La fonction **Is_Cons_Prod_Harmonic** est utilisée pour vérifier l'harmonicité des périodes des tâches consommateurs et producteurs (cf. figure 7.13).

Les valeurs des critères théoriques peuvent être calculées si le taux de production des messages est inférieur au taux de consommation (cf. définitions 13 et 21). Les résultats sont donc obtenus sous contrôle de la procédure **Buffer_Flow_Control** : une exception est levée si la loi de conservation du débit n'est pas respectée (cf. figure 7.13).

Finalement, nous avons ajouté des événements tels que la lecture ou l'écriture de données dans un tampons afin d'effectuer des mesures sur les systèmes étudiés dans cette thèse grâce aux outils paramétrables

```
package Buffers is
   ...
   Function Is_Cons_Prod_Harmonic (
      my_buff : in Buffer_Ptr;
      my_tasks : in Tasks_Set )
    return Boolean;

   Procedure Buffer_Flow_Control (
      my_buff : in Buffer_Ptr;
      my_tasks : in Tasks_Set );
   ...
end Buffers;
```

FIG. 7.13 – Interface des fonctions/procédures de collecte d'informations sur le tampon

7.3 Conclusion

Cheddar est un logiciel sous licence GNU développé pour modéliser, simuler et analyser des applications temps réel. Cheddar répond à trois objectifs : il peut être utilisé dans un cadre pédagogique, servir d'outil de prototypage d'applications temps réel et aider aux travaux de recherche sur l'ordonnancement temps réel.

Nous présentons les outils de simulation et de validation disponibles dans Cheddar. Cheddar intègre la plupart des résultats classiques de l'ordonnancement temps réel des systèmes mono-processeurs et répartis (cf. chapitre 2).

Il est possible de prendre en compte les spécificités de fonctionnement d'une application. L'utilisateur peut définir, entre autres, son propre ordonnanceur et ses propres règles d'activation des tâches grâce à un langage fournit dans Cheddar. Plusieurs exemples de ces fonctionnalités sont donnés dans ce chapitre

Nous avons développé des classes Ada pour étendre les fonctionnalités de Cheddar. Ces nouvelles fonctionnalités permettent d'étudier les systèmes temps réel comprenant des tampons. Les classes $Buffers$ et $Buffers\_Set$ permettent d'éditer des applications comprenant des tampons dans Cheddar. Le simulateur d'ordonnancement des tâches a été modifié pour prendre en compte les éventuelles activations de tâches sur réception de message. La classe $Queueing\_System$ fournit des fonctions/procédures pour calculer les critères de performance nécessaire à l'analyse des tampons.

Chapitre 8

Etude de cas : première approche de l'analyse temporelle d'un plan PILOT

L'objectif de ce chapitre est de fournir une première approche de l'analyse temporelle d'un plan PILOT. Cette approche se base sur les résultats présentés dans les chapitres 3, 4 et 5.

Le langage PILOT est un langage graphique permettant de concevoir et d'exécuter des missions pour des robots mobiles télé-opérés. Ce langage est développé au sein de l'équipe d'acceuil informatique de l'UBO (EA2215) [NLSM03]. Un plan PILOT représente une séquence formée d'actions et de structures de contrôle (conditionnelle, ...). Nous cherchons en particulier à borner le temps d'exécution d'un plan ou encore à vérifier des contraintes de synchronisation entre différentes actions.

Pour évaluer ces critères pour un plan PILOT donné, nous appliquons la démarche suivante :

1. Nous supposons une architecture donnée. En particulier, le robot, son support d'exécution embarqué ainsi que l'environnement de contrôle de PILOT sont modélisés en termes de tâches, de tampons et de canaux de communication. Les paramètres des tâches, tels que la période, la capacité ou les contraintes de précédence, sont connus.

2. A partir de ces informations, nous calculons les bornes maximums sur l'occupation et sur le temps d'attente des tampons.

3. Finalement, nous appliquons l'analyse holistique pour déterminer les temps de réponse des traitements répartis. Nous obtenons ensuite le temps de réponse des actions et structures du langage PILOT. Grâce à ces temps de réponse, nous pouvons valider temporellement un plan PILOT.

Ce chapitre est organisé en trois parties :

- La section 8.1 décrit le langage PILOT et l'architecture logicielle de son environnement de contrôle.

- La section 8.2 présente l'analyse temporelle d'un plan PILOT.

- Finalement, nous concluons.

8.1 Le langage PILOT et son architecture logicielle

Dans cette section, nous décrivons, dans un premier temps le langage PILOT, puis nous détaillons son architecture logicielle.

8.1.1 La langage PILOT

Le langage PILOT est un langage graphique. Il est basé sur la notion d'action.

Une action comporte un ordre exécutable par le robot, une précondition et une ou plusieurs règles de surveillances auxquelles sont associés des traitements.

Deux types d'actions sont distingués : les actions élémentaires et les actions continues. Une action élémentaire a sa propre fin, contrairement à une action continue. Elle se termine généralement lorsque son objectif est atteint. La terminaison d'une action continue est quant à elle déclenchée par une autre primitive du langage. Quelle que soit sa nature, une action ne s'exécute que si sa précondition est vraie. De même, lorsqu'au cours de l'exécution d'une action, une de ses règles de surveillance est vraie, le traitement associé est exécuté.

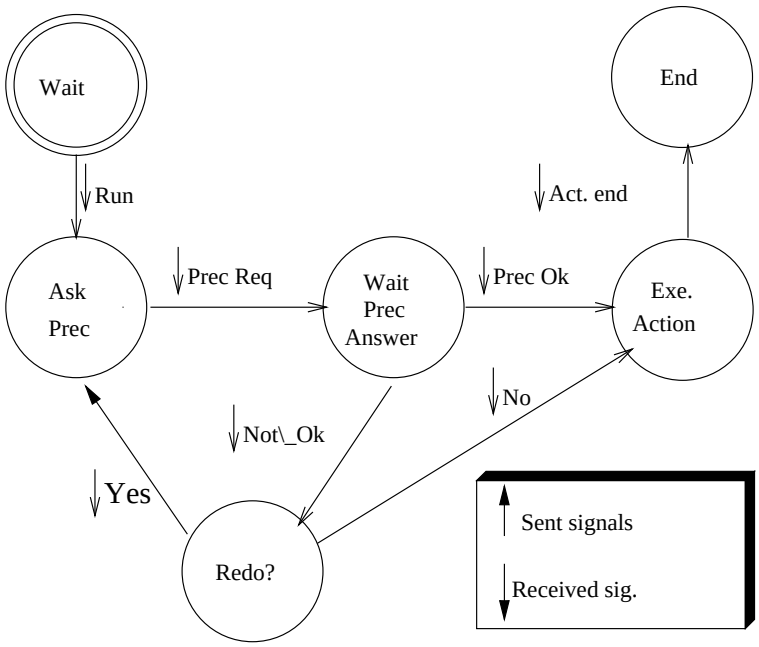

FIG. 8.1 – Automate à état fini

Dans la pratique, les préconditions et les règles de surveillance associées aux actions sont très souvent des conditions sur des valeurs de capteurs. Le traitement par défaut associé aux règles de surveillances est l'arrêt de l'action correspondante. Chaque type d'action est modélisé par un automate d'état fini (cf. figure 8.1). Deux étapes de cet automate sont essentielles. La première, **Ask Prec.** traduit l'évaluation de la précondition avant le lancement de l'action. Si cette précondition est satisfaite (*Prec OK*), alors l'action est exécutée. Sinon, l'utilisateur est averti et peut choisir entre la réévaluation de la précondition (**Redo ?**) et le lancement de l'action sans réévaluation de la précondition (**Exe. Action**). L'état **Exe. Action** est le deuxième état le plus important de l'automate. Quand une

action atteint son but prédéfini (cas des actions élémentaires) ou quand l'une de ses règles de surveillances devient vraie, l'action est stoppée par le système de contrôle (**End**).

	Début de séquence
	Séquentialité
	Fin de séquence
?	Conditionnelle
	Itération
	Conditionnelle
	Parallélisme
	Préemption

FIG. 8.2 – Liste des symboles relatifs aux structures de contrôle de PILOT

Le langage PILOT fournit différentes structures de contrôle pour la construction de plans : séquentialité, conditionnelle, itération, parallélisme et préemption. La figure 8.2 montre les symboles utilisés pour identifier ces structures.

Nous décrivons brièvement ci-après les structures de contrôle du langage PILOT :

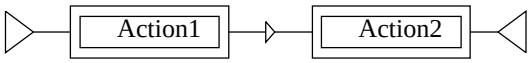

FIG. 8.3 – Séquentialité

- *La séquentialité.* Trois symboles sont utilisés pour la séquentialité : le début de séquence, l'inter-séquence et la fin de séquence. La figure 8.3 montre un exemple de séquence comportant deux actions élémentaires action1 et action2. L'exécution de l'action2 commence après la fin de l'action1.

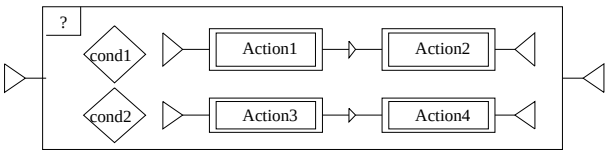

FIG. 8.4 – Conditionnelle

- *La conditionnelle.* Elle est formée d'une ou plusieurs alternatives ordonnées du haut vers le bas et comportant chacune une condition suivie d'une séquence. La première séquence dont la condition est vraie est la seule exécutée. La figure 8.4 montre un exemple de conditionnelle formée de deux alternatives.

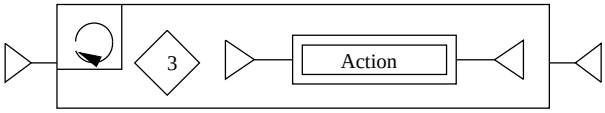

FIG. 8.5 – Itération fixe

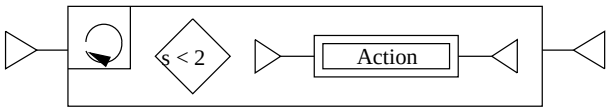

FIG. 8.6 – Itération conditionnelle

- *L'itération.* Elle est formée d'un critère de poursuite suivi d'une séquence. Ce critère peut être soit un nombre d'itérations, soit une expression booléenne. Dans le premier cas, l'itération est dite fixe et dans le second, elle est qualifiée de conditionnelle. La figure 8.5 montre un exemple d'itération fixe. Dans cet exemple, l'action est exécutée trois fois. La figure 8.6 présente quant à elle un exemple d'itération conditionnelle. L'action est exécutée quand la condition s $\prec$ 2 est initialement vraie et est réexécutée à son issue, tant que cette condition est vraie.

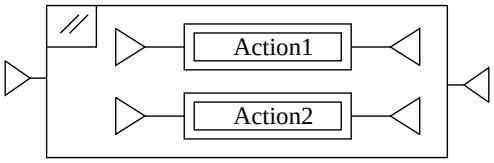

FIG. 8.7 – Parallélisme

- *Le parallélisme.* Il est formé de plusieurs séquences. Les séquences sont exécutées en parallèle. Son exécution se termine lorsque toutes les séquences ont atteint leur fin. La figure 8.7 illustre l'utilisation du parallélisme.

  Dans cet exemple, les actions 1 et 2 s'exécutent en parallèle et l'exécution parallèle se termine lorsque les deux actions sont terminées.

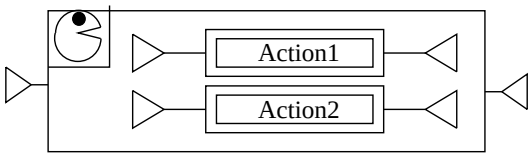

FIG. 8.8 – Préemption

- *La préemption.* Comme le parallélisme, elle est formée de plusieurs séquences dont l'exécution se fait en parallèle, mais contrairement à ce dernier, son exécution se termine dès que l'une de ses séquences atteint sa fin. La figure 8.8 montre un exemple d'utilisation de la préemption.

  Dans cet exemple, la terminaison de l'une des séquences entraîne l'arrêt de l'autre séquence et la fin de l'exécution de la préemption.

8.1.2 Architecture logicielle de l'environnement PILOT

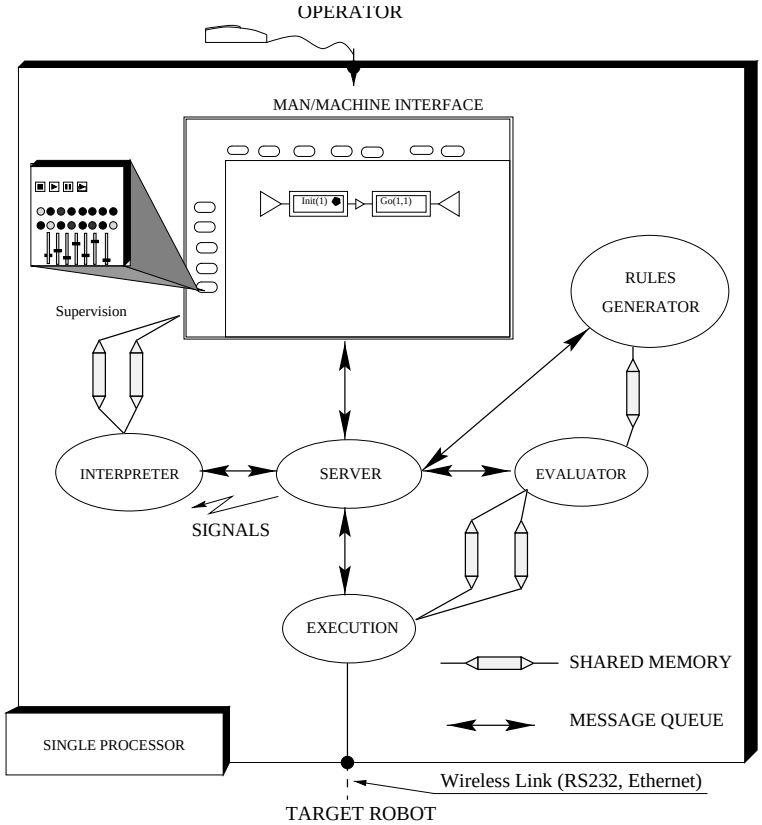

FIG. 8.9 – L'environnement PILOT

Nous présentons dans cette partie l'architecture logicielle de l'environnement PILOT.

Le système de contrôle (cf. figure 8.9) est l'interface entre l'utilisateur et la machine pilotée (Robot Cible). Il comporte six modules exécutés en parallèle :

- Une *Interface Homme-Machine* (IHM),
- Un *Serveur de Communication*,
- Un *Générateur de Règles*,
- Un *Evaluateur*,
- Un *Module d'Exécution* ou *Driver*,
- Et un *Interpréteur*.

Ces modules sont exécutés en parallèle et communiquent par socket et par mémoire partagée. Le système de contrôle peut s'exécuter soit en mode centralisé, soit en mode distribué. Le choix du mode d'exécution est effectué de façon statique (avant la compilation).

L'*IHM* fournit des moyens pour la construction de plans, la création dynamique d'actions (sans recompilation du code), et la modification du plan avant et au cours de l'exécution de ce dernier. Elle intègre également des moyens pour la supervision de l'exécution du plan. L'*IHM* stocke le plan dans une zone de mémoire partagée avec l'*interpréteur*.

L'*interpréteur* lit le plan en mémoire partagée et envoie des ordres (demande d'évaluation de précondition, ordre de démarrage d'une action, ...) aux autres modules afin de réaliser l'exécution du plan.

Le *serveur de communication* gère les communications inter-modules.

Le rôle du *générateur de règles* est de transformer les chaînes de caractères des règles de précondition et de surveillance en arbres binaires. Il stocke le résultat dans une zone de mémoire partagée avec l'*évaluateur*.

L'*évaluateur* évalue les règles de précondition et de surveillance à partir des arbres binaires correspondants.

Le *module d'exécution* réalise l'interface entre le robot et le système de contrôle. Il traduit les ordres de haut niveau du plan en ordres de bas niveau compréhensibles par la machine téléopérée. Le module d'exécution supporte différents protocoles de communication (connexion série, Ethernet, FDDI).

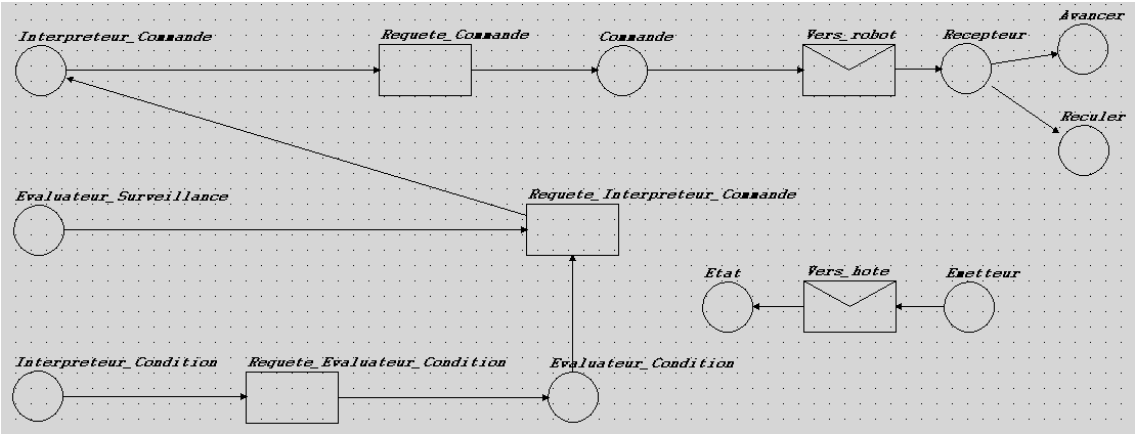

FIG. 8.10 – Contraintes de précédence entre les tâches de l'environnement PILOT

L'environnement PILOT est constitué de deux parties dont une est localisée sur la station de contrôle et l'autre embarquée sur le robot. La station et le robot communiquent à travers un lien série ou une liaison radio. Les services offerts par les deux composantes de l'environnement PILOT sont implantés par un ensemble de tâches.

La figure 8.10 décrit les contraintes de précédence entres les tâches de l'environnement PILOT. Dans cette figure, les tâches de l'environnement PILOT sont représentées par des cercles. Elles communiquent soit par des tampons, soit par des messages transmis à travers une liaison série.

La tâche *Interpreteur* est découpée en deux sous-tâches (*Interpreteur_ Condition* et *Interpreteur_ Commande*) afin de simplifier la modélisation de ses contraintes de précédence. Pour les mêmes raisons, la tâche *Evaluateur* est modélisée en deux tâches : les tâches *Evaluateur_ Condition* et *Evaluateur_ Surveillance*. Enfin, la tâche *Exécution* est modélisée par les tâches *Commande* et *Etat*.

Les tâches *Récepteur*, *Avancer*, *Reculer* et *Emetteur* s'exécutent sur le robot. On ne s'occupe pas du fonctionnement des tâches sur le robot.

Les tâches *Generateur* et *Serveur* n'apparaissent pas directement. La tâche *Serveur* est prise en compte au travers des tampons. La tâche *Generateur* est modélisée par les tâches *Evaluateur_Condition* et *Evaluateur_Surveillance* (cumul des durées de génération des conditions et d'évaluation des règles).

Toutes les tâches placées sur la station de contrôle fonctionnent de la façon suivante :

- La tâche *Interpreteur_Condition* demande l'évaluation de précondition, de conditions d'itération ou de conditionnelle à la tâche *Evaluateur_ Condition*.

- *Evaluateur_ Surveillance* évalue périodiquement les règles de surveillance et transmet les résultats à *Interpreteur_Commande*.

- *Evaluateur_Condition* évalue les préconditions et les expressions booléennes associées au itération et aux instructions conditionnelles, puis, envoie le résultat à la tâche *Interpreteur_Commande*.

- Lorsque les règles de précondition évaluées par *Evaluateur_ Condition* sont vraies, la tâche *Interpreteur_Commande* envoie à la tâche *Commande* les ordres à exécuter par le robot.

- La tâche *Commande* se charge de les transmettre au robot par la liaison série grâce aux messages *Vers_Robot*. Ces messages sont réceptionnés par la tâche *Récepteur*. La tâche *Récepteur* les fait suivre aux tâches qui fournissent les services demandés (ex : les tâches *Avancer*, *Reculer*, ...).

- La tâche *Emetteur* transmet périodiquement l'état du robot à l'application de contrôle. Ces données sont réceptionnées et mémorisées par la tâche *Etat*. On suppose que la tâche Emetteur transmet ces informations à son initiative. En réalité, dans la mise en œuvre actuelle de l'environnement PILOT, la tâche *Emetteur* est activée sur une demande de la tâche *Commande*.

Les tâches accèdent à trois ressources partagées :

- La ressource *Mémoire_ Plan* qui stocke la description actuelle du Plan en cours d'exécution. Cette description peut être modifiée par l'utilisateur grâce à l'interface homme-machine. Cette ressource est accédée par les tâches *Interpréteur1*, *Interpréteur2* et *Ihm*.

- La ressource *Mémoire_ Etat_ Capteurs* mémorise sur la station de contrôle l'état courant des capteurs du robot. Elle est accédée par les tâches *Evaluateur_ Surveillance*, *Evaluateur_ condition*, *Ihm* et *Etat*.

- Enfin, la troisième ressource est la zone de mémoire *Mémoire_ Etat_ Actions* qui stocke l'état courant de l'exécution des actions sur le robot. Elle est accédée par les tâches *Evaluateur_ Surveillance*, *Evaluateur_ condition*, *Ihm* et *Etat*.

8.2 Analyse temporelle d'un plan PILOT

Il s'agit ici de proposer une approche permettant d'obtenir certaines caractéristiques d'un plan PILOT tels que son temps d'exécution maximum ou l'évaluation des synchronisations.

Nous devons dans un premier temps analyser temporellement l'environnement PILOT. Pour cela, nous proposons une méthode basée sur l'analyse Holistique et sur nos propositions de critères maximums de performance. Nous présentons un exemple d'application de cette méthode sur une partie de l'environnement PILOT, à savoir celle localisée sur la station de contrôle.

8.2.1 Analyse temporelle de l'environnement PILOT

Dans cette section, nous donnons une méthode pour analyser temporellement l'environnement PILOT. Nous appliquons ensuite cette méthode à un cas concret.

Nom	P_i/D_i	C_i	Priorité
Commande	20000	120	10
Etat	30000	120	150
Evaluateur_Condition	20000	20	100
Evaluateur_Surveillance	20000	20	100
Interpreteur_Condition	20000	1000	50
Interpreteur_Commande	20000	1000	50

TAB. 8.1 – Paramètres des tâches

Le tableau 8.1 résume les différents paramètres des tâches de l'environnement PILOT. Les temps de réponses des tâches permettent d'exhiber les temps de latence de bout en bout des chaînes de traitements les plus significatives dont :

- La chaîne de traitements exécutant un nouveau plan. Elle est consituée de deux sous chaînes de traitements exécutées l'une à la suite de l'autre. Déterminer le temps de latence sur l'activation d'un plan PILOT requiert d'étudier le temps d'exécution de ces deux sous chaînes de traitements successives. Les sous chaînes sont :

  1. La séquence d'exécution des tâches *Interpreteur_Condition*, *Evaluateur_Condition* et finalement *Interpreteur_Commande*.

  2. La séquence d'exécution des tâches *Evaluateur_Surveillance*, *Interpreteur_Commande* et finalement *Commande*.

- Les chaînes de traitements qui mettent à jour dans l'IHM l'état du robot distant et du plan.

Nous nous sommes limités à l'étude de la première chaîne de traitements. L'approche adoptée est toutefois valable pour toutes les chaînes de traitements.

L'évaluation du temps de réponse nécessite de déterminer pour un message donné et pour chaque tampon traversé dans ces deux chaînes de traitements, le temps pendant lequel le message reste dans le tampon. Les tampons constituant les chaînes de traitements considérés sont les suivants :

- *Requete_Evaluateur_Condition*. Un producteur activé aléatoirement ou périodiquement toutes les 20000 milli-secondes. Un consommateur périodiquement activé toutes les 20000 milli-secondes.

- Les tampons *Tampon_Etat*, *Interpreteur_Commande* et *Requete_Interpreteur_Commande* sont tous les deux accèdés par un seul consommateur et un seul producteur de même période. L'occupation maximum de ces tampons est donc bornée à 2 messages (cf. théorème 11).

Nous appliquons l'analyse Holistique pour déterminer les temps de réponse des tâches et des messages transitant dans le tampon (cf. section 2.4.3). Pour appliquer cette analyse, il est nécessaire de connaître les délais de communication de chacun des tampons.

Nous employons les résultats proposés dans les chapitres précédents (chapitres 3, 4 et 5) pour les calculer. Nous considérons que les tampons de l'environnement PILOT sont des files P/P/1 partagées par une tâche producteur et une tâche consommateur. Dans ce cas, la borne maximum sur le temps d'attente est de $W_{max} = 2.P_{cons}$ (cf. théorème 11). Or, toutes les tâches consommateur ont une période de 20000 ms, les délais de communication sont donc bornés par la valeur 40000 ms.

$$
\begin{array}{l}
J_{commande} = r_{Requete\_Commande} \\
J_{Interpreteur\_Commande} = r_{Requete\_Interpreteur\_Commande} \\
J_{Evaluateur\_Condition} = r_{Requete\_Evaluateur\_Condition} \\
J_{Requete\_Evaluateur\_Condition} = r_{Interpreteur\_Condition} \\
J_{Requete\_Interpreteur\_Commande} = r_{Evaluateur\_Condition} \\
J_{Requete\_Commande} = r_{Interpreteur\_Commande}
\end{array}
$$

FIG. 8.11 – Relations de calcul des J_i pour l'analyse Holistique

Le tableau 8.2 récapitule l'ensemble des résultats (temps de réponse) de l'analyse Holistique. Les valeurs des J_i sont obtenues grâce aux relations de la figure 8.11.

A partir des ces temps de réponse, nous pouvons déduire le temps passé dans chacune des instructions du langage PILOT grâce aux relations que nous présentons dans la section suivante.

Nom	r_i	J_i	r_i	J_i	r_i	J_i	r_i	J_i
C	2260	40000	44300	41140	45440	81180	87480	81320
I_Cond	1140	0	1180	0	1180	0	1180	0
I_Com	1140	40000	41180	40140	41320	80140	81320	81280
E_C	140	40000	40140	41140	41280	41180	41320	41180
E	120	0	120	0	120	0	120	0
R_E_C	40000	1140	41140	1180	41180	1180	41180	1180
R_I_C	40000	140	40140	40140	80140	41280	81280	41320
R_C	40000	1140	41140	41180	81180	41320	81320	81320
C	87620	121320	127620	122460	128760	122500	128800	122500
I_Cond	1180	0	1180	0	1180	0	1180	0
I_Com	82460	81320	82500	81320	82500	81320	82500	81320
E_C	41320	41180	41320	41180	41320	41180	41320	41180
E	120	0	120	0	120	0	120	0
R_E_C	41180	1180	41180	1180	41180	1180	41180	1180
R_I_C	81320	41320	81320	41320	81320	41320	81320	41320
R_C	121320	82460	122460	82500	122500	82500	122500	82500

TAB. 8.2 – Analyse holistique appliquée à l'environnement PILOT

8.2.2 Estimation/Evaluation du temps d'exécution d'un plan

Avant de présenter les relations permettant d'évaluer le temps de réponse d'un plan PILOT, nous introduisons quelques notations :

- A_i constitue le temps d'exécution de l'action élémentaire numéro i.
- Ca_i constitue le temps de réponse de l'action i.
- Cb_i constitue le temps de réponse de la branche i.
- Ta_i constitue soit le temps maximum, soit le temps minimum de réponse de la chaîne de traitements réalisée par l'environnement PILOT afin d'initier l'exécution de l'action i sur le robot et d'en obtenir l'information de terminaison.
- Tb_i constitue une borne sur le temps d'exécution des opérations réalisées au démarrage et à la terminaison d'une boîte PILOT.

Par la suite, en première approche, nous nous limitons aux actions élémentaires.

Les temps de réponse des actions et des structures de contrôle de PILOT sont obtenus grâce aux relations suivantes :

- L'action élémentaire. Pour une action élémentaire, nous avons :

$$Ca_i = A_i + Ta_i$$

- La séquence. Pour une séquence j comportant n actions élémentaires numérotées de 1 à n, le temps de réponse est

$$Cb_j = \sum_{\forall 1 \leq i \leq n} Ca_i$$

- Le parallélisme. Pour une boite parallèle j comportant n séquences (ou branches) numérotées de 1 à n, le temps de réponse est :

$$Cb_j = Tb_j + max_{\forall 1 \leq i \leq n}(Cb_i)$$

- La préemption. Pour une boite de préemption j comportant n séquences (ou branches) numérotées de 1 à n, le temps de réponse est :

$$Cb_j = Tb_j + min_{\forall 1 \leq i \leq n}(Cb_i)$$

- L'itération. On suppose ici une boite d'itération j où le nombre d'itérations maximal est connu. Nous notons ce nombre n. Soit T_j, une borne maximale ou minimale sur le temps d'exécution du test de terminaison de la boucle. Soit Cb_1, le temps de réponse de la séquence itérée.

  Pour la boite j, le temps de réponse est :

$$Cb_j = Tb_j + n.(T_j + Cb_1)$$

- La conditionnelle. On suppose ici une boite conditionnelle j comportant n branches conditionnelles (numérotées de 1 à n). Soit T_i, une borne sur le test de la condition de la branche i. Soit Cb_i, le temps de réponse de la séquence de la branche i.

  Pour la boite j, le temps de réponse est :

$$Cb_j = Tb_j + \sum_{1 \leq i \leq n} T_i + max(Cb_i)$$

8.3 Conclusion

Notre objectif est de fournir une première approche de l'analyse temporelle d'un plan PILOT.

La méthode d'analyse temporelle de l'environnement PILOT est basée sur l'analyse Holistique et sur nos propositions de critères maximums de performance (délai d'attente d'un message, cf. chapitres 3, 4 et 5).

Nous avons appliqué cette méthode à une chaîne de traitements de l'environnement PILOT afin d'extraire différents temps de réponse des tâches de cette chaîne. Le travail présenté n'est pas complet. Il faut ensuite calculer le temps de réponse des actions du langage PILOT à partir des relations proposées. Finalement, il est possible d'obtenir un certain nombre de critères de performance comme le temps d'exécution d'un plan PILOT ou/et de vérifier la synchronisation de diférentes actions.

En outre, nous ne sommes pas encore capable d'analyser complètement l'architecture logicelle. Par exemple, nous n'avons pas encore étudié le cas des tampons dont les flux d'arrivée sont à la fois déterministe et non-déterministe. Ce cas constitue une perspective intéressante pour la suite des travaux.

Chapitre 9

Conclusion et perspectives

Lorsqu'un système est temps réel, il n'est pas suffisant de considérer uniquement l'exactitude des résultats produits par l'application. Il faut également vérifier le respect des contraintes temporelles du système. Dans le cadre de cette thèse, nous avons étudié une méthode, généralement désignée sous le terme de théorie de l'ordonnancement temps réel, permettant de simuler voire vérifier de tels systèmes.

La notion de faisabilité est primordiale dans les systèmes temps réel. Une application est faisable, ou ordonnançable, si toutes les tâches qui la composent respectent leurs contraintes temporelles. Il s'agit de déterminer si les tâches du système modélisé sont exécutées dans les temps.

Cette thèse traite de la faisabilité des systèmes temps réel répartis.

Nous considérons qu'un système réparti est constitué de plusieurs sous-systèmes monoprocesseur reliés par un réseau.

Un sous-système est composé de tâches et de tampons. Nous supposons que les tâches du système sont périodiques et ordonnancées, par exemple, selon un algorithme à priorités fixes. La migration des tâches entre les différents processeurs est interdite. Les tampons collectent des informations délivrées par un autre sous-système au travers du réseau ou périodiquement lors d'une communication entre tâches de ce même sous-système. Les messages arrivent dans le tampon à une cadence donnée. Dans certains systèmes, cette cadence peut être difficile à définir.

Ce modèle d'application correspond à une pratique industrielle que l'on retrouve, par exemple, dans l'avionique modulaire : le système est constitué de sous-systèmes communiquant par l'intermédiaire d'un réseau [Ari97]. Ces sous-systèmes sont éventuellement fournis par différents partenaires industriels. Le fonctionnement global du système est défini par un intégrateur de système.

La présence de tampons apporte une certaine souplesse dans la conception d'une application temps réel dont les tâches communiquent. Les tâches sont activées à leur rythme mais il est nécessaire que la taille des tampons soit suffisante pour qu'il n'y ait pas de situation de débordement. La modularité induite par les tampons facilite le travail de l'intégrateur de système.

La vérification du respect des contraintes temporelles des tâches est accomplie grâce aux méthodes classiques de faisabilité de jeux de tâches périodiques. On cherche à vérifier des contraintes temporelles qui sont locales à chaque sous-système. D'autre part, il est nécessaire que le nombre de messages présents dans le tampon n'excède pas sa taille.

La théorie des files d'attente propose des critères de performance pour dimensionner les tampons. Les tâches qui consomment les messages dans les tampons sont alors soumises à une contrainte de précédence : elles sont activées dès qu'un message arrive dans le tampon.

Malheureusement, dans ce cas, la vérification du respect des contraintes temporelles locales à un sous-système est plus difficile.

Cette thèse explore une voie différente qui permet d'obtenir des garanties sur la faisabilité des tâches au détriment de la simplicité d'analyse des tampons. En particulier, nous supposons qu'il n'y a pas de contrainte de précédence entre les tâches d'un sous-système. Dans ce cas, tester la faisabilité d'un sous-système consiste à vérifier localement le respect des contraintes temporelles des tâches selon les tests classiques de la littérature (cf. section 2.2).

Si les tampons ne sont soumis qu'à la loi de conservation du débit (cf. définition 13 et 21), la faisabilité des tâches en est facilitée. En revanche, la difficulté est reportée sur le dimensionnement des tampons.

Lorsque la durée minimum entre deux arrivées de messages est connue, les critères maximums permettent d'avoir des garanties sur le bon fonctionnement de l'application. Ils sont particulièrement utiles pour les applications où le problème du débordement des tampons est critique. De plus, les bornes exhibées ne supposent pas l'utilisation d'un algorithme d'ordonnancement particulier. Elles peuvent donc être affinées si des hypothèses sont faites sur l'ordre d'exécution des différentes tâches.

Quand il n'existe pas de durée minimum entre deux arrivées de message, il est impossible d'avoir les mêmes garanties que le cas précédent. Les critères moyens ne peuvent en aucun cas être considérés comme des bornes. Par contre, ils peuvent servir à évaluer les caractéristiques du tampon de manière à diminuer le risque de débordement.

Toutefois, pour les applications où le débordement des tampons ne remet pas en cause le bon déroulement de leur exécution, les critères moyens s'avèrent suffisants. C'est particulièrement le cas lorsqu'il est rare que l'occupation du tampon soit supérieure à sa taille moyenne (cas où la probabilité de non débordement est élevée). Ainsi, il est possible de diminuer l'empreinte mémoire d'un sous-système.

Les contributions de cette thèse sont les suivantes :

- Des contributions de modélisation : nous avons proposé un modèle d'application temps réel avec tampons. Ce modèle permet d'utiliser, pour les tâches, les tests de faisabilités classiques de la littérature. En ce qui concerne les tampons de notre application, ils sont modélisés grâce à la théorie des files d'attente. Nous avons proposé une loi de service P afin de prendre en compte le caractère indépendant des tâches périodiques.

- Les contributions théoriques : nous avons étudié deux files d'attente : les files M/P/1 et P/P/1.

Nous proposons une résolution approchée de la file M/P/1 basée sur les files d'attente classiques telles que la file M/G/1 [SLNM04b, LSNM05]. Grâce à l'évaluation du temps de service moyen W_s de la loi P et de sa variance σ_s^2, il est possible d'utiliser les critères de performance de la théorie des files d'attente. Le temps d'attente et le taux d'occupation mesurés lors des simulations diffèrent, selon les jeux de tâches, de 0% à 10% avec les critères théoriques.

Nous proposons une résolution exacte de la file P/P/1 basée sur des résultats des réseaux ATM [LSN+03]. Cette résolution consiste à obtenir des bornes maximums sur le taux d'occupation et sur le temps d'attente des messages dans le tampon. Des simulations menées sur la file P/P/1 confirment les résultats attendus par la résolution exacte.

- Des contributions techniques : nous avons développé de nouvelles fonctionnalités pour Cheddar, un logiciel de modélisation et d'analyse d'applications temps réel [SLNM04a]. Ces fonctionnalités permettent de modéliser et d'analyser les tampons d'un système temps réel. Nous nous sommes basés sur les boites à outils disponibles dans Cheddar pour mettre en œuvre nos algorithmes de simulation des tampons.

Finalement, la présence de tampons apporte des solutions intéressantes à la conception de systèmes temps réel. Leur dimensionnement est un problème intéressant qui va concerner, à notre avis, de plus en plus de systèmes temps réel. Les critères que nous proposons dans le cadre de cette thèse apportent des éléments de solution à ce problème pour certains types de jeu de tâches. Néanmoins, de nombreux points restent à explorer pour généraliser et améliorer la qualité de ces résultats :

- Il faut affiner les approximations effectuées pour le calcul du temps de service moyen de loi P et de sa variance voire, trouver une résolution exacte de la file M/P/1.

- Il manque des critères moyens ou maximums lorsque plusieurs tâches consomment des messages dans un même tampon ou lorsque les délais critiques sont arbitraires. En outre, il faudrait étudier d'autres critères de performance afin d'avoir une analyse plus fine des tampons.

- Il manque une évaluation des critères de performance de la file "MP/P/1". Cette file décrit un tampon recevant à la fois des flux de messages dont le délai inter-arrivée minimum est connu et des flux de messages dont seul le délai inter-arrivée moyen peut être évalué. Nous retrouvons ce type de file dans l'environnement logiciel de PILOT (cf. chapitre 8).

- Nous n'avons pas abordé le cas où les messages possèdent des contraintes temporelles. La faisabilité consisterait, non seulement, à étudier les tâches et les tampons de l'application, mais également, à vérifier les contraintes des messages.

- Une autre perspective serait d'évaluer la possibilité d'obtenir, grâce à nos travaux, les temps de réponse d'un traitement réparti en s'appuyant sur une méthode telle que l'analyse holistique. Dans l'étude de cas PILOT, nous tentons d'appliquer une approche similaire (cf. chapitre 8).

Ce travail nous offre des perspectives à plus long terme. Nous voudrions appliquer les résultats des systèmes temps réel durs à des applications multimédias et/ou interactives pour obtenir des garanties sur leur qualité de service.

Certaines de ces applications contiennent des tampons (socket, ...) et cohabitent avec les autres tâches du système. L'ordonnancement se fait en temps partagé : l'ordonnanceur essaye d'accorder la même fraction de la charge processeur à chaque tâche.

Par conséquent il est difficile de considérer l'importance et l'urgence des différents traitements de l'application. Une tâche du système dont l'échéance est éloignée peut bloquer l'application alors qu'elle doit terminer rapidement un traitement.

Les concepteurs de ces applications estiment parfois qu'il suffit d'augmenter la puissance du processeur pour améliorer les performances de l'application. Or, nous savons que dans certains cas diminuer les temps d'exécution ne mène pas forcément à une amélioration de la qualité de service (anomalie de Graham [Gra01]). En outre, ils laissent généralement le soin aux utilisateurs de configurer les paramètres de l'application pour obtenir une qualité de service satisfaisante (dans les autres cas, les paramètres sont prédéfinis).

Nous pourrions donc proposer une approche basée sur les travaux de cette thèse. L'objectif serait d'obtenir un gain sur les performances de l'application et de proposer des outils de configuration automatiques qui garantissent une qualité de service spécifiée par l'utilisateur.

Liste des publications associées à cette thèse

- Revues :

[1] F. Singhoff, J. Legrand, L. Nana et L. Marcé, "Cheddar : a Flexible Real Time Scheduling Framework", To be published in the Ada Letters journal. Proceedings of the ACM SIGADA International Conference, Atlanta, 15-18 November, 2004.

- Conférences Internationales avec comité de lecture :

[2] J. Legrand, F. Singhoff, L. Nana et L. Marcé, "Performance Analysis of Buffers Shared by Independent Periodic Tasks", To be submitted to 17th Euromicro Conference on Real Time System (ECRTS 2005).

[3] F. Singhoff, J. Legrand, L. Nana et L. Marcé, "Extending Rate Monotonic Analysis when Tasks Share Buffers" DAta Systems in Aerospace conference (DASIA 2004), Nice, Juillet 2004.

[4] J. Legrand, F. Singhoff, L. Nana, L. Marcé, F. Dupont et H. Hafidi, "About Bounds of Buffers Shared by Periodic Tasks : the IRMA project", 15th IEEE-Euromicro International Conference of Real Time Systems, Wip session.

[5] L. Nana, J. Legrand, F. Singhoff et L. Marcé, "Modeling and Testing of PILOT Plans Interpretation Algorithms", Multi-conference on Computational Engineering in Systems Applicatons, CESA'03, IEEE, Lille, Juillet 2003.

- Conférences Nationales avec comité de lecture :

[6] J. Legrand, F. Singhoff, L. Nana, L. Marcé, F. Dupont et H. Hafidi, "Faisabilité d'une application de supervision : le projet IRMA", RTS'03, Paris, Avril 2003

[7] L. Nana, J. Legrand, F. Singhoff et L. Marcé, "Modélisation, simulation et test des algorithmes d'interprétation de plans PILOT", MOSIM'03, 4ème Conférence Francophone de Modélisation et Simulation - Organisation et Conduite d'Activités dans l'Industrie et les Services, Toulouse, Avril 2003.

- Rapports, délivrables contractuels :

[18] J. Legrand, F. Singhoff, L. Nana, L. Marcé. "Cheddar Framework Programmer's Guide", EA 2215 Technical report number legrand-01-2004, July 2004.

[9] F. Singhoff, J. Legrand, L. Nana et L. Marcé, "Cheddar User's Guide", EA 2215 Technical report number singhoff-01-2003, September 2003.

[10] F. Dupont, H. Hafidi, F. Singhoff, L. Marcé et J. Legrand, "Le projet IRMA", Rapport de fin de contrat régionale, Novembre 2002.

Bibliographie

[AB90] N. Audsley and A. Burns. Real-time System Scheduling. Technical Report YCS 134, University of York, 1990.

[ABJ01] J. Andersson, S. Baruah, and J. Jansson. Static-priority scheduling on multiprocessors. In *IEEE real-time systems symposium*, pages 193–202. IEEE Computer Society Press, December 2001.

[ABR+93] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. 1993.

[ABRT93] A. N. Audsley, A. Burns, M. Richardson, and K. Tindell. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, pages 284–292, 1993.

[AD90] M. Alabau and T. Dechaize. Tutorial : ordonnancement temps réel local et réparti. pages 113–139. Colloque de l'AFCET sur le temps réel, Nantes, octobre 1990.

[Ari97] Arinc. *Avionics Application Software Standard Interface*. The Arinc Committee, January 1997.

[AVS02] K. E. Avrachenkov, N. O. Vilchensky, and G. L. Shevlyaokov. Priority queueing with finite buffer size and randomized push-out mechanism. Technical report, INRIA technical Report number 4434, March 2002.

[BD98] J.P. Beauvais and A.M. Déplanche. Affectation de tâches dans un système temps réel réparti. *Technique et Science Informatiques*, 17(1) :87–106, janvier 1998.

[BF03] S. K. Baruah and S. Funk. Task assignment in heterogeneous multiprocessor platforms. Technical report, University of North Carolina, 2003.

[BGJ98] I. Blum, L. Gallon, and G. Juanole. La problématique de l'initialisation du protocole CSMS/CA ARINC 629 CP : modélisation et validation. pages 217–232. Real Time Systems, Paris, January 1998.

[BLA98] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *RTSS*, pages 286–295, 1998.

[BLOS94] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son. Assigning real-time tasks to homogeneous multiprocessor systems. January 1994.

[Bur00] B. Burchell. A3XX Maintenance : A First Look. *Overhaul and Maintenance Revue. URL : www.aviationnow.com*, August 2000.

[BW97] A. Burns and A. Wellings. *Real-time Systems and Programming Languages.* Addison Wesley, 1997.

[CCDP00] P. Chevochot, A. Colin, D. Decotigny, and I. Puaut. Are cots suitable for building distributed fault-tolerant hard real-time systems. *IPDPS 2000 Workshop*, 2000.

[CD96] K. Chen and L. Decreusefond. An Appriximate Analysis of Waiting Time in Multi-classes M/G/1/./EDF Queues. 24(1), May 1996.

[CDKM00] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Ordonnancement temps réel.* Hermès, 2000.

[CEKT02] S. Chakraborty, T. Erlebach, S. Kunzli, and L. Thiele. Schedulability of event-driven code blocks in real-time embedded systems. Technical Report TIK 130, ETH Zurich, 2002.

[CFH+03] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Andersson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. *To appear in Handbook on Scheduling Algorithms, Methods, and Models Joseph Y. Leung (ed.), Chapman Hall/CRC*, 2003.

[CKS02] L. Cucu, R. Kocif, and Y. Sorel. Precedence, periodicity and latency constraints. mars 2002.

[CMB00] D. Chen, A. K. Mok, and S. Baruah. Scheduling distributed real-time tasks in the DGMF model. 2000.

[CP00] P. Chevochot and I. Puaut. Holistic schedulability analysis of a fault-tolerant real-time distributed run-time support. pages 355–362. Proc. of the 7th International Conference on Real-Time Computing Systems and Applications (RTCSA'00), Cheju Island, South Korea, December 2000.

[DB99] I. Demeure and C. Bonnet. *Introduction aux systèmes temps réel.* Collection pédagogique de télécommunications, Hermès, septembre 1999.

[Dec96] T. Decker. Three Popular Avionics Databuses. *Real Time Magazine*, (2) :29–34, April 1996.

[DL78] S. Dhall and C. Liu. On a real time scheduling problem. *Operations Research*, 26 :127–140, 1978.

[FGB01] S. Funk, J. Goossens, and S. Baruah. On-line scheduling on uniform multiprocessors. In *IEEE real-time systems symposium*, pages 183–192. IEEE Computer Society Press, December 2001.

[Gal95] B. O. Gallmeister. *POSIX 4 : Programming for the Real World* . O'Reilly and Associates, January 1995.

[GBF02] J. Goossens, S. Baruha, and S. Funk. Real time scheduling on multiprocessors. mars 2002.

[GCG01] E. Grolleau and A. Choquet-Geniet. Ordonnancement de tâches temps réel en environnement multiprocesseur à l'aide de réseaux de petri. *Real-Time Systems, RTS 2001*, March 2001.

[GCG02] E. Grolleau and A. Choquet-Geniet. Off-line computation of real time schedules using petri nets. Technical report, LISI/ENSMA, 2002.

[GCGC98a] E. Grolleau, A. Choquet-Geniet, and F. Cottet. Cyclicité des ordonnancements au plus tôt des systèmes de tâches temps réel. Poitiers (France), jui 1998.

[GCGC98b] E. Grolleau, A. Choquet-Geniet, and F. Cottet. Ordonnancement optimal des systèmes de tâches temps réel à l'aide de réseaux de petri. Poitiers (France), 1998.

[GGBM91] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real time applications with SIGNAL. INRIA-RENNES, Rapport numéro 1446, 1991.

[GK96] M. Gagnaire and D. Kofman. *Réseaux Haut Débit : réseaux ATM, réseaux locaux, réseaux tout-optiques*. Masson-Inter Editions, Collection IIA, 1996.

[GL01] D. Geniet and G. Largeteau. Validation temporelle de systèmes de tâches temps réel strictes à durées variable à l'aide de langages rationnels. In *Actes MSR 2001*, page 243, Poitiers (France), oct 2001.

[Gra01] R. Graham. Bounds on the performance of scheduling algorithms. In *Computer and Job shop Scheduling Theory*, pages 165–227. John Wiley ans sons, 2001.

[GS96] R. Gupta and M. Spezialetti. A compact task graph representation for real-time scheduling. *Real-Time Systems*, 11(1) :71–102, 1996.

[Ham97] C. Hamann. On the quantitative specification of jitter constrained periodic streams, Janvier 1997.

[JB95] K. Jeffay and D. Bennett. A rate-based execution abstraction for multimedia computing. In *Network and Operating System Support for Digital Audio and Video*, volume 1018, pages 64–75, April 1995.

[JG99] K. Jeffay and S. Goddard. The rate-based execution model, 1999.

[JP86] M. Joseph and P. Pandya. Finding Response Time in a Real-Time System. *Computer Journal*, 29(5) :390–395, 1986.

[JSP91] K. Jeffay, D.L. Stone, and D.E. Poirier. YARTOS : Kernel support for efficient, predictable real-time systems. *Proc. joint Eight IEEE Workshop on Real-Time Operating Systems and Software and IFAC/IFIP Workshop on Real-Time Programming, Atlanta. Real-Time Systems Newsletter*, 7(4) :7–12, May 1991.

[Kle75a] L. Kleinrock. *Queueing Systems : Computer Application*. Wiley-interscience, 1975.

[Kle75b] L. Kleinrock. *Queueing Systems : theory.* Wiley-interscience, 1975.

[KLSC00] T. Kim, J. Lee, H. Shin, and N. Chang. Best case response time analysis for improved schedulability analysis of distributed real-time tasks. 2000.

[KRP+94] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real Time Analysis.* Kluwer Academic Publishers, 1994.

[KS02] A. Koubâa and Y. Song. Evaluation de performances d'ethernet commuté pour des applications temps réel. mars 2002.

[KT96] K. Kawachiya and H. Tokuda. Q-thread : A new execution model for dynamic qos control, avril 1996.

[KU94] P. Koopman and B. Upender. iCommunication Protocols for Embedded Systems. *Embedded System's Programming*, 7(11) :46–58, November 1994.

[Lar96] J. Larsson. Schedulite : a fixed priority scheduling analysis tool. Technical report, Master Thesis of Uppsala University. Report number ASTEC 96/01, October 1996.

[Leb98] L. Leboucher. *Algorithmique et Modélisation pour la Qualité de Service des Systèmes Répartis Temps Réel.* Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications de Paris, septembre 1998.

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

[Leh96] J. P. Lehocsky. Real Time Queueing Theory. pages 186–194. Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96), Washington, DC, USA, December 1996.

[LL73] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment. *Journal of the Association for Computing Machinery*, 20(1) :46–61, January 1973.

[LM80] J.Y.T Leung and M.L. Merril. A note on preemptive scheduling of periodic real time tasks. *Information processing Letters*, 3(11) :115–118, 1980.

[LM00] B. Lisper and P. Mellgren. Response time calculation and priority assignement with integer programming methods. 2000.

[LMM98] S. Lauzac, R. Melhem, and D. Mossé. An efficient RMS admission control and its application to multiprocessor scheduling. 1998.

[Loc86] C. D. Locke. *Best-effort Decision Making for Real-Time Scheduling.* Phd thesis, Dept. of computer science, Carnegie Melon, Pittsburgh, 1986.

[LS95] L. Leboucher and J.B. Stefani. Admission control for end to end distributed bindings. Centre National d'Etudes des Télécommunications (CNET), Lecture Notes in computer science, Teleservice and Multimedia Communications, vol 1052, November 1995.

[LSM+02] J. Legrand, F. Singhoff, L. Marcé, F. Dupont, and H. Hafidi. Le projet irma. Technical report, UBO/TNI, Rapport de fin de contrat régional ITR Recherche, Opération A1CA93, Programme 1045, novembre 2002.

[LSN+03] J. Legrand, F. Singhoff, L. Nana, L. Marcé, F. Dupont, and H. Hafidi. About Bounds of Buffers Shared by Periodic Tasks : the IRMA project. In the 15th Euromicro International Conference of Real Time Systems (WIP Session), Porto, July 2003.

[LSNM05] J. Legrand, F. Singhoff, L. Nana, and L. Marcé. Performance analysis of buffers shared by independent periodic tasks. *En cours de soumission à ECRTS'05*, decembre 2005.

[LW82] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real time tasks. *Performance Evaluation*, pages 237–250, 1982.

[MBCG98] A. Mok, S. Baruah, D. Chen, and S. Gorinsky. Real-time scheduling of multimedia tasks. 1998.

[MC96] A. Mok and D. Chen. A multiframe model for real-time tasks. (CS-TR-96-07), janvier 1996.

[Mok83] A.K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real Time Environnement.* PhD Thesis, Departement of electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, May 1983.

[NLSM03] L. Nana, J. Legrand, F. Singhoff, and L. Marcé. Modeling and Testing of PILOT Plans Interpretation Algorithms. IEEE Multi-conference on computational Engineering in Systems Applications, Lille, juillet 2003.

[OS93] Y. Oh and S. H. Son. Tight performance bounds of heuristics for a real-time scheduling problem. Technical Report CS-92-24, May 1993.

[OS95] Y. Oh and S.H. Son. Fixed-priority scheduling of periodic tasks on multiprocessor systems. 1995.

[PEP99] P. Pop, P. Eles, and Z. Peng. Schedulability-driven communication synthesis for time triggered embedded systems. pages 287–294. Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications, 1999.

[Pet81] J. L. Peterson. *Petri Net theory and the Modelling of Systems.* Prentice Hall, 1981.

[Pua02] I. Puaut. Cache analysis vs static cache locking for schedulability analysis in multitasking real-time systems. *ECRTS2002 Worshop on WCET*, 2002.

[RCK00] P. Richard, F. Cottet, and C. Kaiser. Validation temporelle d'un logiciel temps réel : application à un laminoir inductriel. *CIFA'2000*, juillet 2000.

[RCR01] P. Richard, F. Cottet, and M. Richard. On line Scheduling of Real Time Distributed Computers With Complex Communication Constraints. 7th Int.

Conf. on Engineering of Complex Computer Systems, Skovde (Sweden), June 2001.

[RD01] H. Roux and Anne-Marie Deplanche. Extension des réeaux de Petri T-temporels pour la modélisation de l'ordonnancement de tâches temps réel. In *Actes MSR 2001*, page 327, Nantes (France), oct 2001.

[Rit97] M. Ritter. Congestion detection methods and their impact on the performance of the ABR flow control. 1997.

[Riv98] N. Rivierre. *Ordonnancement temps réel centralisé, les cas préemptifs et non préemptifs.* Thèse de doctorat, Université de Versailles Saint Quentin, février 1998.

[Rob90] T. G. Robertazzi. *Computer Networks and Systems : queueing theory and performance evaluation.* Springer-Verlag, 1990.

[RPGC02] M. Richard, P.Richard, E. Grolleau, and F. Cottet. Contraintes de précédence et ordonnancement mono-processeur. mars 2002.

[SLF92] M. Sidi, H. Levy, and S. Fuhrmann. A Queueing Network with a Single Cyclically Roving Server. *Queueing systems, Theory and Applications*, 11 :121–144, 1992.

[SLNM04a] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar : a Flexible Real Time Scheduling Framework. International ACM SIGADA Conference, Atlanta, November 2004.

[SLNM04b] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Extending Rate Monotonic Analysis when Tasks Share Buffers. In the DAta Systems in Aerospace conference (DASIA 2004), Nice, July 2004.

[SRL90] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols : An Approach to real-time Synchronization. *IEEE Transactions on computers*, 39(9) :1175–1185, 1990.

[SSNB95] J. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo. Implications of Classical Scheduling Results For Real-Time Systems. *IEEE Computer*, 28(6) :16–25, June 1995.

[Sta92] I. Stavrakakis. A Considerate Priority Queueint System with Guaranteed Policy Fairness. pages 2151–2159. In the proceedings of IEEE Infocom'92 Conference, Florence, May 1992.

[Sun97] J. Sun. *Fixed Priority end to end Scheduling in distributed real time systems.* PhD Thesis, University of Illinois, 1997.

[Tak98] H. Takada. Scheduling a task including i/o blockings with the multiframe task model, 1998.

[TBW92] K. Tindell, A. Burns, and A. Welling. Mode changes in priority preemptive scheduled systems. 1992.

[TBW95] K. Tindell, A. Burns, and A. Welling. Calculating controller area network (CAN) message response times. 1995.

[TC94] K. W. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3) :117–134, April 1994.

[Tin92] K. Tindell. An extendible approach for analysing fixed priority hard real-time tasks. 1992.

[TZ99] P. Tsigas and Y. Zhang. Non-blocking data sharing in multiprocessor real-time systems. *RTCSA99*, 1999.

[Vah96] U. Vahalia. *UNIX Internals : the new frontiers*. Prentice Hall, 1996.

[Wil96] T. Williams. Designers looking for the road to distributed real-time systems. *Computer Design's Electronic Systems Technology and Design*, September 1996.

[WP00] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. pages 25–25, 2000.

[XP90] J. Xu and D. Parnas. Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusion Relations. *IEEE Transactions on Software Engineering*, 16(3) :360–369, March 1990.

[YAWM97] A. Youssef, H. Abdel-Wahab, and K. Maly. Inter-stream adaptation over group reservations. Technical Report TR-97-37, Old Dominion University, 30, 1997.

[ZF94] H. Zhang and D. Ferrari. Rate-Controlled Service Disciplines. *In journal of High Speed Netwokrs*, 4(3), 1994.

Annexe A

Calcul de la probabilité d'état de la file M/P/1

Nous proposons une expression de la probabilité d'état pour la file M/P/1. Dans l'état actuel de nos travaux, nous sommes en mesure de donner cette probabilité dans un intervalle de temps t. Mais pour obtenir la probabilité de non débordement, il faudrait que cette probabilité soit indépendante du temps.

Théorème 13 *La probabilité d'avoir k messages dans le tampon à l'instant précédent la consommation n est de :*

$$P_k(r_n + (n-1)P_{cons}) = \sum_{i=0}^{k} Pp_i(r_n + P_{cons} - r_{n-1}).P_{k+1-i}(r_{n-1} + (n-2)P_{cons})$$

Avec

$$P_k(r_n + (n-1)P_{cons} + \delta t) = P_{k+1}(r_n + (n-1)P_{cons} - \delta t)$$

Et

$$P_0(r_n + (n-1)P_{cons} + \delta t) = P_1(r_n + (n-1)P_{cons} - \delta t) + P_0(r_n + (n-1)P_{cons} - \delta t)$$

Eléments de preuve :

Nous cherchons à analyser le comportement de la file entre les 2 dernières consommations. Puis nous appliquons par récurrence le même raisonnement aux consommations précédentes.

Le temps entre 2 consommations successives est de :

$$r^i_{cons} + P_{cons} - r^{i-1}_{cons}$$

La probabilité d'avoir n productions pendant le service courant est de :

$$Pp_n(r^i_{cons} + P_{cons} - r^{i-1}_{cons})$$

La probabilité d'avoir n messages présent dans le tampon avant la i^{eme} consommation est de :

$$P_n(r^i_{cons} + (i-1).P_{cons})$$

Les différents scénarios qui permettent d'avoir k messages présent dans le tampon avant la i^{eme} consommation sont les suivants :

- 0 production entre les 2 dernières consommations et $k+1$ messages présents dans le tampon avant la consommation $i-1$.

$$Pp_0(r^i_{cons} + P_{cons} - r^{i-1}_{cons}).P_{k+1}(r^{i-1}_{cons} + (i-2).P_{cons})$$

- 1 production entre les 2 dernières consommations et k messages présent dans le tampon avant la consommation $i-1$.

$$Pp_1(r^i_{cons} + P_{cons} - r^{i-1}_{cons}).P_k(r^{i-1}_{cons} + (i-2).P_{cons})$$

- 2 productions entre les 2 dernières consommations et $k-1$ messages présent dans le tampon avant la consommation $i-1$.

$$Pp_2(r^i_{cons} + P_{cons} - r^{i-1}_{cons}).P_{k-1}(r^{i-1}_{cons} + (i-2).P_{cons})$$

- ...

- $k-1$ productions entre les 2 dernières consommations et 2 messages présent dans le tampon avant la consommation $i-1$.

$$Pp_{k-1}(r^i_{cons} + P_{cons} - r^{i-1}_{cons}).P_2(r^{i-1}_{cons} + (i-2).P_{cons})$$

- k productions entre les 2 dernières consommations et 1 ou 0 messages présent dans le tampon avant la consommation $i-1$.

$$Pp_k(r^i_{cons} + P_{cons} - r^{i-1}_{cons}).\Big(P_1(r^{i-1}_{cons} + (i-2).P_{cons}) + P_0(r^{i-1}_{cons} + (i-2).P_{cons})\Big)$$

□

Annexe B

Démonstration de la borne sur la charge processeur

Le théorème et la démonstration proviennent de l'article [LL73].

Théorème 14 *Pour un jeu de m tâches périodiques à priorité fixe dont le ratio entre les différentes périodes est inférieur à deux, la charge processeur doit respecter la condition suivante :*

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq m.(2^{\frac{1}{m}} - 1)$$

Eléments de preuve :

Soit T_1, T_2, ..., T_m les m tâches dont les capacités sont C_1, C_2, ..., C_m. Nous considérons que les périodes sont ordonnées selon la règle suivante : $P_m > P_{m-1} > ... > P_2 > P_1$. U est égal à la charge processeur. Nous désirons montrer que :

$$C_1 = P_2 - P_1$$

Nous supposons que :

$$C_1 = P_2 - P_1 + \Delta \quad \Delta > 0$$

Soit,

$$\begin{aligned}
C_1^{'} &= P_2 - P_1 \\
C_2^{'} &= C_2 + \Delta \\
C_3^{'} &= C_3 \\
&\vdots \\
C_{m-1}^{'} &= C_{m-1} \\
C_m^{'} &= C_m
\end{aligned}$$

$U^{'}$ est la charge processeur correspondant à ces nouvelles capacités. Nous avons :

$$U - U^{'} = \left(\frac{\Delta}{P_1}\right) - \left(\frac{\Delta}{P_2}\right) > 0$$

D'autre part, nous supposons que :

$$C_1 = P_2 - P_1 - \Delta \quad \Delta > 0$$

Soit,

$$\begin{aligned}
C_1^{''} &= P_2 - P_1 \\
C_2^{''} &= C_2 - 2\Delta \\
C_3^{''} &= C_3 \\
&\vdots \\
C_{m-1}^{''} &= C_{m-1} \\
C_m^{''} &= C_m
\end{aligned}$$

$U^{'}$ est la charge processeur correspondante. Nous avons :

$$U - U^{''} = \left(\frac{\Delta}{P_1}\right) + \left(\frac{2\Delta}{P_2}\right) > 0$$

Par conséquent, si U est la charge processeur minimum, alors :

$$C_1 = P_2 - P_1$$

De la même manière, nous pouvons montrer que :

$$\begin{aligned}
C_2 &= P_3 - P_2 \\
C_3 &= P_4 - P_3 \\
&\vdots \\
C_{m-1} &= P_m - P_{m-1}
\end{aligned}$$

Soit,

$$C_m = P_m - 2(C_1 + C_2 + ... + C_{m-1})$$

Pour simplifier la notation, nous posons l'équation suivante :

$$g_i = \frac{P_m - P_i}{P_i} \quad i = 1, 2, ..., m$$

Donc,

$$C_i = P_{i+1} - P_i = g_i.P_i - g_{i+1}.P_{i+1} \quad i = 1, 2, ..., m-1$$

Et,

$$C_m = P_m - 2.g_i.P_1$$

Et finalement,

$$\begin{array}{rl} U & = \sum_{i=1}^{m} \left(\frac{C_i}{P_i}\right) = \sum_{i=1}^{m-1} \left(g_i - g_{i+1}\left(\frac{P_{i+1}}{P_i}\right)\right) + 1 - 2.g_1.\left(\frac{P_1}{P_m}\right) \\ & = \sum_{i=1}^{m-1} \left(g_i - g_{i+1}\left(\frac{g_i+1}{g_{i+1}+1}\right)\right) + 1 - 2.\left(\frac{g_1}{g_1+1}\right) \\ & = 1 + g_1.\left(\frac{g_1-1}{g_1+1}\right) + \sum_{i=2}^{m-1} g_i.\left(\frac{g_i - g_{i-1}}{g_i+1}\right) \end{array} \quad \text{(B.1)}$$

La borne sur la charge processeur est égal à 1 si $g_i = 0$ pour tout i.

Pour trouver la borne maximum sur la charge processeur, l'équation B.1 doit être minimisée selon les g_j. Cela peut être fait en affectant aux variables g_j de la dérivée de U une valeur nulle et résolvant l'equation résultante :

$$\frac{\vartheta U}{\vartheta g_j} = \frac{g_j^2 + 2.g_j - g_{j-1}}{(g_j+1)^2} - \frac{g_{j+1}}{g_{j+1}+1} = 0 \quad j = 1, 2, ..., m-1 \quad \text{(B.2)}$$

Avec $g_0 = 1$.

Il peut être montré que la solution générale à l'équation B.2 est :

$$g_j = 2^{\frac{m-j}{m}} - 1 \quad j = 0, 1, ..., m-1$$

Soit,

$$U = m.(2^{\frac{1}{m}} - 1)$$

□

Annexe C

Rappel des notations utilisées

Notation	Dénomination	Définition
T_i	La tâche i	Traitement ou en ensemble de traitements.
r_i	Temps de réponse	Délai entre la date de première activation de la tâche i et l'instant de sa terminaison.
P_i	Période	Délai fixe entre deux activations successives.
C_i	Capacité	Borne sur la durée d'exécution de la tâche i.
D_i	Délai critique	Date à laquelle la tâche i doit avoir terminé son exécution. Elle est relative à l'activation de la tâche i.
s_i	Date de première activation	Date à laquelle la tâche est prête à être exécutée pour la première fois.
J_i	Gigue sur activation	Borne sur la latence entre l'activation théorique de la tâche i et son activation réelle.
B_i	Temps d'interblocage	Durée maximale de blocage de la tâche i par une tâche de priorité inférieure.

Tab. C.1 – Notations