Contribution à la robustesse des systèmes temps réel embarqués - Approches de dimensionnement du mécanisme de protection temporelle d'AUTOSAR OS

Dominique Bertrand

Université de Nantes

École doctorale

SCIENCES ET TECHNOLOGIES
DE L'INFORMATION ET DES MATHEMATIQUES

Année : 2011

Thèse de Doctorat de l'Université de Nantes

Spécialité : AUTOMATIQUE ET INFORMATIQUE APPLIQUÉE

Présentée et soutenue publiquement par :

Dominique Bertrand

le 24 janvier 2011

à l'École Centrale de Nantes

CONTRIBUTION À LA ROBUSTESSE DES SYSTÈMES TEMPS-RÉEL EMBARQUÉS

Approches de dimensionnement du mécanisme de protection temporelle d'AUTOSAR OS

Jury

Rapporteurs :	Françoise SIMONOT-LION	Professeur à l'École des Mines de Nancy
	Pascal RICHARD	Professeur à l'Université de Poitiers
Examinateurs :	Jean-Charles FABRE	Professeur à l'INP de Toulouse
	Yves SOREL	Directeur de recherche à l'INRIA
	Yvon TRINQUET	Professeur à l'Université de Nantes
	Sébastien FAUCOU	Maître de Conférences à l'Université de Nantes
Invité :	Philippe QUÉRÉ	Team Leader chez RENAULT

Directeur de thèse : **Yvon TRINQUET**
Co-encadrant : **Sébastien FAUCOU**
Laboratoire : Institut de Recherche en Communications et Cybernétique de Nantes

Table des matières

Introduction générale **1**

Partie I — Étude préliminaire **5**

Chapitre 1 — Systèmes temps réel embarqués et sûreté de fonctionnement **7**

1.1 Systèmes temps réel embarqués . . . 8
1.1.1 Systèmes embarqués . . . 8
1.1.2 Systèmes temps réel . . . 9
1.1.3 Systèmes temps réel embarqués et automobile . . . 9
1.2 Sûreté de fonctionnement . . . 10
1.2.1 Système et environnement . . . 10
1.2.2 Concepts de base de la SdF . . . 11
1.2.3 Faute, erreur, défaillance . . . 12
1.2.4 Moyens pour assurer la sûreté de fonctionnement . . . 14
1.2.5 SdF et normalisation . . . 16

Chapitre 2 — Protection temporelle **21**

2.1 Contrôle d'échéance . . . 22
2.2 Isolation temporelle . . . 22
2.2.1 Approche dirigée par le temps . . . 22
2.2.2 Ordonnancement hiérarchique . . . 23
2.2.3 Réservation CPU . . . 24
2.3 Protection temporelle dans AUTOSAR OS . . . 24
2.3.1 Le modèle OSEK/VDX . . . 25
2.3.2 La protection temporelle . . . 26

Chapitre 3 — Problématique de notre étude **31**

3.1 Définition du système et de son environnement . . . 31
3.2 Modèle de fautes considéré . . . 32

3.2.1 Sous-estimation du WCET . . . 34
3.3 Problématique . . . 36
3.3.1 Explication par un exemple . . . 36
3.3.2 Position du problème . . . 38

Partie II — Dimensionnement du mécanisme de protection temporelle d'AUTOSAR OS . . . 39

Chapitre 4 — Configuration naïve du mécanisme de protection . . . 41

4.1 Ordonnançabilité des systèmes déterministes . . . 42
4.1.1 Modèle d'application considéré . . . 42
4.1.2 Technique d'analyse d'ordonnançabilité . . . 43
4.2 Analyse . . . 46
4.2.1 Protection temporelle et analyse d'ordonnançabilité . . . 46
4.2.2 Première approche : approche implicite . . . 46
4.3 Simulations . . . 47
4.3.1 Paramètre de génération des applications . . . 47
4.3.2 Résultats de simulations . . . 50
4.4 Discussion . . . 52
4.4.1 À partir d'un exemple . . . 52
4.4.2 Relaxer les budgets – Un but pertinent . . . 54

Chapitre 5 — Configuration par analyse de sensibilité . . . 55

5.1 Analyse de sensibilité . . . 56
5.1.1 État de l'art . . . 56
5.1.2 Exemple de système . . . 57
5.2 Prise en compte de la criticité des tâches . . . 60
5.2.1 Extension du modèle d'application . . . 60
5.2.2 Exemple de système . . . 61
5.3 Prise en compte dans le calcul des budgets . . . 62
5.4 Simulations . . . 63
5.4.1 Ajout de la boîte de calcul des budgets . . . 63
5.4.2 Résultats de simulations . . . 63
5.5 Discussion . . . 67

Chapitre 6 — Configuration par analyse probabiliste . . . 69

6.1 État de l'art . . . 70
6.1.1 Analyse probabiliste et ordonnançabilité . . . 70
6.1.2 Première approche . . . 72
6.1.3 Seconde approche - cas d'un système à tâches concrètes . . . 75
6.1.4 Méthode retenue pour l'étude . . . 79
6.2 Modélisation du système . . . 79
6.2.1 Modification du modèle d'application . . . 79
6.2.2 Système multi-critique et choix des priorités . . . 80

6.2.3 Exemple de système et d'utilisation de l'algorithme . . . 82
6.3 Analyse . . . 83
6.3.1 Étude de systèmes multi-critiques, analyse probabiliste et ordonnançabilité 83
6.3.2 Effet du budget d'exécution sur l'exécution . . . 84
6.3.3 Position du problème . . . 85
6.3.4 Optimisation unidirectionnelle . . . 86
6.3.5 Exemple simple de système . . . 86
6.4 Simulations . . . 90
6.4.1 Paramètre de génération des applications . . . 90
6.4.2 Modification de la boîte de calcul des budgets . . . 92
6.4.3 Résultats de simulations . . . 93
6.5 Discussion . . . 95

Chapitre 7 — Utilisation additionnelle de la surveillance d'échéances 99

7.1 Principe de la surveillance d'échéances . . . 99
7.2 Apport de la surveillance d'échéances pour la sûreté de fonctionnement . . . 100
7.3 Apport de la surveillance d'échéances pour l'analyse . . . 101
7.3.1 Apport pour le calcul du backlog . . . 101
7.3.2 Difficultés . . . 101
7.3.3 Solution retenue . . . 104
7.4 Simulations . . . 105
7.4.1 Paramètre de génération des applications . . . 105
7.4.2 Résultats de simulations . . . 105
7.5 Discussion . . . 109
7.5.1 Analyse probabiliste et surveillance d'échéance . . . 109
7.5.2 Dimensionnement des budgets d'exécution . . . 110

Partie III — Étude de cas . . . 111

Chapitre 8 — Présentation de l'étude . . . 113

8.1 Présentation de la plate-forme matérielle AFP9328 . . . 114
8.2 Présentation du système d'exploitation Trampoline . . . 114
8.2.1 Un système d'exploitation compatible OSEK/VDX . . . 115
8.2.2 L'extension AUTOSAR . . . 116
8.2.3 Module de traces . . . 119
8.3 Portage sur cible AFP9328 . . . 120
8.3.1 Portage de la protection temporelle . . . 120
8.3.2 Portage du module de traces . . . 120
8.4 Précision du mécanisme de protection temporelle . . . 121

Chapitre 9 — Estimation du pire temps d'exécution . . . 123

9.1 Méthodes d'analyse des temps d'exécution . . . 124
9.1.1 Méthodes par analyse statique . . . 124
9.1.2 Méthodes par simulations . . . 124

9.2 Les différentes tâches et comportements temporels . . . 130
9.2.1 Détermination des différents paramètres . . . 130
9.2.2 Observateur de Luenberger . . . 131
9.2.3 Filtre de Kalman . . . 133
9.2.4 Tâche dont le temps d'exécution suit une loi de Gumbel . . . 135
9.2.5 Tâche de type « machine à états » . . . 138
9.3 Discussion . . . 140

Chapitre 10 — Étude d'une application . . . 143

10.1 Présentation de l'application . . . 143
10.2 Configurations du mécanisme de protection . . . 146
10.2.1 Première configuration . . . 146
10.2.2 Étude de sensibilité . . . 148
10.2.3 Étude probabiliste . . . 149
10.3 Discussion . . . 151
10.3.1 Effets des préemptions sur le temps d'exécution . . . 151
10.3.2 Sous-estimations et mécanisme de protection temporelle . . . 152

Conclusion 157

Annexes 159

Annexe A — Simulateur de système temps-réel . . . 161

A.1 Le simulateur TrueTime . . . 161
A.2 Modifications apportées au simulateur TrueTime . . . 164
A.2.1 Version basique (pas de mécanisme de protection) . . . 165
A.2.2 Version Protection par budget d'exécution . . . 165
A.2.3 Résumé des modifications . . . 166

Annexe B — Variables aléatoires et probabilités . . . 167

B.1 Variables aléatoires réelles . . . 167
B.1.1 Axiomatique de Kolmogorov . . . 167
B.1.2 Caractérisation d'une variable aléatoire . . . 168
B.1.3 Moments d'une variable aléatoire . . . 168
B.2 V. a. indépendantes et sommes de v. a. . . . 169
B.3 Variables aléatoires discrètes . . . 169

Annexe C — Cas d'étude simple . . . 171

C.1 Présentation de l'exemple . . . 171
C.2 Exemple de calcul de la probabilité d'ordonnançabilité . . . 172

Annexe D — Méthodes d'optimisation **175**

D.1 Optimisation unidirectionnelle (variable scalaire) 175
D.1.1 Méthode de Newton 175
D.1.2 Méthode par réduction d'intervalle 176
D.1.3 Méthode de Brent 176
D.2 Optimisation multi-directionnelle (variable vectorielle) 177
D.2.1 Méthodes analytiques 177
D.2.2 Méthodes heuristiques 178
D.2.3 Méta-heuristiques et algorithme évolutionnaire 178

« Rien ne sert de courir, il faut partir à point »

Le Lièvre et la tortue - Jean de La Fontaine

Introduction générale

Les systèmes informatiques embarqués occupent une place considérable dans notre vie quotidienne. Le développement de microprocesseurs de plus en plus petits et puissants et de moins en moins onéreux et consommateurs d'énergie en est la première cause. De tels composants semblent parfaitement destinés à remplacer ou à améliorer les performances de ce qui, auparavant, était réalisé de façon mécanique ou hydraulique. Les systèmes informatiques embarqués ont percé dans des domaines très variés. Hier, la première mission Apollo faisait appel à l'informatique. Aujourd'hui des téléphones mobiles de plus en plus complexes et polyvalents, des appareils électroménagers intelligents, des rames de métro sans conducteur, ..., voient le jour. Dans les domaines où l'informatique embarquée s'est engouffrée, il ne faut pas oublier le domaine de l'automobile qui, depuis quelques années, utilise ces technologies afin d'améliorer la sécurité, le confort ou la consommation. Depuis le premier véhicule automobile fonctionnel inventé par Joseph Cugnot en 1769, depuis la première utilisation de l'informatique embarquée pour commander l'injection de carburants en 1967, la route est bien longue. Parmi les technologies développées depuis les débuts de l'électronique embarquée, nous pouvons citer l'aide au freinage ABS conçu en 1973, l'airbag installé pour la première fois en 1981, le contrôleur de trajectoire ESP conçu en 1992, *etc.* Le nombre de fonctionnalités augmentant, la taille du code embarqué dans les véhicules fait de même. Depuis deux décennies, la tendance est exponentielle et la taille du code embarqué passe de 1.1 Ko pour une Citroën CX en 1980 à 100 millions de lignes de code pour une voiture de première classe en 2010.

Aujourd'hui, de multiples composants automobiles informatisés sont développés. Ces composants peuvent être critiques (éléments de sécurité) ou non (éléments de confort). La notion de criticité est liée aux dommages que peut entraîner une défaillance du composant. Plus une fonction est critique, plus celle-ci est donc assujettie à une sûreté de fonctionnement forte. Au cours de ces dernières années, la fiabilité des nouvelles technologies a connu quelques déboires. Ces soucis ont touché des composants de confort non critiques (verrouillage central, lève-vitres), mais également des composants aux comportements plus alarmants (un siège électrique qui avance seul), voire dangereux (baisse de puissance moteur soudaine sur autoroute). La fiabilité logicielle et matérielle est donc une condition nécessaire au développement de l'informatique embarquée, afin de garantir un bon fonctionnement, et ce, quels que soient les aléas de l'environnement. Dans ce cadre, une norme, l'ISO 26262 (ISO 26262, 2010) vient d'être publiée, qui vise à définir des niveaux de criticité et des contraintes sur le processus de développement des systèmes embarqués du domaine automobile.

L'observation des architectures embarquées automobile nous montre qu'elles évoluent actuellement d'une architecture fédérée à une architecture intégrée. Pour l'architecture fédérée, le constructeur joue le rôle d'un intégrateur qui assemble autour d'un réseau commun différentes unités de contrôles électroniques (ECUs), celles-ci étant conçues et validées par les équipementiers à partir des spécifications du constructeur (notamment celles qui régissent les échanges inter-calculateurs). Compte-tenu de la multiplication des fonctionnalités basées sur du logiciel, une telle architecture devient difficile à utiliser, notamment pour des raisons économiques. L'évolution se fait donc vers une architecture intégrée qui va permettre d'exécuter plusieurs prestations (applications) sur une même ECU. On passe donc progressivement du multiplexage de signaux entre calculateurs vers un multiplexage d'applications sur les calculateurs.

Cette modification profonde a bien sûr un lourd impact sur le processus. Elle entraîne la nécessité de standards, d'une part pour assurer l'interopérabilité entre composants cohabitants, et d'autre part pour améliorer la sûreté de fonctionnement globale du système. En effet, le multiplexage sur une même ECU d'applications de niveaux de criticité différents nécessite des services qui ne sont pas nécessaires dans une architecture fédérée. Ainsi, pour faciliter l'intégration de fonctions d'origines diverses, il est important de disposer des moyens techniques permettant d'assurer la ségrégation spatiale et la ségrégation temporelle entre les entités. Le périmètre de la notion d'entité s'étend, depuis l'application complète fournie par un équipementier, jusqu'à une granularité beaucoup plus fine : celle de la tâche gérée par un système d'exploitation temps réel.

C'est dans cette optique que le standard AUTOSAR (AUTomotive Open System ARchitecture (AUTOSAR, 2008)) spécifie l'architecture des systèmes informatiques embarqués automobiles afin de faciliter le développement de systèmes multi-concepteurs, de permettre la réutilisation de composants (standards) sur étagères et d'accroître la flexibilité de tout le processus de développement.

Dans notre étude, nous nous intéressons au mécanisme de protection temporelle proposé par le standard AUTOSAR OS. Celui-ci permet d'isoler chaque fonction (tâche) de l'application sur le plan du temps réel. Il permet de détecter et confiner une erreur avant qu'une défaillance plus importante ne se produise. Dans notre recherche, nous nous intéressons plus particulièrement au contrôle du temps maximal d'exécution pour chaque fonction. Si l'exécution d'une fonction n'est pas terminée lorsque son budget d'exécution est épuisé, alors une erreur est signalée. Déterminer les budgets nécessite la construction d'un modèle d'application basé sur un ensemble de paramètres à identifier. Cette identification comporte quelques difficultés techniques, notamment la détermination d'une estimation du pire temps d'exécution. Le but de notre travail est de proposer des stratégies de dimensionnement du mécanisme de protection temporelle qui tiennent compte de ces difficultés et de fournir ainsi des méthodes de calcul du budget d'exécution de chaque tâche qui soient sûres et robustes.

Notre étude se compose de trois parties. La première constitue une étude préliminaire permettant de spécifier les concepts de base. Nous définissons ce qu'est un système temps réel embarqué et les différentes concepts fondamentaux de la sûreté de fonctionnement. Nous nous référons très largement aux définitions proposées par Laprie *et al.* (1995). Nous présentons ensuite un état de l'art de différents mécanismes de contrôle temporel de l'exécution d'un système. Nous nous intéressons plus particulièrement au mécanisme de protection temporelle proposé par le standard AUTOSAR OS et modélisons le comportement de ce mécanisme. Nous terminons la première partie en délimitant le système étudié et son environnement, en

déterminant le modèle de fautes considéré et en posant la problématique générale de notre travail.

La seconde partie représente la partie centrale de la thèse. Elle permet de définir des méthodes de dimensionnement du mécanisme de protection temporelle d'AUTOSAR OS. Le chapitre 4 montre qu'un mécanisme de protection temporelle non correctement dimensionné peut entraîner une diminution de la qualité de service du système. Le chapitre 5 permet de définir une première méthode de dimensionnement basée sur les études de sensibilité proposées par Bini (2004) et Bini *et al.* (2006). Cette méthode permet de définir des budgets relaxés pour les tâches non critiques, tout en garantissant le contrôle des défaillances. Le chapitre 6 propose une seconde méthode basée sur une vision probabiliste du problème. Nous avons ainsi défini une méthode de calcul des budgets d'exécution basée sur l'analyse d'ordonnançabilité probabiliste proposée par Diaz *et al.* (2002, 2004). Néanmoins, cette méthode souffre de différents problèmes sur le plan calculatoire (complexité et sûreté des calculs) et conceptuel (possibilité d'occurrence de défaillance mal contrôlée dans le système). Nous montrons, chapitre 7, que la mise en place d'un mécanisme de surveillance d'échéance permet de résoudre ces différents problèmes.

La troisième partie vient compléter le travail d'analyse en mettant en place les différents éléments à travers une étude de cas sur une plate-forme réelle d'exécution. Le chapitre 8 présente la plate-forme matérielle utilisée pour l'étude (l'AFP9328 d'Armadeus System) et le système d'exploitation temps réel Trampoline. Trampoline est un noyau de système d'exploitation temps réel en licence libre, conçu par l'équipe Systèmes Temps Réel de l'IRCCyN et aligné sur le standard AUTOSAR OS. Il a fallu réaliser le portage de différents éléments (protection temporelle, module de trace d'exécution) pour la cible envisagée. Le chapitre 9 présente la mise en place d'une étude d'estimation des temps d'exécution. Cette estimation est réalisée par tests sur plate-forme de différents programmes permettant de définir des profils d'exécution afin de modéliser le comportement temporel de quelques tâches, ceci en vue de construire une application. Cette application est construite, chapitre 10, en déterminant les budgets d'exécution alloués à chaque tâche. Elle permet d'observer le comportement du mécanisme de protection.

Première partie

Étude préliminaire

« De tout inconnu le sage se méfie. »

Le renard, le loup, et le cheval - Jean de La Fontaine

Chapitre 1

Systèmes temps réel embarqués et sûreté de fonctionnement

Sommaire

1.1 Systèmes temps réel embarqués 8

1.1.1 Systèmes embarqués 8

1.1.2 Systèmes temps réel 9

1.1.3 Systèmes temps réel embarqués et automobile 9

1.2 Sûreté de fonctionnement 10

1.2.1 Système et environnement 10

1.2.2 Concepts de base de la SdF 11

1.2.3 Faute, erreur, défaillance 12

1.2.4 Moyens pour assurer la sûreté de fonctionnement 14

1.2.5 SdF et normalisation 16

Nous travaillons sur la robustesse des systèmes temps réel embarqués. La robustesse fait plus largement partie du domaine de la sûreté de fonctionnement. Nous rappelons dans ce chapitre les différentes notions de la sûreté de fonctionnement définies notamment par Laprie *et al.* (1995) ainsi que les différents standards internationaux caractérisant le domaine.

Le chapitre s'articule comme suit. La section 1 donne une définition de ce qu'est un système temps réel embarqué. Puis nous introduisons, section 2, les notions de base de la sûreté de fonctionnement.

1.1 Systèmes temps réel embarqués

1.1.1 Systèmes embarqués

Un système embarqué peut être défini comme un système électronique et informatique autonome dédié à une tâche bien précise. Ses ressources sont généralement limitées (volume et consommation limitées). Le premier système embarqué reconnu comme tel est le système de guidage des missions lunaires Apollo ("Apollo Guidance Computer" ou AGC) dès 1967 (première mission Apollo), développé par Charles Stark Draper du MIT, chargé du système de guidage inertiel du module. L'AGC était un ordinateur multitâches réalisant des traitements en temps réel. L'unité comportait 64 ko de mémoire morte, contenant l'ensemble des programmes, et 4 ko de mémoire vive. Le processeur était constitué de plus de 5000 portes logiques réalisées à l'aide de circuits intégrés. L'ensemble pesait environ 35 kg.

De nos jours, les systèmes embarqués se retrouvent dans de nombreux domaines. Parmi ceux-ci nous pouvons citer quelques exemples : le domaine de l'astronautique (fusées, satellites artificiels, sondes spatiales), du transport (automobile, rail, aéronautique), militaire (missiles, radars), des télécommunications (téléphonie, serveurs, box), de l'électroménager (téléviseurs, fours à micro-ondes, machines à laver), du multimédia (console de jeux, assistant personnel, navigateur GPS), *etc.* Depuis le développement de l'informatique miniature dû à l'invention du circuit intégré en 1958, puis du microprocesseur vers 1973, celle-ci s'immisce dans une large gamme de systèmes du plus gros (Airbus A380, fusée Arianne 4) au plus petit (téléphone portable).

Les systèmes embarqués exécutent des tâches prédéfinies selon un cahier des charges dont les contraintes peuvent être :

- de coût : le plus faible possible surtout si le système est produit en grande série ;
- d'empreinte spatiale : une taille généralement réduite ;
- d'empreinte mémoire : un espace mémoire limité ;
- de consommation énergétique : la plus faible possible due à l'utilisation généralement de batteries pour unique alimentation ;
- temporel : les temps d'exécution et l'échéance temporelle d'un service sont déterminés (les délais sont connus ou bornés a priori). Cette contrainte fait que généralement de tels systèmes ont des propriétés temps réel ;
- de sûreté de fonctionnement : certains systèmes embarqués subissant une défaillance peuvent mettre des vies humaines en danger. Ils sont alors dits critiques et ne doivent "jamais" faillir (même si le zéro défaillance est impossible) ;
- de sécurité : certains systèmes peuvent se révéler porteurs d'informations confidentielles pour leurs utilisateurs.

Ces contraintes sont généralement cumulées et deviennent de plus en plus strictes avec le temps et les avancées technologiques.

Dans ce mémoire, nous nous concentrons sur la contrainte de sûreté de fonctionnement (aspects temporels) pour des systèmes temps réel embarqués critiques, plus particulièrement dans le domaine automobile.

1.1.2 Systèmes temps réel

On peut distinguer trois classes de systèmes informatiques : les systèmes transformationnels (calculs scientifiques, bases de données) qui gèrent des programmes dont les résultats sont calculés à partir de données disponibles dès l'initialisation du programme et les instants de production des résultats ne sont pas contraints ; les systèmes interactifs (logiciels de bureautique, systèmes transactionnels) qui gèrent des programmes dont les résultats sont fonction de données produites par l'environnement du programme et les instants de production des résultats ne sont pas contraints ; les systèmes réactifs (systèmes critiques, automatisation, protocole) qui gèrent des programmes dont les résultats sont fonction de données produites par l'environnement du programme (le procédé à contrôler) et les instances de production des résultats sont contraints par les dynamiques du procédé. Dans ce dernier cas, on a affaire à des systèmes temps réel.

Le comportement d'un système informatique est qualifié de "temps réel" lorsqu'il est assujetti à l'évolution dynamique d'un procédé qui lui est connecté et qu'il doit piloter en "réagissant" à tous ses changements d'état.

On distingue le temps réel strict ou dur (hard real-time) et le temps réel souple ou mou (soft real-time) suivant l'importance accordée aux contraintes temporelles. Le temps réel strict ne tolère aucun dépassement de ces contraintes, ce qui est souvent le cas lorsque de tels dépassements peuvent conduire à des situations critiques, voire catastrophiques. À l'inverse, le temps réel souple s'accommode de dépassements des contraintes temporelles dans certaines limites au-delà desquelles le système devient inutilisable.

Pour le développement de systèmes de plus en plus complexes (multitâches, multi-processeur), l'utilisation d'un système d'exploitation (operating system ou OS) devient indispensable. Le système d'exploitation permet, comme son nom l'indique, d'exploiter les ressources matérielles de l'ordinateur ; il sert d'interface entre le matériel et les logiciels applicatifs. Parmi les services que peut fournir un système d'exploitation, nous pouvons citer la gestion du temps accordé pour l'exécution de chaque tâche (ordonnancement) ; la gestion des mémoires, la gestion des périphériques, *etc.*

De nombreux systèmes d'exploitation temps réel existent. Les plus connus sont QNX, RTLinux (extension temps réel du noyau Linux) ou VxWorks. D'autres systèmes d'exploitation plus spécifiques (car conformes à un standard) sont également disponibles comme osCAN ou MICROSAR OS de l'entreprise Vector, ou RTA-OSEK ou RTA-OS de l'entreprise ETAS pour l'automobile, le système d'exploitation INTEGRITY®-178B de l'entreprise Green Hills Software pour l'avionique (utilisé notamment dans l'Airbus A380).

1.1.3 Systèmes temps réel embarqués et automobile

L'histoire de l'électronique automobile commence probablement en 1967 lorsque l'injection électronique remplace l'injection mécanique dans le but d'améliorer le rendement moteur, grâce à un calculateur électronique. Depuis, l'électronique et les calculateurs s'immiscent dans de nombreux dispositifs et plus particulièrement dans les dispositifs de sécurité comme l'airbag (première commercialisation en 1981) ou le système d'anti-blocage des freins ABS (mis au

point par l'entreprise Bosch en 1973).

De nos jours, de plus en plus de fonctions sont installées à bord d'un véhicule automobile. Toujours pour la sécurité du conducteur et de ses passagers, de nombreux dispositifs sont inventés : à l'ABS viennent s'ajouter l'anti-patinage ASR et le contrôle de trajectoire ESP (Bosch 1995) ; un système d'airbag amélioré (airbags passagers, latéraux, *etc.*) ; des systèmes d'aide à la conduite comme l'allumage automatique des phares, l'essuyage automatique des vitres, le limiteur et le régulateur de vitesse, la vision de nuit ; et un confort au volant amélioré avec des systèmes de sièges de plus en plus perfectionnés, une climatisation et des systèmes de guidage GPS. Cette liste n'est pas du tout exhaustive et le nombre de dispositifs mis à disposition croît régulièrement.

Tous ces dispositifs sont contrôlés par des calculateurs (Electronic Control Unit ou ECU). Auparavant un calculateur était généralement dédié à une seule fonction. Cependant, avec le nombre croissant de dispositifs, les contraintes économiques et l'évolution des unités de calcul, chaque calculateur est (ou sera) affecté à différentes fonctions, celles-ci pouvant être plus ou moins critiques.

Dans ce mémoire, nous nous intéressons à l'aspect sûreté de fonctionnement et la continuité de service délivrée par du logiciel embarqué dans un véhicule automobile.

1.2 Sûreté de fonctionnement

La majorité des concepts présentés ci-dessous sont tirés du *Guide de la Sûreté de Fonctionnement* de Laprie *et al.* (1995) et des articles qui ont suivi de Aviznienis *et al.* (2004) et Arlat *et al.* (2006).

1.2.1 Système et environnement

Un système est une entité qui interagit avec d'autres entités (autres systèmes) matérielles, logicielles ou humaines et le monde physique. Ces autres systèmes constituent l'environnement du système étudié. La frontière du système est la limite commune entre le système et l'environnement. La fonction du système est ce que l'on attend du système en terme de fonctionnalités et de performances et est décrite par la spécification fonctionnelle. Le comportement d'un système est ce qu'il fait et peut être décrit par une séquence d'états. L'ensemble des états constitue l'espace d'état. La structure (architecture) d'un système est ce qui lui permet de générer son comportement. Un système peut être vu comme un ensemble de composants interconnectés en vue d'interagir. Chaque composant est alors un nouveau système, etc. La décomposition s'arrête quand un composant est considéré comme un composant atomique.

Le service délivré par un système est le comportement perçu par son ou ses utilisateurs. Un utilisateur est un autre système qui reçoit un service d'un système. L'interface de service est la partie de la frontière du système où le service est délivré. L'état externe est la partie de l'espace d'état qui est perçue à l'interface de service, la partie restante étant l'état interne. Un système implémente généralement plus d'une fonction, et délivre plus d'un service.

La figure 1.1 représente schématiquement l'ensemble des éléments énoncés ci-dessus. Les blocs pointillés à l'intérieur du système représentent ses différents composants et leurs inter-

actions. L'environnement étant également un système, il peut lui même être décomposé en plusieurs composants.

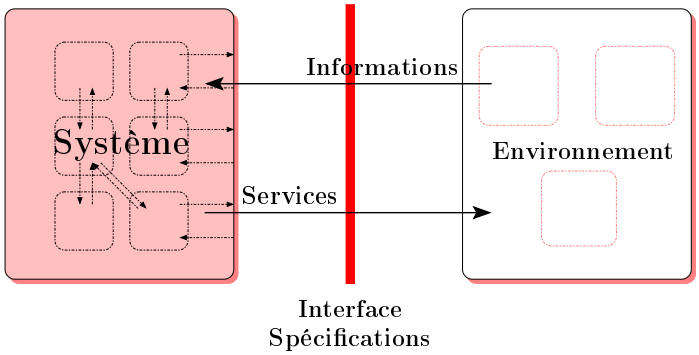


Figure 1.1 – Définition du système et de son environnement

1.2.2 Concepts de base de la SdF

Définition 1.1. *(Laprie et al., 1995) La sûreté de fonctionnement est la propriété qui permet aux utilisateurs d'un système de placer une confiance justifiée dans le service qu'il leur délivre. La sûreté de fonctionnement peut être alors vue comme l'aptitude à éviter les défaillances du service qui sont plus fréquentes ou plus graves que le niveau acceptable.*

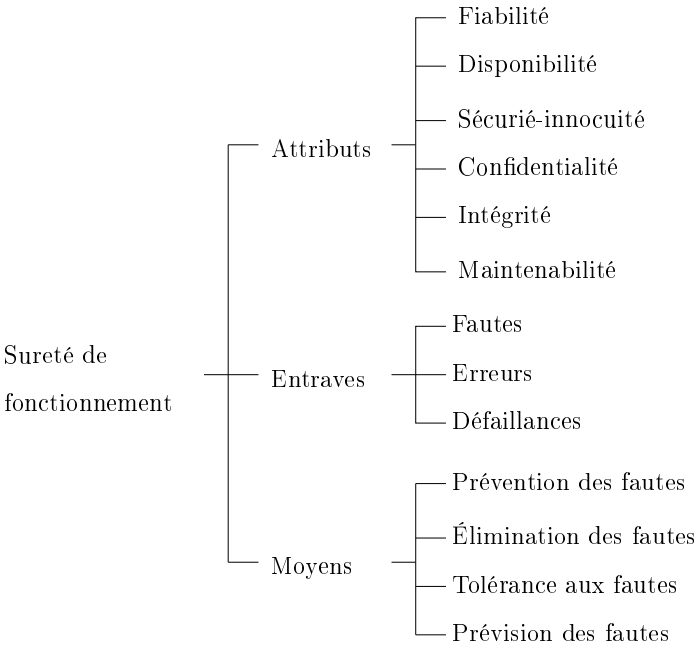


Figure 1.2 – Arbre de la sûreté de fonctionnement (Laprie *et al.*, 1995)

La sureté de fonctionnement peut être vue selon différentes propriétés qui permettent de définir ses attributs :

- la disponibilité : le fait d'être prêt à l'utilisation ;
- la fiabilité : la continuité de service ;
- la sécurité-innocuité : la non-occurrence de défaillances aux conséquences catastrophiques sur l'environnement ;
- la confidentialité : la non-occurrence de divulgations non-autorisées de l'information ;
- l'intégrité : la non-occurrence d'altérations inappropriées d'informations ;
- la maintenabilité : l'aptitude aux réparations et aux évolutions.

L'association à la confidentialité de l'intégrité et de la disponibilité conduit à la sécurité-immunité (attribut composite).

Un système est sujet à une défaillance lorsque son comportement n'est pas conforme à sa spécification. L'importance relative des critères dépend : de la nature de l'application, des exigences des utilisateurs et des conditions d'utilisation (environnement, etc.). Par exemple, dans le monde des systèmes embarqués en général, la fiabilité et la disponibilité sont deux critères importants ; dans le monde des communications (ex : commutateur téléphonique) c'est la disponibilité ; dans le domaine des bases de données, ce sont la disponibilité et la sécurité-immunité ; dans le domaine des transports (exemple : navigation, guidage, freinage) : la fiabilité, la sécurité-innocuité ainsi que la disponibilité sont les attributs les plus importants.

Au delà des attributs primaires déjà cités ci-dessus, nous pouvons définir des attributs secondaires qui affirment ou spécialisent les attributs primaires. Un exemple d'attribut secondaire est la robustesse.

Définition 1.2. *(Laprie et al., 1995) La robustesse est l'aptitude à conserver un service correct (sûreté de fonctionnement) en présence de fautes externes (une classe spécifique de fautes).*

D'autres attributs secondaires peuvent être définis à partir de la sécurité-immunité comme la responsabilité (disponibilité et intégrité de la personne qui a effectué une opération) ; l'authenticité (intégrité du contenu et de l'origine d'un message, et éventuellement d'autres informations comme l'instant d'émission) ; la non-réfutabilité (disponibilité et intégrité de l'émetteur d'un message ou du destinataire).

1.2.3 Faute, erreur, défaillance

Une défaillance du système survient lorsque le service délivré dévie de l'accomplissement de la fonction du système, c'est-à-dire de ce à quoi le système est destiné.

Défaillance

Les défaillances peuvent être classées selon plusieurs points de vue. Ces points de vue sont au nombre de quatre : le domaine de défaillance (défaillance en valeur, temporelle ou erratique), la détectabilité (défaillance signalée ou non), la perception de la défaillance (défaillance cohérente, le service incorrect est perçu identiquement par tous les utilisateurs, ou non), les conséquences de la défaillance (défaillance mineure ou catastrophique). La notion de sévérité de défaillance permet de définir la criticité.

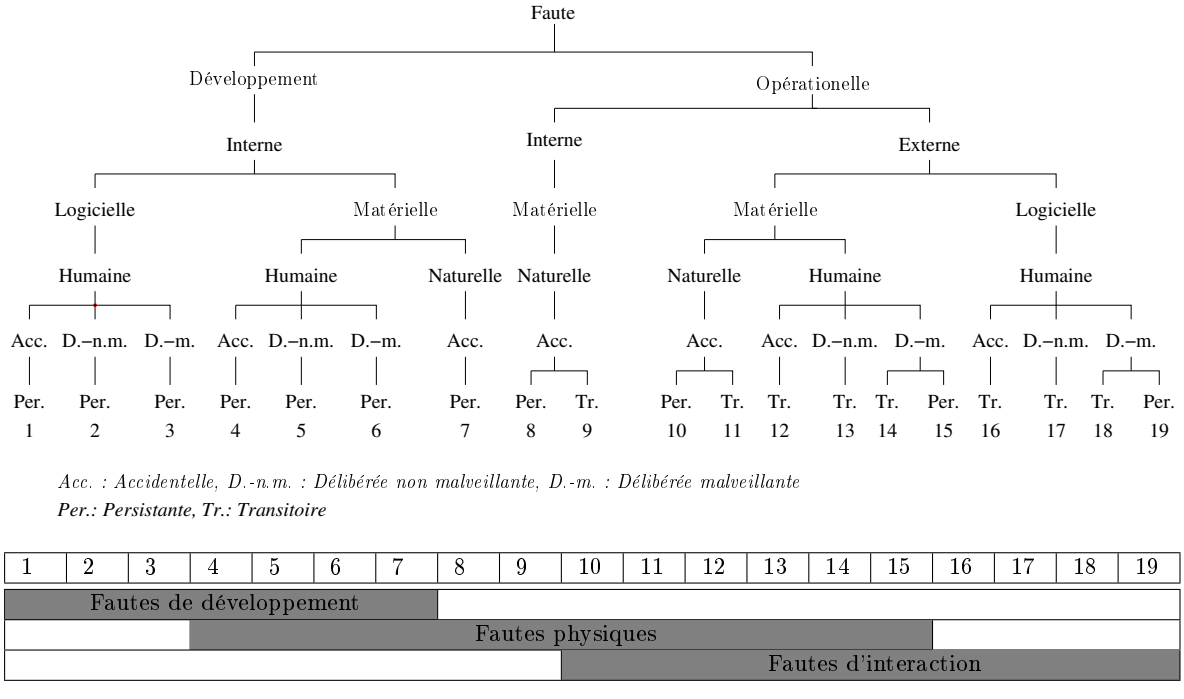
Définition 1.3. *(Laprie et al., 1995) La criticité d'un système est la plus haute sévérité de son mode de défaillance. La relation entre le mode de défaillance et la criticité dépend de l'application considérée.*

Erreur

Une erreur est une partie de l'espace d'états qui peut mener à une défaillance. La classification des erreurs est relative aux défaillances qu'elles peuvent causer : erreur en valeur ou temporelle, erreur détectée ou latente, erreur cohérente ou non, erreur mineure ou catastrophique.

Faute

Les fautes peuvent être classées selon huit points de vue : cause phénoménologique (faute naturelle ou humaine), nature (faute accidentelle, délibérée et non-malveillante, délibérée et malveillante), phase de création ou occurrence (faute de développement ou opérationnelle), frontière du système (faute interne ou externe), persistance (faute permanente ou transitoire), dimension (faute matérielle ou logicielle). Toutes les combinaisons ne sont pas possibles et trois groupes majeurs non-exclusifs peuvent être définis : les fautes de développement, les fautes physiques et les fautes d'interaction. La figure 1.3 représente l'ensemble des fautes qui peuvent exister sur un système en général.



Acc. : Accidentelle, D.-n.m. : Délibérée non malveillante, D.-m. : Délibérée malveillante
Per.: Persistante, Tr.: Transitoire

Figure 1.3 – Type de fautes rencontrées (Arlat *et al.*, 2006)

Une faute est dite active quand elle produit une erreur ; sinon elle est dite dormante. L'activation d'une faute est l'application d'une entrée sur un composant (sens large du terme)

amenant une faute dormante à devenir active. Une propagation d'erreur à travers un composant est causée par une transformation successive d'erreurs en d'autres erreurs. La propagation d'erreur à l'interface utilisateur cause une défaillance. L'activation d'une faute peut donc causer une erreur qui peut produire par propagation une défaillance. Mais cette défaillance peut alors causer d'autres fautes dans d'autres composants qui peuvent activer des erreurs, etc. On peut transcrire ce phénomène par la chaine suivante :

$$\ldots \longrightarrow \text{faute} \overset{activation}{\longrightarrow} \text{erreur} \overset{propagation}{\longrightarrow} \text{défaillance} \overset{cause}{\longrightarrow} \text{faute} \longrightarrow \ldots.$$

Une perturbation électromagnétique (faute) peut provoquer une corruption de l'EEPROM et l'exécution d'un code erroné (par exemple en changeant la valeur d'une constante) qui va durer plus longtemps. Cette exécution erronée peut être vue comme l'activation d'une faute entraînant une erreur temporelle. Le délai apporté au système peut, par propagation, entraîner la défaillance du service fourni par le code ou d'un autre service du système. La perturbation électromagnétique, cause de la défaillance, peut être classifiée en temps que faute opérationnelle, externe, matérielle, naturelle, accidentelle et transitoire (type 11 du tableau 1.3). Toutes les perturbations électromagnétiques n'engendrent pas de défaillance et la chaîne peut être interrompue. De plus, des moyens de protections peuvent être mis en place afin de limiter les effets, de briser la chaîne et, par la même, d'améliorer la sûreté de fonctionnement du système.

1.2.4 Moyens pour assurer la sûreté de fonctionnement

Quatre principaux moyens peuvent être utilisés afin d'assurer une bonne sûreté de fonctionnement : la prévention des fautes, la tolérance aux fautes, l'élimination des fautes et la prévision des fautes.

Prévention des fautes

La prévention des fautes consiste à empêcher l'introduction de fautes dans le système. Ceci peut être accompli par l'utilisation de méthodologies de conception de système contrôlées et de bonnes techniques de mise en œuvre.

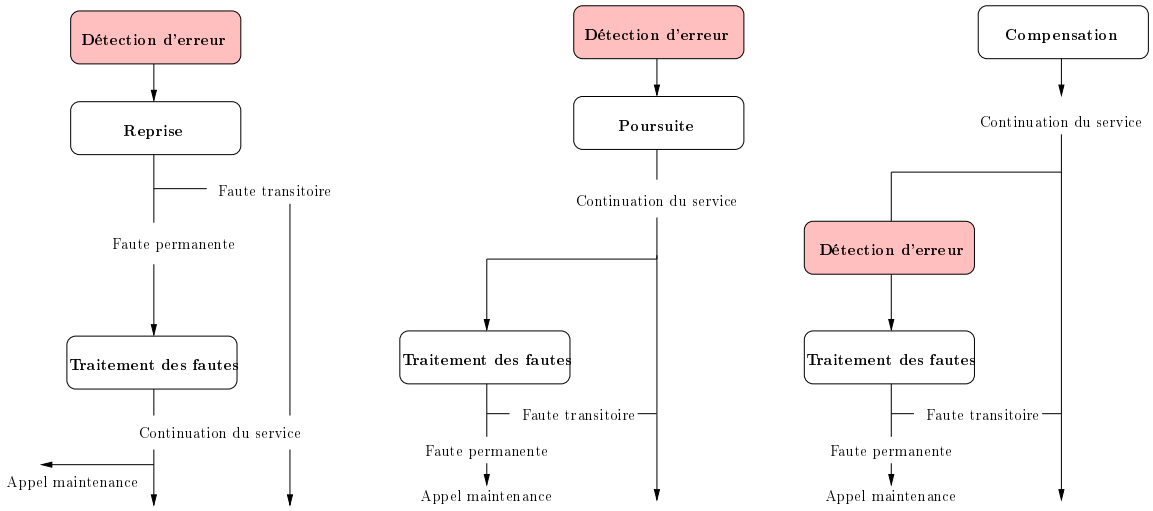
Tolérance aux fautes

La tolérance aux fautes est la façon d'éviter l'occurrence d'une défaillance de service en présence de fautes. La tolérance aux fautes peut être mise en œuvre sur les deux niveaux, erreurs et fautes, par le traitement des erreurs ou le traitement des fautes. Ces deux méthodes peuvent être combinées (traitement des erreurs puis traitement des fautes).

1. *Traitement des erreurs :* Le traitement des erreurs est mené en trois temps : la détection d'erreur, le diagnostic d'erreur et le recouvrement d'erreur.
   - Plusieurs méthodes sont utilisées pour la détection d'erreurs. On peut citer, par exemple, les codes détecteurs d'erreurs (redondance dans la représentation de l'information), la duplication et comparaison (deux ou plusieurs unités indépendantes en parallèle), les contrôles temporels et d'exécution (chien de garde, vérification du temps de réponse), le contrôle de vraisemblance (contrôle des valeurs en sortie), le contrôle

de données structurées, le diagnostic en ligne (programme d'auto-test afin de révéler les fautes dormantes), *etc.* ;
- Le diagnostic d'erreur permet d'estimer les dommages créés par l'erreur qui est détectée et par les erreurs éventuellement propagées avant la détection ;
- De même que pour la détection d'erreurs, plusieurs méthodes sont utilisées pour le recouvrement. On peut citer, par exemple, la reprise ou rollback recovery, figure 1.4(a), (où le système est ramené dans un état sain survenu avant l'occurrence d'erreur ; ceci passe par l'établissement de points de reprise, qui sont des états d'un processus qui peuvent ultérieurement nécessiter d'être restaurés), la poursuite ou rollforward recovery, figure 1.4(b), (où un nouvel état est trouvé à partir duquel le système peut fonctionner), la compensation d'erreur, figure 1.4(c), (où l'état erroné comporte suffisamment de redondance pour permettre la transformation de l'état erroné en un état exempt d'erreur).

2. *Traitement des fautes :* Le traitement des fautes est mené en deux temps : le diagnostic des fautes et la passivation des fautes.
   - Le diagnostic des fautes consiste à déterminer les causes des erreurs, en termes de localisation et de nature ;
   - Le but de la passivation des fautes est de prévenir les fautes avant qu'elles ne se propagent. Plusieurs méthodes peuvent être utilisées. On peut citer, par exemple, l'isolation (exclusion physique ou logique du composant fautif), la reconfiguration (modification de la structure interne du système), la ré-initialisation (choisit, met à jour et enregistre la nouvelle configuration).



(a) Reprise (b) Poursuite (c) Compensation

Figure 1.4 – Mécanisme de recouvrement d'erreur (Aviznienis *et al.*, 2004)

La tolérance aux fautes est un concept récursif et il est essentiel que les mécanismes qui implémentent la tolérance aux fautes soient protégés contre des fautes qui pourraient les affecter (Muller *et al.*, 1996).

Élimination des fautes

L'élimination des fautes pendant la phase de développement consiste au respect de trois étapes : la vérification (les critères de sélection concernent le respect de certaines propriétés), le diagnostic (le système vérifie-t-il toutes les propriétés ?) et la correction. Plusieurs techniques de vérification peuvent être mises en œuvre : vérification sans exécution dite vérification statique (analyse statique, model checking), vérification avec exécution dite vérification dynamique (exécution symbolique, tests). L'élimination des fautes pendant la phase d'utilisation du système consiste en la correction ou la maintenance préventive du système.

Prévision des fautes

La prévision des fautes est conduite en évaluant le comportement du système en présence de fautes. Cette évaluation a deux aspects : une évaluation qualitative ou ordinale (identifie, classifie et ordonne les modes de défaillances ou les combinaisons d'événements qui peuvent mener à des défaillances), une évaluation quantitative ou probabiliste (évalue le degré de satisfaction de certains des attributs de la sûreté de fonctionnement).

1.2.5 SdF et normalisation

Plusieurs normes de sûreté de fonctionnement existent et chaque domaine se dote de sa norme spécifique. Ces normes découlent pour la plupart de la norme générique CEI 61508 (Functional safety of electrical/electronic/programmable electronic safety-related systems) (CEI 61508, 1999).

Norme CEI 61508

La norme générique CEI 61508 (CEI 61508, 1999) s'appuie sur une approche basée sur la prévention des risques et la détermination du niveau d'intégrité :

- en terme de conséquence (nombre de personnes × gravité des blessures) ;
- de façon quantitative (risque quantifiable, taux de cible de défaillances par heure) ;
- de façon qualitative (conséquence, fréquence et durées d'expositions, probabilité d'un événement initiateur, probabilité d'une occurrence non voulue).

Elle définit alors des niveaux d'intégrité (System Integrity Level : SIL, SIL 1 à 4) qui peuvent être apparentés aux niveaux de criticités (SIL4 le plus critique). Ceux-ci définissent des niveaux de défaillances selon le niveau d'intégrité demandé. Ces niveaux sont définis à partir de quatre paramètres :

- Les conséquences : $C1$ (blessures légères), $C2$ (blessures permanentes sérieuses pour une ou plusieurs personnes ; décès d'une personne), $C3$ (décès de quelques personnes), $C4$ (nombre de morts conséquent)
- La fréquence : $F1$ (rare ou plus souvent), $F2$ (fréquent ou permanent) ;
- La contrôlabilité : $P1$ (possible sous certaines circonstances), $P2$ (la plupart du temps impossible) ;
- La probabilité d'occurrence : $W1$ (très faible), $W2$ (faible), $W3$ (relativement importante).

Le tableau 1.1 représente alors le niveau SIL en fonction des paramètres énoncés ci-dessus.

Conséquences	Fréquence	Contrôlabilité	Probabilité d'occurrence		
			W3	W2	W1
C1	*	*	-	-	-
C2	F1	P1	SIL 1	-	-
		P2	SIL 1	SIL 1	-
	F1	P1	SIL 2	SIL 1	SIL 1
		P2	SIL 3	SIL 2	SIL 1
C3	F1	*	SIL 3	SIL 3	SIL 2
	F2	*	SIL 4	SIL 3	SIL 3
C4	*	*	-	SIL 4	SIL 3

Tableau 1.1 – Définition des niveaux de criticité SIL (CEI 61508, 1999)

La norme définit deux modes de fonctionnement : à la demande qui introduit la notion de PFD (Probability of Failure on Demand) exprimée en taux de défaillances par an ; et en continu qui introduit la notion de taux de défaillances dangereuses (λ_d) exprimé en 10^{-x} par heure. Le tableau 1.2 donne la classification quantitative des niveaux de criticité.

Niveau d'intégrité	Fonctionnement en continu (taux de défaillances par heure)	Fonctionnement à la demande (probabilité de défaillance par heure)
SIL 4	$10^{-8} < \lambda_d \leq 10^{-9}$	$10^{-4} < PFD \leq 10^{-5}$
SIL 3	$10^{-7} < \lambda_d \leq 10^{-8}$	$10^{-3} < PFD \leq 10^{-4}$
SIL 2	$10^{-6} < \lambda_d \leq 10^{-7}$	$10^{-2} < PFD \leq 10^{-3}$
SIL 1	$10^{-5} < \lambda_d \leq 10^{-6}$	$10^{-1} < PFD \leq 10^{-2}$
SIL 0	Aucune exigence	

Tableau 1.2 – Classification quantitative des niveaux de criticité (CEI 61508, 1999)

Normes spécifiques et dérivées de la norme CEI 61508

La norme CEI 61508 a donné naissance à plusieurs normes dérivées, spécifiques à certains domaines. La figure 1.5 représente un aperçu de certaines de ces normes concernant la sûreté de fonctionnement.

Les exigences au niveau sûreté de fonctionnement dans le domaine ferroviaire sont définies par le CENELEC (European Committee for Electrotechnical Standardization). Les trois normes EN 50126, EN 50128 and EN 50129 représentent la base du procédé de certification en matière de fiabilité. Les normes EN 50128 ("Software for railway control and protection systems") et EN 50129 ("Safety related electronic systems for signaling") représentent l'application des règles proposées dans la norme IEC 61508 pour le domaine ferroviaire. La norme EN50128 (EN

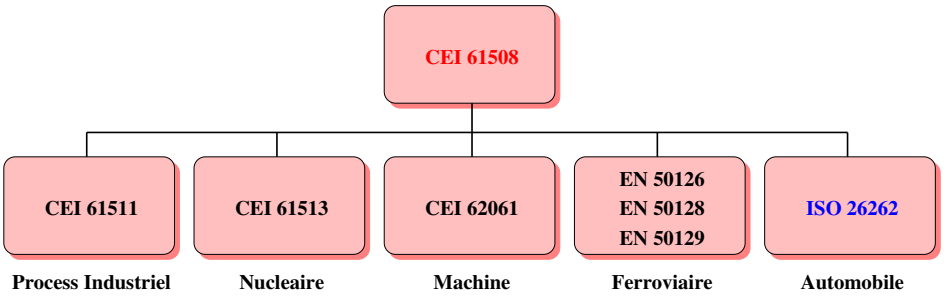


Figure 1.5 – La norme CEI 61508 et ces dérivées

50128, 2001) prend en considération cinq niveaux d'intégrité de la sécurité du système. Les différents systèmes sont catégorisés dans l'un ou l'autre de ces différents niveaux, des systèmes très critiques (SIL-4) tels que ceux affectés à la signalisation de sécurité jusqu'aux systèmes de gestion de l'information non critique (SIL-0). Le tableau 1.3 donne la classification quantitative spécifique des niveaux d'intégrité dans ce domaine.

Niveau d'intégrité de la sécurité	Probabilité maximale de défaillance	Facteur de réduction du risque demandé
SIL 4	$10^{-4} < p \leq 10^{-5}$	10^5 à 10^4
SIL 3	$10^{-3} < p \leq 10^{-4}$	10^4 à 10^3
SIL 2	$10^{-2} < p \leq 10^{-3}$	10^3 à 100
SIL 1	$10^{-1} < p \leq 10^{-2}$	100 à 10
SIL 0	Aucune exigence	

Tableau 1.3 – Classification quantitative des niveaux de criticité dans le domaine ferroviaire (EN 50128)

Les exigences au niveau sûreté de fonctionnement dans le domaine de l'aviation civile sont définies par les normes DO 178B (DO-178B, 1992) et ED-12B (ED-12B, 1999) Elles sont définies par 5 niveaux de criticité (de A à E) :

- Niveau A : Problème catastrophique - Sécurité du vol ou atterrissage compromis - Crash de l'avion
- Niveau B : Problème majeur entraînant des dégâts sérieux voire la mort de quelques occupants
- Niveau C : Problème sérieux entraînant un dysfonctionnement des équipements vitaux de l'appareil
- Niveau D : Problème pouvant perturber la sécurité du vol
- Niveau E : Problème sans effet sur la sécurité du vol

Ces 5 niveaux sont aussi appelés niveaux DAL (Design Assurance Level). Les niveaux sont établis par les études de sécurité. Ces études fixent alors le niveau DAL pour le matériel et le logiciel conformément aux normes de sécurité (EUROCAE ED-79 et SAE ARP4754 "Certification considerations for Highly-Integrated and Complex Aicraft Systems") ou directives de l'avionneur (ABD100, ABD200,...). Le niveau DAL d'un sous-système peut être différent du

niveau système à la condition que le niveau DAL du système soit atteint par une architecture matérielle/logicielle adéquate.

Dans le domaine automobile, la norme ISO 26262 (ISO 26262, 2010) s'inspire de la norme générique précédente pour définir des niveaux de criticité, ASIL (Automotive System Integrity Level). Quatre niveaux ASIL sont définis ASIL A à D, ASIL A définissant le plus bas niveau et ASIL D le plus haut niveau de criticité. En plus de ces niveaux de criticité, une classe QM (Quality Management) est définie . Une exigence notée QM n'est pas considérée comme une exigence de sûreté de fonctionnement. Ces niveaux sont définis à partir de trois paramètres :

- La sévérité : $S0$ (pas de blessé), $S1$ (blessés légers), $S2$ (blessés graves), $S3$ (blessés mortellement)
- La fréquence : $E1 = 0.001$ (événements rares : < 1 par an d'utilisation du véhicule), $E2 = 0.01$ (quelquefois : $< 1\%$ du temps d'utilisation du véhicule), $E3 = 0.1$ (assez souvent : 1% à 10% du temps d'utilisation du véhicule), $E4 = 1$ (souvent : 10% à 100% du temps d'utilisation du véhicule) ;
- La contrôlabilité : $C1 = 0.01$ (simplement contrôlable : moins d'une personne sur 100 n'est pas capable de contrôler la situation), $C2 = 0.1$ (normalement contrôlable : moins d'1 personne sur 10 n'est pas capable de contrôler la situation), $C3 = 1$ (incontrôlable : le conducteur moyen n'est pas capable de contrôler la situation).

Le tableau 1.4 représente alors le niveau ASIL en fonction des 3 paramètres énoncés ci-dessus.

Sévérité	Fréquence	Contrôlabilité		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	ASIL A
	E4	QM	ASIL A	ASIL B
S2	E1	QM	QM	QM
	E2	QM	QM	ASIL A
	E3	QM	ASIL A	ASIL B
	E4	ASIL A	ASIL B	ASIL C
S3	E1	QM	QM	ASIL A
	E2	QM	ASIL A	ASIL B
	E3	ASIL A	ASIL B	ASIL C
	E4	ASIL B	ASIL C	ASIL D

Tableau 1.4 – Définition des niveaux de criticité ASIL (ISO 26262, 2010)

« La méfiance est mère de la sûreté »

Le Chat et un vieux rat - Jean de La Fontaine

Chapitre 2

Protection temporelle

Sommaire

2.1 Contrôle d'échéance 22

2.2 Isolation temporelle 22

2.2.1 Approche dirigée par le temps 22

2.2.2 Ordonnancement hiérarchique 23

2.2.3 Réservation CPU 24

2.3 Protection temporelle dans AUTOSAR OS 24

2.3.1 Le modèle OSEK/VDX 25

2.3.2 La protection temporelle 26

La protection temporelle peut prendre plusieurs formes. Elle peut être réalisée au niveau de la défaillance (contrôle d'échéance) ou prévenir la défaillance en empêchant la propagation d'erreurs temporelles. Cette césure de la chaîne de défaillance passe par une isolation temporelle. Nous entendons par isolation temporelle le fait d'éviter la propagation d'erreur temporelle entre instances.

Dans notre étude, nous nous intéressons plus particulièrement au domaine automobile qui est régit par un ensemble de standards. Un des standard conduisant le monde automobile actuel est le standard AUTOSAR. Celui-ci définit des mécanismes de protection temporelle afin de se prémunir de défaillances.

Le chapitre s'articule comme suit. La section 1 introduit le contrôle d'échéance et montre ses limites quant à une protection temporelle efficace. Puis nous introduisons, section 2, les grandes approches d'isolation temporelle existantes. La section 3 permet de définir plus précisément le mécanisme de protection d'AUTOSAR OS, objet de notre étude.

2.1 Contrôle d'échéance

Le contrôle d'échéance (deadline monitoring) consiste à détecter l'exécution d'une instance au-delà de son échéance. Il permet de détecter une défaillance temporelle du système, mais est insuffisant pour assurer le confinement des erreurs. En effet, quand une échéance n'est pas respectée ce peut être dû à une erreur temporelle introduite par une autre tâche qui a interféré trop longtemps avec la tâche en défaut.

Exemple . Soit le système composé de trois tâches dont les différents paramètres sont présentés tableau 2.1. La politique d'ordonnancement retenue est préemptive, à priorités fixes, et les priorités sont choisies selon l'algorithme optimal dans notre cas "Rate Monotonic" (plus petite période, plus grande priorité).

Tâche	Priorité	Temps d'exécution	Échéance = Période
τ_1	Haute	1	5
τ_2	Moyenne	3	10
τ_3	Basse	5	15

Tableau 2.1 – Exemple de configuration

Supposons que toutes les tâches sont prêtes à s'exécuter à la date zéro. La figure 2.1 représente le chronogramme de l'exécution du système. Nous pouvons voir que toutes les tâches sont exécutées avant leur échéance.

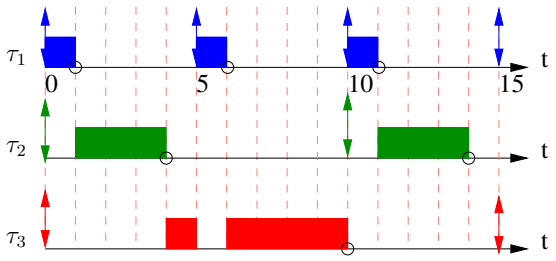


Figure 2.1 – Un exemple d'exécution du système présenté table 2.1

Maintenant considérons le cas où les tâches τ_1 et τ_2 ne se comportent pas correctement et ne respectent pas les paramètres donnés tableau 2.1. La figure 2.2 montre un exemple de l'insuffisance du contrôle d'échéance : si les tâches τ_1 et τ_2 respectent leurs échéances, leur comportement incorrect entraîne un dépassement d'échéance de la tâche τ_3.

◊

2.2 Isolation temporelle

2.2.1 Approche dirigée par le temps

L'approche dirigée par le temps ("time triggered") a été proposée par l'équipe de Hermann Kopetz à l'Université de technologie de Vienne (Kopetz, 1991; Kopetz *et al.*, 1995). Pour un

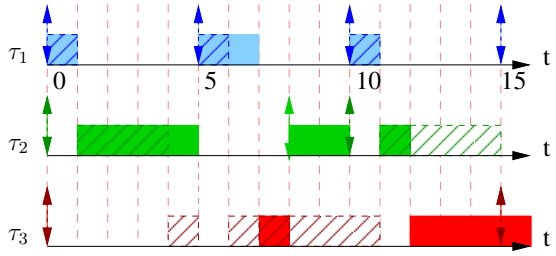


Figure 2.2 – Insuffisance du contrôle d'échéance

système totalement dirigé par le temps, toutes les activités du noyau (activations de tâches, changements de contexte, communications entre tâches) sont dirigées par un ordonnancement cyclique calculé hors-ligne. Pour garantir une complète prédictibilité, un système de surveillance est utilisé afin de détecter toutes violations à l'exécution des comportements temporels souhaités. L'isolation temporelle est donc réalisée par construction et tous les types de contraintes temporels peuvent être appliqués (latences, gigue et échéance).

Dans le domaine de l'automobile, une tentative visant à standardiser l'approche dirigée par le temps a été proposée à travers la spécification OSEKtime (2001). Une application OSEKtime est composée de deux types de tâches : les tâches TT (Time Triggered) dont l'exécution est contrôlée par le noyau et les tâches ET (Event Triggered : dirigées par les événements) qui ont les propriétés du standard OSEK/VDX (2005). Les tâches TT sont activées par le noyau suivant une table d'ordonnancement cyclique. Si plusieurs tâches sont activées en même temps, elles sont ordonnancées suivant une politique préemptive LIFO (la dernière tâche activée est la première à être exécutée). Afin de garantir une exécution des tâches TT déterministe, un contrôle d'échéance (deadline monitoring) est utilisé. Si l'exécution d'une instance de tâche TT n'est pas terminée à l'échéance, une routine de recouvrement est activée. Les tâches ET sont ordonnancées selon la politique classique d'OSEK/VDX dans les temps creux laissés par les tâches TT. $^{\text{TTP}}$OS (TTP OS, 2010) est un exemple de produit commercial conforme à OSEKtime.

Parmi d'autres tentatives pour adapter l'approche dirigée par le temps au domaine de l'automobile, nous pouvons citer PharOS (Chabrol *et al.*, 2009). PharOS est un noyau dérivé d'OASIS (David *et al.*, 1998) conçu pour adresser les besoins du domaine nucléaire. PharOS est principalement une technologie dirigée par le temps, même s'il offre un support pour des fonctions ET sous la forme de procédures d'interruption. Basé sur un ordonnancement statique, le noyau est en charge d'activer les modules et de transférer les données entre modules. Lorsque plusieurs modules sont activés en même temps, ils sont ordonnancés selon la politique d'ordonnancement EDF. Si l'exécution d'un module n'est pas achevée lors de son échéance, la tâche comportant le module est arrêtée et réintégrée dans le système d'une façon cohérente sur le plan temporel. De plus, une action de recouvrement est activée.

2.2.2 Ordonnancement hiérarchique

L'ordonnancement hiérarchique est une solution bien connue pour l'intégration de système. Il est utilisé par exemple dans l'avionique modulaire intégrée (IMA pour Integated Modular Avionic) à travers le standard ARINC (APEX pour APplication EXecutive) (ARINC, 1997)

pour le partitionnement et l'ordonnancement du système. Un noyau minimal est en charge de l'ordonnancement des partitions selon une politique prévisible. Le noyau minimal est également en charge de transmettre les différentes requêtes d'interruption aux partitions qui ont souscrit à la requête, de synchroniser l'accès aux ressources partagées entre partitions et de transmettre les données entre partitions. Chaque partition contient un sous-système complet (*i.e.* une application) avec un ordonnanceur local. Chaque partition peut utiliser une politique d'ordonnancement différente, adaptée à ses besoins.

Quelques tentatives ont été proposées visant à appliquer un tel modèle dans le domaine de l'automobile. Un exemple est le noyau DECOS Core OS (Schlager *et al.*, 2006). Ce système d'exploitation temps réel a été proposé dans le projet DECOS (Dependable Embedded Component and Systems), qui avait pour but de construire des architectures intégrées basées sur des technologies dirigées par le temps. Une des applications était le domaine de l'automobile. Dans COS (Core OS), les partitions sont ordonnancées selon une table cyclique calculée hors-ligne. Chaque partition occupe un ou plusieurs créneaux du cycle. Lorsqu'un un créneau est terminé, le noyau passe à la partition associée au prochain créneau. Les changements de contexte étant dirigés par le temps, l'isolation temporelle est assurée. COS implémente un modèle de communication basé sur sur des messages passant entre partitions. Pour favoriser le développement séparé des sous-systèmes, COS virtualise les périphériques partagés. Chaque partition peut donc potentiellement abriter un système d'exploitation local.

Une initiative pour permettre l'ordonnancement hiérarchique dans le standard AUTOSAR est proposée par Åsberg *et al.* (2009). Afin d'y parvenir, quelques modifications sont à apporter à l'architecture. La proposition consiste à utiliser un ordonnancement global compatible avec le modèle de ressources périodiques proposé par Shin et Lee (2003). Dans ce modèle, chaque partition reçoit périodiquement un montant constant de temps CPU. Une partition est stoppée lorsqu'elle a exploité son temps CPU, puis relancée au début de la prochaine période. Les partitions sont ordonnancées selon une politique préemptive à priorités fixes. Le système peut contenir un mélange de partitions et de tâches classiques. D'autres modifications sont nécessaires comme l'implémentation d'un protocole de synchronisation permettant le partage de ressources entre partitions.

2.2.3 Réservation CPU

Les systèmes temps réel intégrant des mécanismes basés sur une réservation de ressources ont été introduits dans les années 1990 par Mercer *et al.* (1994). Pour ces systèmes, chaque tâche reçoit X unités de temps d'une ressource R sur chaque période T. R peut être n'importe quelle ressource sous le contrôle du système d'exploitation temps réel : processeur, mémoire, *etc.* Une majorité d'études dans ce domaine se focalise sur la réservation de temps CPU. Une vue d'ensemble intéressante est donnée par Mancina *et al.* (2009).

2.3 Protection temporelle dans AUTOSAR OS

Dans la suite, nous allons souvent nous référer au standard AUTOSAR OS(AUTOSAR, 2008) c'est pourquoi nous décrivons dans cette partie les principales caractéristiques de cette

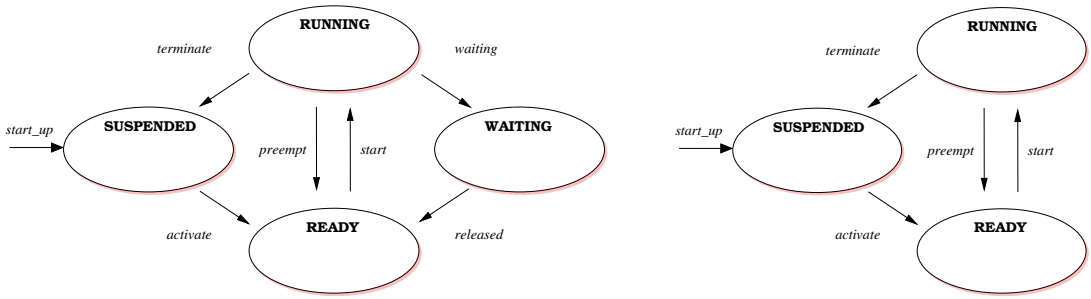
spécification.

Ses principales caractéristiques sont :
- configuration statique ;
- ordonnancement à priorité statique ;
- fonctions de protection (sur la mémoire, le temps, etc.) en ligne ;
- calibré pour des contrôleurs de petite capacité (puissance, mémoire).

Ces caractéristiques définissent le type de système d'exploitation utilisé à l'heure actuelle sur les unités de contrôle électronique (ECUs) dans le domaine de l'automobile.

2.3.1 Le modèle OSEK/VDX

Le modèle de tâche d'AUTOSAR reprend celui d'OSEK/VDX (OSEK/VDX, 2005). Un système OSEK/VDX est composé d'un ensemble de tâches et d'interruptions (ISR) concurrentes. Deux types de tâche sont distingués : les tâches basiques et les tâches étendues. Les tâches étendues sont différenciées des tâches basiques car elles peuvent attendre un événement. Le modèle d'état représenté figure 2.3(a) correspond à celui d'une tâche étendue. Dans le cas d'une tâche basique, l'état *waiting* n'existe pas (figure 2.3(b)).



(a) Tâche étendue (b) Tâche basique

Figure 2.3 – Modèle d'exécution des tâches dans OSEK/VDX OS

La figure 2.4 représente une séquence d'exécution possible, avec les différents états et transitions dans le cas d'une tâche basique.

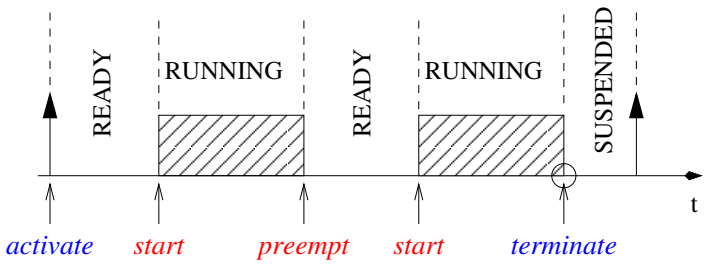


Figure 2.4 – Exemple d'exécution d'une tâche basique

L'activation d'une tâche est réalisée en utilisant le service *ActivateTask* du système d'exploitation. Dès qu'une tâche commence son exécution, elle consomme une des requêtes d'activation présente dans la file d'attente. Le nombre maximum de requêtes mémorisables dans la file d'attente est déterminé par un attribut de la tâche.

La terminaison d'une tâche est réalisée en utilisant le service *TerminateTask*. Une tâche ne peut être terminée que par elle-même. OSEK apporte la possibilité de lier la terminaison d'une tâche et l'activation d'une autre par le service *ChainTask*.

Les contraintes d'exclusions mutuelles sont gérées par la mise en place de ressources. Une ressource est partagée par plusieurs tâches et peut être prise (*GetRessource*) ou relâchée (*ReleaseRessource*) par une tâche. La gestion des ressources est réalisée en adoptant le protocole à priorité plafond d'OSEK (protocole IPCP (Burns et Wellings, 2001).

Un événement est émis par une tâche (*SetEvent*) et est reçu par une autre qui l'attend (*WaitEvent*). Les événements d'OSEK/VDX sont de type « privé ». La suppression d'un événement n'est pas implicite (appel au service *ClearEvent*).

2.3.2 La protection temporelle

Dans le modèle AUTOSAR, les sources d'erreurs entraînant potentiellement des défaillances temporelles sont de trois types :

1. Le temps d'exécution de la tâche dépasse la valeur du WCET annoncé (ou configuré) ;
2. La durée de blocage d'une ressource par une tâche dépasse la valeur annoncée ;
3. Une tâche est activée à une fréquence supérieure à la fréquence annoncée.

Pour détecter les erreurs temporelles, AUTOSAR OS définit trois mécanismes de protection :

1. La protection du temps d'exécution : il garantit une valeur statique maximale, appelée *Budget d'exécution*, sur la durée d'exécution des instances de la tâche ;
2. La protection du temps de blocage : il garantit une valeur statique maximale, appelée *Budget de blocage*, sur la durée des :
   - prises de ressources par une tâche/OsIsr2 (OS Interrupt Service Routine type 2) ;
   - suspensions par une Tâche/OsIsr des interruptions de l'OS ;
   - suspensions/annulations de toutes les interruptions par une Tâche/OsIsr.
3. La protection de l'intervalle entre deux activations successives : il garantit une valeur statique minimale, appelée *Fenêtre de temps (Time Frame)*, sur le temps s'écoulant entre :
   - deux activations d'une tâche/OsIsr2.

Le mécanisme est donc configuré à l'aide d'un ensemble de constantes à fixer par l'utilisateur durant la phase de développement. Les différentes constantes (paramètres) sont présentées tableau 2.2.

Paramètres à fixer pour chaque tâche	
TIMEFRAME	Intervalle minimal entre deux créations de deux instances d'une même tâche
EXECUTIONBUDGET	Temps d'exécution maximal d'une instance de tâche
MAXALLINTERRUPTLOCKTIME	Temps maximal de blocage d'une interruption par une instance de tâche
MAXOSINTERRUPTLOCKTIME	Temps maximal de blocage d'une interruption de catégorie 2 par une instance de tâche
Paramètres à fixer pour chaque tâche par ressource	
RESOURCELOCKTIME	Temps maximal de blocage d'une ressource par une tâche

Tableau 2.2 – Paramètres de configuration du mécanisme de protection temporelle

Certains paramètres sont à fixer pour chaque tâche et d'autres pour chaque ressource (partie supérieure et inférieure du tableau). Dans notre étude, nous nous concentrons sur des tâches indépendantes. Les seules paramètres qui nous intéressent sont donc le TIMEFRAME et l'EXECUTIONBUDGET. Par la suite, ces paramètres seront souvent notés respectivement F et B. Ces paramètres étant propre à chaque tâche τ_i, nous noterons F_i et B_i ces paramètres.

Nous avons modélisé le comportement du mécanisme de protection temporelle tel qu'il est implanté dans Trampoline. Nous avons utilisé un formalisme de type modèle état / transition, enrichi de variables continues pour modéliser les chiens de garde (type automate hybride (Alur *et al.*, 1993)). Le but est de partir du graphe représentant les différents états et transitions d'une tâche (tâche basique ou étendue d'OSEK/VDX) sans protection temporelle et d'y ajouter les modifications induites par le mécanisme.

La figure 2.5 présente un modèle de fonctionnement pour la surveillance de la fréquence maximale d'activation (variable f, borne F) et de la durée maximale d'exécution (variable b, borne B) dans le cas d'une tâche basique. Les états (ovales) grisés vides sont les états associés à l'exécution de la fonction de recouvrement de l'application fournie par l'utilisateur (`ProtectionHook`). Chaque détection d'erreur par le mécanisme de protection temporelle fait appel à une fonction de recouvrement. Celle-ci peut définir le redémarrage de l'application (`PRO_SHUTDOWN`), la terminaison de l'instance en cours (`PRO_TERMINATE8TASK`) ou la non prise en compte d'une activation (`PRO_IGNORE`). Chaque détection entraîne également l'émission d'une erreur : `E_OS_PRO_TIME` pour un dépassement de budget d'exécution ou `E_OS_PRO_ARRIVAL` pour une activation trop rapprochée.

Pour rester lisible, le fonctionnement du service de mémorisation des requêtes d'activation (qui reçoit les signaux *Queue* et émet *Dequeue*) n'est pas représenté. Le modèle ne fait pas apparaitre le fait que les signaux d'activation sont filtrés par le service de mémorisation avant d'être relayé au service de surveillance temporelle.

Le modèle pour les OsIsr se déduit facilement de celui donné pour les tâches basiques. La principale différence réside dans le service de mémorisation des requêtes d'activation dont la sémantique n'est pas standardisée dans le cas des Isr.

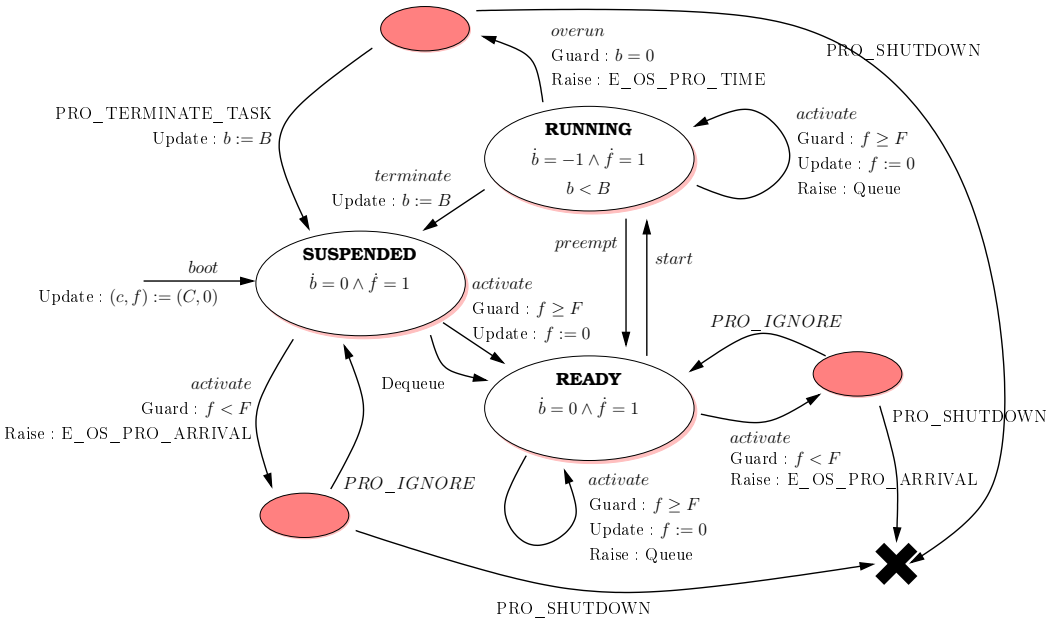


Figure 2.5 – Modèle du fonctionnement de la surveillance de la fréquence d'activation et de la durée d'exécution d'une tâche basique dans Trampoline

La figure 2.6 présente un modèle de fonctionnement pour la surveillance de la fréquence maximale d'activation (variable f, borne F) et de la durée maximale d'exécution (variable b, borne B) dans le cas d'une tâche étendue. En comparant les deux modèles, on s'aperçoit facilement que les activations des modules inclus dans les tâches ne sont pas surveillées de la même façon selon que la tâche est basique ou étendue.

En résumé, le comportement du mécanisme de protection temporelle dépend des paramètres configurés statiquement. Dans ce mémoire, nous proposons des méthodes permettant de fixer la valeur d'un de ces paramètres : le budget d'exécution.

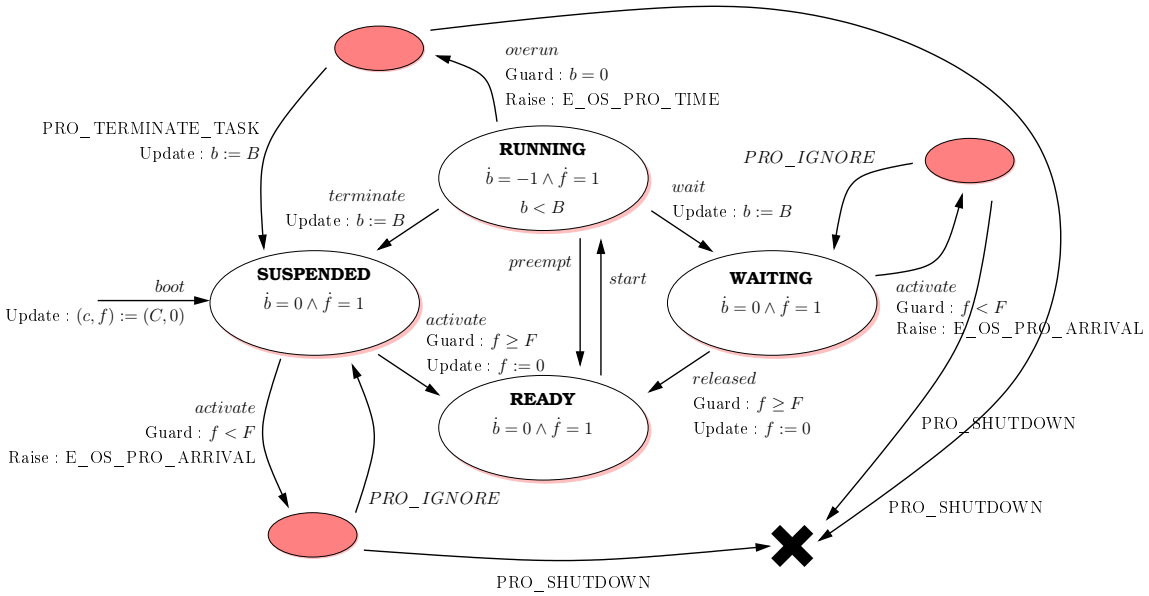


Figure 2.6 – Modèle du fonctionnement de la surveillance de la fréquence d'activation et de la durée d'exécution d'une tâche étendue dans Trampoline

« Le savant n'est pas l'homme qui fournit les vraies réponses, c'est celui qui pose les vraies questions »

Le Cru et le cuit - Claude Lévi-Strauss

Chapitre 3

Problématique de notre étude

Sommaire

3.1 Définition du système et de son environnement 31
3.2 Modèle de fautes considéré 32
3.2.1 Sous-estimation du WCET 34
3.3 Problématique 36
3.3.1 Explication par un exemple 36
3.3.2 Position du problème 38

Le présent chapitre sert à positionner notre étude en définissant le système étudié ainsi que son environnement (section 1). Nous contribuons à la robustesse temporelle des système temps réel embarqués. Une définition du modèle d'erreur adoptée (section 2) est nécessaire afin de bien cibler notre travail. Nous pouvons alors poser la problématique général de notre recherche (section 3).

3.1 Définition du système et de son environnement

Le système considéré dans notre étude est la partie du noyau d'un système d'exploitation qui implémente l'ordonnancement. Sa fonction est d'ordonnancer, autrement dit de multiplexer des traitements dans le temps (et éventuellement dans l'espace dans le cas d'un système multiprocesseur). Ce système est utilisé par des applications et d'autres parties du système d'exploitation. Pour notre étude, les applications et la plate-forme matérielle forment l'environnement du système.

Les applications sont constituées d'un ensemble de tâches à exécuter au cours du temps. Les requêtes d'activation de ces tâches représentent les entrées du système, celui-ci fixant une séquence temporelle d'exécution (sortie du système) afin de déterminer à chaque instant quelle instance de tâche occupe le (ou les) processeur(s). Dans cette étude, nous nous limitons

au cas mono-processeur. Étant donné que nous travaillons sur des systèmes temps réel, le temps d'exécution de chaque instance doit respecter certaines contraintes temporelles. Parmi celles-ci, nous pouvons définir l'échéance. L'échéance d'une instance est le temps maximal acceptable pour finir l'exécution de celle-ci (le temps séparant la requête d'activation et la fin de l'exécution est appelé temps de réponse de l'instance). Si l'exécution d'une instance ne respecte pas son échéance, des problèmes peuvent survenir pouvant avoir, dans certains cas (instances critiques), des conséquences graves. L'ordonnanceur doit ainsi fournir une séquence d'exécution cohérente par rapport aux différentes contraintes temporelles des applications.

La figure 3.1 représente schématiquement l'ensemble des éléments énoncés ci-dessus.

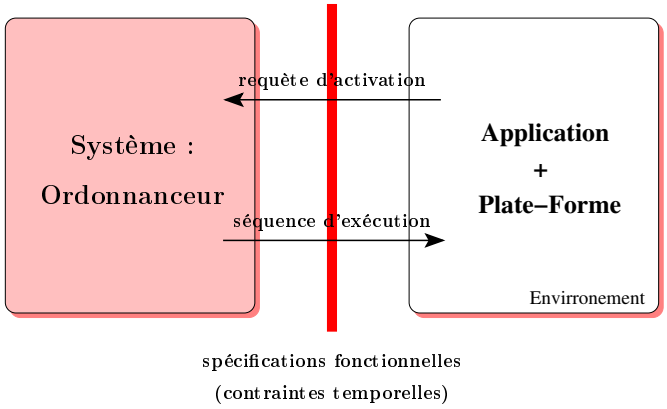


Figure 3.1 – Définition de notre système et de son environnement

Notre travail s'inscrit directement dans le but d'améliorer la robustesse de notre système c'est-à-dire le fait de conserver un ordonnancement fournissant une qualité contrôlée en présence de fautes provenant de l'application ou de la plate-forme matérielle. L'ensemble des fautes spécifiquement prises en compte est répertorié dans la section suivante.

3.2 Modèle de fautes considéré

Définissons précisément chaque terme énoncé section 1.2.3 dans le cas spécifique de notre étude.

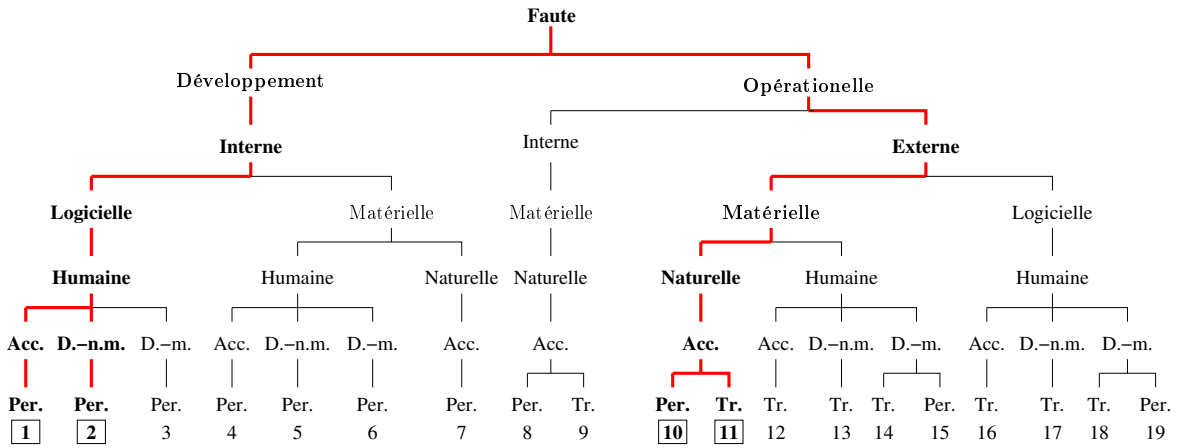
Nous parlons de défaillance lorsque l'exécution d'une des instances constituant l'application ne respecte pas son échéance.

Les erreurs qui peuvent conduire à une défaillance du service d'ordonnancement sont celles qui peuvent provoquer l'allongement du temps de réponse d'une instance. Nous répertorions trois principales causes :

- la création prématurée d'une instance diminuant la date d'arrivée d'activation d'une instance et pouvant par là même augmenter le temps de réponse d'une instance moins prioritaire. Le temps de réponse augmentant, l'instance peut manquer son échéance ;
- l'utilisation du processeur par l'instance en cours pendant une durée trop importante (supérieure à l'estimation de son pire temps d'exécution) entraînant un délai d'exécution

sur une tâche moins prioritaire et donc une augmentation du temps de réponse ;
- l'utilisation du processeur par l'instance en cours alors qu'elle occupe une ressource ou que les interruptions sont masquées, pendant une durée supérieure au pire temps d'exécution estimé de cette section critique.

Chaque erreur est causée par le comportement d'une instance incapable de se comporter conformément aux hypothèses. En contexte automobile, les fautes qui peuvent provoquer ces erreurs sont :
- un défaut d'un capteur, qui émet des signaux à destination du contrôleur d'interruption à une fréquence supérieure aux hypothèses ;
- une perturbation électromagnétique, qui conduit le contrôleur d'interruption à détecter et relayer une requête d'interruption non commandée par un périphérique ;
- une défaillance physique d'un circuit du microcontrôleur, qui conduit à l'exécution d'un code erroné ;
- une perturbation électromagnétique, qui provoque une corruption de l'EEPROM et l'exécution d'un code erroné (par exemple en changeant la valeur d'une constante) ;
- une défaillance humaine ou logicielle lors de la phase de conception, qui conduit à une sous-estimation du pire temps d'exécution d'un job ou d'une section critique.



Acc. : Accidentelle, D.-n.m. : Délibérée non malveillante, D.-m. : Délibérée malveillante
Per.: Persistante, Tr.: Transitoire

Figure 3.2 – Type de fautes considérées dans l'ensemble des fautes rencontrées

Il est bien évident que seul un sous-ensemble des occurrences de ces fautes conduit à des erreurs temporelles au niveau de l'ordonnanceur (les autres conduisent à des erreurs à d'autres niveaux, ou sur d'autres composants du système).

Cette analyse nous conduit au modèle d'erreur du tableau 3.1. Parmi les caractérisations possibles des fautes, nous retenons celle basée sur la dimension de la faute :
- les fautes matérielles (fautes de types 10 et 11 figure 3.2) ne peuvent pas être tolérées par le système sans l'utilisation d'une technique adéquate (redondance temporelle et/ou spatiale, checkpointing, etc.) ;

faute						erreur
libéllé	**cause**	**nature**	**occurrence**	**persistance**	**dimension**	
défaut capteur	naturelle	accidentelle	opérationnelle	permanente ou transitoire	matérielle	suractivation
perturbation élec.	naturelle	accidentelle	opérationnelle	transitoire	matérielle	
défaillance circuit	naturelle	accidentelle	opérationnelle	permanente ou transitoire	matérielle	surexécution
corruption EEPROM	naturelle	accidentelle	opérationnelle	permanente	matérielle	
sous-estimation WCET	humaine	accidentelle ou délibérée	développement	permanente	logicielle	

Tableau 3.1 – Modèle de faute et modèle d'erreur

– les fautes de développement logicielles (fautes de type 1 et 2 figure 3.2, celles-ci étant externes du point de vue du système retenu dû au découpage ordonnancement/applications), c'est-à-dire les fautes d'estimation du WCET dans notre cas, peuvent être tolérées naturellement par le système dans une certaine mesure.

3.2.1 Sous-estimation du WCET

Les erreurs liées à une sous-estimation du WCET sont celles qui ont la plus forte probabilité d'être activées. En effet, parmi les occurrences de fautes matérielles, seule une petite partie se propage jusqu'à l'interface de l'ordonnanceur (par exemple, une corruption du compteur ordinal conduira aussi bien à une erreur d'accès mémoire qu'à une sur-exécution). Par contre, une sous-estimation du WCET pendant la phase de développement peut entraîner facilement une surexécution, car le système est généralement dimensionné par rapport à la valeur pire-cas estimée.

Il existe deux grands types d'approche pour l'estimation du WCET :
– mener une campagne de tests en exécutant le code sur la plate-forme cible (réelle ou simulée) ;
– utiliser une technique d'analyse statique.

La première approche est simple à mettre en œuvre. Elle présente le désavantage de ne pas pouvoir fournir une borne fiable : dans le cas général, il est en effet impossible de garantir que les tests réalisés couvrent toutes les configurations possibles du logiciel et de l'architecture matérielle, et donc en particulier la configuration menant au pire temps d'exécution.

La seconde approche résout ce problème en utilisant des abstractions du code et de l'architecture matérielle pour produire un système suffisamment simple pour être exploré de manière exhaustive (Shaw, 1989; Kirner et Puschner, 2005). Elle présente plusieurs désavantages : tout d'abord, sa mise en œuvre n'est pas triviale et nécessite la plupart du temps l'intervention de l'intégrateur qui doit annoter le code pour que l'outil soit capable de réaliser les bonnes abstractions ; ensuite, les abstractions utilisées ne sont pas exactes et introduisent des confi-

gurations non accessibles par le système réel ; enfin, les mécanismes des processeurs modernes (pipeline, prédicteur de branchement, mémoires caches, etc.) sont complexes à modéliser.

Ainsi, l'estimation fournie par un outil de calcul de WCET (n') est (qu') une borne supérieure de la valeur réelle du WCET. La qualité de cette borne dépend de nombreux facteurs : qualité et quantité des annotations, complexité du code, complexité de l'architecture matérielle, technique utilisée par l'outil, etc.

Sehlberg (2005) réalise des comparaisons entre le WCET mesuré par simulation et le WCET estimé par l'outil commercial aiT sur une application issue du domaine des véhicules de construction (des détails sur l'architecture matérielle et sur le logiciel sont donnés dans le papier). Les résultats sont les suivants : le WCET estimé en fournissant à aiT un maximum d'annotations est 16% supérieur à la plus grande valeur mesurée. En fournissant le minimum d'annotations, le WCET estimé est près de 20% supérieur à la plus grande valeur mesurée.

Cassé *et al.* (2010) réalisent des comparaisons entre le WCET mesuré par simulation et le WCET estimé par l'outil universitaire OTAWA sur une application issue du domaine automobile et sur différents types d'architectures matérielles (des détails sur l'architecture matérielle et sur le logiciel sont donnés dans le papier). Les résultats obtenus montrent que le WCET estimé est 10% à 75% supérieur à la plus grande valeur mesurée.

En l'état actuel des connaissances, il semble donc impossible d'obtenir des valeurs de WCET qui sont à la fois sûres et précises. De nombreux travaux sont en cours pour résoudre ce problème, en explorant l'utilisation de techniques hybrides (génération de configuration par analyse statique puis exécution sur cible réelle ou simulée (Bernat *et al.*, 2002; Colin et Petters, 2003) ou génération de configuration par simulation et analyse statique des blocs (David et Puaut, 2004)), ou encore de techniques de model-checking (Cassez, 2010).

Comme souligné par Vestal (2007) et Baruah et Vestal (2008) une sous-estimation peu fiable peut être délibérée. Il est alors envisageable de sous-estimer le WCET des tâches les moins critiques afin de :

- minimiser le niveau de performance requis concernant le processeur ;
- minimiser l'effort requis pour produire une estimation précise et sûre du WCET.

Dans ce cas, la sous-estimation vient du fait de la non prise en compte d'une de valeurs d'exécution dont la probabilité d'occurrence est très faible. Un dépassement de la valeur maximale configurée peut être considérée comme une erreur. Cependant, il est alors nécessaire d'inclure un mécanisme de protection temporelle garantissant l'isolation temporelle des tâches et le recouvrement en cas de détection d'erreur.

Avec cette approche, le niveau de criticité influence le niveau d'assurance de l'estimation du temps d'exécution. Le modèle proposé par Vestal (2007), fait l'hypothèse que plus une tâche est critique, plus l'effort produit pour déterminer le pire temps d'exécution de celle-ci est important. Pour les tâches critiques, la connaissance d'une borne supérieure sûre du pire temps d'exécution est habituellement déterminée via des techniques d'analyses statiques. Par contre, pour les tâches non-critiques, l'étude du temps d'exécution est traditionnellement réalisée par tests sur les paramètres d'entrée. Dans ce cas, seule une estimation de la distribution des temps d'exécution est connue. Les pire-cas étant difficiles à produire, cette distribution peut être incomplète et le pire-cas non représenté.

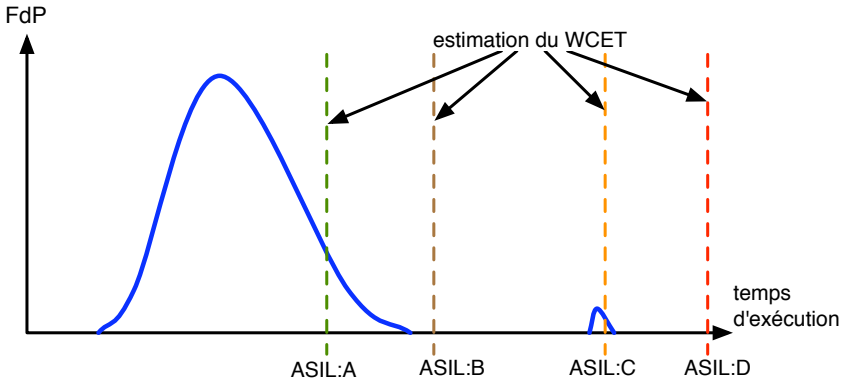


Figure 3.3 – Exemple d'estimation du WCET en fonction du niveau de criticité

La figure 3.3 représente un exemple d'estimation du WCET en fonction du niveau de criticité (ici les niveaux de criticité considérés sont les niveaux ASIL proposés par la norme ISO 26262 voir section 1.2.5). Nous reprenons les principes énoncés par Vestal (2007) qui envisage une sous-estimation possible du WCET dans le cas de tâches non-critiques (ASIL A) du fait des méthodes moins coûteuses utilisées pour réaliser l'estimation. Plus le niveau de criticité est important plus l'estimation est précise. Dans le cas d'une tâche très critique (ASIL D), l'estimation est sur-estimée (borne supérieure) ce qui permet d'avoir toute confiance dans la valeur pire-cas.

3.3 Problématique

Le mécanisme de protection temporelle considéré est celui proposé par AUTOSAR (2008) (présenté section 2.3). Notre étude porte sur l'identification du comportement du mécanisme de protection et de son dimensionnement.

3.3.1 Explication par un exemple

Soit l'application A composée de quatre tâches périodiques τ_1 à τ_4 dont les paramètres d'exécution sont présentés tableau 3.2. L'application est ordonnancée par un ordonnancement préemptif à priorités statiques.

	Priorité	Période	Échéance	Temps d'exécution
τ_1	4	5	5	1
τ_2	3	12	12	3
τ_3	2	25	25	5
τ_4	1	32	32	8

Tableau 3.2 – Paramètres d'exécution de l'application A

Le chronogramme, figure 3.4, représente la séquence d'exécution des différents travaux (ou différentes instances).

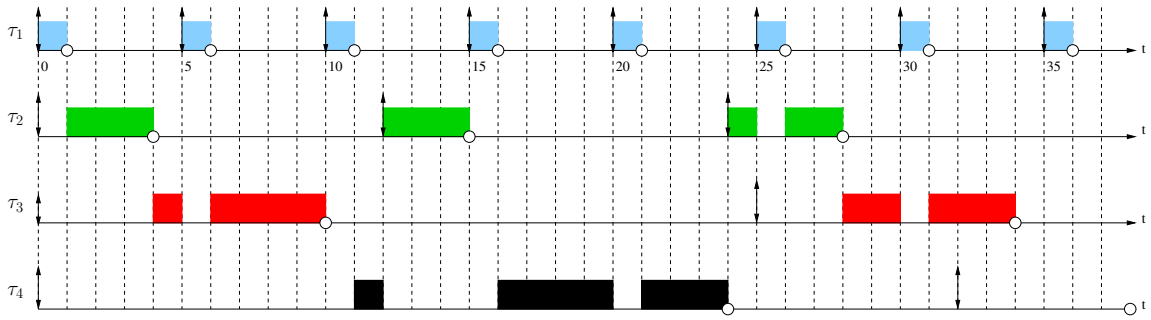


Figure 3.4 – Séquence d'exécution obtenue pour l'application A

Dans ce cas, tous les travaux respectent leur échéance puisqu'aucun travail ne se termine (rond vide) après son échéance (flèche vers le bas). Le temps s'écoulant entre la date d'activation (flèche vers le haut) et la terminaison d'un travail est appelé temps de réponse. Par exemple, pour la tâche τ_4, le travail étudié a un temps de réponse de 24 unités de temps.

Faisons varier le temps d'exécution des instances de τ_1. Chaque instance de τ_1 est affectée d'un temps d'exécution C_1. Nous pouvons alors déterminer, par calcul (voir section 4.1), le pire temps de réponse R_4 des instances de la tâche τ_4 en fonction du temps d'exécution C_1. La figure 3.5 représente cette évolution.

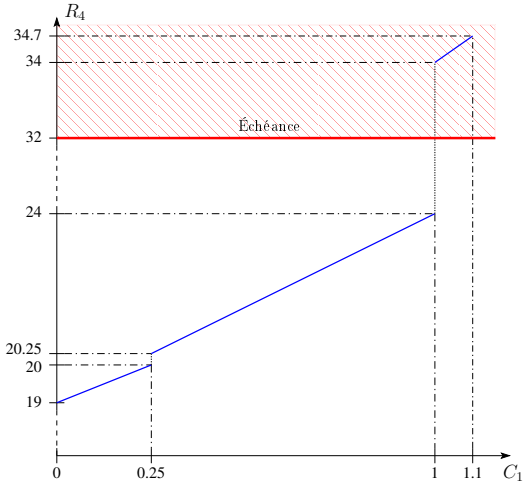


Figure 3.5 – Évolution du pire temps de réponse de la tâche τ_3 en fonction du temps d'exécution des instances de τ_1

Nous pouvons observer des non-linéarités (sauts) dans l'évolution du pire temps de réponse. Ceci est dû à l'ajout d'une interférence d'une instance plus prioritaire sur le temps de réponse.

L'échéance étant fixée à 32 le temps de réponse ne doit pas dépasser cette valeur. Pour obtenir $R_4 \leq 32$, nous ne devons pas dépasser la valeur de 1 unité de temps pour le temps d'exécution des instances de τ_1.

Le budget d'exécution de τ_1, B_1 peut ainsi être fixé à la valeur de 1 unité de temps. Cependant, cette valeur n'est valable que si les autres instances respectent les valeurs de temps d'exécution utilisées pour l'analyse. Ici, on pourrait donc fixer les valeurs des budgets d'exécution tels que $B_1 = 1,\ B_2 = 3,\ B_3 = 5,\ B_4 = 8$.

3.3.2 Position du problème

Le but de notre étude est de proposer des techniques de dimensionnement du budget d'exécution de chaque tâche de l'application en fonction de ses paramètres d'exécution.

La suite de l'étude est alors décomposée en trois grandes sections présentées dans la suite de ce rapport :

1. Dans un premier temps, nous allons tester le mécanisme de protection avec certains dimensionnements et observer son comportement en présence de fautes. Pour cela, nous aurons besoin de modéliser un système comportant des fautes afin de simuler au mieux le comportement du système.

2. Les conclusions de la première étape nous amènerons à proposer des méthodes de calcul des budgets d'exécution et de dimensionner au mieux le mécanisme de protection temporelle.

3. La dernière partie du rapport présente une étude de cas qui illustre l'utilisation des méthodes de calcul des budgets que nous proposons afin de configurer un système temps réel embarqué utilisant un noyau Trampoline.

Deuxième partie

Dimensionnement du mécanisme de protection temporelle d'AUTOSAR OS

« On hasarde de perdre en voulant trop gagner »

Le Héron - Jean de La Fontaine

Chapitre 4

Configuration naïve du mécanisme de protection

Sommaire

4.1 Ordonnançabilité des systèmes déterministes 42
4.1.1 Modèle d'application considéré 42
4.1.2 Technique d'analyse d'ordonnançabilité 43
4.2 Analyse 46
4.2.1 Protection temporelle et analyse d'ordonnançabilité 46
4.2.2 Première approche : approche implicite 46
4.3 Simulations 47
4.3.1 Paramètre de génération des applications 47
4.3.2 Résultats de simulations 50
4.4 Discussion 52
4.4.1 À partir d'un exemple 52
4.4.2 Relaxer les budgets – Un but pertinent 54

Le présent chapitre est consacré à l'étude du mécanisme de protection dimensionné de façon « naïve », c'est-à-dire en utilisant comme budget d'exécution le WCET annoncé pour chaque tâche. Cette étude fera appel à un outil de simulation afin de déterminer le comportement d'un système ainsi configuré.

Le chapitre s'articule comme suit. La section 1 permet de spécifier le modèle considéré et de faire un rappel sur les différentes techniques d'analyse d'ordonnançabilité. Puis nous introduisons, section 2, le dimensionnement du budget retenu dans ce chapitre. Des simulations, section 3, permettent d'afficher le comportement du système en présence ou non du mécanisme de protection. Puis nous discutons, section 5, des résultats obtenus et des perspectives possibles.

4.1 Ordonnançabilité des systèmes déterministes

4.1.1 Modèle d'application considéré

Nous supposons le système composé d'un ensemble de n tâches, $S = \{\tau_1, \ldots, \tau_n\}$. Trois types de tâche peuvent être discernés selon leur loi d'activation : les tâches périodiques, les tâches sporadiques et les tâches apériodiques. Les tâches périodiques sont caractérisées par une période d'activation ; les tâches sporadiques garantissent un intervalle minimum entre deux activations appelés pseudo-période. Aucune précision n'est donnée sur l'activation des tâches apériodiques. Dans cette étude, nous nous limitons au cas des tâche périodiques. Une tâche sporadique peut être considérée dans le pire cas comme une tâche périodique de période égale à sa pseudo-période.

Chaque tâche périodique τ_i peut être vue comme une séquence infinie d'instances $\tau_{i,j}$. Chaque instance $\tau_{i,j}$ est activée à la date $a_{i,j} = a_{i,j-1} + T_i$, doit être terminée avant son échéance absolue $d_{i,j} = a_{i,j} + D_i$ et possède un temps d'exécution $c_{i,j}$. Son exécution se termine à la date $f_{i,j}$, le temps de réponse $r_{i,j}$ étant la durée séparant l'activation de la terminaison d'une instance. La figure 4.1 représente un exemple d'exécution d'une instance d'une tâche périodique ainsi que ses différents paramètres. Le temps séparant le bloc (1) du bloc (2) représente un moment de préemption de l'instance considérée. La date de commencement $s_{i,j}$ dans cet exemple n'est pas simultanée avec l'activation car d'autres tâches doivent s'exécuter.

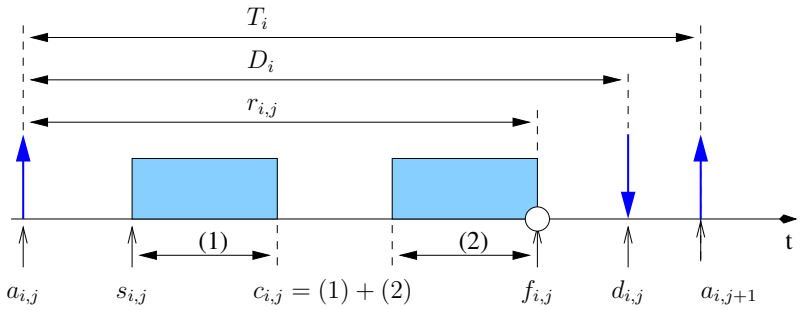


Figure 4.1 – Tâche périodique et ses différents paramètres

Nous considérons que les tâches sont ordonnancées par un ordonnancement préemptif à priorité fixe. Dans ce cas, chaque tâche est affectée d'une priorité π_i qui détermine quelle tâche doit s'exécuter à chaque instant. Dans la suite de ce rapport, sauf notification contraire, les tâches sont classées par ordre de priorité décroissante (la tâche la plus prioritaire est la tâche τ_1)

Chaque tâche τ_i est ainsi caractérisée par sa période T_i, son pire temps d'exécution C_i, son niveau de criticité L_i, sa priorité π_i et son échéance relative D_i. Nous nous limitons à l'étude de tâches à échéance contrainte ($D_i \leq T_i$) ou à échéance sur requête ($D_i = T_i$). Un système S peut donc être vu comme un ensemble de n tâches, chaque tâche τ_i étant caractérisée par un 4-uplet $\big(C_i, D_i, T_i\pi_i\big)$.

$$S = \{\tau_i\}_{i=1\ldots n} = \big\{\big(C_i, D_i, T_i, \pi_i\big)\big\}_{i=1\ldots n}$$

Ce modèle représente le modèle de base considéré. Celui-ci sera complété à travers les différents chapitres de notre étude.

4.1.2 Technique d'analyse d'ordonnançabilité

Définissons tout d'abord les notions de base de l'analyse d'ordonnançabilité.

Définition 4.1. *Un système composé de n tâches, $S = \{\tau_1, \ldots, \tau_n\}$, est dit ordonnançable ssi il existe une politique d'ordonnancement tel que chaque instance de tâche $\tau_{i,j}$ respecte son échéance*

Définition 4.2. *Un système $S = \{\tau_1, \ldots, \tau_n\}$ est dit faisable par une certaine politique d'ordonnancement P ssi ce système soumis à la politique P est ordonnançable.*

Faisons maintenant un petit rappel historique des principales méthodes d'analyses d'ordonnançabilité. Nous nous limitons au cas de systèmes composés uniquement de tâches périodiques. Dans ce cas d'étude, les principaux paramètres considérés pour chaque tâche τ_i sont sa période T_i, son pire temps d'exécution C_i, sa priorité π_i et son échéance relative D_i.

Liu et Layland (1973) posent les bases de l'analyse d'ordonnançabilité. Ils supposent un système composé d'un ensemble de tâches périodiques, indépendantes, à échéance sur requête, ordonnancé par un ordonnancement mono-processeur préemptif à priorités fixes. Sous ces hypothèses, ils définissent l'instant critique et prouvent les théorèmes suivants

Définition 4.3. *L'instant critique de niveau de i est défini comme étant l'instant d'activation de l'instance de la tâche de niveau de priorité π_i ayant le plus important temps de réponse*

Théorème 4.1. *Pour un ensemble de tâches périodiques à échéance sur requête, ordonnancé par une politique d'ordonnancement à priorité fixe, l'instant critique d'une tâche survient lorsque la date d'activation d'une de ses instances coïncide avec celles d'une instance de toutes les autres tâches plus prioritaires qu'elle.*

Théorème 4.2. *Dans la classe des politiques d'ordonnancement préemptifs à priorités fixes, et pour une des configurations de tâches indépendantes périodiques, synchrone et à échéance sur requêtes, la politique « Rate Monotonic » (plus la période est petite, plus la priorité affectée est grande) est "optimale" c'est-à-dire que si une telle configuration de tâches est ordonnançable en appliquant une affectation arbitraire des priorités, elle l'est aussi pour la politique « Rate Monotonic ».*

Théorème 4.3. *Pour un ensemble de n tâches périodiques indépendantes à échéance sur requête ordonnancé par une politique d'ordonnancement « Rate Monotonic », une condition suffisante d'ordonnançabilité est*

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n.(2^{1/n} - 1) \tag{4.1}$$

L'hypothèse d'échéance sur requête peut être relaxée. Si nous considérons le cas de tâches à échéance contrainte ($D_i \leq T_i$), la politique d'ordonnancement "optimale" est « Deadline Monotonic » (plus l'échéance relative est petite, plus la priorité affectée est grande), et la condition suffisante d'ordonnançabilité est

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

Les conditions d'ordonnançabilité énoncées ci-dessus ne sont que des conditions suffisantes et non nécessaire.

Exemple . Prenons l'exemple du système composé de 3 tâches périodiques de paramètres d'exécution tels que

$$\begin{array}{llll} T_1 = 4; & C_1 = 1; & D_1 = T_1; & \pi_1 = 3; \\ T_2 = 6; & C_2 = 2; & D_2 = T_2; & \pi_2 = 2; \\ T_3 = 8; & C_3 = 2; & D_3 = T_3; & \pi_3 = 1; \end{array}$$

L'utilisation processeur pour ce système est égale à

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} = \frac{5}{6} \approx 0.83 \quad > \quad 3.(2^{1/3} - 1) \approx 0.78$$

Or la séquence synchrone composant le pire cas d'exécution représentée figure 4.2 démontre que le système est ordonnançable.

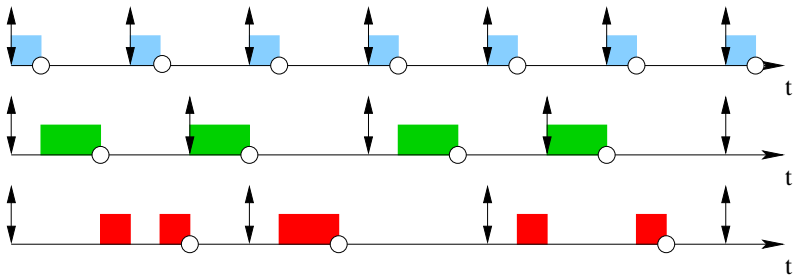


Figure 4.2 – Cas de système ordonnançable avec $U > n.(2^{1/n} - 1)$

◊

Dans les années 1980, des conditions nécessaires et suffisantes ont été proposées. Les premières sont basées sur la charge processeur de niveau i, W_i. Lehoczky *et al.* (1989) énoncent le théorème suivant pour des tâches périodiques indépendantes à échéance sur requête. Ce théorème est étendu par Nassor et Bres (1991) pour des tâches à échéance contrainte.

Théorème 4.4. *Un ensemble de n tâches périodiques indépendantes à échéance contrainte ordonnancé par une politique d'ordonnancement à priorités fixes est ordonnançable si et seulement si*

$$\forall i = 1, \ldots, n, \quad \exists t \in [0, D_i], \quad W_i(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{T_i} \right\rceil C_j \leq t \tag{4.2}$$

L'ensemble des instants à examiner est réduit aux points d'ordonnancement des tâches plus prioritaires (Lehoczky *et al.*, 1989).

Théorème 4.5. *Un ensemble de n tâches périodiques indépendantes à échéance contrainte ordonnancées par une politique d'ordonnancement à priorités fixes est ordonnançable si et seulement si*

$$\max_{1 \leq i \leq n} \min_{t \in E_i} W_i(t) \leq t \tag{4.3}$$

avec

$$E_i = \left\{ k.T_j \mid j = 1, \ldots, i \ ; \ k = 1, \ldots, \left\lceil \frac{D_i}{T_j} \right\rceil \right\}$$

Exemple . Reprenons l'exemple de système exposé ci-dessus et traçons la valeur de $W_3(t)$ pour $t \in [0, T_3]$. L'ensemble des points d'ordonnancement à étudier est

$$\begin{aligned} E_3 &= \left\{ k.T_j \mid j = 1, \ldots, 3 \ ; \ k = 1, \ldots, \left\lceil \frac{D_3}{T_j} \right\rceil \right\} \\ &= \{0, 4, 6, 8\} \end{aligned}$$

Nous devons alors calculer $W_3(t)$ pour chaque date contenue dans E_3. Par exemple, à la date $t = 6$

$$\begin{aligned} W_3(6) &= \sum_{j=1}^{3} \left\lceil \frac{t}{T_i} \right\rceil C_j \\ &= 2.C_1 + C_2 + C_3 = 6 \end{aligned}$$

Après avoir calculé chaque point, nous obtenons la courbe présentée figure 4.3. Le pointillé

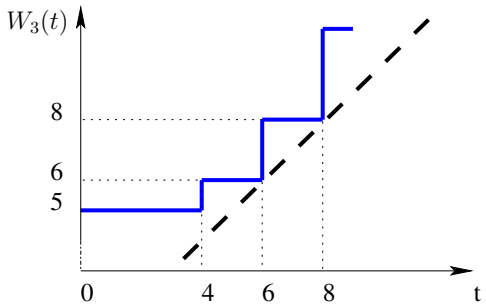


Figure 4.3 – Évolution de $W_3(t)$ sur l'intervalle $[0, T_3]$

gras représente la bissectrice $(y = t)$. $W_3(t)$ coupe cette courbe en $t = 6$. Il existe donc $t \in E_3$ tel que $W_3(t) \leq t$: la tâche τ_3 est ordonnançable.

$\Diamond$

Pendant ce temps, un autre groupe de chercheurs formule une méthode de décision basée sur le calcul du pire temps de réponse d'une instance. Indépendamment, Joseph et Pandya (1986) et Audsley *et al.* (1991) formulent le théorème suivant

Théorème 4.6. *Pour un ensemble de n tâches périodiques indépendantes à échéances contraintes ordonnancées par une politique d'ordonnancement à priorités fixes, le pire temps de réponse d'une tâche τ_i est tel que*

$$R_i = Ci + \sum_{\tau_i \in hp(i)} \left\lceil \frac{R_i}{T_i} \right\rceil C_j \tag{4.4}$$

avec $hp(i) = \{\{\tau_j\}_{j=1,\ldots,n} \mid \pi_j < \pi_i\}$

Lehoczky (1990) relâche définit l'instant synchrone pour des systèmes à échéances sur requêtes ou contraintes

Théorème 4.7. *Pour un ensemble de tâches périodiques ordonnancées par une politique d'ordonnancement à priorité fixe, le pire temps de réponse de τ_i est celui de la première instance de τ_i dans la période d'activité synchrone de niveau i, soit $R_i = R_{i,1}$*

Tindell *et al.* (1994) étend les contraintes d'échéances sur requête à D_i ou d'échéances contraintes en fournissant un test exact dans le cas de tâches à échéances quelconques.

Exemple . Reprenons l'exemple du système exposé précédemment et calculons le pire temps de réponse de la tâche τ_3. Le calcul est itératif en prenant comme initialisation la valeur d'exécution C_3

$$\begin{aligned} R_3^1 &= C_3 = 2 \\ R_3^2 &= C_3 + \left\lceil \frac{R_3^1}{T_1} \right\rceil + \left\lceil \frac{R_3^1}{T_2} \right\rceil = 5 \\ R_3^3 &= C_3 + \left\lceil \frac{R_3^2}{T_1} \right\rceil + \left\lceil \frac{R_3^2}{T_2} \right\rceil = 6 \\ R_3^4 &= C_3 + \left\lceil \frac{R_3^3}{T_1} \right\rceil + \left\lceil \frac{R_3^3}{T_2} \right\rceil = 6 = R_3^3 = R_3^* \end{aligned}$$

Le pire temps de réponse de la tâche τ_3 est égal à 6. Le chronogramme 4.2 illustre cette propriété. Ce pire temps de réponse est présent lors de la première activation à l'instant synchrone et est bien égal à 6 unités de temps. ◊

4.2 Analyse

4.2.1 Protection temporelle et analyse d'ordonnançabilité

Dans le cas d'une étude d'ordonnançabilité, nous venons de voir que les principaux paramètres considérés pour chaque tâche τ_i sont sa période T_i, son pire temps d'exécution C_i, sa priorité π_i et son échéance relative D_i.

Ces études sont donc réalisées dans le pire cas et garantissent une ordonnançabilité sûre du système (aucune instance ne rate son échéance) en absence de fautes. Si un système composé uniquement de tâches indépendantes ordonnancé par une politique préemptive à priorité fixe et si celui-ci est ordonnançable en considérant un temps d'exécution de chaque instance égal au pire temps d'exécution alors il l'est avec des valeurs d'exécution inférieures.

Le budget d'exécution est une valeur limite d'exécution. Fixer cette valeur au pire temps d'exécution permet de garantir une ordonnançabilité sûre puisque chaque instance dépassant son budget est considérée comme fautive et est arrêtée. Comme la spécification AUTOSAR OS peut le suggérer, cette valeur peut donc être prise pour configurer les budgets d'exécution.

4.2.2 Première approche : approche implicite

Comme nous l'avons présenté section 2.3.2, le mécanisme de protection temporelle proposé par AUTOSAR OS est basé sur trois budgets affectés à chaque instance :

- le budget d'exécution, $EXECUTION\_BUDGET$, garantissant le maximum de temps d'exécution d'une instance ;
- la fenêtre temporelle, $TIMEFRAME$, garantissant un intervalle de temps minimum entre deux activations ;
- le budget de blocage, $BLOCKING\_TIME\_BUDGET$, garantissant un temps de verrouillage maximum d'une ressource ou d'un masquage d'interruptions.

Dans notre étude, nous nous restreignons aux tâches indépendantes. Aucune ressource ne peut donc être présente. Une première approche consiste à configurer le mécanisme de protection selon le schéma suivant :

$$\begin{cases} TIMEFRAME & = T_i \\ EXECUTION\_BUDGET & = C_i \\ BLOCKING\_TIME\_BUDGET & = NotUsed \end{cases}$$

Nous allons alors évaluer cette configuration sur un ensemble de systèmes contenants des fautes.

4.3 Simulations

4.3.1 Paramètre de génération des applications

Pour évaluer les différentes stratégies de configuration du mécanisme de protection temporelle, nous avons développé un environnement de simulation (voir annexe A). Cet outil est composé de deux parties : le générateur d'architecture et le simulateur d'applications temps-réel.

Générateur d'architecture

Le générateur d'architecture a été implanté sous Matlab. Il permet de générer un ensemble de tâches. Chaque architecture est composée de n tâches dont chaque paramètre ($\tau_i = \{C_i, T_i, D_i, L_i, \pi_i\}$) doit être fixé. Pour la génération des différents paramètres, nous fixons les règles suivantes.

La période, T_i, de chaque tâche est choisie aléatoirement dans une liste de 18 valeurs. Ces valeurs sont choisies par l'algorithme proposé par Goossens et Macq (2001) afin de minimiser l'hyperpériode. Dans notre étude, les valeurs possibles sont tirées dans l'ensemble

$$\begin{aligned} T &= \left\{2^{u_2} \times 3^{u_3} \times 5^{u_5} \mid \{u_2, u_3\} \in \{0,1,2\}^2,\ u_5 \in \{0,1\}\right\} \\ &= \{1,2,3,4,5,6,9,10,12,15,18,20,30,36,45,60,90,180\} \end{aligned}$$

L'hyperpériode $\overline{T}$ dans le cas d'un système synchrone est égale au plus petit commun multiple (ppcm) des périodes. Dans notre cas d'étude, comme chaque valeur est composée de puissance de nombres premiers, l'hyperpériode ne peut pas dépasser la valeur maximale possible pour une période, $\max(T) = 180$ unité de temps. Le pire temps d'exécution, C_i, est fixé par un des algorithmes proposé par Bini et Buttazzo (2004) (UScaling, UFitting et UUniFast) qui permet de fixer des valeurs d'exécution en fonction de la charge globale processeur U désirée.

Les échéances sont définies telles que le ratio $\eta = D_i/T_i$ soit le même pour chaque tâche. Nous supposons les tâches à échéances contraintes, le paramètre η est donc tel que $\eta \leq 1$ (dans le cas particulier où $\eta = 1$, nous avons $D_i = T_i$, cas des tâches à échéance sur requêtes). Les priorités sont affectées selon l'algorithme "deadline monotonic" (algorithme optimal dans le cas où $\eta \leq 1$, section 4.1).

Le comportement du générateur est ainsi contrôlé par trois paramètres : le nombre de tâches n, l'utilisation globale processeur $U = \sum_{i=1}^{n} C_i/T_i$ et la sévérité des contraintes temporelles η.

La figure 4.4 représente la répartition des principaux paramètres générés sur un ensemble de 50 architectures, les paramètres du générateur étant fixés à $n = 10$, $U = 0.9$ et $D_i/T_i = 0.75$. Chaque colonne (abscisse) correspond à une architecture, l'ordonnée à la répartition. Les croix bleues claires représentent l'ensemble des valeurs tirées. Les triangles et ronds bleus représentent respectivement les valeurs maximales et minimales pour chaque architecture. Les croix noires représentent les valeurs moyennes pour chaque architecture et la ligne pointillée la moyenne sur l'ensemble des tâches générées.

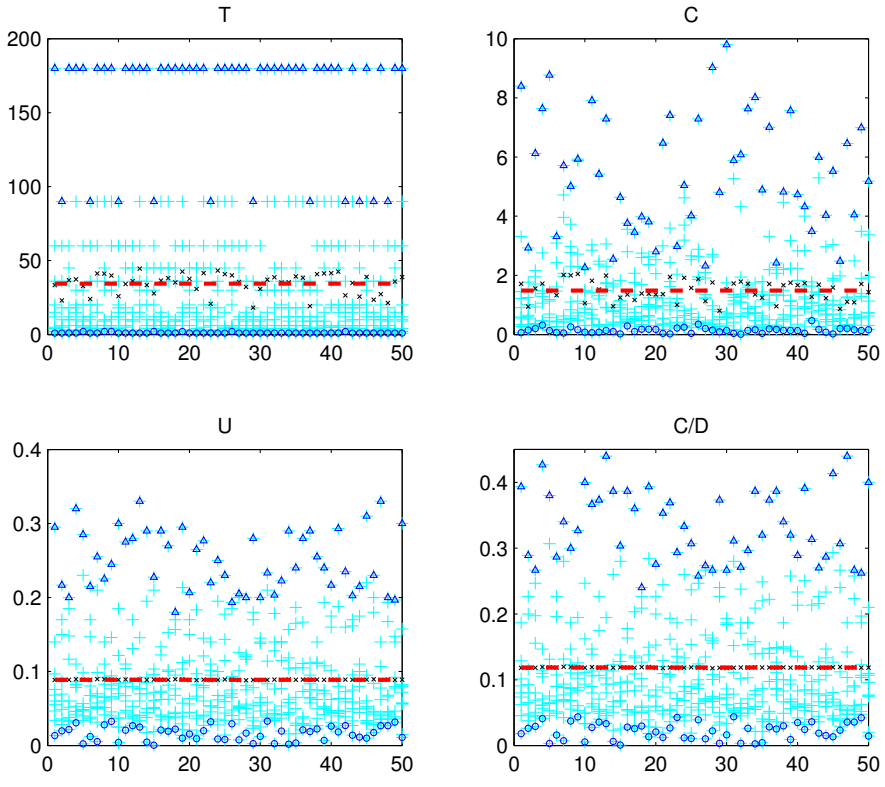


Figure 4.4 – Répartition des différents paramètres de modélisation (50 architectures)

Le tableau 4.1 présente un récapitulatif des valeurs maximales, minimales, des moyennes et écarts-types obtenus pour T_i et C_i/D_i sur l'ensemble des 50 architectures générées.

	η	max	min	m	σ
T_i	*	180	1	34,35	49,09
C_i/D_i	1	0.3300	0.0004	0.0889	0.0721
	0.75	0.4400	0.0006	0.1185	0.0961
	0.67	0.4925	0.0007	0.1327	0.1076

Tableau 4.1 – Comportement du générateur d'architecture ($n = 10$, $U = 0.9$ et $D_i/T_i \in \{1, 0.75, 0.67\}$)

Modèle de fautes

Concernant la simulation des fautes, nous modélisons la possibilité de sous-estimation du WCET mais intégrons également le fait que le pire temps d'exécution n'est pas atteint par chaque instance. Puisqu'il est impossible de simuler le système dans sa période complète d'activité, nous décidons d'augmenter artificiellement la fréquence d'apparition de fautes. Comme nous voulons "stresser" la protection temporelle, nous avons volontairement augmenté l'importance des fautes (i.e. la différence entre la valeur estimée du WCET et la valeur d'exécution). Le temps d'exécution d'une instance d'une tâche τ_i est donc choisi dans un intervalle $[c^{min}, c^{max}] = [\gamma^{min}.C_i, \gamma^{max}.C_i]$. Nous choisissons une distribution uniforme paramétrée par γ^{min} et γ^{max}, paramètres de la simulation. Nous avons donc

$$\forall i \in \{1, \ldots, n\}, \quad \forall j, \quad c_{i,j} = \text{rand}_u\ (\gamma^{min}.C_i, \gamma^{max}.C_i)$$

La figure 4.5 représente un exemple de choix des paramètres γ^{min} et γ^{max} par rapport à une fonction de probabilité réelle (idem à la figure 3.3).

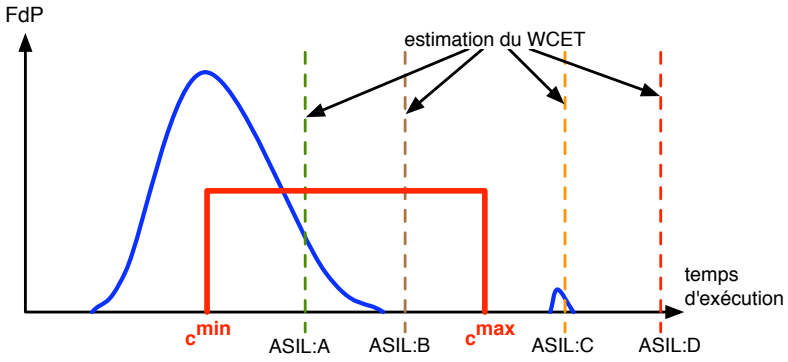


Figure 4.5 – Répartition des différents paramètres de modélisation (50 architectures)

D'autres distributions (loi normale, par exemple) ont été testées mais les résultats obtenus sont quasi-identiques et une distribution uniforme est simple à mettre en place et à analyser.

Simulateur de système temps-réel

Le simulateur est basé sur la boîte à outil Matlab/Simulink, TrueTime (Ohlin *et al.*, 2007), avec l'apport de quelques modifications permettant d'implémenter le mécanisme de protection

temporelle. Des mécanismes de détection des erreurs (temps d'exécution supérieur au budget) et de détection de défaillance (échéance ratée) sont mis en place pour extraire les événements pertinents des simulations. Ces deux mesures permettent de caractériser la qualité d'un algorithme de configuration de la protection temporelle. Un résumé de la composition de la boîte à outil ainsi qu'un aperçu des différentes modifications apportées est présenté annexe A.

La simulation est réalisée en définissant une période d'étude $\overline{T}_{etude}$. Cette période d'étude est fonction du nombre d'activations de tâches simulées et de l'hyperpériode.

$$\overline{T}_{etude} = \{k.\overline{T} \mid nbActivations_{[0,k.\overline{T}]} \geq M\}$$

où M est le nombre minimum d'activations que nous souhaitons simuler (toutes tâches confondues).

La figure 4.6 représente l'ensemble du dispositif de simulation (générateur + simulateur), ses entrées (paramètres) et ses sorties (mesures).

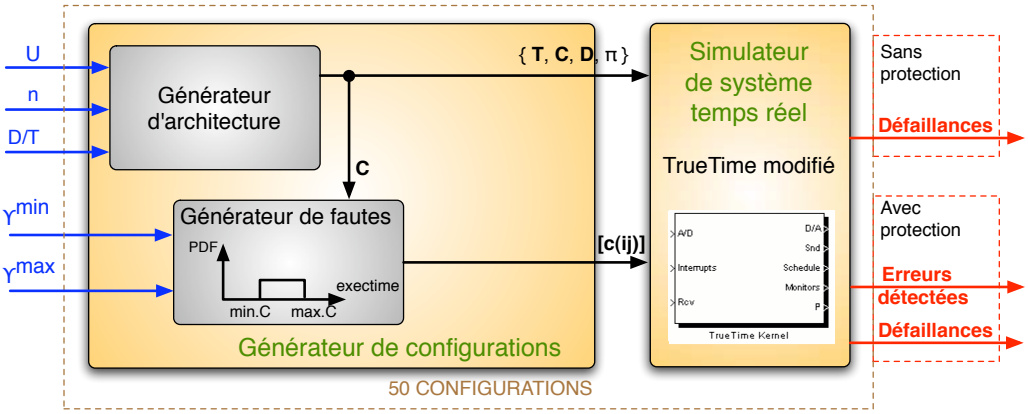


Figure 4.6 – Générateur et simulateur d'applications

4.3.2 Résultats de simulations

Dans un premier temps, le générateur est configuré avec les paramètres d'entrée fixés à $n = 10$, $U = 0.9$ et $\eta = D_i/T_i = 0.67$. Le simulateur est configuré tel que $\gamma^{min} = 0.95$ et $\gamma^{max} = 1.15$: les temps d'exécution tirés aléatoirement peuvent être plus petits ou plus grands que la valeur du WCET estimée et utilisée pour dimensionner le système. Chaque simulation est stoppée après 5000 activations d'instances. 50 architectures sont générées et simulées avec et sans recours à la protection temporelle.

Le mécanisme de protection temporelle est configuré selon le schéma décrit précédemment :

$$\begin{cases} TIMEFRAME & = T_i \\ EXECUTION\_BUDGET & = C_i \end{cases}$$

Le nombre de violations d'échéance est mesuré avec et sans mécanisme de protection, ainsi que le nombre d'instances tuées par la protection temporelle avant leur terminaison. Ces deux

mesures permettent d'évaluer la complétude du mécanisme (toutes les défaillances sont elles éliminées ?) et sa précision (toutes les erreurs détectées entraînent-elles une défaillance ?).

Les résultats sont donnés figure 4.7. Le nombre d'instances tuées par le mécanisme de protection (temps d'exécution supérieur au budget) sur l'ensemble des instances générées est tracé pour chaque architecture (ligne cercle, échelle de droite). Le nombre d'instances dépassant leur échéance sans mécanisme de protection est tracé pour chaque architecture (ligne continue, échelle de gauche). Les lignes pointillés représentent les valeurs moyennes de ces deux mesures. Les résultats montrent que l'immense majorité des fautes simulées (dépassement de

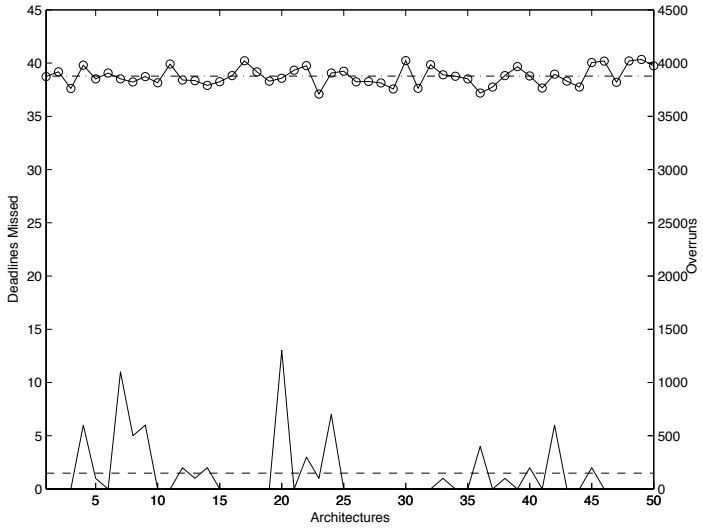


Figure 4.7 – Sur-exécutions et échéances ratées dans le cas où $D/T = 0.67$

l'estimation du pire temps d'exécution) n'entraînent aucune défaillance. Plus précisément, le nombre moyen d'échéances ratées sans recours au mécanisme de protection (système libre) est de 1.48 à comparer avec le nombre moyen de sur-exécutions détectées qui est de 3887. La plupart du temps, il existe un temps creux, durant lequel aucune tâche ne devrait s'exécuter, qui permet au système, normalement surchargé, d'absorber silencieusement les fautes.

	η	max	min	m	σ
Sur-exécutions	*	4034	3709	3887	86
Échéances ratées (avec protection)	*	0	0	0	0
Échéances ratées (sans protection)	1	2	0	0.04	0.28
	0.75	11	0	1.28	2.69
	0.67	13	0	1.48	2.89

Tableau 4.2 – Sur-exécutions et échéances ratées dans le cas où $D/T = \{1, 0.75, 0.67\}$

Le tableau 4.2 résume les valeurs maximales, minimales, moyennes et écarts-types de ces deux mesures pour les différentes valeurs de $\eta \in \{1, 0.75, 0.67\}$.

Comme nous pouvions le pressentir, le nombre de violations d'échéances diminue lorsque $\eta = D/T$ augmente. En effet, cela suppose que le système est moins contraints (échéance plus lointaine) ce qui laisse plus de marge à chaque instance pour s'exécuter.

La plupart des systèmes considérés, malgré une utilisation processeur U élevée ($U = 0.9$), possède des temps libres qui permettent à l'ordonnanceur d'absorber la majorité des fautes. Cependant, rater une échéance peut avoir une conséquence non-négligeable pour certaines tâches du système (tâches critiques). Dans ce cas, la protection temporelle permet d'empêcher la propagation d'erreur ("effet domino") entre les tâches et de garantir l'ordonnançabilité de chaque instance de tâche.

4.4 Discussion

Prenons un exemple de système généré par le générateur d'architecture et analysons les résultats observés en fonction de l'ordonnançabilité du système.

4.4.1 À partir d'un exemple

Exemple . Soit S (système généré n°7) le système composé de 10 tâches périodiques indépendantes $S = \{\tau_1, \ldots, \tau_{10}\}$, classées par ordre de priorités décroissantes, dont les principaux paramètres sont présentés tableau 4.3. Une estimation du pire-temps d'exécution $\hat{C}$ pour chaque tâche est tirée telle que l'utilisation processeur global soit d'environ 0.9 (ici, $U_{base} = 0.8965$).

Les temps d'exécution sont tirés selon une loi uniforme avec $C^{min} = \lceil 0.95\, C_{base} \rceil$ et $C^{max} = \lceil 1.15\, C_{base} \rceil$ (sous-estimation du pire temps d'exécution). Les valeurs maximales et minimales sont calculées pour chaque tâche voir tableau. Nous obtenons ainsi une utilisation processeur comprise entre $U^{min} = 0.8586$ et $U^{max} = 1.0394$. Nous considérons que les temps d'exécution des instances de chaque tâche lors de l'exécution sont distribués uniformément entre ces deux valeurs : l'utilisation processeur moyenne du système est donc de $\overline{U} = 0.9490$.

Nous calculons le pire temps de réponse de chaque tâche en considérant les valeurs minimales, maximales et moyennes d'exécution. Nous obtenons respectivement les valeurs R^{min}, R^{max} et $\overline{R}$. Nous nous apercevons que dans le cas de temps d'exécution minimaux, le système est totalement ordonnançable car tous les pire-temps de réponse sont inférieurs aux échéances. Dans le cas d'un système moyennement surchargé (prise en considération de $\overline{C}$), seules les instances émanant de la tâche τ_{10} peuvent être défaillantes. Dans le cas de la prise en compte du pire cas (C^{max}), les instances τ_8 et τ_9 dépassent leur échéance et le pire temps de réponse de la tâche τ_{10} peut être infini (phénomène de famine : les instances des autres tâches monopolisent le processeur). Cependant, même si toutes les instances $\tau_{i,j}$ avaient un temps d'exécution égal à C_i^{max}, seule une instance de la période d'étude (hyperpériode) peut être défaillante. Les temps d'exécution étant tirés aléatoirement, le pire temps de réponse peut ne jamais survenir durant une simulation.

Les résultats de simulation illustrent ces remarques. En effet, le pire-cas n'est jamais survenu puisque les tâches τ_8 et τ_9 n'ont jamais failli. Par contre la tâche τ_{10} a failli quasiment à chaque activation. Le système a été simulé sur 14 hyperpériodes (l'hyperpériode étant ici égale

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}
T	100	200	300	600	1500	2000	3600	6000	9000	18000
D	67	150	225	450	1125	1500	2700	4500	6750	13500
$\hat{C}$	5	51	63	42	8	197	207	396	472	571
Modèle de fautes										
C^{min}	5	49	60	40	8	188	197	377	449	543
$\overline{C}$	5.5	54	66.5	44.5	9	207.5	218	416.5	496	600
C^{max}	6	59	73	49	10	227	239	456	543	657
Pire temps de réponse										
$\hat{R}$	5	56	124	166	174	556	1114	2583	3599	5986
R^{min}	5	54	119	159	167	533	1062	1789	3386	5392
$\overline{R}$	5.5	59.5	131.5	176	185	589	1179.5	2930	5359.5	**14000**
R^{max}	6	65	144	193	268	982	1710	**5327**	**15898**	-
Résultats de simulation										
Nb acti	2520	1260	840	420	168	126	70	42	28	14
Sur-exec	1886	930	633	316	119	91	55	34	22	12
Échéances ratées	0	0	0	0	0	0	0	0	0	12
Relaxation possible										
$\hat{C}_+$	5	53	66	44	8	207	217	416	496	600
Sur-exec	1886	685	433	220	119	63	38	21	13	8
$\hat{R}_+$	5	58	129	173	181	580	1162	2883	5279	11893

Tableau 4.3 – Exemple de système généré et résultats

à 18000). La simulation représente le cas moyen (où seule la tâche τ_{10} peut rater son échéance) et la probabilité de connaître une défaillance de la tâche τ_8 ou τ_9 étant très faible, le temps de simulation nécessaire à son apparition est très grand. Le nombre de sur-exécutions détectées par le mécanisme de protection est quant à lui important (4098 sur-exécutions détectées).

◊

Les résultats obtenus par simulation mettent en évidence les limites du schéma de configuration du mécanisme de protection temporelle proposé dans cette section. Le système comporte des temps creux qui peuvent être utilisés afin de relaxer les budgets. Or le schéma utilisé n'exploite pas ces temps creux. Ainsi, si l'on configure le système utilisé comme exemple avec les budgets relaxés données en bas du tableau, toutes les tâches respectent leur échéance ($R < D$) et le nombre d'erreurs détectées est moins important (le nombre de sur-exécutions

détectées passe de 4098 à 3486 soit une diminution d'environ 15%).

4.4.2 Relaxer les budgets – Un but pertinent

Les expériences menées démontrent que laisser le système sans mécanisme de protection temporelle peut être plus favorable que d'essayer de le maîtriser trop strictement par ce mécanisme maladroitement dimensionné. Un mécanisme comme la surveillance d'échéance considérée comme non adaptée par le standard AUTOSAR OS peut paraître plus appropriée (en appliquant le principe qui consiste à traiter un pire cas qui n'apparaît que très rarement).

Dans ce mémoire, nous travaillons sur une relaxation possible des budgets d'exécution du mécanisme de protection proposé par le standard afin d'équilibrer au mieux la balance erreurs détectées/défaillances d'une instance. Dans ce cadre, l'objectif est de parvenir à récupérer le plus de temps creux possibles afin d'utiliser ceux-ci pour relâcher les contraintes du mécanisme de protection.

Deux approches sont proposées. La première est une approche déterministe qui garantit au système muni du mécanisme de protection un contrôle total des défaillances temporelles. La seconde est une approche probabiliste qui garantit une probabilité d'ordonnançabilité de chaque instance du système. Nous considérons dans ce mémoire des systèmes multi-critiques où des tâches critiques et non-critiques cohabitent. Dans ce cadre, nous apporterons des éléments de réponse sur la prise en compte des niveaux de criticité dans le choix des budgets d'exécution.

« Limiter la pauvreté sans limiter la richesse »

Les Misérables - Victor Hugo

Chapitre 5

Configuration par analyse de sensibilité

Sommaire

5.1 Analyse de sensibilité 56
5.1.1 État de l'art 56
5.1.2 Exemple de système 57
5.2 Prise en compte de la criticité des tâches 60
5.2.1 Extension du modèle d'application 60
5.2.2 Exemple de système 61
5.3 Prise en compte dans le calcul des budgets 62
5.4 Simulations 63
5.4.1 Ajout de la boîte de calcul des budgets 63
5.4.2 Résultats de simulations 63
5.5 Discussion 67

Le présent chapitre est consacré à la relaxation des budgets d'exécution basée sur une étude déterministe du système.

Plusieurs études ont été réalisées sur les mécanismes de réservation de ressources dans le cas d'un système ordonnancé par un ordonnanceur préemptif à priorités dynamiques EDF. Dans ce cas, la récupération des temps creux est assez simple. Une méthode efficace pour réaliser la protection temporelle sous un ordonnancement EDF est l'utilisation d'un serveur à bande passante fixe (Constant Bandwidth Server) introduit par Abeni *et al.* (1998); Abeni et Buttazzo (2004). En conjonction avec le serveur, des mécanismes de récupération des temps inutilisés sont mis en place comme l'algorithme CASH (CApacity SHaring) introduit par Caccamo *et al.* (2000) ou l'algorithme BASH (BAndwidth SHaring) introduit par Caccamo *et al.* (2005).

Dans notre étude, nous nous intéressons aux systèmes ordonnancés par un ordonnancement à priorités fixes. Dans ce cadre, la recherche et la répartition des temps creux sont un peu moins aisées. En effet, la tâche en exécution n'est plus la tâche ayant la plus petite laxité mais celle qui détient la plus haute priorité, constante fixée au démarrage.

Le présent chapitre est consacré à la relaxation des budgets par récupération statique des budgets en utilisant des résultats issus d'analyse de sensibilité (Bini, 2004; Bini *et al.*, 2006). Cette technique permet d'augmenter la taille des budgets, de réduire le nombre de faux-positifs (détection d'une erreur ne menant pas à une défaillance) et ainsi d'améliorer la qualité de service de l'ordonnanceur équipé du mécanisme de protection temporelle.

Le chapitre est organisé comme suit. La section 1 permet de faire un rappel sur l'analyse de sensibilité. Puis nous introduirons en section 2, le modèle de tâche retenu pour notre étude. La section 3 permet d'adapter les résultats aux systèmes multi-critiques. Des simulations en section 4 permettent de comparer les différentes techniques de configurations. Puis nous discuterons en section 5 des résultats obtenus et des perspectives possibles.

5.1 Analyse de sensibilité

5.1.1 État de l'art

Bini (2004) et Bini *et al.* (2006) ont travaillé sur l'analyse de la sensibilité d'un système. Cette analyse permet de connaître dans quelle mesure le système reste faisable (au sens de l'ordonnancement) lors de variations des paramètres d'une tâche (T_i, C_i ou D_i). Le modèle d'application retenu est composé d'un ensemble $S = \{\tau_1, \ldots, \tau_n\}$ de n tâches périodiques et indépendantes. Chaque tâche τ_i est caractérisée par sa période T_i, son pire temps d'exécution C_i et son échéance relative D_i. L'ordonnancement est mono-processeur à priorité statique, préemptif. Sous ces hypothèses, ils démontrent le théorème suivant

Théorème 5.1. *(Bini, 2004) Un ensemble de tâches $S = \{\tau_1, \ldots, \tau_n\}$ périodiques classées dans l'ordre des priorités décroissantes ($\pi_1 < \cdots < \pi_n$) est ordonnançable sous un ordonnancement à priorité fixe préemptif si et seulement si*

$$\forall i \in \{1, \ldots, n\},\ \exists t \in \text{schedP}_i, \quad C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t \tag{5.1}$$

où schedP_i *est un ensemble de points d'ordonnancement définis par* $\text{schedP}_i = \mathcal{P}_{i-1}(D_i)$, *et* $\mathcal{P}_i$ *est défini comme suit*

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1}\left(\left\lfloor \frac{t}{T_i} \right\rfloor T_i\right) \cup \mathcal{P}_{i-1}(t) \end{cases}$$

En utilisant la notation vectorielle et les opérateurs min, et max, l'équation (5.1) peut être écrite comme suit

$$\max_{i=1,\ldots,n} \min_{t \in \text{schedP}_i} \mathbf{n_i} \cdot \mathbf{C_i} \leq t \tag{5.2}$$

où $\mathbf{n_i} = (\lceil \frac{t}{T_1} \rceil, \lceil \frac{t}{T_2} \rceil, \ldots, \lceil \frac{t}{T_{i-1}} \rceil, 1)$ et $\mathbf{C_i} = \{C_1, \ldots, C_i\}$.

Ce théorème représente une version améliorée de la condition d'ordonnançabilité proposée par Lehoczky *et al.* (1989) (théorème 4.5) avec une diminution du nombre des points d'ordonnancement à analyser ($schedP_i \subseteq E_i$).

Il définit alors la définition de X-space afin de représenter l'espace admissible (faisable) de variation des paramètres du modèle.

Définition 5.1. *Soit* $\mathbf{X} \in \{\mathbf{T}, \mathbf{C}, \mathbf{D}\}$ *(*$\mathbf{X} = \{X_i\}_{1 \leq i \leq n}$*) choisi comme paramètre d'un système composé de* n *tâches* $S = \{\tau_1, \ldots, \tau_n\}$*. Le domaine de faisabilité* X*-space correspond à l'ensemble des valeurs de* $\mathbf{X}$ *tel que le système* S *soit faisable.*

Dans notre étude, nous nous intéressons au C-space qui définit l'ensemble des valeurs de temps d'exécution pour chaque tâche tel que le système S reste faisable.

À partir d'un système $S = \{\tau_1, \ldots, \tau_n\}$ faisable dont le vecteur des temps d'exécution est $\mathbf{C} = \{C_1, \ldots, C_n\}$, nous pouvons définir un nouveau système S' également faisable dont le vecteur des temps d'exécution est $\mathbf{C'} = \{C'_1, \ldots, C'_n\}$ (les autres paramètres $\mathbf{T}$ et $\mathbf{D}$ restant inchangés) tel que

$$\mathbf{C'} = f(\mathbf{C}) \tag{5.3}$$

où f est une fonction de $\mathbb{R}^n$ dans $\mathbb{R}^n$ définissant les variations admissibles des paramètres d'exécution C_i. Pour notre problème, le but est de relaxer au maximum les valeurs de $\mathbf{C}$ jusqu'à parvenir à la limite d'ordonnancement du système (borne du C-space). Dans ce cadre, deux études particulières peuvent être intéressantes :

Relaxation proportionnelle Un cas d'étude est de choisir la fonction f telle que

$$f : \mathbf{C} \mapsto f(\mathbf{C}) = (1 + \lambda) \,.\, \mathbf{C}$$

Cette étude correspond à une relaxation proportionnelle sur chaque composante. Dans ce cas, le montant de la variation λ admissible peut être déterminé en appliquant la condition nécessaire et suffisante donnée par l'équation (5.2). S' est alors ordonnançable si et seulement si :

$$\lambda \leq \min_{i=1,\ldots,n} \max_{t \in \text{schedP}_i} \frac{t}{\mathbf{n_i} \cdot \mathbf{C_i}} - 1 \triangleq \lambda^{max} \tag{5.4}$$

Relaxation mono-dimensionnelle Un autre cas d'étude est de choisir la fonction f telle que

$$f : \mathbf{C} \mapsto (f(\mathbf{C}))_i = C_i + \delta_{ik} \,.\, \Delta C_k \quad \text{avec } \delta_{ik} = \begin{cases} 1 & \text{si } i = k \\ 0 & \text{sinon} \end{cases}$$

Cette relaxation permet de ne faire varier qu'une seule valeur C_k du vecteur de temps d'exécution, laissant les autres inchangées. Dans ce cas, $\mathcal{S}'$ est ordonnançable si et seulement si

$$\Delta C_k \leq \min_{i=k,\ldots,n} \max_{t \in \text{schedP}_i} \frac{t - \mathbf{n_i} \cdot \mathbf{C_i}}{\lceil t/T_k \rceil} \triangleq \Delta C_k^{max} \tag{5.5}$$

5.1.2 Exemple de système

Prenons l'exemple d'un système composé de deux tâches, $S = \{\tau_1, \tau_2\}$ (τ_1 plus prioritaire que τ_2) avec τ_1 tel que $T_1 = D_1 = 9.5$ et $T_2 = D_2 = 22$. Soit C_1 et C_2 des paramètres tels que

le système S soit faisable par une politique d'ordonnancement préemptif à priorités statiques. Appliquons les critères d'ordonnançabilité énoncés par l'équation (5.1). Le calcul de schedP_1 et schedP_2 donne

$$\begin{aligned} \text{schedP}_1 &= \mathcal{P}_0(D_1) = \{D_1\} \\ \text{schedP}_2 &= \mathcal{P}_1(D_2) = \mathcal{P}_0(D_2) \cup \mathcal{P}_0\left(\left\lfloor \frac{D_2}{T_1} \right\rfloor .T_1\right) = \{2T_1, D_2\} \end{aligned}$$

Nous obtenons alors les contraintes suivantes pour les durées d'exécution C_1 et C_2

- τ_1 faisable $\iff \exists t \in \text{schedP}_1 = \{D_1\}, \quad C_1 \leq t$
- τ_2 faisable $\iff \exists t \in \text{schedP}_2 = \{2T_1, D_2\}, \quad C_2 + \left\lceil \frac{t}{T_1} \right\rceil C_1 \leq t$

Ce qui donne le système d'équation suivant

$$\begin{cases} C_1 \leq D_1 \\ \left\| \begin{array}{l} C_2 + 2C_1 \leq 2T_1 \\ ou \\ C_2 + 3C_1 \leq D_2 \end{array} \right. \end{cases}$$

La figure 5.1 représente graphiquement le C-space (partie en dessous des limites, lignes rouges continues), domaine de faisabilité du système étudié.

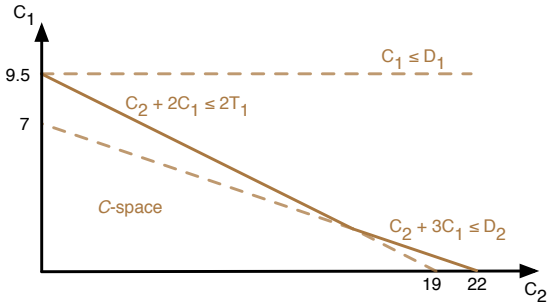


Figure 5.1 – Exemple de $\mathcal{C}$-space

Reprenons le système défini ci-dessus. Le système S est défini tel que $C_1 = 4$ et $C_2 = 6$. Dans ce cas, nous obtenons le chronogramme d'exécution présenté figure 5.2. Nous pouvons remarquer la présence d'un temps creux entre les dates $t = 14$ et $t = 19$.

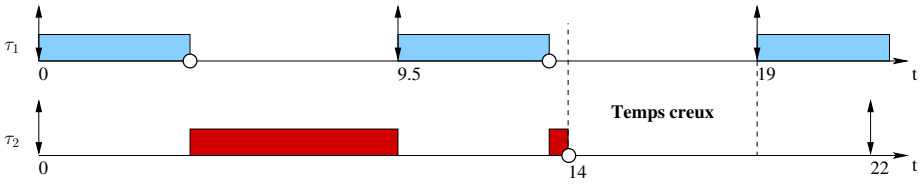


Figure 5.2 – Présence d'un temps creux

Le système S est faisable et nous voulons savoir quelles sont les limites de cette faisabilité. Dans le plan C_1/C_2 présenté figure 5.3, le point $(C_1 = 4, C_2 = 6)$ est à l'intérieur du C-space.

Plusieurs directions de recherche de limite du C-space peuvent être prises en compte. Les points (1), (2) et (2') sont des exemples de points limites trouvés selon certaines directions.

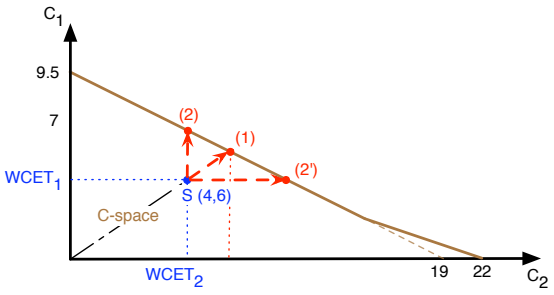


Figure 5.3 – Exemple de relaxation

Appliquons maintenant les formules données ci-dessus pour les deux cas particuliers présentés. Pour l'étude proportionnelle, nous avons

$$\lambda^{max} = \min\left[\frac{D_1}{C_1} - 1, \max\left(\frac{D_2}{\lceil D_2/T_1\rceil C_1 + C_2} - 1, \frac{2T_1}{\lceil 2T_1/T_1\rceil C_1 + C_2} - 1\right)\right]$$

$$\lambda^{max} = \min\left[\frac{55}{40}, \max\left(\frac{4}{18}, \frac{5}{14}\right)\right] = \frac{5}{14} \approx 0.357$$

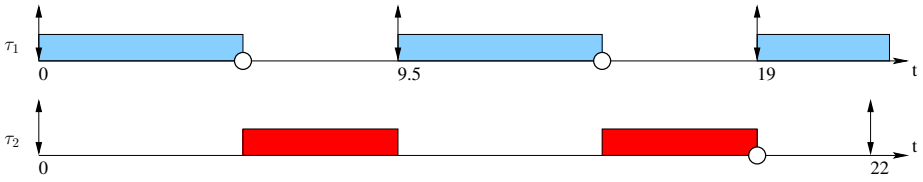


Figure 5.4 – Relaxation proportionnelle

Dans ce cas, nous obtenons un système S' dont les temps d'exécution relaxés sont $C'_1 = (1 + \lambda^{max}).C_1 = 5.43$ et $C'_2 = 8.14$. Ce point de fonctionnement correspond au point (1) sur la figure 5.3. Le chronogramme figure 5.4 montre une partie d'exécution (instant critique) des tâches τ_1 et τ_2 dans ce cas. Le temps creux présent précédemment est comblé.

Pour l'étude mono-dimensionnelle, nous obtenons

$$\begin{aligned}
\Delta C_1^{max} &= \min\left[\frac{D_1 - C_1}{\lceil D_1/T_1\rceil}, \max\left(\frac{D_2 - \lceil D_2/T_1\rceil .C_1 - C_2}{\lceil D_2/T_1\rceil}, \frac{2T_1 - \lceil 2T_1/T_1\rceil .C_1 - C_2}{\lceil 2T_1/T_1\rceil}\right)\right] \\
&= \min\left[5.5, \max\left(\frac{4}{3}, \frac{5}{2}\right)\right] = \frac{5}{2} = 2.5 \\
\Delta C_2^{max} &= \max\left(\frac{D_2 - \lceil D_2/T_1\rceil .C_1 - C_2}{\lceil D_2/T_2\rceil}, \frac{2T_1 - \lceil 2T_1/T_1\rceil .C_1 - C_2}{\lceil 2T_1/T_2\rceil}\right) \\
&= \max(4, 5) = 5
\end{aligned}$$

Dans ce cas, on obtient soit le système S'_1 dont les temps d'exécution relaxés sont $C'_1 = C_1 + \Delta C_1^{max} = 6.5$ et $C'_2 = C_2 = 6$ (point de fonctionnement (2) figure 5.3, chronogramme figure 5.5), soit le système S'_2 dont les temps d'exécution relaxés sont $C'_1 = C_1 = 4$ et $C'_2 = C_2 + \Delta C_2^{max} = 11$ (point de fonctionnement (2') figure 5.3, chronogramme figure 5.6).

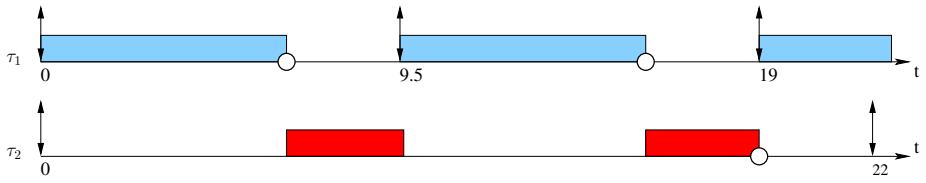


Figure 5.5 – Relaxation mono-dimensionnelle (uniquement C_1)

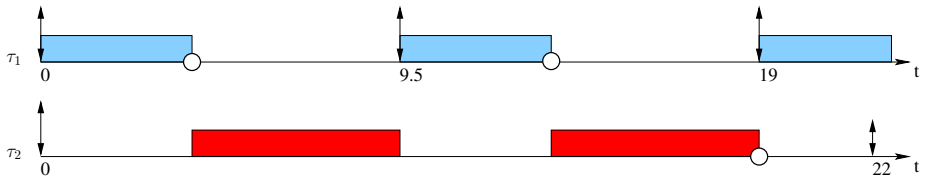


Figure 5.6 – Relaxation mono-dimensionnelle (uniquement C_2)

L'application des formules fournies ci-dessus permet de déterminer plusieurs cas limites d'exécution. Dans ces cas d'étude, les temps creux (instants pendant lesquels le processeur n'exécute aucune tâche) sont réduits à des valeurs correspondants à des minima locaux.

5.2 Prise en compte de la criticité des tâches

Afin d'étudier plus particulièrement dans quelle mesure nous pouvons appliquer les études de sensibilité présentées ci-dessus pour relaxer les budgets d'exécution, nous devons, dans un premier temps, spécifier le modèle d'application que nous considérons.

5.2.1 Extension du modèle d'application

En plus des paramètres C_i, T_i et D_i, nous assignons à chaque tâche un niveau de criticité $L_i \in \mathbb{L}$. Plus une tâche est critique, plus son niveau de criticité, L_i est grand. Nous supposons que plus une tâche est critique, plus le niveau de confiance sur les valeurs des paramètres de cette tâche est élevé (Vestal, 2007). Dans la suite de cette étude, nous considérons uniquement deux types de tâches : les tâches critiques ($L_i = 1$) et les tâches non-critiques ($L_i = 0$).

L'étude de systèmes déterministes est réalisée en prenant en compte le pire temps d'exécution C_i. Si le système est ordonnançable en considérant ces valeurs d'exécution, il l'est dans les autres cas. Cependant, la détermination du pire temps d'exécution est assez difficile à réaliser (voir section 3.2.1). La remarque énoncée précédemment sur la confiance en les paramètres d'une tâche est notamment applicable au WCET. Pour une tâche critique ($L_i = 1$), cette valeur est censée être une borne sûre. Pour une tâche non-critique ($L_i = 0$), cette valeur est par exemple le pire cas observé à l'occasion d'une campagne de tests, ce qui ne constitue pas une borne sûre. Dans la suite, cette remarque est déterminante dans le choix de la stratégie

de relaxation des budgets.

Un système S peut donc être vu comme un ensemble de n tâches, chaque tâche τ_i étant caractérisée par un 5-uplet $\big(C_i, D_i, T_i, L_i, \pi_i\big)$.

$$S = \{\tau_i\}_{i=1\ldots n} = \big\{\big(C_i, D_i, T_i, L_i, \pi_i\big)\big\}_{i=1\ldots n}$$

5.2.2 Exemple de système

Exemple . Considérons le système S composé de trois tâches ordonnancées sur un unique processeur à l'aide d'un ordonnancement préemptif à priorités statiques. Les caractéristiques principales des tâches τ_1, τ_2, τ_3 sont fournies tableau 5.1.

Tâche	π_i	$D_i = T_i$	C_i	L_i
τ_1	3	5	1	0
τ_2	2	10	3	1
τ_3	1	15	5	0

Tableau 5.1 – Exemple de configuration

Analysons le pire-cas d'exécution. Dans ce cas d'étude, celui-ci correspond au cas synchrone (voir section 4.1) où toutes les tâches sont activées simultanément (considérée comme la date zéro). La figure 5.7 représente le chronogramme d'exécution du système. Nous pouvons voir que toutes les tâches sont exécutées avant leur échéance.

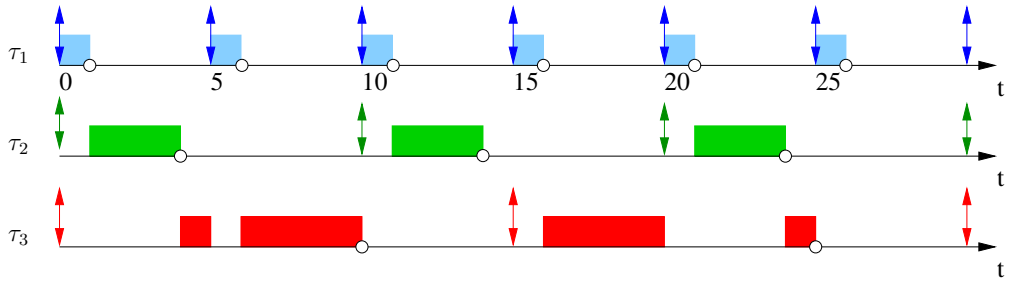


Figure 5.7 – Pire cas d'exécution d'un système déterministe

La tâche τ_2 étant une tâche critique ($L_2 = 1$), son pire temps d'exécution $C_2 = 3$ est une valeur sûre. Au contraire, les temps d'exécution des tâches τ_1 et τ_2 non-critiques ($L_1 = L_3 = 0$) peuvent être des valeurs non sûres. Nous pouvons donc supposer que dans certains cas leur exécution peut être plus importante que la valeur considérée pour le dimensionnement. Si nous considérons une sous-estimation de l'ordre de 10% pour les tâches non-critiques, nous pouvons montrer que ce système est encore ordonnançable. En effet, dans ce cas, les temps d'exécution de tâches non-critiques, τ_1 et τ_3, peuvent atteindre les valeurs respectives $C_1 = 1.1$ et $C_3 = 5.5$. Dans ce cas, une étude d'ordonnançabilité en prenant comme valeur de référence de temps d'exécution $\{C_1 = 1.1,\ C_2 = 3,\ C_3 = 5.5\}$ montre que ce système reste ordonnançable. Le

calcul des pire temps de réponse donne :

$$\begin{aligned} R_1 &= C_1 = 1.1 \leq D_1 \\ R_2 &= C_2 + C_1 = 4.1 \leq D_2 \\ R_3 &= C_3 + 2C_2 + 3C_1 = 14.8 \leq D_3 \end{aligned}$$

Le chronogramme représentant le scénario synchrone de ce système est présenté figure 5.8.

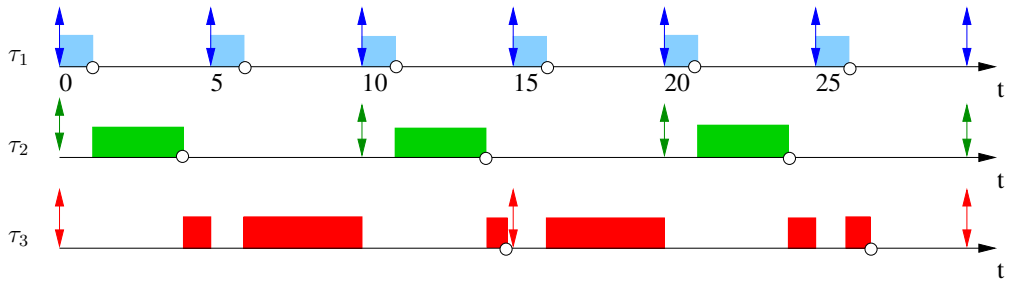


Figure 5.8 – Pire cas d'exécution augmenté ($C_{nc} + 10\%$)

◊

Cet exemple confirme que même si certains pire temps d'exécution sont sous-estimés, le système peut, dans une certaine mesure, absorber des erreurs d'estimation.

5.3 Prise en compte dans le calcul des budgets

L'analyse de sensibilité peut être modifiée afin de prendre en compte le caractère multi-critique d'un système. Comme nous l'avons exposé dans notre modélisation, le pire temps d'exécution d'une tâche critique peut être supposé estimé de manière sûre (généralement surdimensionné). Aucun dépassement de cette valeur ne peut exister au cours d'une exécution. Il n'est donc pas utile de relaxer les budgets des tâches critiques. Ainsi l'ensemble des temps creux peut être distribué entre les tâches non critiques. Nous proposons de le distribure proportionnellement au WCET estimé, en adaptant la fonction présentée équation (5.3)

Notons w_i le poids de chaque tâche τ_i. La direction de relaxation peut être dirigée par ces poids. Nous pouvons définir la fonction f (équation (5.3) section 5.1.1)

$$f : \mathbf{C} \mapsto (f(\mathbf{C}))_i = C_i + \lambda . w_i . C_i \tag{5.6}$$

avec

$$\lambda = \min_{i=1,\ldots,n} \max_{t \in schedP_i} \frac{t - \mathbf{n_i} \cdot \mathbf{C_i}}{\mathbf{n_i} \cdot \mathbf{C_i^w}}, \quad \mathbf{C^w} = \{w_i \ . \ C_i\}_{i \in \{1,\ldots,n\}} \tag{5.7}$$

Preuve : En utilisant l'équation (5.2), S' est ordonnançable si et seulement si

$$\max_{i=1,\dots,n} \min_{t \in schedP_i} \mathbf{n_i} \cdot \mathbf{C'_i} \leq t$$

$$\max_{i=1,\dots,n} \min_{t \in schedP_i} \mathbf{n_i} \cdot (\mathbf{C_i} + \lambda \mathbf{w_i}.\mathbf{C_i}) \leq t$$

$$\max_{i=1,\dots,n} \min_{t \in schedP_i} \lambda \leq \frac{t - \mathbf{n_i} \cdot \mathbf{C_i}}{\mathbf{n_i} \cdot \mathbf{w_i}.\mathbf{C_i}}$$

$$\lambda \leq \min_{i=1,\dots,n} \max_{t \in schedP_i} \frac{t - \mathbf{n_i} \cdot \mathbf{C_i}}{\mathbf{n_i} \cdot \mathbf{w_i}.\mathbf{C_i}} \triangleq \lambda_w^{max}$$

Les variables schedP$_i$ et $\mathbf{n}_i$ sont définies section 5.1.1.

Nous pouvons fixer une valeur de w_i plus importante pour les tâches non-critiques afin de relaxer au mieux leur budget d'exécution. Si nous considérons l'estimation sûre du WCET pour les tâche critiques, nous pouvons fixer $w_i = 0$ pour ces tâches. Si le système ne comporte qu'une seule tâche non-critique, nous sommes dans le cas d'une relaxation uni-dimensionnelle. Si le système ne comporte que des tâches non-critiques, nous sommes dans le cas d'une relaxation proportionnelle. La méthode proposée est une généralisation permettant de choisir la meilleure direction de recherche pour la relaxation des budgets dans le cas d'un système multi-critiques.

5.4 Simulations

5.4.1 Ajout de la boîte de calcul des budgets

Le calcul du budget, dans cette section, n'est pas uniquement le résultat de la détermination de l'estimation du pire temps d'exécution. Celui-ci dépend de la stratégie adoptée : configuration implicite naïve, proportionnelle, relative aux niveaux de criticité. Nous devons donc créer une nouvelle fonction permettant de calculer le budget d'exécution de chaque tâche (calcul du paramètre λ).

La figure 5.9 représente l'ensemble du dispositif de simulation (générateur + simulateur), ses entrées (paramètres) et ses sorties (mesures). Le nouveau bloc permet de calculer les budgets d'exécution et a été, tout comme le générateur, implémenté sous Matlab.

5.4.2 Résultats de simulations

Systèmes mono-critique

Pour la première expérience, nous simulons exactement les mêmes architectures (mêmes paramètres pour chaque tâche, temps d'exécution inclus) que pour l'étude implicite naïve. Nous rappelons les paramètres du générateur d'architecture : $n = 10, U = 0.9$ et $D_i/T_i = 0.67$. Le simulateur est configuré avec $\gamma^{min} = 0.95$ et $\gamma^{max} = 1.15$.

Le mécanisme de protection temporelle est maintenant configuré selon le schéma suivant :

$$\begin{cases} TIMEFRAME & = T_i \\ EXECUTION\_BUDGET & = B_i = (1+\lambda).C_i \end{cases}$$

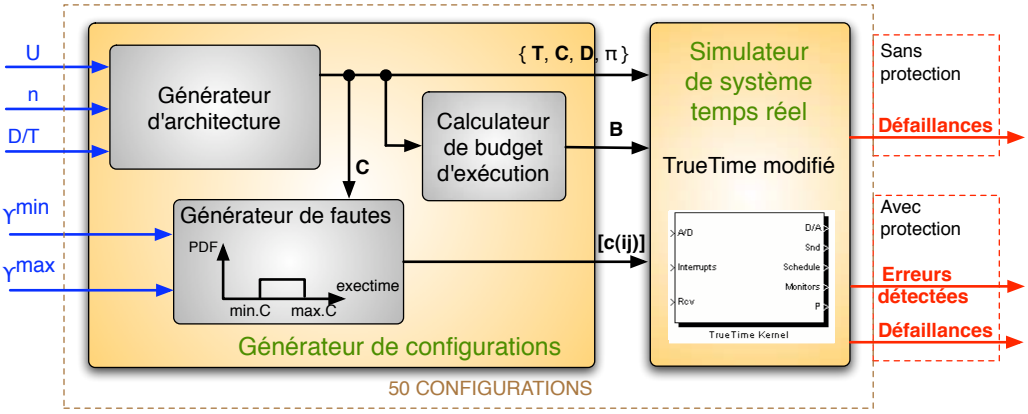


Figure 5.9 – Générateur et simulateur d'applications

Le nombre de violations d'échéance est mesuré avec et sans mécanisme de protection ainsi que le nombre d'instances tuées par la protection temporelle avant leur terminaison.

Les résultats sont donnés figure 5.10. Le nombre d'instances tuées par le mécanisme de protection (temps d'exécution supérieur au budget) sur l'ensemble des instances générées est tracé pour chaque architecture (ligne cercle, échelle de droite). Le nombre d'instances dépassant leur échéance sans mécanisme de protection est tracé pour chaque architecture (ligne continue, échelle de gauche). Les lignes pointillés représentent les valeurs moyennes de ces deux mesures.

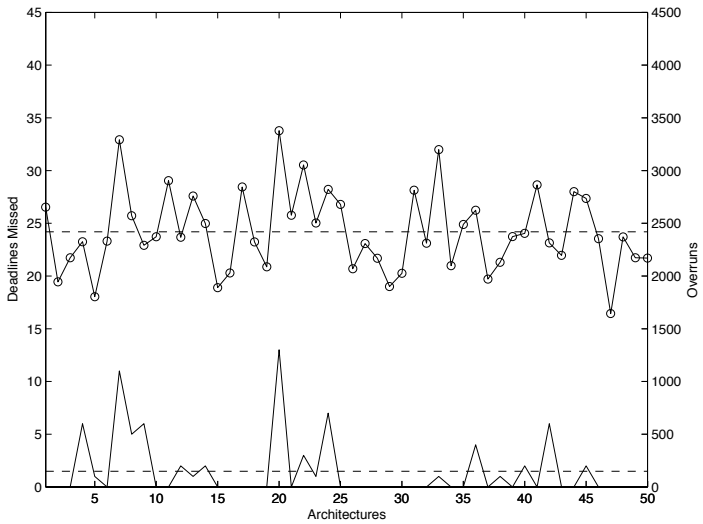


Figure 5.10 – Sur-exécutions et échéances ratées dans le cas où $D/T = 0.67$

Le tableau 5.2 résume les valeurs maximales, minimales, moyennes et écarts-types de ces deux mesures pour les différentes valeurs de $\eta = (D/T) \in \{1, 0.75, 0.67\}$.

La valeur de λ dépendant de l'échéance, le nombre de dépassements de budget dépend de la valeur de η ce qui explique la division des résultats pour les trois cas sur la première ligne

	η	max	min	m	σ
Sur-exécutions	1	2566	752	1650	536
	0.75	3361	1544	2380	400
	0.67	3378	1644	2420	86
Échéances ratées (avec protection)	*	0	0	0	0
Échéances ratées (sans protection)	1	2	0	0.04	0.28
	0.75	11	0	1.28	2.69
	0.67	13	0	1.48	2.89

Tableau 5.2 – Sur-exécutions et échéances ratées dans le cas où $D/T = \{1, 0.75, 0.67\}$

du tableau 5.2. Comme nous pouvons le voir, le nombre de dépassements de budgets décroit en fonction de la valeur η. En effet, de meilleurs résultats sont observés pour les plus fortes valeurs de η. Ceci s'explique par le fait que plus η est élevé, plus l'échéance est lointaine et moins le système est contraint. La valeur du budget est donc plus importante et le nombre de dépassements moins élevé. Lorsque $D_i/T_i = 1$, le nombre moyen de dépassements de budgets est de 1650, à comparer avec la valeur 3887 obtenue avec une configuration implicite naïve (voir tableau 4.2 section 4.3.1).

Le taux de défaillance est nul, la détection reste donc complète. La précision du mécanisme de protection temporelle est améliorée.

Systèmes multi-critiques

Pour la seconde expérience, nous considérons des systèmes multi-critiques. Par rapport aux remarques apportées sur la précision de l'estimation en fonction de la criticité, les temps d'exécution d'une tâche critique et d'une tâche non-critique sont tirés différemment. Les paramètres du générateur d'architecture sont toujours le mêmes : $n = 10$, $U = 0.9$ et $D_i/T_i = 0.67$. Le modèle de faute est différent : nous considérons des valeurs de γ^{min} et γ^{max} par niveau de criticité. Nous considérons dans cette étude 4 niveaux de criticité (ASIL A, B, C, D). La répartition du nombre de tâches par niveau de criticité et des valeurs de γ^{min} et γ^{max} sont résumées tableau 5.3.

ASIL	Temps d'exécution	$n_{niveauASIL}$	Poids (w)
A	$rand(0.95, 1.25) \times WCET$	1	2
B	$rand(0.95, 1.15) \times WCET$	2	1
C, D	$rand(0.95, 1) \times WCET$	3, 4	0

Tableau 5.3 – Répartition des niveaux de criticité et temps d'exécution

Pour ces simulations, nous faisons les hypothèse suivantes, une tâche ayant un niveau de criticité ASIL C ou D a une estimation sûre de son temps d'exécution ($\gamma^{max} = 1$). Une

relaxation de budget n'est pas nécessaire : $w = 0$. Une tâche de niveau de criticité ASIL A ou B peut avoir un pire temps d'exécution sous-estimé ($\gamma^{max} > 1$). Une relaxation des budgets est possible ($w = \{1, 2\}$). Le mécanisme de protection temporelle est maintenant configuré selon le schéma suivant :

$$\begin{cases} TIMEFRAME & = T_i \\ EXECUTIONBUDGET & = B_i = (1 + \lambda.w_i).C_i \end{cases}$$

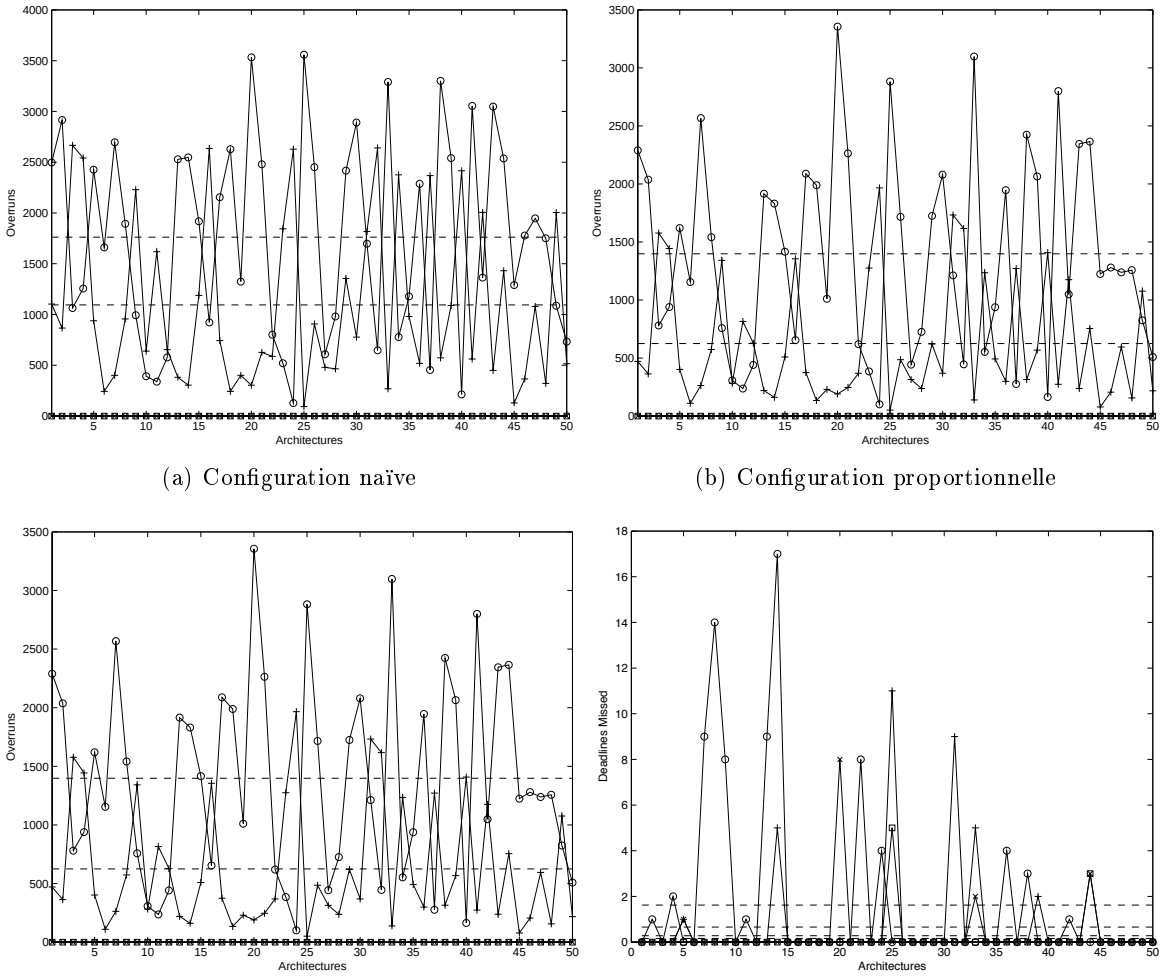
Le nombre de violations d'échéances est mesuré avec et sans mécanisme de protection ainsi que le nombre de instances tuées par la protection temporelle avant leur terminaison.

Les résultats sont donnés figure 5.11. Le nombre d'instances tuées par le mécanisme de protection (temps d'exécution supérieur au budget) sur l'ensemble des instances générées est tracé pour chaque architecture, chaque niveau de criticité et chaque dimensionnement de budget (configuration naïve implicite, configuration proportionnelle, configuration pondérée par les niveaux de criticité). Le nombre d'instances dépassant leur échéance sans mécanisme de protection est tracé pour chaque architecture et chaque niveau de criticité (figure 5.11(d)). Les lignes pointillés représentent les valeurs moyennes de ces deux mesures. Les niveaux de criticité sont tracés comme suit : ASIL D $\rightarrow \times$, ASIL C $\rightarrow \square$, ASIL B $\rightarrow +$ et ASIL A $\rightarrow \circ$.

	ASIL	*max*	*min*	*m*	σ
Sur-exécutions (config. implicite)	C et D	0	0	0	0
	B	2666	93	1094	818
	A	3559	127	1761	981
Sur-exécutions (config. proportionnelle)	C et D	0	0	0	0
	B	1966	51	624	519
	A	3356	101	1398	854
Sur-exécutions (config. pondérée)	C et D	0	0	0	0
	B	1966	15	682	818
	A	2630	73	956	721
Échéance ratées (sans protection)	D	8	0	0.28	1.23
	C	5	0	0.16	0..82
	B	11	0	0.66	2.19
	A	17	0	1.62	3.76

Tableau 5.4 – Sur-exécutions et échéances ratées dans le cas d'un système multi-critique

Le tableau 5.4 résume les valeurs maximales, minimales, moyennes et écarts type de ces mesures. Le nombre de dépassement de budget des instances de niveau de criticité ASIL A décroit avec la nouvelle stratégie considérée ($1761 \rightarrow 1398 \rightarrow 956$). En effet, ces instances ont accès à une relaxation plus importante ($w = 2$). Le nombre de sur-exécutions des instances de niveau de criticité ASIL B est plus important que celui détecté avec la méthode proportionnelle ($682 > 624$). Ceci est dû au choix des poids w_i qui doit être approprié par rapport au système. Un équilibre doit donc être trouvé pour la répartition des relaxations selon le niveau de criticité. Les instances de niveaux de criticité ASIL C et D ne peuvent pas dépasser leur budget par hypothèse ($\gamma^{max} = 1$). L'analyse du taux d'échéances ratées nous montre qu'une tâche non-critique peut entraîner la défaillance d'une tâche critique par propagation même si toutes



(a) Configuration naïve

(b) Configuration proportionnelle

(c) Configuration pondérée

(d) Pas de budget

Figure 5.11 – Sur-exécutions et échéances ratées dans le cas d'un système multi-critique

les tâches critiques sont correctement dimensionnées. Nous voyons ici l'importance de la protection temporelle afin de garantir un taux d'échec nul au moins au niveau des tâches critiques.

5.5 Discussion

Les méthodes de dimensionnement proposées dans ce chapitre permettent d'améliorer la précision du mécanisme de protection temporelle d'AUTOSAR OS dans le cas d'un système dont nous ne connaissons pas précisément les paramètres temporels (sous-estimation des temps d'exécution possible), ceci aussi bien dans le cas mono-critique que dans le cas multi-critique. Dans le second cas, la certitude d'estimation du pire cas pour les tâches critiques permet de relaxer plus largement le budget des tâches non-critiques dont l'effort effectué pour déterminer le temps d'exécution peut être limité (ce qui entraîne la possibilité de sous-estimation).

Des travaux parallèles à notre étude ont été publiés sur des études de sensibilité de systèmes multi-critiques par Dorin *et al.* (2009). Le papier propose une méthode différente de calcul des marges possibles pour chaque temps d'exécution de tâche. Cette relaxation prend en compte la modélisation proposée par Vestal (2007) mais uniquement dans le cas d'une relaxation mono-directionnelle (uniquement un budget d'exécution peut être relaxé). Malgré le choix décisif des poids, notre méthode, a l'avantage de pouvoir redistribuer un temps inutilisé pour chaque tâche proportionnellement à sa criticité. La distribution proportionnelle au WCET estimé pour les tâches non critiques est un choix qu'on peut remettre en question (voir chapitre 10).

Notre étude se limite au cas d'une étude statique de dimensionnement. En effet, le budget de temps non utilisé par une instance de tâche n'est pas récupéré. Ce phénomène peut être rapporté au cas des serveurs à scrutation qui perdent leur capacité en cas d'absence de tâche en attente. Cependant récupérer dynamiquement les parties de budgets non utilisées n'est pas chose facile puisque nous sommes dans le cas d'un ordonnanceur à priorité fixe. Quelques pistes pourraient être explorées (Davis *et al.*, 1993; Bougueroua *et al.*, 2007). Néanmoins, le monde automobile et plus généralement le domaine des systèmes critiques, afin de préserver un maximum de déterminisme, préfère que tous les paramètres impliqués dans l'ordonnancement soient connus au démarrage du système.

Une autre piste peut être explorée : celle d'une étude probabiliste. En effet, nous considérons que les temps d'exécution dans le cas d'une tâche non-critique sont incertains et sont généralement déterminés par tests. Une modélisation de ces temps d'exécution peut alors être de considérer des variables aléatoires. Nous exprimons dans le chapitre suivant comment cela peut être pris en compte dans la configuration du mécanisme de protection temporelle d'AUTOSAR OS.

« Toute connaissance dégénère en probabilité »

Traité de la nature humaine (1740) - David Hume

Chapitre 6

Configuration par analyse probabiliste

Sommaire

6.1 État de l'art 70
6.1.1 Analyse probabiliste et ordonnançabilité 70
6.1.2 Première approche 72
6.1.3 Seconde approche - cas d'un système à tâches concrètes 75
6.1.4 Méthode retenue pour l'étude 79
6.2 Modélisation du système 79
6.2.1 Modification du modèle d'application 79
6.2.2 Système multi-critique et choix des priorités 80
6.2.3 Exemple de système et d'utilisation de l'algorithme 82
6.3 Analyse 83
6.3.1 Étude de systèmes multi-critiques, analyse probabiliste et ordonnançabilité 83
6.3.2 Effet du budget d'exécution sur l'exécution 84
6.3.3 Position du problème 85
6.3.4 Optimisation unidirectionnelle 86
6.3.5 Exemple simple de système 86
6.4 Simulations 90
6.4.1 Paramètre de génération des applications 90
6.4.2 Modification de la boîte de calcul des budgets 92
6.4.3 Résultats de simulations 93
6.5 Discussion 95

Le précédent chapitre laisse entrevoir deux pistes à explorer dans le but de relaxer les budgets d'exécution des tâches : redistribution dynamique des budgets et étude probabiliste du problème. Cependant, la relaxation dynamique ne semble pas viable à l'heure actuelle dans

le monde automobile. Le présent chapitre est donc consacré à la relaxation des budgets d'exécution basée sur une étude probabiliste du système.

Plusieurs études probabilistes ont été réalisées sur des systèmes aux paramètres incertains. Nous pouvons citer notamment les travaux de Cucu et Tovar (2006) dans le cas de période d'activation incertaine ou les travaux de Tia *et al.* (1995) et Diaz *et al.* (2002) dans le cas de temps d'exécution incertains.

Le présent chapitre est consacré à la relaxation des budgets par récupération des temps creux en utilisant des résultats d'analyse probabiliste. En effet, si nous acceptons de dimensionner le système en utilisant des paramètres non sûrs pour certaines tâches, il peut paraître acceptable que certaines instances de ces tâches ne soient pas ordonnançables, si (1) cela n'affecte pas les tâches critiques et (2) cela bénéficie globalement au système (augmentation des budgets).

Le chapitre s'articule comme suit. La section 1 permet de faire un rappel sommaire sur les différentes études probabilistes proposées dans la littérature. Puis nous introduisons en section 2 le modèle de tâche retenu pour notre étude. La section 3 permet d'appliquer les résultats d'étude probabiliste dans le cas de systèmes multi-critiques afin de relaxer les budgets des tâches non-critiques. Des simulations en section 4 permettent d'afficher une comparaison entre un système dimensionné par une étude déterministe et probabiliste. Puis nous discutons en section 5 des résultats obtenus et des perspectives possibles.

6.1 État de l'art

6.1.1 Analyse probabiliste et ordonnançabilité

Une analyse déterministe nécessite de connaître parfaitement l'ensemble des paramètres de chaque tâche $\{T, C, D\}$ ou de considérer le pire cas (Liu et Layland, 1973; Audsley, 1991; Lehoczky *et al.*, 1989; Bini, 2004). Le test de faisabilité d'une instance d'une tâche τ_i de priorité π_i est généralement basé sur un calcul de son temps de réponse ou de la charge processeur de niveau i. Ces deux grandeurs sont, dans ce cas, des constantes notées respectivement $r_{i,j}$ et $W_i(t)$. Les deux méthodes sont les suivantes :

- Analyse du temps de réponse : une instance $\tau_{i,j}$ est ordonnançable si et seulement si :

$$r_{i,j} \leq D_i$$

- Analyse de la charge processeur : une instance $\tau_{i,j}$ est ordonnançable si et seulement si :

$$\exists t \in [a_{i,j}, a_{i,j} + D_i], \quad W_i(t) \leq t$$

Quand un des paramètres de la tâche (T, C ou D) n'est pas parfaitement connu, celui-ci peut être représenté par une variable aléatoire ($\mathcal{T}$, $\mathcal{C}$ ou $\mathcal{D}$). Dans ce cas, le temps de réponse $r_{i,j}$ de l'instance $\tau_{i,j}$ de même que la charge processeur $W_i(t)$ sont des variables aléatoires. Nous notons par la suite $\mathcal{R}_{i,j}$ et $\mathcal{W}_i(t)$ ces variables aléatoires.

Dans ce cas, le test d'ordonnançabilité n'a plus vraiment de sens. En effet, pour décider l'ordonnançabilité d'une tâche, il suffit d'extraire le pire cas du modèle probabiliste et de l'analyser avec l'une des méthodes déterministes évoquées ci-dessus. Dans le cas d'une étude probabiliste, nous nous intéressons plutôt à des systèmes non ordonnançables en considérant les pires cas pour lesquels nous cherchons à déterminer la probabilité qu'une tâche soit ordonnançable.

Cette notion de probabilité d'ordonnançabilité est assez délicate à exprimer pour une tâche. En effet, même si l'on peut généralement réduire l'étude à une séquence finie d'instances, comment définir de façon globale sur une tâche la probabilité d'ordonnançabilité ? Définir celle-ci comme une moyenne peut être trompeur car si une tâche comporte deux instances dans la période d'étude l'une totalement ordonnançable et l'autre totalement non-ordonnançable, comment peut-on qualifier cette tâche par une probabilité d'ordonnançabilité d'un demi ? D'une manière ou d'une autre, parler d'ordonnançabilité au niveau tâche entraîne une perte partielle de connaissance du système.

Cependant, il est facile de définir la probabilité d'ordonnançabilité d'une instance, $S_{\tau_{i,j}}$, en fonction du temps de réponse.

Définition 6.1. *La probabilité d'ordonnançabilité d'une instance $\tau_{i,j}$, $S_{\tau_{i,j}}$ est telle que*

$$S_{\tau_{i,j}} = \mathbb{P}(\mathcal{R}_{i,j} \leq D_i)$$

La question de la représentation et de l'interprétation des résultats d'une analyse probabiliste au niveau tâche reste ouverte. Cependant certains auteurs ont proposé des réponses. Ainsi, des études ont déjà été réalisées comme le cas de système où T est une variable aléatoire par Cucu et Tovar (2006) ou lorsque C est une variable aléatoire par Tia *et al.* (1995) et Diaz *et al.* (2002). Le tableau 6.1 résume une partie des études effectuées.

Article	$t_{i,j}$		$c_{i,j}$		Cas		Étude possible	
	T_i	$\mathcal{T}_i$	C_i	$\mathcal{C}_i$	non concret	concret	$r_{i,j}$	$\mathcal{S}_{\tau_{i,j}}$ ou borne
Cucu et Tovar (2006)		x	x		x		x	
Tia *et al.* (1995)	x			x	x			x
Diaz *et al.* (2002)	x			x		x	x	x

Tableau 6.1 – Exemples de cas d'études

Notre étude s'intéresse à des systèmes dont les temps d'exécution sont incertains (seul $\mathcal{C}$ est une variable aléatoire, les autres paramètres étant supposés parfaitement connus). Les deux dernières études peuvent alors constituer un point de départ pour notre problème. Ces deux études font l'hypothèse de tâches périodiques et indépendantes. La première étude (Tia *et al.*, 1995) est basée sur l'analyse de la charge processeur et considère un système composé de tâches non-concrètes en analysant le système à l'instant critique (instant où toutes les tâches sont activées simultanément). La méthode proposée fournit alors une borne minimale sur l'ordonnançabilité d'une tâche. La seconde étude (Diaz *et al.*, 2002) considère le cas de systèmes composés de tâches concrètes (offsets ϕ constants et connus). Cette méthode permet de calculer le temps de réponse $\mathcal{R}_{i,j}$ de chaque travail d'une tâche et donc sa probabilité d'ordonnançabilité.

6.1.2 Première approche

Les études menées par Tia *et al.* (1995) apportent une méthode de décision sur l'ordonnançabilité de chaque tâche du système dans le cas d'une application composée de tâches non concrètes, périodiques, indépendantes et ordonnancées sur un unique processeur selon une politique préemptive à priorités fixes.

Dans le cas d'une étude déterministe, un majorant de la charge processeur $w_i(t)$ sur tout intervalle $[a_{i,j}, f_{i,j}[$ est donné par :

$$w_i(t) \leq c_{i,j} + \sum_{k=1}^{i-1} \sum_{l=1}^{\lceil t/T_k \rceil} c_{k,l}$$

Dans le cas d'une étude pire cas, la valeur retenue pour le temps d'exécution d'une instance est une borne maximale $c_i = \max_j c_{i,j}$. L'équation précédente devient :

$$w_i(t) \leq \overline{w_i}(t) = c_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{T_k} \right\rceil .c_k \tag{6.1}$$

Le travail $\tau_{i,j}$ respecte son échéance s'il existe une date t dans $[a_{i,j}, d_{i,j}]$ telle que l'inégalité $w_i(t) \leq t$ soit vérifiée. Si $\overline{w_i}(t)$ vérifie cette propriété alors $w_i(t)$ également (équation (6.1)).

Dans le cas où les temps d'exécution sont des variables aléatoires, $\overline{w_i}(t)$ est une variable aléatoire. Notons $\overline{\mathcal{W}_i}(t)$ cette variable aléatoire. Celle-ci peut s'exprimer comme suit :

$$\overline{\mathcal{W}_i}(t) = \mathcal{C}_i \otimes \bigotimes_{k=1}^{i-1} \left\lceil \frac{t}{T_k} \right\rceil .\mathcal{C}_k \tag{6.2}$$

Le signe $\otimes$ correspond au produit de convolution de variables aléatoires. Le passage aux produits de convolution n'est possible que dans le cas de variables aléatoires considérées indépendantes (voir annexe B).

La probabilité $S_{\tau_{i,j}}$ qu'une instance de la tâche τ_i soit ordonnançable est minorée par :

$$S_{\tau_{i,j}} \geq \max_{t \in E} \mathbb{P}(\overline{\mathcal{W}_i}(t) \leq t) \tag{6.3}$$

où E est l'ensemble des dates d'activations des instances des tâches plus prioritaires $\{T_k, 2T_k, \ldots, \lfloor D_i/T_k \rfloor T_k, \forall k \in \{1, \ldots, i-1\}\} \cup D_i$. La borne donnée par l'équation (6.1) considère implicitement que l'activation de l'instance $\tau_{i,j}$ a lieu à l'instant critique. Cet instant critique, dans un tel système, représente le cas synchrone où toutes les tâches s'activent au même instant. Il est bien connu depuis (Liu et Layland, 1973) que cet instant correspond au pire cas concernant le temps de réponse des tâches dans le cas de tâches périodiques, indépendantes, non concrètes, à échéance contrainte ($D_i < T_i$) ou à échéance sur requête ($D_i = T_i$).

L'équation (6.3) donne ainsi une borne inférieure sur la probabilité d'ordonnançabilité de chaque instance $\tau_{i,j}$. Il ne s'agit que d'une borne et non d'un calcul exact. En effet, le test proposé consiste à étudier la valeur de $\mathcal{W}_i(t)$ à des dates précises (instants d'activation). Pour

chaque valeur de t, $\mathcal{W}_i(t)$ peut prendre différentes valeurs, correspondant à différentes trajectoires. En effet, chaque séquence d'exécution apporte une valeur pour la charge processeur celle-ci étant pondérée par la probabilité d'occurrence de cette séquence. Or il est possible que certaines trajectoires correspondent à des scenarii où l'instance de la tâche étudiée est déjà terminée. En ignorant le passé, le test est donc pessimiste.

Exemple . Considérons l'exemple du système $\mathcal{S}$ composé de 3 tâches τ_1, τ_2 et τ_3. Chaque tâche est ici représentée uniquement par son temps d'exécution, sa période et son échéance relative ($\tau = (\mathcal{C}, T, D)$) tel que

$$\tau_1 = (\begin{pmatrix} 1 & 2 \\ 0.50 & 0.50 \end{pmatrix}, 3, 3),\ \tau_2 = (\begin{pmatrix} 1 & 3 \\ 0.50 & 0.50 \end{pmatrix}, 8, 8),\ \tau_3 = (\begin{pmatrix} 2 \\ 1.00 \end{pmatrix}, 12, 12)$$

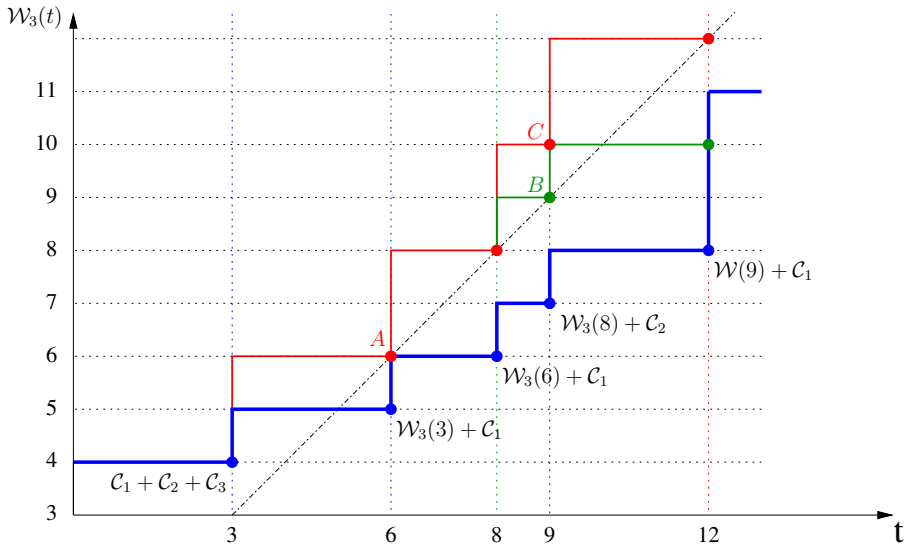


Figure 6.1 – Explication de l'inexactitude de la solution de Tia

La figure 6.1 représente une partie des valeurs possibles pour la charge processeur $\mathcal{W}_3(t)$ aux différents points d'ordonnancement ($E_3 = \{3, 6, 8, 9, 12\}$). La courbe bleue (trait épais) représente les valeurs minimales de la charge en chaque point (correspondant à la séquence d'exécution composée des temps d'exécution les plus faibles). Cette courbe passe en dessous de la bissectrice en $t = 6$ (point A). Cette trajectoire est donc ordonnançable c'est-à-dire que quelle que soit la séquence d'exécution ultérieure à la date $t = 6$ et comportant précédemment cette séquence, celle-ci est ordonnançable. Or, la trajectoire parvenant à la valeur C issue du point A, n'est plus ordonnançable (point au-dessus de la bissectrice). Cette non prise en compte du passé amène donc le pessimisme énoncé ci-dessus. La méthode proposée par Diaz *et al.* (2002), présentée section 6.1.3, permet de calculer le temps de réponse d'une instance et par conséquent une valeur exacte de son ordonnançabilité. La valeur obtenue pour la probabilité minimale d'ordonnançabilité d'une instance de la tâche τ_3 est telle que $S_{\tau_{3,j}} \geq 0.60937$.

De plus, la méthode de calcul proposée ne prend pas en compte les résidus (charge processeur résiduelle) dus à la non-ordonnançabilité possible des instances des tâches. En effet,

lorsqu'une instance ne finit pas son exécution avant son échéance, elle peut imposer un délai sur l'exécution des tâches moins prioritaires.

Prenons le cas de la séquence d'exécution suivante pour l'exemple précédent :

$$c_1 = \{2, 1, 2, 2, 2, 1, 2, 1 \ldots\}, c_2 = \{3, 1, 3, \ldots\}, c_3 = \{2, 2, \ldots\}$$

Nous obtenons le chronogramme présenté figure 6.2. Dans ce cas, les échéances de certaines tâches ne sont pas respectées mais il n'existe aucun résidu à la fin de l'hyperpériode.

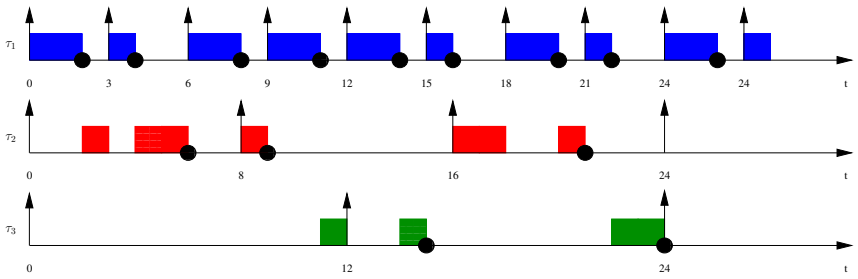


Figure 6.2 – Exemple d'exécution sans présence de résidu

Prenons maintenant le cas de la séquence d'exécution suivante :

$$c_1 = \{2, 2, 2, 2, 2, 1, 2, 1, \ldots\}, \quad c_2, c_3 \text{ inchangés}$$

Nous obtenons le chronogramme présenté figure 6.3. Dans ce cas, les temps de réponses des instances de τ_2 sont plus importants ce qui retarde l'exécution des instances de la tâche τ_3. À la fin de l'hyperpériode, il reste une part d'exécution de l'instance $\tau_{3,2}$ à exécuter. Un résidu est donc présent et les temps de réponse des instances présentes dans la deuxième hyperpériode seront plus importants que si l'on considère un système sans résidu. L'instant critique déterminé par Liu et Layland (1973) dans le cas déterministe est donc plus complexe. Dans certains cas il faut observer plusieurs hyperpériodes afin d'en dégager le pire-cas.

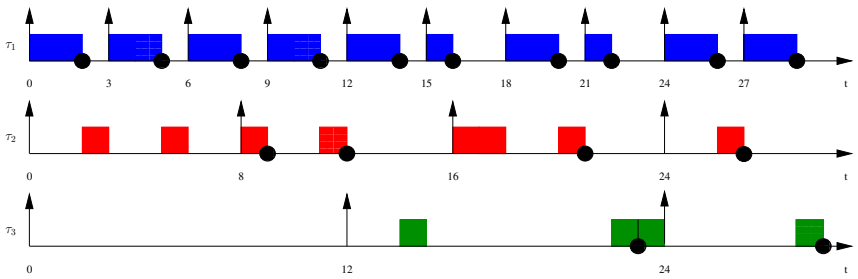


Figure 6.3 – Exemple d'exécution avec résidus

◊

Cet exemple illustre que la méthode proposée par Tia *et al.* peut amener à des résultats incorrects.

6.1.3 Seconde approche - cas d'un système à tâches concrètes

Les études menées par Diaz *et al.* (2002, 2004); López *et al.* (2008) apportent une méthode de décision de l'ordonnançabilité d'une tâche dans le cas d'une application composée de tâches concrètes, périodiques, indépendantes et ordonnancées sur un unique processeur selon une politique à priorité fixe préemptive. Le modèle est identique à celui proposé ci-dessus, cependant, dans ce cas d'étude, les offsets des tâches sont considérés connus et fixés. Le modèle est complété en introduisant un nouveau paramètre pour chaque tâche qui est son offset (ou retard au démarrage) ϕ_i. La date d'activation de l'instance $\tau_{i,j}$, $a_{i,j}$ est donc $a_{i,j} = \phi_i + j.T_i$.

Afin de simplifier les notations, un simple indice sera parfois utilisé pour définir la date d'activation d'une instance de tâche. Nous noterons a_k cette date d'activation. Ce simple indice correspond à l'ordre d'activation des instances des tâche considérées en faisant l'hypothèse $j_1 < j_2 \Rightarrow a_{j_1} \leq a_{j_2}$.

Le temps de réponse d'une instance $\mathcal{R}_{i,j}$ est déterminé par l'équation suivante :

$$\mathcal{R}_{i,j} = \mathcal{W}_\pi(a_{i,j}) \otimes \mathcal{C}_i \otimes \mathcal{J}_{i,j} \tag{6.4}$$

où $\mathcal{W}_\pi(t)$ est le π-level backlog (*cf.* ci-après) et $\mathcal{J}_{i,j}$ est la charge générée par les instances plus prioritaires que $\tau_{i,j}$ et qui s'exécutent pendant $[a_{i,j}, a_{i,j} + \mathcal{R}_{i,j}]$.

Le π-level backlog $\mathcal{W}_\pi(t)$ définit la somme des exécutions restantes à la date t des instances de tâches dont la priorité est supérieure ou égale à π. Cette variable est calculée en utilisant la procédure itérative suivante (López *et al.*, 2008).

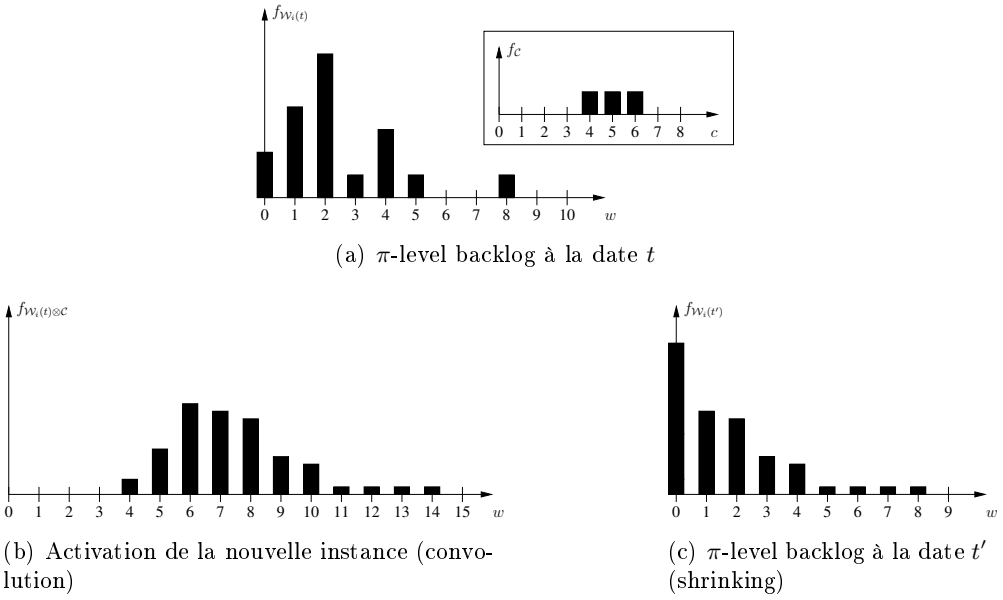
$$\begin{array}{lcl} \mathcal{W}_\pi(a_{k_0}) & = & 0 \\ \mathcal{W}_\pi(a_k) & = & \text{shrink } (\mathcal{W}(a_{k-1}) \otimes \mathcal{C}_{k-1}, a_k - a_{k-1}) \end{array} \tag{6.5}$$

k_0 est la date d'activation de la première instance de tâche dont la priorité est supérieure ou égale à π activée dans l'hyperpériode considérée. La procédure est ainsi itérée pour chaque instance activée avant t et de priorité supérieure ou égale à π. La fonction shrink$(\cdot)$ produit une variable aléatoire telle que :

$$f_{\text{shrink}(\mathcal{W},\Delta)}(x) = \begin{cases} 0 & \text{si } x < 0 \\ \displaystyle\sum_{w=-\infty}^{0} f_{\mathcal{W}}(w + \Delta) & \text{si } x = 0 \\ f_{\mathcal{W}}(x + \Delta) & \text{si } x > 0 \end{cases} \tag{6.6}$$

Cette transformation permet de reculer toutes les valeurs supérieures à Δ de Δ et d'accumuler toutes les valeurs inférieures à Δ en 0. La figure 6.4 illustre sur un exemple la procédure de calcul du π-level backlog sur une itération.

Le temps de réponse $\mathcal{R}_{i,j}$ est alors calculé en ajoutant le temps d'exécution de l'instance $\tau_{i,j}$ au backlog. Ceci n'est valable que si aucune interférence de tâches plus prioritaires n'a lieu. On note $\mathcal{R}_{i,j}^{[a_{i,j},a_{j+1}]} = \mathcal{W}(a_{i,j}) \otimes \mathcal{C}_i$, le temps de réponse partiel où a_{j+1} est la date d'activation de la première instance d'une tâche plus prioritaire à τ_i s'activant juste après celle-ci et $a_j = a_{i,j}$. L'exposant représente ainsi le domaine de validité du temps de réponse partiel calculé. Pour



(a) π-level backlog à la date t

(b) Activation de la nouvelle instance (convolution)

(c) π-level backlog à la date t' (shrinking)

Figure 6.4 – Exemple d'une itération de calcul du π-level backlog (Diaz *et al.*, 2002)

déterminer le temps de réponse du travail $\tau_{i,j}$, il faut ainsi ajouter itérativement les différentes interférences des tâches plus prioritaires. La procédure est définie par

$$\begin{array}{lcl} \mathcal{R}_{i,j}^{[a_{i,j},a_{j+1}]} & = & \mathcal{W}_\pi \otimes \mathcal{C}_i \\ \mathcal{R}_{i,j}^{[a_{i,j},a_{k+1}]} & = & \mathrm{AF}\left(\mathcal{R}_{i,j}^{[a_{i,j},a_{k+1}]}, a_k - a_{i,j}, \mathcal{C}_k\right) \end{array} \tag{6.7}$$

où AF est défini comme suit

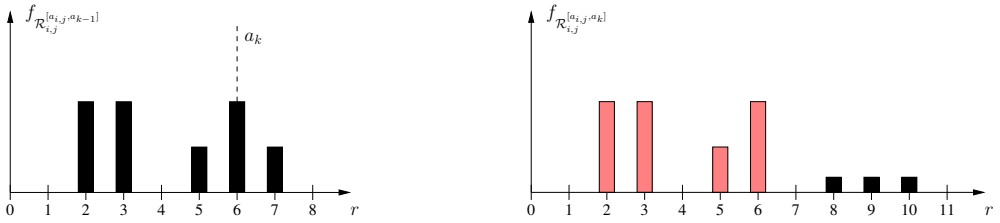
$$f_{\mathrm{AF}(\mathcal{R},\Delta,\mathcal{C})}(x) = \begin{cases} f_{\mathcal{R}}(x) & \text{pour } x \leq \Delta \\ \displaystyle\sum_{i=\Delta+1}^{\infty} f_{\mathcal{R}}(i).f_{\mathcal{C}}(x-i) & \text{pour } x > \Delta \end{cases} \tag{6.8}$$

La figure 6.5 illustre sur un exemple la procédure de calcul du temps de réponse sur une itération.

Ces études permettent également de fixer une fenêtre d'étude des instances à analyser. En effet, Diaz *et al.* (2002) démontre la stabilité du calcul de temps de réponse dans le cas où l'utilisation processeur moyenne $\overline{U}$ est inférieure à 1. Cette utilisation moyenne est définie par

$$\overline{U} = \bigotimes_{i=1}^{n} \mathcal{U}_i = \bigotimes_{i=1}^{n} \frac{\mathcal{C}_i}{T_i} \tag{6.9}$$

Dans ce cas, il existe un état stationnaire des temps de réponse des instances qui peut être calculé en déterminant l'état stationnaire des résidus d'exécution en fin d'une hyperpériode (steady state backlog). Les résidus s'accumulent jusqu'à atteindre un état stationnaire. Le calcul des temps de réponse des instances dans une hyperpériode en prenant en compte les



(a) Temps de réponse partiel date a_{k-1}

(b) Temps de réponse partiel date a_k

Figure 6.5 – Exemple d'une itération de calcul du temps de réponse (Diaz *et al.*, 2002)

résidus de l'état stationnaire représente ainsi le "pire-cas" (des cas plus favorables peuvent être présents avant l'accumulation complète des résidus).

Plusieurs méthodes de calcul de cet état stationnaire sont apportées (analytique et itérative). Dans (Diaz *et al.*, 2004), la seule méthode générale retenue est celle dite itérative car l'étude analytique s'avère lourde en calculs et mal conditionnée. La méthode itérative consiste à calculer les résidus (backlog) au bout d'une, puis deux, puis ... hyperpériodes jusqu'à obtention d'un état stationnaire. Un critère de convergence comme l'erreur quadratique entre deux itérations successives peut constituer le point d'arrêt du calcul de l'état stationnaire. L'état stationnaire pour chaque tâche $\mathcal{B}_\pi^{stat}$ déterminé, celui-ci impacte toutes les instances à analyser.

$$\mathcal{R}_{i,j} = \mathcal{B}_\pi^{stat} \otimes \mathcal{W}_\pi(a_{i,j}) \otimes \mathcal{C}_i \otimes \mathcal{J}_{i,j} \tag{6.10}$$

Dans le cas où l'utilisation processeur maximale U^{max} est inférieure à 1, l'étude (Diaz *et al.*, 2002) indique que l'état stationnaire est atteint à la fin de la première hyperpériode. Cette utilisation maximale est définie par

$$\overline{U} = \sum_{i=1}^{n} U_i^{max} = \sum_{i=1}^{n} \frac{C_i^{max}}{T_i} \tag{6.11}$$

L'étude de tels systèmes dans le cas d'une politique d'ordonnancement préemptive à priorités fixes est intéressante puisque certains systèmes dont l'utilisation maximale est inférieure à 1 peuvent être non ordonnançables (Audsley *et al.*, 1993).

Le tableau 6.2 répertorie les différents cas d'étude. Dans ce tableau, $\overline{T}$ représente la valeur de l'hyperpériode. Celle-ci dépend du système considéré. Dans le cas synchrone, l'hyperpériode est égale au plus petit commun multiple (ppcm) des périodes des tâches considérées ($\overline{T} = \text{ppcm}_{\{i=1,\ldots,n\}}\{T_i\}$). Dans le cas de tâches concrètes, l'hyperpériode est telle que $\overline{T} = \max_{\{i=1,\ldots,n\}}\{\phi_i\} + 2 * \text{ppcm}_{\{i=1,\ldots,n\}}\{T_i\}$. Le cas où $U^{max} \leq 1$ permet de ne pas calculer l'état stationnaire ce qui entraîne une importante simplification des calculs.

Exemple . Reprenons l'exemple présenté partie 6.1.2. Le développement des calculs est proposé annexe C. L'exemple est composé de tâches synchrones ($\phi_i = 0$). L'utilisation globale moyenne est $\overline{U} = 0.9167 \leq 1$ et l'utilisation maximale $U^{max} = 1.2083 > 1$. Nous sommes dans le cas ④ du tableau 6.2. L'analyse doit être réalisée sur une hyperpériode en calculant

Utilisation processeur	$U^{max} \leq 1$	$\overline{U} \leq 1$
Tâches concrètes	Période d'étude : $2.\overline{T}$ Pas de calcul du steady-state ①	Période d'étude : $\overline{T}$ Calcul du steady-state ②
Cas synchrone	Période d'étude : $\overline{T}$ Pas de calcul du steady-state ③	Période d'étude : $\overline{T}$ Calcul du steady-state ④

Tableau 6.2 – Différents cas d'étude et fenêtre d'analyse des instances

au préalable l'état stationnaire.

Le calcul de l'état stationnaire est déterminé par itération comme expliqué ci-dessus. La figure 6.6 représente le résultat des calculs du backlog à la fin de chaque itération $(1, 2, \ldots\ k$ hyperpériodes). La fonction de répartition du steady-state backlog est représentée par la courbe en rouge (gras). Elle est obtenue au bout de $k = 9$ itérations.

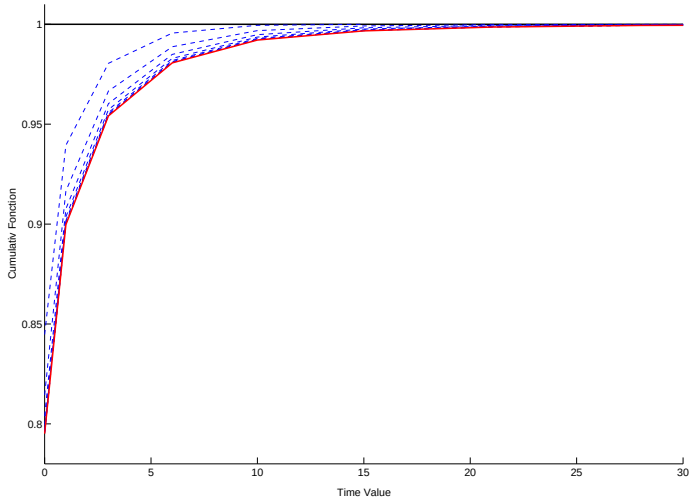


Figure 6.6 – Itérations de calcul de l'état stationnaire

Cet exemple permet de montrer le calcul des résidus pouvant être présents à la fin d'une hyperpériode et entraîner un délai sur le temps de réponse des instances. La méthode proposée ci-dessus est appliquée au système étudié. La valeur obtenue pour la probabilité minimale d'ordonnançabilité d'une instance de la tâche τ_3 est 0.40586. Ce résultat est inférieur à la valeur obtenue par la méthode proposée section 6.1.2. Ceci montre que la méthode développée par Tia *et al.* (1995) produit des résultats erronés car elle ne tient pas compte des résidus.

◊

6.1.4 Méthode retenue pour l'étude

Au vue des conclusions sur ces deux méthodes d'analyse probabilistes d'ordonnaçabilité, nous retenons pour la suite de l'étude la méthode probabiliste proposée par Diaz *et al.* (2002).

6.2 Modélisation du système

6.2.1 Modification du modèle d'application

En plus des paramètres C_i, T_i et D_i, nous assignons à chaque tâche un retard au démarrage ou offset ϕ_i. Les date d'activation des instances d'une tâche périodique τ_i de période T_i sont telles que $a_{i,j} = \phi_i + j * T_i$. Dans le cas de tâches concrètes les offsets sont considérés connus et fixés. Un système est dit synchrone si $\forall i, \phi_i = 0$.

Le niveau de criticité L_i associé à chaque tâche τ_i caractérise le niveau d'assurance des contraintes temporelles. Nous assignons à chaque niveau de criticité une probabilité d'ordonnançabilité à respecter $O_{l=L_i}$.

Le niveau de criticité est également associé au niveau d'assurance de l'estimation du temps d'exécution. Pour les tâches critiques, nous supposons la connaissance d'une borne supérieure sûre, C_{sup}, du pire temps d'exécution. Pour les tâches non critiques, nous supposons la connaissance d'une estimation de la variable aléatoire, $\mathcal{C}_i$, dont le temps d'exécution d'un travail $\tau_{i,j}$ est une réalisation. Cette variable peut être représentée soit par sa fonction de densité de probabilité $f_{\mathcal{C}_i}$, soit par sa fonction de répartition $F_{\mathcal{C}_i}$ (voir annexe B). Cette variable aléatoire peut être échantillonnée afin de réduire la taille des distributions et donc réduire la complexité des calculs (Refaat et Hladik, 2010). Dans la suite, nous utilisons la notation matricielle suivante :

$$\mathcal{C}_i = \begin{pmatrix} c_k \\ \mathbb{P}(\mathcal{C}_i = c_k) \end{pmatrix}_{k \in [1,\dots,K_i]} \tag{6.12}$$

où c_i est un temps d'exécution possible de la tâche τ_i, $\mathbb{P}(\mathcal{C}_i = c_k)$ la probabilité que le temps d'exécution soit égal à c_k et K_i le nombre de valeurs de temps d'exécution pour la tâche.

Le temps d'exécution de chaque tâche est également caractérisé par une fonction $\mathfrak{C}_i$ (Baruah et Vestal, 2008) de $\mathbb{N}$ dans $\mathbb{R}^+$ qui spécifie un pire temps d'exécution $\mathfrak{C}_i(l)$ par niveau de criticité $l \in \mathbb{L}$. Cette fonction respecte la propriété suivante :

$$\forall i \in \{1 \dots n\}, \forall l \in \mathbb{L},\ \mathfrak{C}_i(l) \leq \mathfrak{C}_i(l+1)$$

Chaque fonction $\mathfrak{C}_i$ est à mettre en relation avec l'estimation des temps d'exécution et de leur précision. Dans notre étude, nous considérons pour les tâches critiques la fonction $\mathfrak{C}_i$ telle que

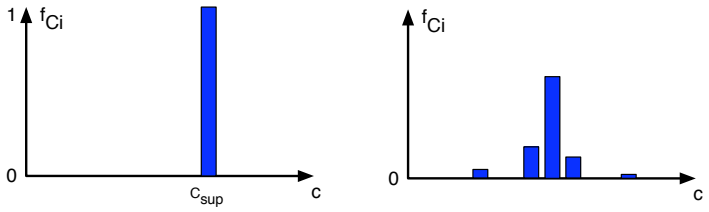
$$\mathfrak{C}_i(l=0) = \mathfrak{C}_i(l=1) = C_{sup} \tag{6.13}$$

En effet, le bon fonctionnement de ces tâches doit être sûr et la prise en compte de la borne supérieure permet de le garantir. Pour les tâches non-critiques, nous choisissons de définir la fonction $\mathfrak{C}_i$ de la façon suivante

$$\begin{cases} \mathfrak{C}_i(l=0) = \overline{C},\ \mathbb{P}(\mathcal{C}_i \leq \overline{C}) \geq p_i \\ \mathfrak{C}_i(l=1) = \zeta_i.\overline{C},\ \zeta_i \geq 1 \end{cases} \tag{6.14}$$

La valeur $\overline{C}$ représente un pire temps d'exécution "moyen" dont le paramètre p_i fixe la fiabilité. Le paramètre ζ_i permet de fixer une marge de sécurité. Ce paramètre doit être choisi suffisamment grand afin de s'assurer que la valeur $\zeta_i.\overline{C}$ soit, d'une façon certaine, une borne supérieure de la valeur d'exécution de la tâche (Cassé *et al.*, 2010).

La figure 6.7 représente les deux types de distributions utilisées (tâche critique et non-critique), ainsi que les valeurs correspondantes de la fonction $C(l = \cdot)$.



(a) Tâche critique (b) Tâche non-critique

Figure 6.7 – Exemple de fonctions de densité de probabilité pour le temps d'exécution

En résumé, un système S peut donc être vu comme un ensemble de n tâches, chaque tâche τ_i étant caractérisée par un 7-uplet $\big((\mathcal{C}_i, \mathfrak{C}_i), \phi_i, D_i, T_i, L_i, \pi_i \big)$.

$$S = \{\tau_i\}_{i=1\ldots n} = \big\{ \big((\mathcal{C}_i, \mathfrak{C}_i), \phi_i, D_i, T_i, L_i, \pi_i \big) \big\}_{i=1\ldots n}$$

6.2.2 Système multi-critique et choix des priorités

Nous utilisons les définitions proposées par Baruah et Vestal (2008) sur les systèmes multi-critiques et les notions de faisabilité correspondantes.

Définition 6.2. *Quelque soit le système multi-critique S composé de tâches sporadiques, on peut définir le système traditionnel $\overline{S}$ tel que*

$$\overline{S} = \big\{ \big((\mathcal{C}_i, \overline{C_i}), \phi_i, D_i, T_i, \pi_i \big) \big\}_{i=1\ldots n}$$

où $\overline{C_i} = C_i(l = L_i)$ - dans ce cas le système traditionnel ne comporte plus de niveau de criticité

Définition 6.3. *Un système multi-critique S composé de tâches sporadiques est dit faisable si et seulement si le système traditionnel $\overline{S}$ correspondant est faisable.*

Exemple . Soit le système synchrone présenté par Baruah et Vestal (2008) composé de tâches $S = \{\tau_1, \tau_2\}$ tel que

$$\begin{array}{llllll} \mathfrak{C}_1(l=0) = 2, & \mathfrak{C}_1(l=1) = 2, & \phi_1 = 0, & D_1 = T_1 = 4, & L_1 = 1 \\ \mathfrak{C}_2(l=0) = 2, & \mathfrak{C}_2(l=1) = 5, & \phi_2 = 0, & D_2 = T_2 = 7, & L_2 = 0 \end{array}$$

Dans le cas d'une étude d'ordonnançabilité classique, le pire cas doit être considéré. Le système étudié est donc le système suivant : $S_1 = \{(2,4,4),(5,7,7)\}$. Dans ce cas, aucune configuration ne permet d'ordonnancer le système avec un ordonnanceur de type FFP.

Dans le cas d'une étude d'ordonnançabilité multi-critiques, le système traditionnel étudié est le système suivant : $S_1 = \overline{S} = \{(2,4,4),(2,7,7)\}$. Dans ce cas, la configuration pour laquelle la tâche τ_1 est affectée de la plus forte priorité est ordonnançable.

Cet exemple permet de montrer que l'étude de systèmes multi-critiques et la méthode d'affectation des priorités proposée par Vestal *et al.* permet d'ordonnancer des systèmes non-ordonnançables avec une définition classique de l'ordonnançabilité d'une tâche.

◊

Nous utilisons l'algorithme proposé par Vestal (2007), basé sur l'algorithme d'Audsley (Audsley, 1991), pour fixer la priorité des tâches. Cet algorithme est basé sur la notion de faisabilité d'un système donné par la définition 6.3.

Le choix des priorités est simple. On note τ_{cur} l'ensemble des tâches non encore affectées d'une priorité. Initialement, aucune priorité n'est affectée aux tâches du système : $\tau_{cur} = S$. L'affectation des priorités se fait de la plus petite à la plus grande (*i.e.* la première tâche affectée est la moins prioritaire). À chaque itération, l'ordonnançabilité de chaque tâche est déterminée. Nous obtenons deux ensembles distincts : l'ensemble des tâches non-ordonnançables τ_{hi} et celui des tâches ordonnançables τ_o tel que

$$\tau_{cur} = \tau_{hi} \cup \tau_o, \quad \tau_{hi} \cap \tau_o = \emptyset$$

Une tâche τ_i de l'ensemble τ_o est sélectionnée pour affectation. En effet, pour l'ensemble des tâches non-ordonnançables, la priorité de chaque tâche devra être supérieure, aucune tâche de cet ensemble ne peut donc être choisie pour affectation. Comme nous ne voulons affecter qu'une tâche par niveau de priorité, un classement doit être réalisé dans l'ensemble des tâches ordonnançables. L'ordre choisi est basé sur le niveau de criticité de chaque tâche ordonnançable : la tâche ayant le niveau de criticité le plus faible est sélectionnée. Ce choix est spécifique à notre étude. En effet, il permet de dégager un maximum de budget d'exécution car une tâche critique placée à un niveau de priorité bas contraint plus fortement le système, celle-ci devant avoir une ordonnançabilité sûre. Si deux tâches ont le même niveau de criticité un choix arbitraire sera effectué.

Le pseudo-algorithme du choix des priorités est donné dans l'Algorithme 1. La commande initiale pour l'affectation est AugmentedAudsley$(S, 1)$.

Le choix de la sélection de la tâche ordonnançable par niveau de criticité est spécifique à notre étude de choix des budgets d'une tâche. D'autres choix peuvent être effectués comme celui proposé par Vestal (2007) qui permet de dégager une laxité maximale pour chaque tâche. Quant à notre choix, il permet de dégager davantage de budget d'exécution pour les tâches non-critiques (but de notre étude).

Algorithme 1: AugmentedAudsley (τ_{cur}, p)

```
/* But : Affecter les priorités aux tâches contenues dans τ_cur en fonction
   de leur niveau de criticité                                          */
début
    τ_o ← ∅ ;
    pour τ ∈ τ_cur faire
        si τ est ordonnançable alors
            τ_o ← τ_o ∪ {τ} ;
        fin
    fin
    si τ_o = ∅ alors
        " Système non ordonnançable " ;
    sinon
        Choisir la tâche τ ayant la plus basse criticité, lui affecter π = p ;
        τ_cur ← τ_cur \ {τ} ;
        si τ_cur = ∅ alors
            " Système ordonnançable " ;
        sinon
            AugmentedAudsley(τ_cur, p + 1) ;
        fin
    fin
fin
```

6.2.3 Exemple de système et d'utilisation de l'algorithme

Exemple . Soit le système synchrone S tel que :

$$\mathcal{C}_1 = \begin{pmatrix} 2210 & 2730 & 2860 & 2989 & 3380 \\ 0.10 & 0.37 & 0.30 & 0.13 & 0.10 \end{pmatrix}, \quad T_1 = D_1 = 10000, \quad L_1 = 0$$
$$\mathcal{C}_2 = \begin{pmatrix} 1615 & 1995 & 2090 & 2185 & 2470 \\ 0.18 & 0.26 & 0.36 & 0.18 & 0.02 \end{pmatrix}, \quad T_2 = D_2 = 30000, \quad L_2 = 0$$
$$\mathcal{C}_3 = \begin{pmatrix} 12600 \\ 1.00 \end{pmatrix}, \quad T_3 = D_3 = 60000, \quad L_3 = 1$$
$$\mathcal{C}_4 = \begin{pmatrix} 10600 \\ 1.00 \end{pmatrix}, \quad T_4 = D_4 = 60000, \quad L_4 = 1$$
$$\mathcal{C}_5 = \begin{pmatrix} 20570 & 25410 & 26620 & 27829 & 31460 \\ 0.05 & 0.11 & 0.38 & 0.31 & 0.15 \end{pmatrix}, \quad T_5 = D_5 = 120000, \quad L_5 = 0$$

La première étape consiste à déterminer la fonction $\mathfrak{C}(l = \cdot)$ des tâches non-critiques par rapport à leur FdP d'exécution. Les paramètres p_i et ζ_i (équation 6.14) pour les tâches non-critiques sont respectivement fixés à 80% et 1.25. À partir de ces valeurs, nous obtenons le système multi-critiques suivant :

$$\begin{array}{llll} \mathfrak{C}_1(l=0) = 2989, & \mathfrak{C}_1(l=1) = 3736, & T_1 = D_1 = 10000, & L_1 = 0 \\ \mathfrak{C}_2(l=0) = 2185, & \mathfrak{C}_2(l=1) = 2731, & T_2 = D_2 = 30000, & L_2 = 0 \\ \mathfrak{C}_3(l=0) = 12600, & \mathfrak{C}_3(l=1) = 12600, & T_3 = D_3 = 60000, & L_3 = 1 \\ \mathfrak{C}_4(l=0) = 10600, & \mathfrak{C}_4(l=1) = 10600, & T_4 = D_4 = 60000, & L_4 = 1 \\ \mathfrak{C}_5(l=0) = 27829, & \mathfrak{C}_5(l=1) = 34786, & T_5 = D_5 = 120000, & L_5 = 0 \end{array}$$

Appliquons l'algorithme de choix des priorités au système S considéré. Le tableau 6.3 représente le résultat de chaque itération et le choix de la tâche sélectionnée. Le système étudié est or-

	τ_1	τ_2	τ_3	τ_4	τ_5
Iteration 1					**ordo** ($\pi_5 = 1$)
Iteration 2			ordo	**ordo** ($\pi_4 = 2$)	-
Iteration 3		**ordo** ($\pi_2 = 3$)	ordo	-	-
Iteration 4		-	**ordo** ($\pi_3 = 4$)	-	-
Iteration 5	**ordo** ($\pi_1 = 5$)	-	-	-	-

Tableau 6.3 – Exemple d'application de l'algorithme de choix des priorités

donnançable car chaque tâche est affectée d'une priorité. L'itération 4 montre le choix de la tâche de niveau de criticité inférieure τ_2.

◊

6.3 Analyse

Le but de cette étude est de relâcher au mieux les budgets d'exécution tout en garantissant un niveau de sûreté suffisant pour chaque tâche en rapport à son niveau de criticité. Comme nous l'avons dit précédemment nous utilisons, dans cette étude, seulement deux niveaux de criticité, soit deux types de tâches : les tâches critiques et les tâches non-critiques.

6.3.1 Étude de systèmes multi-critiques, analyse probabiliste et ordonnançabilité

Dans le cas des tâches critiques, une ordonnançabilité sûre doit être garantie et ce quelle que soit la valeur considérée pour son exécution. Dans ce cas, une étude déterministe classique doit être utilisée. Nous utilisons la condition d'ordonnançabilité énoncée par Bini (2004); Bini *et al.* (2006).

Dans le cas des tâches non-critiques, nous souhaitons garantir que la tâche reste dans un domaine acceptable d'ordonnançabilité (respect des probabilités d'ordonnançabilité optimales $O_{l=L_i}$ par niveau de criticité).

Nous pouvons définir le domaine d'acceptabilité comme suit :

Définition 6.4. *Le système est acceptable du point de vue de l'ordonnancement si et seulement si :*

$$\forall \tau_i \in S, \begin{cases} \tau_i \text{ ordonnançable}, & \text{si } \tau_i \text{ critique} \\ \mathcal{S}_{\tau_i} \leq O_{l=L_i}, & \text{si } \tau_i \text{ non-critique} \end{cases}$$

6.3.2 Effet du budget d'exécution sur l'exécution

Dans le cas d'une tâche critique, le budget est fixé en utilisant les résultats exprimés par Bertrand *et al.* (2009) pour le cas d'un système multi-critique, c'est-à-dire que son budget est fixé à son WCET estimé (qui est une borne supérieure sûre de son temps d'exécution maximal). Son temps d'exécution n'est donc pas affecté.

Dans le cas d'une tâche non-critique, le budget d'exécution est choisi de sorte que le système reste acceptable. La valeur de celui-ci peut être plus petite que la valeur maximale contenue dans la FdP d'exécution de la tâche considérée. Le budget d'exécution du mécanisme de protection temporelle proposé par AUTOSAR contraint le temps d'exécution tel que

$$\forall i \in \{1 \dots n\}, \forall j,\ c_{i,j} \leq B_i$$

La FdP d'exécution peut ainsi être tronquée lorsque le mécanisme de protection est appliqué. En effet, toutes les valeurs d'exécution qui sont supérieures à la valeur du budget ne sont pas acceptables. Ces valeurs sont donc ramenées à la valeur du budget B (les tâches sont stoppées par détection d'erreur *i.e.* lorsque $c_{i,j} > B$). Nous pouvons représenter la modification de la FdP par la fonction suivante

$$\text{trunc}(\mathcal{C}_i, B) = \begin{pmatrix} \begin{pmatrix} c_k \\ \mathbb{P}(\mathcal{C}_i = c_k) \end{pmatrix}_{c_k < B} & \begin{matrix} B \\ \mathbb{P}(\mathcal{C}_i \geq B) \end{matrix} \end{pmatrix}$$

Exemple . Prenons l'exemple d'une tâche ayant un temps d'exécution représenté par la FdP suivante

$$\mathcal{C}_i = \begin{pmatrix} 4 & 7 & 8 & 10 & 12 \\ 0.05 & 0.22 & 0.50 & 0.15 & 0.08 \end{pmatrix}$$

Si le budget d'exécution de cette tâche B_i est fixé à 9, la distribution tronquée devient

$$\text{trunc}(\mathcal{C}_i, 9) = \begin{pmatrix} 4 & 7 & 8 & 9 \\ 0.05 & 0.22 & 0.50 & 0.23 \end{pmatrix}$$

◊

L'utilisation d'un budget d'exécution B_i pour chaque tâche τ_i entraîne une modification globale du système considéré. Nous notons $\mathbf{B}$ le vecteur des budgets et $S_{(\mathbf{B})}$ le système modifié. Ce système est défini par :

$$S_{(\mathbf{B})} = \left\{ \left(\left(\text{trunc}(\mathcal{C}_i, B_i), \mathfrak{C}_i \right), D_i, T_i, L_i, \pi_i \right) \right\}_{i=1\dots n} \tag{6.15}$$

On définit également le système déterministe pire cas $S^{max}_{(\mathbf{B})}$ tel que

$$S^{max}_{(\mathbf{B})} = \left\{ \left(B_i, D_i, T_i, L_i, \pi_i \right) \right\}_{i=1\dots n} \tag{6.16}$$

Lorsque le budget d'exécution d'une tâche est modifié, la FdP d'exécution de celle-ci est modifiée et par la même son ordonnançabilité ainsi que celles de toutes les tâches moins prioritaires. Nous utilisons cette propriété pour déterminer un budget d'exécution pour chaque tâche du système.

6.3.3 Position du problème

Nous cherchons ainsi à optimiser la taille des budgets B_i de chaque tâche non-critique (au nombre de $\tilde{n}$) en fonction du critère d'ordonnançabilité énoncé par la définition 6.4. Le problème peut s'écrire sous la forme du problème d'optimisation à variables vectorielles suivant :

$$\underset{\mathbf{B}\in\mathbb{E}^{\tilde{n}}}{\text{Argmin}} \|f(\mathbf{B})\| \tag{6.17}$$

où $\mathbf{B}$ est le vecteur de budgets des tâches non-critiques $\{B_i\}_{1\leq i\leq\tilde{n}}$, E est défini plus loin, et f est la fonction définie par :

$$\begin{array}{rccl} f: & \mathbb{X}^{\tilde{n}} & \to & \mathbb{R}^n \\ & \mathbf{B} & \mapsto & (f(\mathbf{B}))_i = \left\{ \begin{array}{ll} sched_{\tau_i}(\mathbf{B}), & \text{pour } i \text{ tel que } \tau_i \text{ critique} \\ S_{\tau_i}(\mathbf{B}) - O_{l=L_i}, & \text{pour } i \text{ tel que } \tau_i \text{ non-critique} \end{array} \right. \end{array} \tag{6.18}$$

Dans le cas général, nous pouvons définir la fonction dans le domaine des réels ($\mathbb{X} = \mathbb{R}$). Dans notre étude, nous nous limitons aux nombres entiers ($\mathbb{X} = \mathbb{N}$). Ce nombre peut être un nombre de cycle d'horloge ou une unité de temps restant à définir. La fonction f reprend les critères d'ordonnançabilité définis par (6.4). La première partie concernant les tâches critiques représente leur ordonnançabilité. La fonction *sched* est telle que

$$sched_{\tau_i}(\mathbf{B}) = \left\{ \begin{array}{ll} 1, & \text{si } \tau_i \text{ dans } S^{max}_{(\mathbf{B})} \text{ est ordonnançable} \\ -\infty, & \text{sinon} \end{array} \right. \tag{6.19}$$

La deuxième partie, concernant les tâches non-critiques, définit la distance entre la probabilité d'ordonnançabilité et la probabilité optimale d'ordonnançabilité du niveau de criticité de la tâche considérée, $O_{l=L_i}$, qui est l'objectif à atteindre. Cette équation doit prendre en compte la présence du mécanisme de budget d'exécution et de son effet sur la modification du système. $S_{\tau_i}(\mathbf{B})$ représente donc la probabilité d'ordonnançabilité de la tâche τ_i dans le système $S_{(\mathbf{B})}$. Cette valeur peut être déterminée par la méthode énoncée section 6.1.3 (Diaz *et al.*). Les hypothèses et remarques énoncées dans ces sections doivent être prises en compte afin d'utiliser correctement celles-ci.

Le domaine de recherche de l'algorithme d'optimisation (6.17) doit alors être défini. Nous fixons pour chaque budget d'exécution de tâche non-critique, une valeur maximale qui est la valeur de la fonction $\mathfrak{C}(l = 1)$ (valeur maximale d'exécution dans le cas le plus critique) et, une valeur minimale qui est la valeur de la fonction $\mathfrak{C}(l = 0)$ (valeur moyenne d'exécution dans le cas non-critique). Nous avons

$$\mathbb{E}_i = [\![\mathfrak{C}_i(l = 0), \mathfrak{C}_i(l = 1)]\!] \tag{6.20}$$

Les tâches critiques quant à elles ont un budget fixé à la valeur maximale C_{sup} et leur budget n'a pas besoin d'être optimisé.

Dans le cas d'un espace fini, toutes les normes sont équivalentes. Nous choisissons la norme 1 dans l'espace des entiers naturels ($\|\cdot\| = \|\cdot\|_1$). Nous rappelons que

$$\forall x = \{x_1, \ldots, x_n\} \in \mathbb{N}^n, \quad \|x\|_1 = |x_1| + \cdots + |x_n|$$

6.3.4 Optimisation unidirectionnelle

Nous proposons de réduire le problème d'optimisation multi-variable ($B \in \mathbb{E}^n$) en problème d'optimisation unidirectionnelle. L'exploration de l'espace d'état est restreint à une unique direction choisie de même que dans le cas multi-critiques déterministe (voir section 5.3). Le vecteur B est tel que

$$\forall i \in \{1, \ldots, n\}, \quad B_i = (1 + \lambda.w_{l=L_i}).\mathfrak{C}_i(l = L_i) \tag{6.21}$$

où $w_{l=L_i}$ est la pondération sur la relaxation des tâches de niveau de criticité L_i et λ le taux de relaxation. Dans notre étude, nous considérons uniquement 2 niveaux de criticité (critique et non-critique) avec des pondérations telles que $w_{l=0} = 1$ et $w_{l=1} = 0$ (borne sûre). Le vecteur B peut se réduire à l'étude de

$$\begin{cases} B_{\text{critique}} & = \mathfrak{C}(l = 1) \\ B_{\text{non-critique}} & = (1 + \lambda).\mathfrak{C}(l = 0) \end{cases} \tag{6.22}$$

Nous pouvons ainsi réduire notre étude d'optimisation comme suit

$$\underset{\lambda \in [0, \lambda_{max}]}{\text{Argmin}} \left\| g(\lambda) \right\|, \quad g(\lambda) = f(\mathbf{B}), \quad \mathbf{B} = \left\lfloor (1 + \lambda).\mathfrak{C}(\mathbf{l} = \mathbf{0}) \right\rfloor \tag{6.23}$$

Dans ce cas, λ est un scalaire réel compris entre 0 (aucune augmentation) et λ_{max} (augmentation maximale). λ_{max} est défini comme suit

$$\lambda_{max} = \min_{1 \leq i \leq \tilde{n}} \mathfrak{C}(l = 1)/\mathfrak{C}(l = 0)$$

Nous utilisons une méthode classique d'optimisation de fonction scalaire ($\|g\| : \mathbb{R} \to \mathbb{R}$) pour résoudre le problème (6.23). La fonction g n'étant pas continue, il est difficile de connaître une dérivée de cette fonction. Nous allons nous orienter vers des méthodes par réduction d'intervalle. Celles-ci consistent à partir d'un intervalle de départ, $\Delta_0 = [\lambda_p^{min}, \lambda_p^{max}]$ et de réduire celui-ci jusqu'à obtenir une précision voulue (voir annexe D). La méthode retenue est la méthode du nombre d'or qui est une méthode à réduction d'intervalle régulier comparable à la dichotomie mais ne calculant qu'un seul point à chaque itération.

6.3.5 Exemple simple de système

Exemple (Impact des distributions de temps d'exécution)**.** Reprenons l'exemple présenté section 5.1.1 de deux tâches périodiques indépendantes dont les caractéristiques sont données tableau 6.4. Nous considérons le cas de deux tâches non-critiques. La représentation graphique du domaine d'ordonnançabilité, $\mathcal{C}$-space, en fonction des temps d'exécution C_1 et C_2 calculés précédemment, est redonnée figure 6.8. Nous ne considérons ici que des valeurs entières. Afin d'augmenter la précision des calculs, nous supposerons toutes les valeurs multipliées par 100 ($C_1 = 9.5 * 100 = 950, \ldots$).

Dans cette situation, l'affectation statique des budgets réalisée par une analyse déterministe multi-critique (voir section 5.1.1) nous donne des valeurs de $B_{d_1} = 543$ et $B_{d_2} = 814$ pour les tâches τ_1 et τ_2.

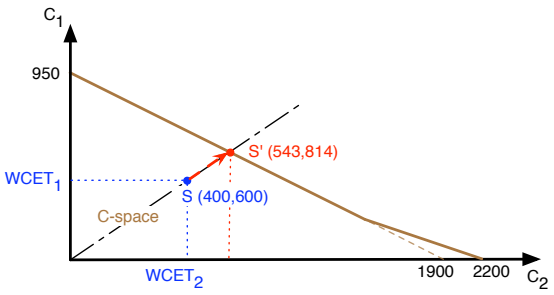


Figure 6.8 – Exemple de $\mathcal{C}$-space

Représentons maintenant chaque temps d'exécution par une variable aléatoire discrète $\mathcal{C}_i$ comportant m_i valeurs. Les distributions sont choisies telles qu'elles suivent une loi normale (discrétisée) de moyenne $\overline{c}_i$ et d'écart type σ_i, $\mathcal{C}_i \sim \mathcal{N}_d(\overline{c}_i, \sigma_i^2, m)$. L'ensemble des paramètres de chaque tâche est résumé tableau 6.4. La figure 6.9 représente la fonction de densité de probabilité du temps d'exécution $\mathcal{C}_2$ dans les cas particuliers où $\overline{c}_2 = B_{s_2}$ et $\sigma_2 = 0.01.\overline{c}_2$. Le nombre de valeurs m est fixé à 5 ou 10 valeurs.

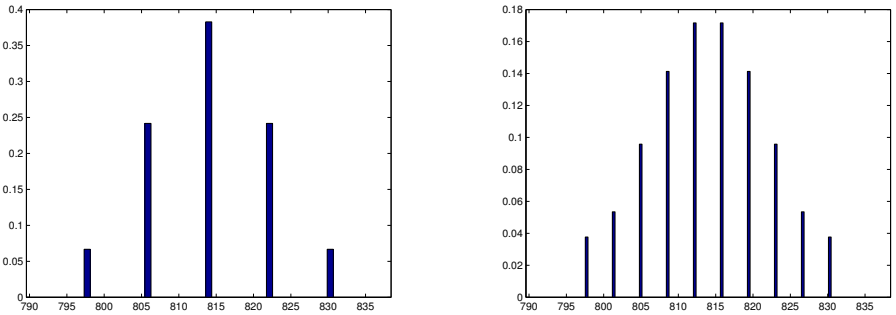
	T	D	$\overline{C}$	B_d	$\mathcal{C}$
τ_1	950	950	400	543	$\mathcal{N}_d(\overline{c}_1, \sigma_1^2, m)$
τ_2	2100	2100	600	814	$\mathcal{N}_d(\overline{c}_2, \sigma_2^2, m)$

Tableau 6.4 – Ensemble des paramètres de l'application

À partir de ces distributions, nous pouvons déterminer la probabilité de non-ordonnançabilité de la tâche τ_2 (la moins prioritaire). Nous choisissons d'utiliser la méthode proposée par Diaz *et al.* (2002), méthode retenue pour notre étude. Les résultats obtenus sont présentés figure 6.10.

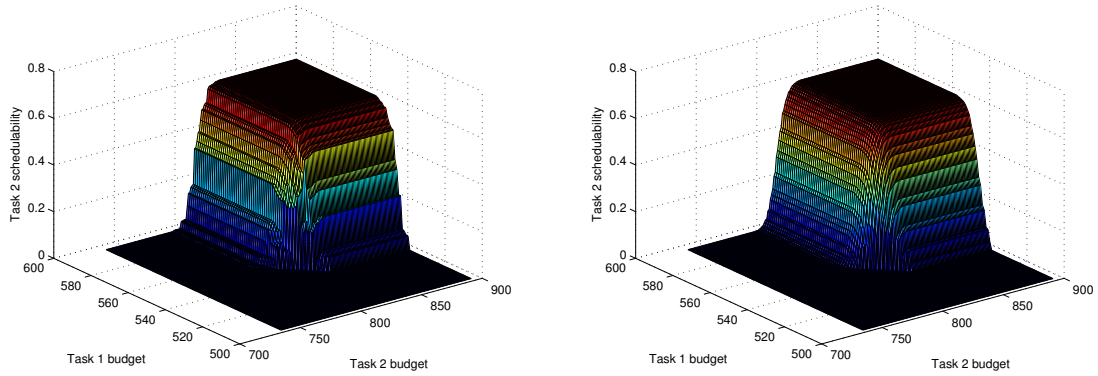
Nous pouvons remarquer que des paliers apparaissent dus au caractère discret des distributions (le nombre de valeurs m de temps d'exécution possible pour une instance de tâche influençant sur le nombre de paliers). Plus la valeur du budget d'exécution augmente pour l'une des deux tâches, plus les contraintes temporelles imposées aux tâches diminuent : la probabilité de non-ordonnançabilité de la tâche τ_2 augmente.

Nous pouvons également remarquer en traçant les lignes de niveaux d'ordonnançabilité, figure 6.11, que la zone dans laquelle le système est ordonnançable (probabilité de non-ordonnançabilité nulle) est plus importante que la zone d'ordonnançabilité définie par le $\mathcal{C}$-space (cas déterministe). Ce phénomène s'explique par le caractère discret et fini des FdP d'exécution : $\mathcal{C}$ possède une valeur minimale, C^{min}, et une valeur maximale, C^{max}, d'exécution. Dans le cas où $B_i \geq C_i^{max}$, le budget d'exécution n'a aucun effet sur l'exécution de la tâche considérée ($\text{trunc}(\mathcal{C}, B) = \mathcal{C}$). Dans le cas où $B_i \leq C_i^{min}$, la seule valeur d'exécution possible est fixée par la valeur du budget d'exécution ($\text{trunc}(\mathcal{C}, B) = (B; 1)$). Ces deux remar-



(a) Cas pour $m = 5$ (b) Cas pour $m = 10$

Figure 6.9 – Fonctions de probabilité (FdP) d'exécution considérées



(a) Cas pour $m = 5$ (b) Cas pour $m = 10$

Figure 6.10 – Probabilités de non-ordonnançabilité de τ_2 (ordonnée) pour les deux FdP considérées en fonction de la valeur du budget d'exécution (abscisse)

ques expliquent la présence d'un domaine d'ordonnançabilité agrandi (plan en bas à gauche) et un palier supérieur (probabilité maximale de non-ordonnançabilité en haut à droite).

◊

Le but de notre étude est d'optimiser au mieux les budgets afin de s'approcher au plus près d'une ligne de niveau ($S_{\tau_i} = O_{l=L_i}$). Dans l'exemple présenté, seule la tâche τ_2 est considérée (représentation de $1 - S_{\tau_i}$), car la tâche τ_1 reste toujours ordonnançable dans la zone d'étude. Dans le cas général, il est préférable d'analyser l'ensemble des distances séparant le point étudié des lignes de l'objectif recherché (comme énoncé par l'équation (6.18)).

Exemple (Utilisation de la méthode de Diaz *et al.* (2002) pour la recherche des budgets)**.** Reprenons l'exemple précédent, dans ce cas où les paramètres $\overline{c}_i$, σ_i et m_i sont fixés respectivement à $0.9.B_{d_i}$, $0.1.\overline{c}_i$ et 10.

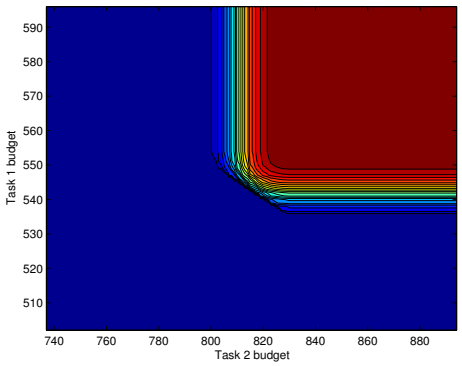


Figure 6.11 – Représentation par lignes de niveaux des résultats et $\mathcal{C}$-space

La figure 6.12 représente l'évolution de la probabilité d'ordonnançabilité de la tâche τ_2, S_{τ_2} en fonction du choix des budgets. Le zoom représente une partie de la courbe correspondant à une transition tâche 2 ordonnançable sûrement/probabilité d'ordonnançabilité.

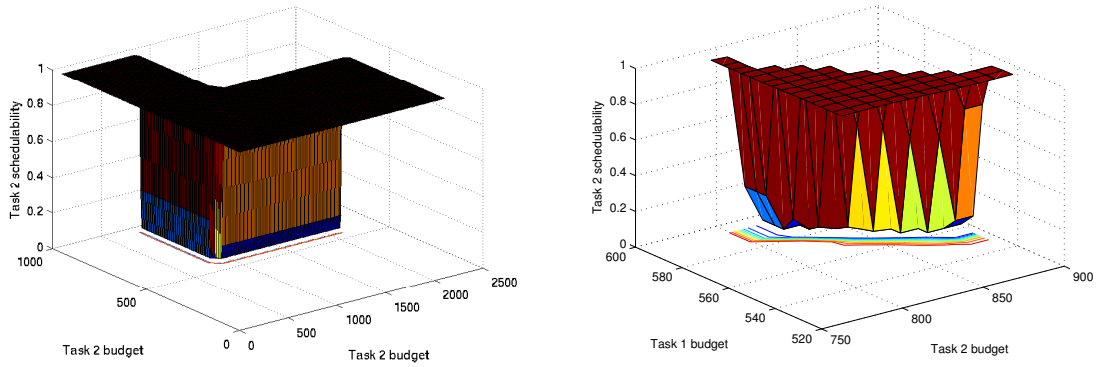
La figure 6.13 représente l'évolution du critère $crit(\mathbf{B}) = \|f(\mathbf{B})\|$ (f définie par l'équation (6.18)) en fonction du choix des budgets. La probabilité optimale d'ordonnançabilité des tâches est fixée à $O_{l=0} = 99\%$ (τ_1 et τ_2 sont non-critiques). Lorsque le système est ordonnançable la valeur du critère $crit$ est de 2% ($S_{\tau_i} = 100\%$). Si les budgets augmentent, le système se rapproche des conditions de fonctionnement désirées (points bas, couleur) puis le système devenant de moins en moins ordonnançable la valeur de la fonction augmente. Le but de l'optimisation présentée dans cette étude est de trouver un minimum de ce critère.

Si nous considérons une optimisation uni-directionnelle, le critère devient $\widetilde{crit}(\lambda) = \|g(\lambda)\|$ (g définie par l'équation (6.23)). Les figures 6.14 et 6.15 représentent respectivement l'ordonnançabilité de la tâche 2 et l'évolution du critère en fonction du paramètre λ (qui fixe les budgets d'exécution).

L'étude de l'optimisation uni-directionnelle est beaucoup plus facile. Le but est toujours de minimiser le critère mais l'étude d'une fonction scalaire minimise très grandement le temps de calcul. Dans cet exemple, le système est composé de deux tâches ce qui reste raisonnable. Cependant, dans la réalité le nombre de tâches est plutôt de l'ordre de 10 ou 20. L'optimisation uni-directionnelle est dans ce cas bien plus avantageuse. En réalité, lorsque nous parlons d'étude uni-directionnelle cela correspond à une unique direction de recherche sur la courbe présentée figure 6.13 (intersection de la courbe avec un plan).

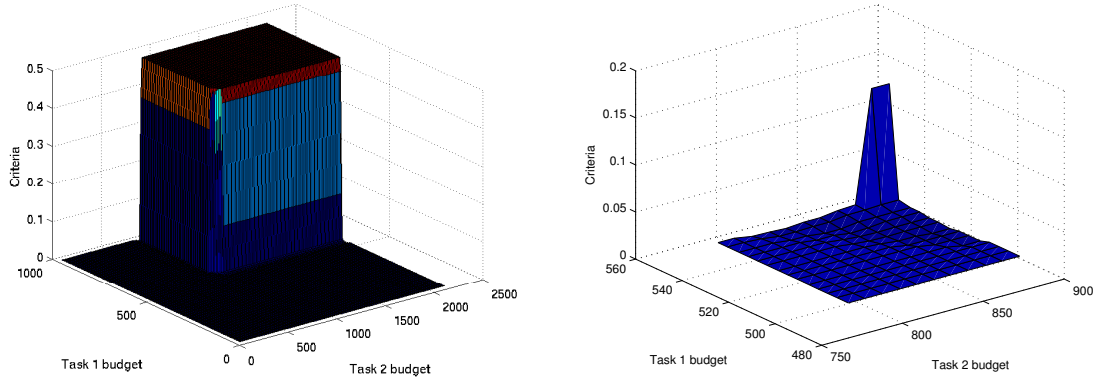
$\Diamond$

Cet exemple permet d'illustrer en deux dimensions (deux tâches) comment les budgets d'exécution doivent être choisis en fonction du résultat sur l'ordonnançabilité de l'ensemble des tâches du système. Dans la section suivante, nous allons générer un certain nombre de configurations afin de déterminer le gain obtenu sur la relaxation des budgets par l'utilisation de cette méthode.



(a) Ordonnançabilité (b) Zoom

Figure 6.12 – Ordonnançabilité de τ_2 (axe z) en fonction du choix des budgets (axe x et y)



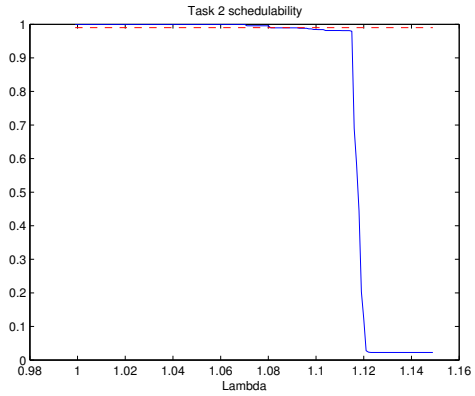
(a) Critère (b) Zoom sur minimum local

Figure 6.13 – Valeur du critère g (axe z) en fonction du choix des budgets (axe x et y)

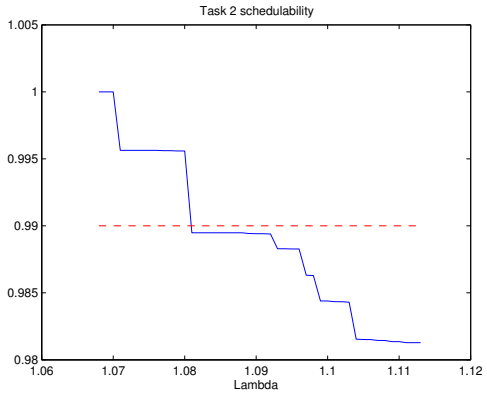
6.4 Simulations

6.4.1 Paramètre de génération des applications

Afin de simuler le comportement du mécanisme de protection dimensionné à l'aide des méthodes présentées ci-dessus, nous générons aléatoirement un ensemble d'applications. Notre générateur est basé sur celui présenté section 4.3.1. Nous considérons, de même que dans le chapitre précédent, des systèmes synchrones ($\forall i, O_i = 0$). Les principaux paramètres d'entrée sont : le nombre de tâches n, le nombre de tâches non-critiques $\tilde{n}$ et l'utilisation processeur de l'ensemble des tâches U. À l'aide de ces données, un ensemble de paramètres T_i (période), D_i (échéance), L_i (niveau de criticité), $\tilde{C}_i$ pour chaque tâche τ_i est déterminé. Les tâches sont supposées à échéance sur requête ($D = T$). La génération des valeurs d'exécution moyenne $\tilde{C}_i$ permet juste de donner un ordre de grandeur pour les temps d'exécution tel que l'utilisation processeur soit de l'ordre de U. Il faut alors générer pour chaque tâche sa fonction de probabilité d'exécution.

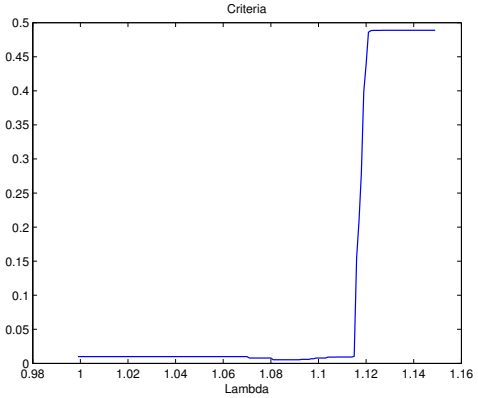


(a) Ordonnançabilité

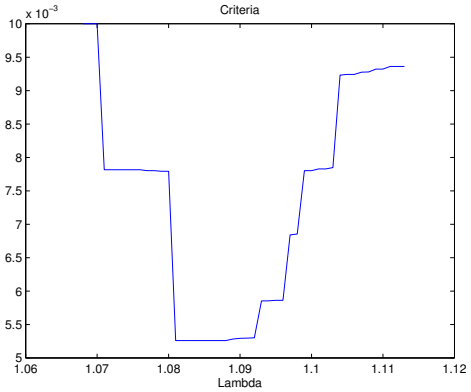


(b) Zoom sur minimum local

Figure 6.14 – Ordonnançabilité de τ_2 (ordonnée) en fonction du choix de λ (abscisse)



(a) Critère



(b) Zoom sur minimum local

Figure 6.15 – Valeur du critère g (ordonnée) en fonction du choix de λ (abscisse)

Pour une tâche critique, nous utilisons la valeur $\tilde{C}_i$ pour définir l'unique valeur d'exécution correspondant à une borne supérieure

$$\mathcal{C}_i = \begin{pmatrix} \tilde{C}_i \\ 1.00 \end{pmatrix}$$

Pour une tâche non-critique, la valeur $\tilde{C}_i$ permet de définir l'ensemble des valeurs d'exécution composant la FdP d'exécution. Nous fixons le nombre de valeurs à 5 tel que ces valeurs soient $\{0.85.\tilde{C}_i,\ 1.05.\tilde{C}_i,\ 1.10.\tilde{C}_i,\ 1.15.\tilde{C}_i,\ 1.30.\tilde{C}_i\}$. Les probabilités correspondantes sont tirées aléatoirement selon les critères énoncés ci-dessous. On a :

$$\mathcal{C}_i = \begin{pmatrix} 0.85.\tilde{C}_i & 1.05.\tilde{C}_i & 1.10.\tilde{C}_i & 1.15.\tilde{C}_i & 1.30.\tilde{C}_i \\ p_1 & p_2 & p_3 & p_4 & p_5 \end{pmatrix}$$

avec

$$\begin{cases} p_1 + p_2 + p_3 + p_4 + p_5 = 1.00 \\ p_1 + p_5 = 0.20 \end{cases}$$

À partir des distributions, nous calculons les valeurs de la fonction d'exécution $\mathfrak{C}_i(l = \cdot)$ comme expliqué section 6.2.3. Les valeurs des paramètres p_i et ζ (équation 6.14) pour les tâches non-critiques sont respectivement fixées à 80% et 1.25.

Les priorités π_i de chaque tâche sont alors déterminées à l'aide de l'algorithme présenté section 6.2.2. Nous obtenons ainsi un système tel que

$$S = \{\tau_i\}_{i=1...n} = \left\{ \left((\mathcal{C}_i, \mathfrak{C}_i), O_i, D_i, T_i, L_i, \pi_i \right) \right\}_{i=1...n}$$

6.4.2 Modification de la boîte de calcul des budgets

Le calcul du budget dépend de la stratégie adoptée : configuration par analyse déterministe (méthode multi-critique présentée dans le chapitre précédent) ou probabiliste (méthode présentée dans ce chapitre). Nous devons donc mettre à jour le calculateur de budget d'exécution de chaque tâche (calcul des paramètres $\lambda_{deterministe}$ et $\lambda_{probabiliste}$).

Dans cette étude, nous ne faisons plus appel au simulateur d'application temps réel TrueTime. En effet, l'objectif dégagé à travers les chapitres précédents est de relaxer au mieux les budgets d'exécution. Notre mesure principale est donc le taux de relaxation obtenu sur chaque tâche non-critique de l'application (les tâches critiques n'étant pas relaxées).

La figure 6.16 représente l'ensemble du dispositif de simulation (générateur principal + générateur de distributions pour les temps d'exécution), ses entrées (paramètres) et ses sorties (mesures). Le bloc de calcul des budgets d'exécution a été mis à jour.

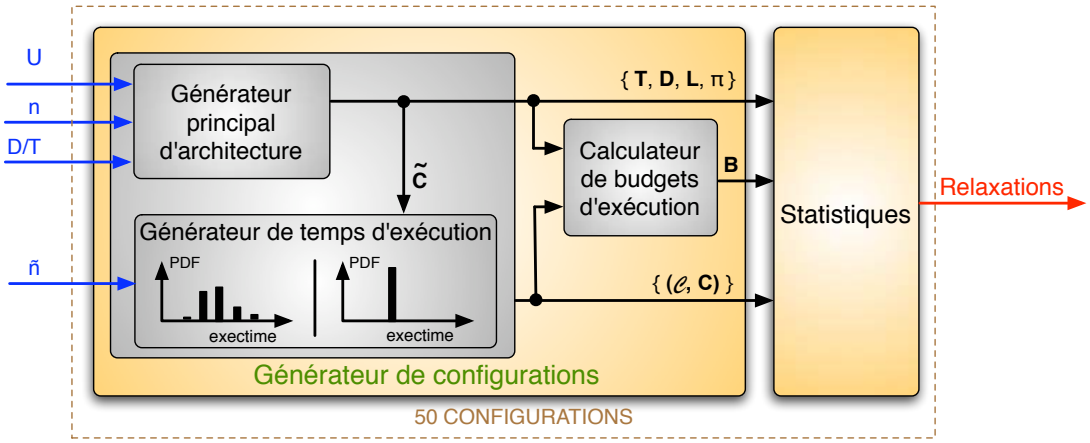


Figure 6.16 – Générateur d'applications et budgets d'exécution

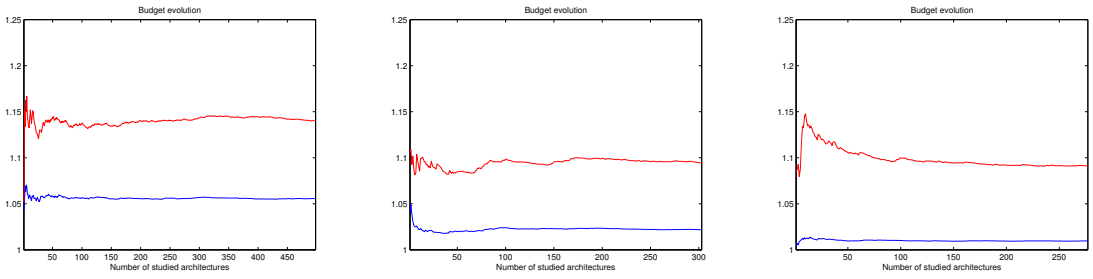
6.4.3 Résultats de simulations

Nous étudions dans la suite plusieurs ratios entre tâches critiques et non-critiques. Nous fixons le nombre de tâches à $n = 10$ et le nombre de tâches non-critiques à $\tilde{n} = \{2, 5, 8\}$ afin d'illustrer les différents comportements de la méthode d'optimisation en présence d'une majorité ou d'une minorité de tâches critiques.

Le but de l'étude est de relaxer les budgets d'exécution afin d'améliorer la qualité de service du mécanisme de détection d'erreurs lorsqu'il est utilisé sur un système mal déterminé, tout en garantissant un comportement temporel acceptable. La mesure effectuée sur chaque tâche non-critique τ_i est définie par le rapport entre le nouveau budget estimé et la valeur du temps d'exécution $\mathfrak{C}_i(cl = 0)$ (valeur du budget implicite). La zone de recherche de B_i est réalisée dans l'intervalle $[\![\mathfrak{C}_i(l = 0), \mathfrak{C}_i(l = 1)]\!] = [1, 1.25].\mathfrak{C}_i(l = 0)$, la mesure est toujours comprise entre 1 et 1.25. Le tableau 6.5 présente le taux moyen de relaxation obtenue sur l'ensemble des tâches non-critiques. Toutes les configurations représentent des systèmes qui sans aucune protection temporelle ne respectent pas les contraintes d'ordonnançabilité demandées par l'utilisateur ($S_{\tau_i} \geq 99\%$ pour les tâches non-critiques). Les résultats sont présentés pour chaque valeur fixée pour $\tilde{n}$ et pour les différentes méthodes d'analyses étudiées (analyse de sensibilité ou probabiliste).

Cas	Mesure	Analyse de sensibilité	Analyse probabiliste
$\tilde{n} = 2$	Relaxation	1.055	1.145
	Temps de calcul	-	3s
$\tilde{n} = 5$	Relaxation	1.022	1.097
	Temps de calcul	-	9m04s
$\tilde{n} = 8$	Relaxation	1.010	1.091
	Temps de calcul	-	12m01s

Tableau 6.5 – Valeurs moyennes des relaxations obtenues et des temps de calculs

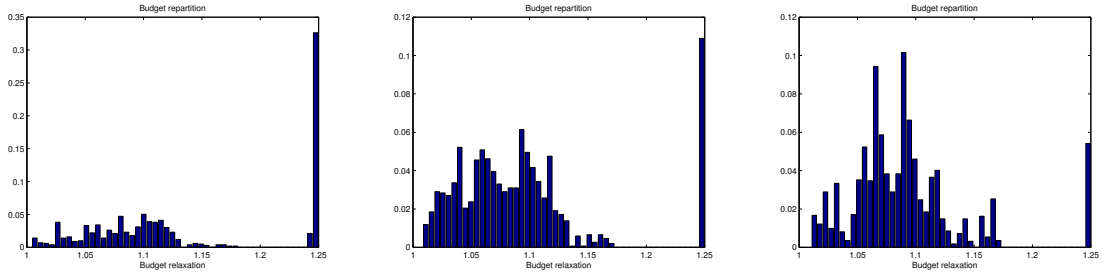


(a) $\tilde{n} = 2$ (b) $\tilde{n} = 5$ (c) $\tilde{n} = 8$

Figure 6.17 – Évolution de la moyenne en fonction du nombre d'applications étudiées (courbe basse : analyse de sensibilité, courbe haute : analyse probabiliste)

Le nombre d'architectures à analyser afin d'obtenir une moyenne représentative a été

défini en traçant l'évolution de la mesure moyenne en fonction du nombre d'architectures. La figure 6.17 présente cette évolution. Ce nombre dépend du nombre de tâches non-critiques considérées. La simulation des systèmes aux tâches non-critiques prépondérantes aboutissent à une moyenne plus rapidement stable (200-250 systèmes à analyser dans le cas $\tilde{n} = 8$ et 500 systèmes à analyser dans le cas $\tilde{n} = 2$). La méthode probabiliste donne lieu à des résultats intéressants puisqu'une relaxation relativement importante (de l'ordre de 10%) en ressort d'où une amélioration de la qualité de service. Nous pouvons remarquer que plus le nombre de tâches non-critiques est faible ($\tilde{n}$ petit), plus le gain l'est également. Ceci est dû à la même remarque que celle énoncée dans le cas déterministe : la marge globale du système est répartie sur un plus petit nombre de tâches donc plus importante. La figure 6.18 permet de visualiser la répartition du taux de relaxation sur l'ensemble des tâches non-critiques étudiées dans le cas d'une analyse probabiliste.



(a) $\tilde{n} = 2$ (b) $\tilde{n} = 5$ (c) $\tilde{n} = 8$

Figure 6.18 – Répartition des valeurs de gains obtenus

Nous pouvons remarquer qu'une configuration où les tâches non-critiques sont minoritaires dégage beaucoup plus souvent de gros budgets qu'une configuration inverse (remarque précédente). La distribution des valeurs de relaxation (presque du tout ou rien) dans le cas $\tilde{n} = 8$ peut également expliquer la difficulté à converger vers une moyenne stable. Dans le cas où les tâches non-critiques sont prépondérantes, la répartition ressemble à une loi normale centrée sur la valeur moyenne (+9%) avec un pic également présent sur la valeur quasi maximale de +25% fixée par la zone de recherche ($[\![\mathfrak{C}_i(l=0), 1.25.\mathfrak{C}_i(l=0)]\!]$). Dans ce cas, la valeur moyenne donnée tableau 6.5 représente une valeur représentative et donne ainsi une bonne mesure de la qualité de cette méthode.

Nous avons voulu alors présenter les résultats obtenus par niveau de priorité. Nous avons ici simulé des applications comportant chacune 10 tâches affectées de priorités π_i différentes. Nous rappelons que plus la valeur de π_i est importante plus la tâche est prioritaire. De plus, pour les résultats nous avons pris en compte uniquement les tâches non-critiques. La présentation de la répartition des tâches non-critiques selon le niveau de priorité permet de connaître le pourcentage de tâches prises en compte. Plus le pourcentage est bas, plus la moyenne des valeurs étudiées est potentiellement biaisée (moins de points en présence). Les résultats sont présentés figures 6.19, 6.20, 6.21 pour les différentes valeurs de $\tilde{n}$.

Dans un premier temps, nous pouvons remarquer figure 6.19 que la répartition des tâches

non-critiques n'est pas uniforme sur l'ensemble des niveaux de priorité. Ces tâches sont plutôt réparties sur les niveaux de priorité les plus bas. Ce résultat démontre que l'algorithme d'affectation des priorités joue bien son rôle en affectant des priorités élevées aux tâches critiques afin de dégager un maximum de marge. La partie relaxation des budgets représente les valeurs moyennes (trait continu) selon la méthode utilisée (noir/croix : étude déterministe, bleu/carré : étude probabiliste). Les valeurs maximales et minimales sont également représentées par des points haut et bas. La partie droite (priorités élevées) de cette courbe est biaisée puisque le nombre de tâches étudiées est assez faible (aucune tâche non-critique de niveau de priorité 10 n'est générée). Nous ne pouvons pas réellement conclure sur la réalité d'une décroissance de la courbe c'est-à-dire une plus grande relaxation pour les tâches de priorités plus faibles. Cependant, il paraîtrait normal que ce soit le cas puisque la présence d'une tâche non-critique à une priorité élevée suppose qu'une tâche critique se situe à un niveau de priorité moins élevé, donc une relaxation peut en être affectée (voir section 6.2.2).

Les figures 6.20 et 6.21 confirment l'action du mécanisme de choix des priorités : la part des tâches non-critiques aux priorités plus élevées est plus faible même si la différence tend à s'amenuiser (nombre de tâches non-critiques plus élevé). Plus la part des tâches non-critiques est importante, moins les résultats sur la mesure de la relaxation de budget sont donc biaisés. Plus $\tilde{n}$ est grand, plus la relaxation des budgets est uniforme sur l'ensemble des priorités. Dans le cas où $\tilde{n} = 8$, ce taux est quasiment toujours égal à la valeur moyenne donnée précédemment.

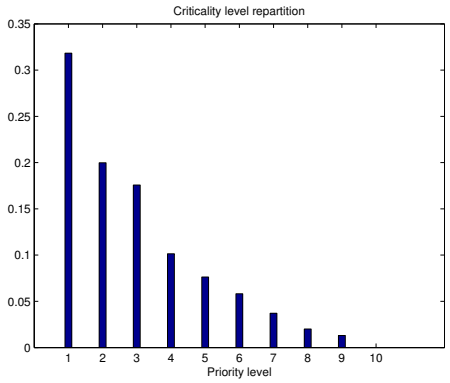
6.5 Discussion

Comme nous pouvons le remarquer par les simulations réalisées, le dimensionnement du mécanisme de protection d'AUTOSAR OS par une analyse probabiliste permet d'étirer le domaine d'acceptabilité et donc de relaxer le budget d'exécution (jusqu'à +14% par analyse probabiliste au lieu de +5% par une étude de sensibilité). Cette relaxation est réalisée uniquement pour les tâches non-critiques puisque l'estimation du pire temps d'exécution des tâches critiques est supposée exacte (voire largement sur-estimée).

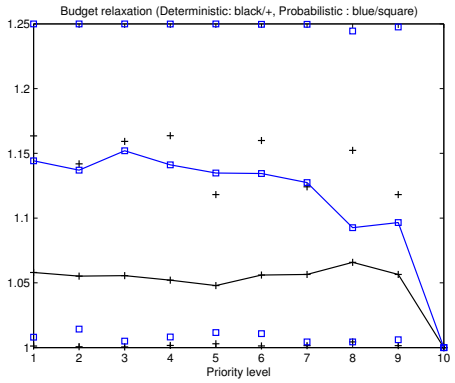
Bien entendu, la relaxation des budgets obtenue par analyse probabiliste dépend des valeurs d'exécution et plus précisément des distributions des variables aléatoires des tâches non-critiques (celles-ci étant estimées par mesure du temps d'exécution). Les profils d'exécution générés et étudiés essaient de représenter au mieux le comportement d'un programme réel.

La relaxation des budgets dépend également de la priorité des tâches critiques. En effet, une tâche critique doit être pleinement ordonnançable quel que soit le temps d'exécution des instances de tâches plus prioritaires qui peuvent la préempter. Si dans le système, une tâche critique est la tâche la moins prioritaire, l'analyse probabiliste revient à une analyse déterministe (de sensibilité dans le meilleur des cas). Pour parer à cette éventualité, nous avons pris en compte ce paramètre dans le choix des priorités et plus précisément dans le choix arbitraire laissé dans certains cas par l'algorithme d'affectation des priorités proposé par Vestal *et al.*. Le but est alors d'affecter les plus hautes priorités aux tâches critiques.

Cependant, la tâche la moins prioritaire est systématiquement la tâche qui est la plus exposée à des défaillances. Ceci vient de la politique d'ordonnancement à priorités fixes retenue. Dans le cas de système ordonnancé par une politique à priorités dynamiques, la priorité des

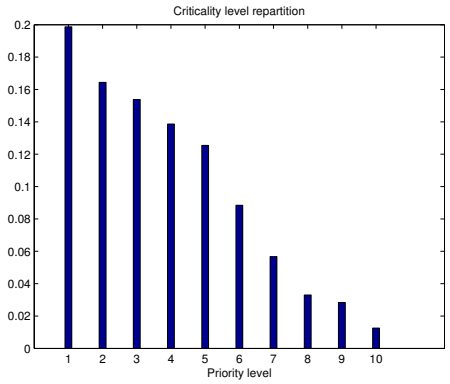


(a) Répartition des tâches non-critiques par niveau de priorité

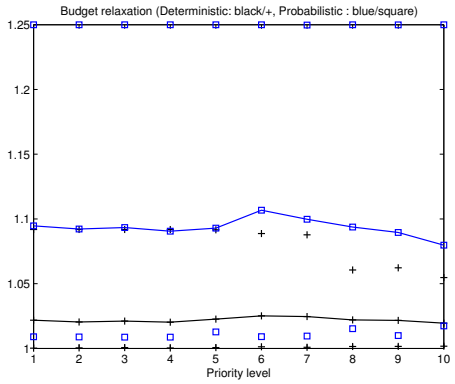


(b) Relaxation des budgets par niveau de priorité

Figure 6.19 – Cas $\tilde{n} = 2$ (tâches critiques prépondérantes)

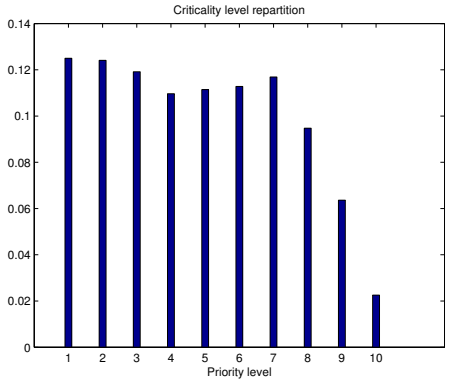


(a) Répartition des tâches non-critiques par niveau de priorité

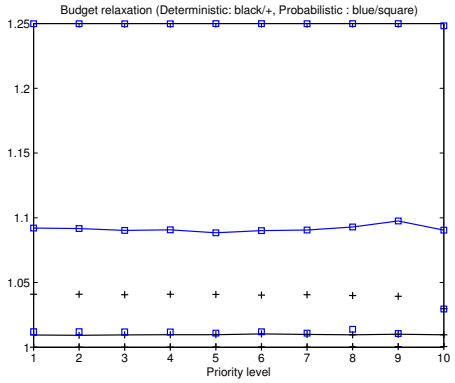


(b) Relaxation des budgets par niveau de priorité

Figure 6.20 – Cas $\tilde{n} = 5$ (équirépartition des tâches critiques et non-critiques)



(a) Répartition des tâches non-critiques par niveau de priorité



(b) Relaxation des budgets par niveau de priorité

Figure 6.21 – Cas $\tilde{n} = 8$ (tâches non-critiques prépondérantes)

tâches variant au cours de l'exécution, la tâche impactée par les défaillances n'est pas toujours la même.

Dans le cas d'une analyse probabiliste, nous laissons certaines instances de tâches non-critiques s'exécuter au delà de leur échéance. Ce taux (probabilité d'ordonnançabilité d'une instance) est contrôlé et assure un comportement acceptable pour les autres instances. Cependant, l'exécution au delà d'une échéance est une défaillance. Cette exécution ne devrait pas avoir lieu et donc être stoppée. Cette remarque nous amène à nous poser la question de l'intérêt de la mise en place d'un mécanisme de protection supplémentaire pour contrôler ces défaillances.

« Toute puissance est faible, à moins que d'être unie »

Le Vieillard et ses Enfants - Jean de La Fontaine

Chapitre 7

Utilisation additionnelle de la surveillance d'échéances

Sommaire

7.1 Principe de la surveillance d'échéances 99
7.2 Apport de la surveillance d'échéances pour la sûreté de fonctionnement 100
7.3 Apport de la surveillance d'échéances pour l'analyse 101
7.3.1 Apport pour le calcul du backlog 101
7.3.2 Difficultés 101
7.3.3 Solution retenue 104
7.4 Simulations 105
7.4.1 Paramètre de génération des applications 105
7.4.2 Résultats de simulations 105
7.5 Discussion 109
7.5.1 Analyse probabiliste et surveillance d'échéance 109
7.5.2 Dimensionnement des budgets d'exécution 110

7.1 Principe de la surveillance d'échéances

Un mécanisme de surveillance d'échéances a pour but de détecter si une instance de tâche est toujours active à la date de son échéance absolue. De même que pour le mécanisme de protection basé sur un budget d'exécution présenté jusqu'ici dans ce mémoire, la surveillance d'échéances permet de détecter un mauvais comportement du système. Cependant dans ce cas, le mécanisme permet de détecter une défaillance et non une erreur. Un traitement de la défaillance doit ensuite être réalisé. Ce traitement est généralement basé dans un premier temps par l'arrêt de l'instance fautive, puis, si nécessaire, par l'exécution d'une procédure de recouvrement.

Exemple . Reprenons l'exemple présenté dans la partie 6.1.2 dont la séquence d'exécution sans mécanisme de protection est redonnée figure 7.1.

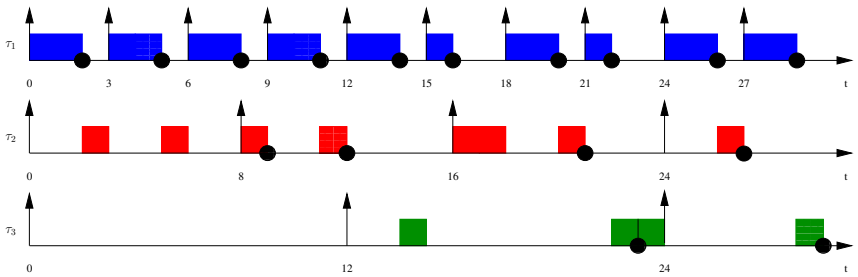


Figure 7.1 – Exemple d'exécution sans recours à la surveillance d'échéances

La figure 7.2 illustre l'action du la surveillance d'échéances sur la même séquence d'exécution des tâches.

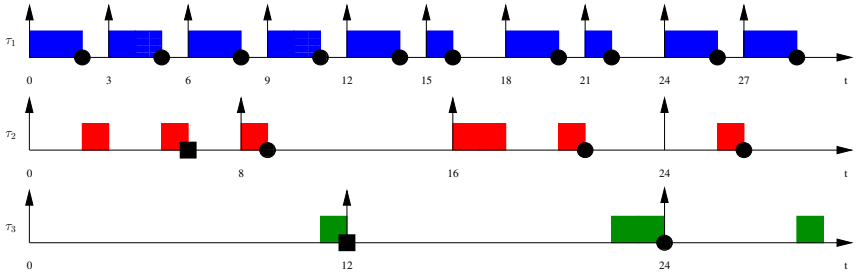


Figure 7.2 – Même exemple en utilisant la surveillance d'échéances

Nous pouvons remarquer que le temps de réponse de l'instance $\tau_{3,2}$ est bien plus court ($r_{3,2} = 24$ au lieu de 30) du fait de l'action de la surveillance d'échéances sur les instances $\tau_{2,1}$ et $\tau_{3,1}$. Aucun résidu d'exécution, dans ce cas, n'est à déplorer à la fin de l'hyperpériode ($\overline{T} = 24$), contrairement au cas précédent où l'instance $\tau_{3,3}$ ne s'étant pas terminée avant la fin de l'hyperpériode laissait un résidu d'exécution.

◊

7.2 Apport de la surveillance d'échéances pour la sûreté de fonctionnement

La méthode proposée au chapitre précédent se base sur une possibilité de défaillance (exécution d'une instance au-delà de son échéance relative) pour les tâches non-critiques. Même si l'analyse garantit une ordonnançabilité complète des tâches critiques (utilisation du pire-cas), il n'est pas prudent de laisser une défaillance se produire sans en contrôler immédiatement les conséquences.

Le mécanisme de surveillance d'échéances permet de contrôler ces défaillances en les détectant et en agissant selon les spécifications demandées par l'utilisateur (dans notre cas, une instance de tâche défaillante est stoppée).

Le mécanisme permet d'introduire un meilleur contrôle du comportement du système lors de son fonctionnement.

7.3 Apport de la surveillance d'échéances pour l'analyse

L'analyse probabiliste n'a de sens que dans le cas de systèmes à tâches concrètes (synchrones ou non). Nous restreignons donc notre étude à l'impact de la surveillance d'échéances sur ce type de systèmes.

7.3.1 Apport pour le calcul du backlog

Si nous considérons qu'une instance fautive est stoppée, l'utilisation de la surveillance d'échéances implique qu'aucun résidu d'exécution ne peut être présent au delà de l'échéance d'une instance de tâche. Dans le cas d'un système à échéance contrainte ($D_i < T_i$) ou à échéance sur requête ($D_i = T_i$), cette propriété implique qu'aucun résidu d'instance antérieure d'une même tâche n'est présent lors de l'activation d'une nouvelle instance. Ce mécanisme garantit donc un résidu d'exécution stationnaire dès la première hyperpériode $\overline{T}$ (résidus nuls dans le cas d'un système synchrone). Nous pouvons en déduire que la période d'étude d'un tel système peut se limiter à deux hyperpériodes $\overline{T}$. Nous rappelons que, dans le cas synchrone, l'hyperpériode est égale à $\overline{T} = \text{ppcm}_{\{i=1,\dots,n\}}\{T_i\}$, et, dans le cas de tâches concrètes, l'hyperpériode est telle que $\overline{T} = \max_{\{i=1,\dots,n\}}\{\phi_i\} + 2 * \text{ppcm}_{\{i=1,\dots,n\}}\{T_i\}$.

Exemple . Ceci peut être remarqué sur l'exemple présenté. En effet, figure 7.2, aucun résidu ne persiste lors de la prochaine hyperpériode contrairement au cas sans recours à la surveillance d'échéances présenté figure 7.1.

◊

Le calcul du steady-state backlog est le principal inconvénient de la méthode proposée par Diaz *et al.* (2002). En effet, comme nous l'avons précisé au chapitre précédent le calcul itératif est assez lent et se base sur un point d'arrêt ne garantissant pas une estimation du steady-state sûre. Le fait de ne plus avoir à considérer son calcul est donc un point important pour la sûreté de fonctionnement.

7.3.2 Difficultés

La prise en compte de la surveillance d'échéances dans l'étude d'ordonnançabilité est un peu plus compliquée que pour le mécanisme de budget. En effet, le mécanisme basé sur le budget d'exécution impose un temps d'exécution maximal pour chaque instance de tâche. Ce temps maximal peut être alors directement utilisé comme pire temps d'exécution pour une étude déterministe d'ordonnançabilité. Dans le cas d'utilisation de la surveillance d'échéances, le mécanisme de protection est appliqué uniquement sur les instances fautives (exécution au-delà de l'échéance). La seule façon de prendre en compte son effet est de dérouler le scénario.

En effet, une instance fautive de tâche arrêtée par le mécanisme peut diminuer le temps de réponse d'une tâche moins prioritaire et donc augmenter sa probabilité d'ordonnançabilité. Ce résultat peut être prise en considération uniquement en examinant le scénario.

Le calcul du temps de réponse de l'instance proposée par Diaz *et al.* (2002) reste valide uniquement dans le cas où aucune instance de tâche interférant avec l'exécution de l'instance analysée n'est stoppée par le mécanisme de protection. Dans le cas contraire, la probabilité d'ordonnançabilité calculée précédemment n'est qu'une borne inférieure à la valeur exacte en présence de la surveillance d'échéances. En effet, dès que le mécanisme de protection agit, il réduit l'interférence sur l'instance analysée, donc réduit son temps de réponse et donc agit sur sa probabilité d'ordonnançabilité.

L'impact de la surveillance d'échéances peut être mesuré si l'on parvient à déterminer son action sur chaque instance. Plus précisément, pour connaître l'ordonnançabilité d'une tâche, il faut connaître l'action du mécanisme sur toutes les instances de tâches plus prioritaires et les instances antérieures de la tâche analysée. Cette action correspond à une troncature de la distribution du temps d'exécution.

Une méthode permettant de déterminer la probabilité d'ordonnançabilité d'une instance est d'analyser les différentes séquences d'exécution possibles pour les instances de tâches plus prioritaires. Pour chaque séquence d'exécution, il est facile de déterminer comment se comporte l'instance étudiée (une instance de tâche plus prioritaire peut être arrêtée par le mécanisme) et de déterminer sa probabilité d'ordonnançabilité (fonction de sa FdP de temps d'exécution). Cependant, la dimension des fonctions de probabilité d'exécution en jeu et le nombre de séquences à examiner peuvent être très importants, et le temps d'analyse du système peut devenir relativement long.

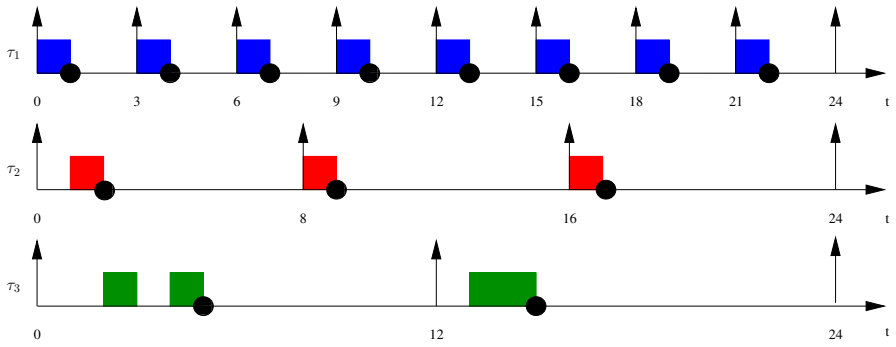
Exemple . Considérons l'exemple du système $\mathcal{S}$ composé de 3 tâches τ_1, τ_2 et τ_3. Chaque tâche est ici représentée uniquement par son temps d'exécution, sa période et son échéance relative ($\tau = (\mathcal{C}, T, D)$) :

$$\tau_1 = \left(\begin{pmatrix} 1 & 2 \\ 0.70 & 0.30 \end{pmatrix}, 3, 3 \right), \ \tau_2 = \left(\begin{pmatrix} 1 & 3 \\ 0.50 & 0.50 \end{pmatrix}, 4, 4 \right), \ \tau_3 = \left(\begin{pmatrix} 1 & 2 \\ 0.50 & 0.50 \end{pmatrix}, 12, 12 \right)$$

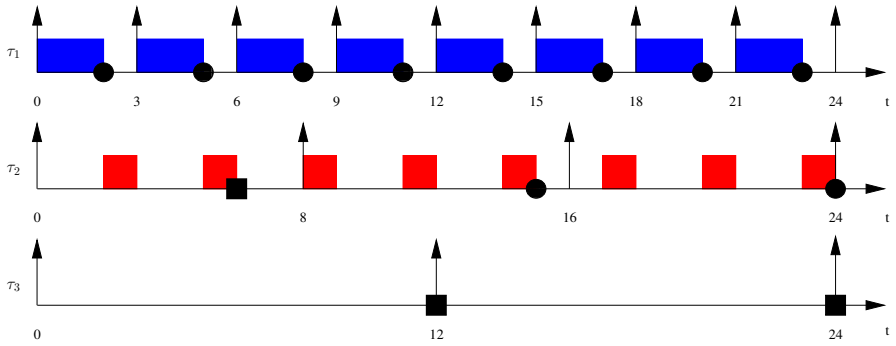
La figure 7.3 représente quelques temps de réponse des instances de τ_3 en fonction de différentes séquences d'exécution pour τ_1 et τ_2.

La séquence d'exécution présentée figure 7.3(a) représente la séquence comportant le moins de charge processeur (temps d'exécution minimaux). Dans ce cas, les deux instances de la tâche τ_3 respectent leur échéance quelle que soit la valeur du temps d'exécution considérée pour celle-ci. Aucune action de la surveillance d'échéances n'est réalisée. La probabilité d'ordonnançabilité des instances de τ_3 dans cette configuration est donc de $S_{\tau_{3,1}} = S_{\tau_{3,2}} = 1$. La probabilité d'avoir la séquence d'exécution considérée pour les tâches τ_1 et τ_2 est de $\big(\mathbb{P}(\mathcal{C}_1 = 1)\big)^8 \ . \ \big(\mathbb{P}(\mathcal{C}_2 = 1)\big)^3$.

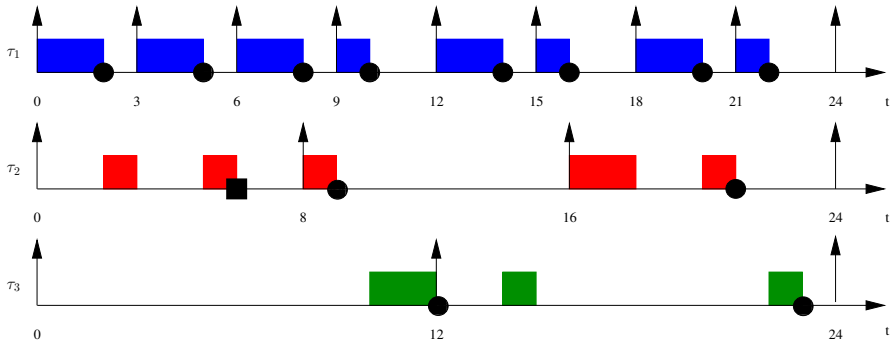
La séquence d'exécution présentée figure 7.3(b) représente la séquence comportant le plus de charge processeur (temps d'exécution maximums). Dans ce cas, aucune instance de la



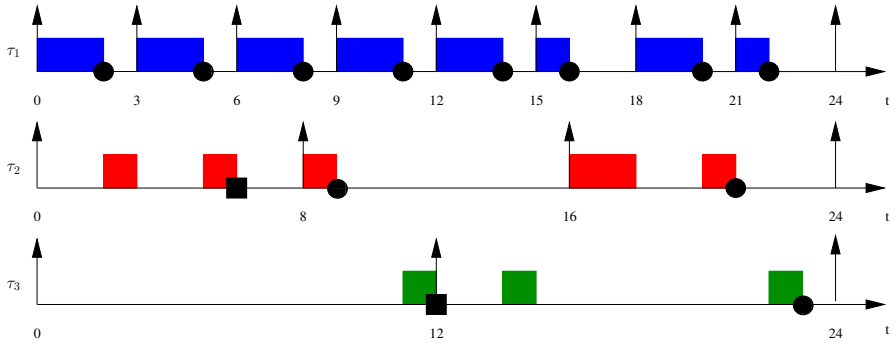
(a) τ_1 séquence $\{1, 1, 1, 1, 1, 1, 1, 1\}$, τ_2 séquence $\{1, 1, 1\}$



(b) τ_1 séquence $\{2, 2, 2, 2, 2, 2, 2, 2\}$, τ_2 séquence $\{3, 3, 3\}$



(c) τ_1 séquence $\{2, 2, 2, 1, 2, 1, 2, 1\}$, τ_2 séquence $\{3, 3, 3\}$



(d) τ_1 séquence $\{2, 2, 2, 2, 2, 1, 2, 1\}$, τ_2 séquence $\{3, 3, 3\}$

Figure 7.3 – Temps de réponse des instances de τ_3 pour quelques séquences d'exécution

tâche τ_3 respecte son échéance quelle que soit la valeur du temps d'exécution considérée pour celle-ci. La surveillance d'échéances agit sur les deux instances de τ_3 ainsi que l'instance $\tau_{2,1}$. La probabilité d'ordonnançabilité des instances de τ_3 dans cette configuration est donc de $S_{\tau_{3,1}} = S_{\tau_{3,2}} = 0$. La probabilité d'avoir la séquence d'exécution considérée pour les tâches τ_1 et τ_2 est de $\big(\mathbb{P}(\mathcal{C}_1 = 2)\big)^8 \,.\, \big(\mathbb{P}(\mathcal{C}_2 = 1)\big)^3$.

Les séquences d'exécution présentées figure 7.3(c) et 7.3(d) représentent des séquences comportant des temps d'exécution maximaux pour les instances de τ_2 mais des temps d'exécution comportant les deux valeurs possibles pour les instances de τ_1.

Dans les deux cas, la surveillance d'échéances agit uniquement sur la première instance de τ_2. Dans le premier cas, les deux instances de τ_3 ne sont pas affectées par la surveillance d'échéances puisque la séquence comportant les temps maximaux pour les instances de τ_3 est ordonnançable. Néanmoins sans l'action de la surveillance d'échéances sur l'instance $\tau_{2,1}$, les instances de τ_3 ne seraient pas ordonnançables. La probabilité d'ordonnançabilité des instances de τ_3 dans cette configuration est donc de $S_{\tau_{3,1}} = S_{\tau_{3,2}} = 1$. La probabilité d'avoir la séquence d'exécution considérée pour les tâches τ_1 et τ_2 est de $\big(\mathbb{P}(\mathcal{C}_1 = 2)\big)^5 \,.\, \big(\mathbb{P}(\mathcal{C}_1 = 1)\big)^3 \,.\, \big(\mathbb{P}(\mathcal{C}_2 = 1)\big)^3$.

Dans le deuxième cas, seule la deuxième instance de τ_3 n'est pas affectée par la surveillance d'échéances, le temps maximal d'exécution pour l'instance $\tau_{3,1}$ est de 1 unité de temps. La probabilité d'ordonnançabilité des instances de τ_3 dans cette configuration est donc de $S_{\tau_{3,1}} = 1/2$ ($\mathbb{P}(\mathcal{C}_3 = 1) = 1/2$) et $S_{\tau_{3,2}} = 1$. La probabilité d'avoir la séquence d'exécution considérée pour les tâches τ_1 et τ_2 est de $\big(\mathbb{P}(\mathcal{C}_1 = 2)\big)^6 \,.\, \big(\mathbb{P}(\mathcal{C}_1 = 1)\big)^2 \,.\, \big(\mathbb{P}(\mathcal{C}_2 = 1)\big)^3$.

Le nombre de séquences à analyser sur cet exemple simple est fonction de la dimension de la FdP d'exécution pour les instances de τ_1 et τ_2, K_1 et K_2 et du nombre d'instances dans la période d'étude n_1 et n_2. Dans notre cas, ce nombre est de $K_1^{n_1} * K_2^{n_2} = 2048$ séquences. Si le nombre d'instances plus prioritaires augmentent, ce nombre de séquences à analyser augmentent considérablement.

◊

7.3.3 Solution retenue

Dans le cas de la méthode proposée par Diaz *et al.* (2002, 2004), l'utilisation de la surveillance d'échéances empêchant l'accumulation de résidu à la fin d'une hyperpériode, le backlog $\mathcal{B}_\pi^{stat}$ est nul. L'état stationnaire n'est donc pas à calculer ce qui permet de diminuer les temps de calculs.

De plus, le calcul de l'état stationnaire étant réalisé par itérations successives au bout de 1, ..., m hyperpériodes jusqu'à parvenir au critère de convergence ne fournit pas une borne sûre (même si l'état obtenu est proche de l'état stationnaire, il se peut qu'il soit sous-estimé et entraîne une légère sur-estimation de la probabilité d'ordonnançabilité). La possibilité de supprimer ce calcul peut être un atout important.

Pour mesurer le gain obtenu, nous avons testé plusieurs systèmes. Le tableau 7.1 présente le temps moyen de calcul nécessaire pour déterminer l'ensemble des probabilités d'ordon-

nançabilité (ou borne) de chaque tâche d'une application selon que nous utilisons ou non la surveillance d'échéances. Pour ces tests, une application est composée d'un ensemble de $n = 10$ tâches synchrones périodiques et indépendantes. Chaque tâche est représentée par les paramètres déterministes T_i, D_i et la variable aléatoire $\mathcal{C}_i$. Nous supposons le cas de tâches à échéance sur requêtes ($D_i = T_i$) et les périodes tirées dans un ensemble de valeurs minimisant l'hyperpériode. Nous considérons deux distributions différentes pour les temps d'exécution : une distribution "déterministe" composée d'une unique valeur d'exécution et un distribution discrète composée de 5 valeurs d'exécution. Nous notons $\tilde{n}$ le nombre de tâches utilisant une distribution discrète.

Case	Méthode de Diaz *et al.* (2002)	Méthode de Diaz *et al.* (2002) + DM
$\tilde{n} = 2$	28ms	6ms
$\tilde{n} = 5$	14s	3s

Tableau 7.1 – Temps moyen de calcul d'une architecture selon la méthode

Le gain en temps de calcul est important de même que le gain en sûreté de fonctionnement.

Néanmoins, comme nous l'avons montré précédemment les valeurs obtenues pour les probabilités d'ordonnançabilité si l'on ne modifie pas la technique ne sont que des bornes minimales. Une étude visant à évaluer par simulation le pessimisme de cette borne serait une perspective intéressante.

7.4 Simulations

7.4.1 Paramètre de génération des applications

Les architectures générées et utilisées pour le calcul des budgets d'exécution sont identiques à celles exploitées au chapitre précédent section 6.4.

Nous rappelons que, dans cette étude, nous ne faisons plus appel au simulateur d'application temps réel TrueTime. Notre mesure principale reste le taux de relaxation obtenu sur chaque tâche non-critique de l'application (les tâches critiques n'étant pas relaxées) et les temps moyens de calcul sur l'ensemble des applications étudiées.

7.4.2 Résultats de simulations

Nous fixons, comme précédemment, le nombre de tâches à $n = 10$ et le nombre de tâches non-critiques à $\tilde{n} = \{2, 5, 8\}$ afin d'illustrer les différents comportements de la méthode d'optimisation en présence d'une majorité ou d'une minorité de tâches critiques.

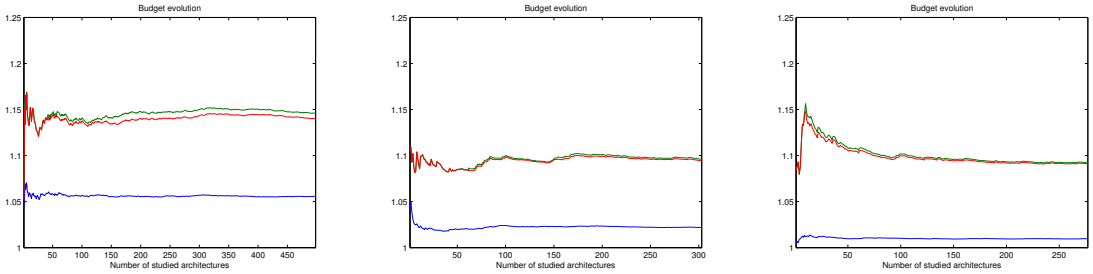
Le tableau 7.2 présente le taux moyen de relaxation obtenue sur l'ensemble des tâches non-critiques. Toutes les configurations représentent des systèmes qui sans aucune protection temporelle ne respectent pas les contraintes d'ordonnançabilité demandées par l'utilisateur ($S_{\tau_i} \geq 99\%$ pour les tâches non-critiques). Les résultats sont présentés pour chaque valeur fixée

pour $\tilde{n}$ (nombre de tâches non-critiques) et pour les différentes méthodes d'analyses étudiées (analyse de sensibilité, probabiliste avec ou sans utilisation du mécanisme de surveillance des échéances).

Cas	Mesure	Analyse de sensibilité	Analyse probabiliste (seule)	Analyse probabiliste (utilisation de DM)
$\tilde{n} = 2$	Relaxation Temps de calcul	1.055 -	1.145 3s	1.147 0.25s
$\tilde{n} = 5$	Relaxation Temps de calcul	1.022 -	1.097 9m04s	1.099 55s
$\tilde{n} = 8$	Relaxation Temps de calcul	1.010 -	1.091 12m01s	1.093 2m14s

Tableau 7.2 – Valeurs moyennes des relaxations obtenues et des temps de calculs

Nous pouvons remarquer que les résultats obtenus à l'aide des deux analyses probabilistes (utilisation ou non du mécanisme de surveillance des échéances) sont assez proches. L'analyse utilisant un deuxième mécanisme de protection permet de dégager légèrement plus de marge sur les budgets puisqu'aucun résidu d'exécution ne peut être présent à la fin d'une hyperpériode ce qui entraîne des temps de réponse moins importants et ainsi une relaxation plus conséquente. Cette méthode de calcul permet, en outre, de réduire très largement les temps de calculs (jusqu'à 6 fois) ce qui peut être un avantage important pour le concepteur de l'application qui doit fournir le dimensionnement du mécanisme de protection.



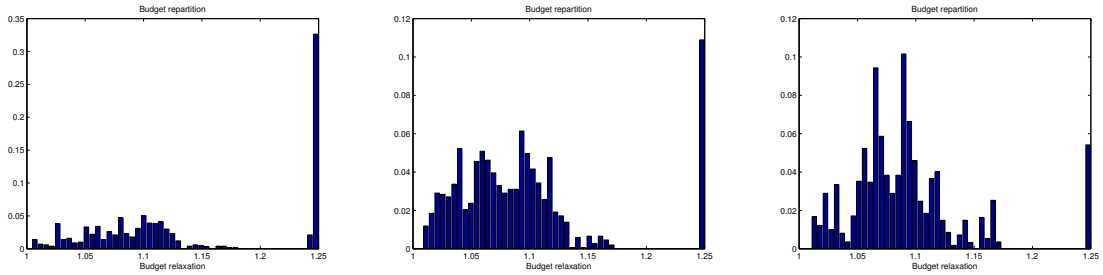
(a) $\tilde{n} = 2$ (b) $\tilde{n} = 5$ (c) $\tilde{n} = 8$

Figure 7.4 – Évolution de la moyenne des budgets en fonction du nombre d'applications étudiées (courbe basse : analyse de sensibilité, courbe haute : analyse probabiliste)

La figure 7.4 présente l'évolution de la valeur moyenne de la relaxation en fonction du nombre d'architectures analysées. Le but est de déterminer le nombre d'applications à étudier afin d'obtenir une moyenne représentative. Le nombre d'architectures nécessaire afin d'obtenir une valeur moyenne viable dépend du nombre de tâches non-critiques considérées. Nous observons les mêmes résultats qu'au chapitre précédent : le nombre d'architectures à analyser croît avec le nombre de tâches critiques (les valeurs présentées ci-dessus sont issues des moyennes

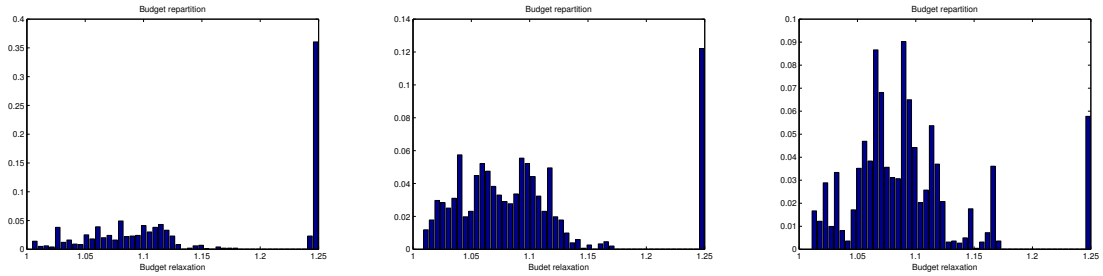
sur l'ensemble des applications).

La figure 6.18 permet de visualiser la répartition du taux de relaxation sur l'ensemble des tâches non-critiques étudiées dans le cas d'une étude probabiliste avec la surveillance des échéances. Nous rappelons figure 7.5, le cas sans la surveillance des échéances afin de pouvoir les comparer plus facilement.



(a) $\tilde{n} = 2$ (b) $\tilde{n} = 5$ (c) $\tilde{n} = 8$

Figure 7.5 – Répartition des valeurs de gains obtenus sans la surveillance des échéances

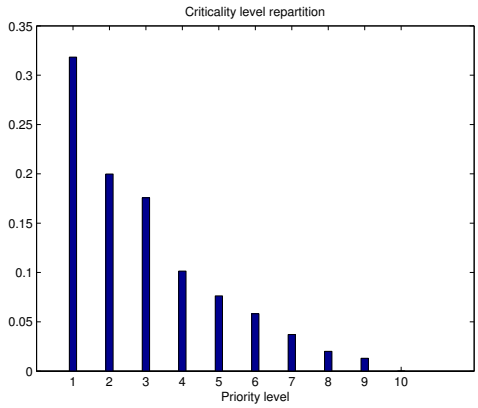


(a) $\tilde{n} = 2$ (b) $\tilde{n} = 5$ (c) $\tilde{n} = 8$

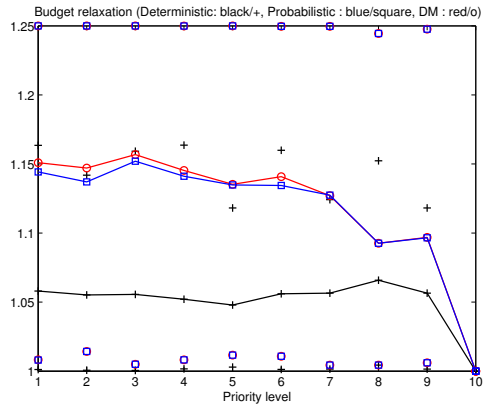
Figure 7.6 – Répartition des valeurs de gains obtenus avec la surveillance des échéances

Nous pouvons observer que les répartitions avec et sans utilisation du mécanisme de surveillance des échéances sont très proches. De même que pour l'analyse précédente, une configuration où les tâches non-critiques sont minoritaires dégage beaucoup plus souvent de gros budgets (budget maximum de l'ordre +25% fixé par la zone de recherche) qu'une configuration majoritaire où la distribution des budgets est plus ramassée sur une valeur moyenne (de l'ordre +10%).

Les résultats par niveau de priorité sont présentés figures 7.7, 7.8 et 7.9 pour les différentes valeurs de $\tilde{n}$. De même que pour les valeurs moyennes, les résultats obtenus à l'aide des deux analyses probabilistes (utilisation ou non du mécanisme de surveillance des échéances) sont très proches. L'analyse utilisant un deuxième mécanisme de protection permet de dégager légèrement plus de marge sur les budgets, cette différence se réduisant lorsque le nombre de tâches non-critiques augmente. Le gain est également plus perceptible pour les tâches les moins prioritaires pour lesquelles il est plus facile de dégager du budget (voir remarque sur ce sujet au chapitre précédent partie simulation).

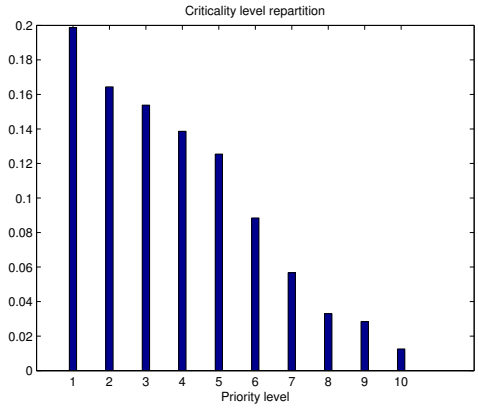


(a) Répartition des tâches non-critiques

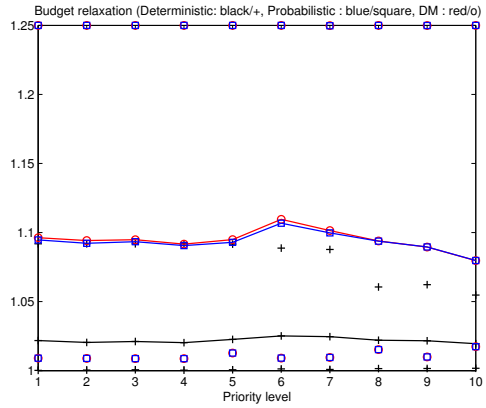


(b) Relaxation des budgets

Figure 7.7 – Cas $\tilde{n} = 2$ (tâches critiques prépondérantes)

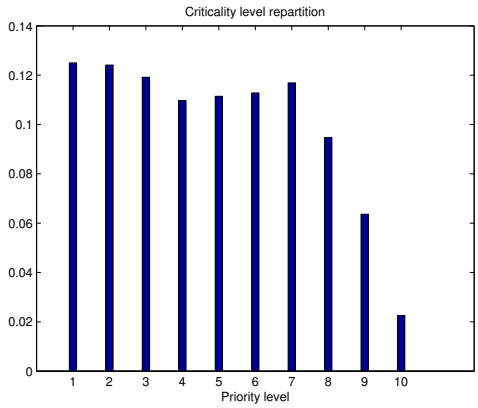


(a) Répartition des tâches non-critiques

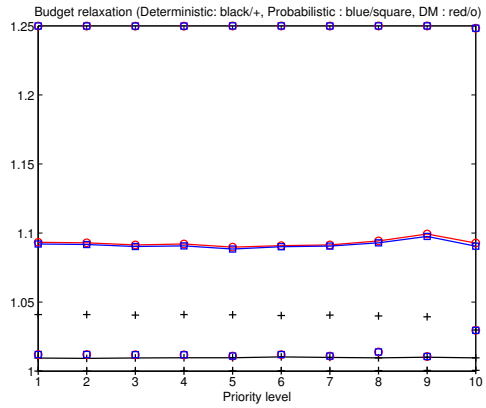


(b) Relaxation des budgets

Figure 7.8 – Cas $\tilde{n} = 5$ (équi-répartition des tâches critiques et non-critiques)



(a) Répartition des tâches non-critiques



(b) Relaxation des budgets

Figure 7.9 – Cas $\tilde{n} = 8$ (tâches non-critiques prépondérantes)

Nous pouvons en conclure que la méthode par analyse probabiliste adjointe au mécanisme de surveillance de l'échéance permet de dégager légèrement plus de budget. Cependant, il ne faut pas oublier que toute tâche fautive est stoppée et que le mécanisme de surveillance des échéances augmente le nombre d'instances tuées.

7.5 Discussion

7.5.1 Analyse probabiliste et surveillance d'échéance

À travers les chapitres 6 et 7, nous avons mis en évidence une méthode d'analyse probabiliste (Diaz *et al.*, 2002) de systèmes dont les temps d'exécution sont incertains (représentés par des variables aléatoires). Nous avons développé une méthode de dimensionnement des budgets d'exécution des tâches (et plus spécifiquement des tâches non-critiques) basée sur cette analyse.

Cette méthode permet dans une certaine mesure de laisser une tâche (non-critique) dépasser son échéance. Un mécanisme de surveillance de l'échéance (DM) peut alors être mis en place pour augmenter le déterminisme de l'ordonnancement. Nous avons essayé de déterminer l'impact d'un tel mécanisme sur l'analyse probabiliste. L'analyse exacte est difficile mais l'étude approchée permet d'obtenir une borne inférieure d'ordonnançabilité relativement précise et surtout plus rapides en temps de calcul.

Le tableau 7.3 résume l'ensemble des méthodes probabilistes étudiées ainsi que leur domaine de validité.

		Méthode Diaz seule		Méthode Diaz + DM
Un des offset $\neq 0$	Contrainte	$U^{max} \leq 1$	$\overline{U} \leq 1$	U quelconque
	Période d'étude	$2.\overline{T}$	$\overline{T}$	$\overline{T}$
	Steady-state	Non	Oui	Non
	Borne/v. exacte	Exact	Approchée	Borne inférieure
Tous les offsets $= 0$	Contrainte	$U^{max} \leq 1$	$\overline{U} \leq 1$	U quelconque
	Période d'étude	$\overline{T}$	$\overline{T}$	$\overline{T}$
	Steady-state	Non	Oui	Non
	Borne/v. exacte	Exact	Approchée	Borne inférieure

Tableau 7.3 – Méthodes d'analyses probabilistes selon les différentes contraintes

Dans le cadre d'une étude probabiliste où les temps d'exécution sont considérés comme des variables aléatoires, la méthode proposée par Diaz *et al.* (2002) permet de déterminer les temps de réponses de chaque instance et ainsi d'apporter une réponse sur l'ordonnançabilité de celle-ci. Cette méthode est soumise à quelques contraintes. Tout d'abord, le système doit avoir une utilisation processeur moyenne $\overline{U}$ inférieure à 1. Cette condition n'est pas réellement restrictive puisqu'un système qui ne satisfait pas cette condition n'est en moyenne pas ordonnançable. Deux cas sont ensuite à distinguer selon que l'utilisation processeur maximale U^{max} est inférieure ou pas à 1. Dans le premier cas, un calcul itératif du steady-state backlog

est nécessaire entraînant une approximation sur le calcul du temps de réponse (approximation non nécessairement pessimiste). Dans le second cas, le calcul du stedy-state backog se résume à calculer le backlog au bout d'une hyperpériode (calcul exact).

L'apport du mécanisme de surveillance d'échéances, permet de renforcer le déterminisme mais la prise en compte de l'impact de celui-ci sur une étude probabiliste du temps de réponse est assez compliquée. Néanmoins, le calcul d'une borne inférieure (pessimiste) d'ordonnançabilité pour chaque instance peut être déterminée sans difficulté. La méthode proposée ne nécessite plus de calcul du steady-state backlog ce qui a l'avantage de ne plus obtenir d'approximations non nécessairement pessimistes et d'obtenir des temps de calcul bien plus avantageux.

Pour que l'analyse probabiliste ait un sens, le système doit être composé uniquement de tâches concrètes. Le cas synchrone est un cas particulier où tous les offsets sont nuls. Ce dernier cas, permet de réduire l'analyse en minimisant la période d'étude.

7.5.2 Dimensionnement des budgets d'exécution

Le mécanisme de surveillance des échéances, considéré comme non suffisant pour contrôler la robustesse temporelle d'un système, peut être mis en place en addition d'un mécanisme de protection basé sur des budgets d'exécution afin de contrôler au mieux la sûreté de fonctionnement. Ce mécanisme de surveillance n'est pas suggéré par le standard AUTOSAR OS qui considère un dimensionnement sûr du mécanisme de protection (cas déterministe), mais pourrait être envisagé dans le cas d'une étude probabiliste visant à toujours relaxer au maximum les budgets en garantissant une sûreté de fonctionnement suffisante.

Troisième partie

Étude de cas

« À la source de toute connaissance, il y a une idée, une pensée, puis l'expérience vient confirmer l'idée. »

Claude Bernard

Chapitre 8

Présentation de l'étude

Sommaire

8.1 Présentation de la plate-forme matérielle AFP9328 114
8.2 Présentation du système d'exploitation Trampoline 114
8.2.1 Un système d'exploitation compatible OSEK/VDX 115
8.2.2 L'extension AUTOSAR 116
8.2.3 Module de traces 119
8.3 Portage sur cible AFP9328 120
8.3.1 Portage de la protection temporelle 120
8.3.2 Portage du module de traces 120
8.4 Précision du mécanisme de protection temporelle 121

Nous souhaitons finaliser cette étude par l'implémentation et l'expérimentation du mécanisme de protection temporelle sur une cible réelle afin de montrer qu'il est possible de mettre en place les différentes configurations de dimensionnement présentées dans la partie précédente.

Cette étude passe par le choix d'une plate-forme matérielle d'exécution et d'un système d'exploitation temps réel compatible AUTOSAR. Dans ce cadre, nous avons choisi d'utiliser la plate-forme matérielle AFP9328 (à disposition dans l'équipe) et le système d'exploitation Trampoline développé essentiellement par les membres de l'équipe. Cependant, la mise en place du mécanisme de protection temporelle sur la plate-forme retenue n'est pas réalisée. Il nous faut donc développer le portage de celui-ci.

Le chapitre s'articule comme suit. La section 1 permet de faire une présentation de la plate-forme matérielle AFP9328 afin d'apporter les principaux éléments qui nous intéressent. Puis nous introduisons, section 2, le système d'exploitation temps réel Trampoline et plus spécifiquement la partie protection temporelle. La section 3 est consacrée au portage des différents éléments sur la cible AFP9328.

8.1 Présentation de la plate-forme matérielle AFP9328

Notre implémentation est basée sur une plate-forme AFP9328. L'APF9328 est une carte à microprocesseur de taille réduite équipée d'un microprocesseur ARM9 à 200MHz, de 16Mo SDRAM, de 16 Mo de FLASH, d'un port Ethernet 10/100Mbits, et d'un FPGA Spartan 3 (200K portes). Elle est facilement intégrable dans un système embarqué grâce notamment à ses régulateurs et ses convertisseurs de niveau (RS232/USB). La figure 8.1 présente une photo de la plate-forme.

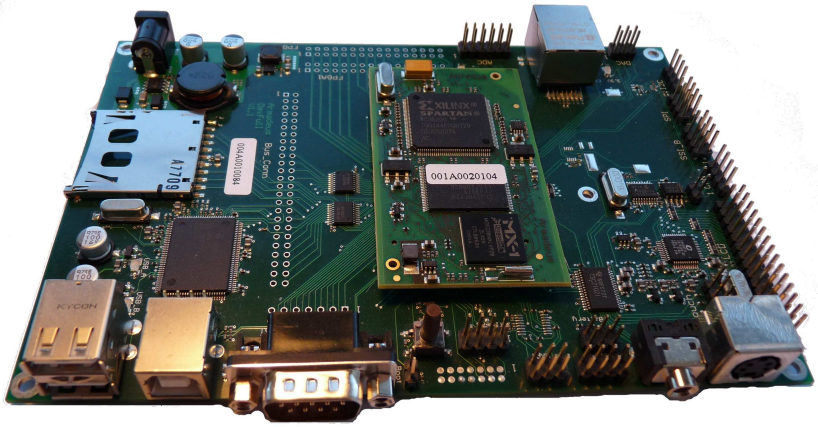


Figure 8.1 – Plate-forme matérielle d'exécution

Le transfert des programmes est réalisé via la liaison série RS232. De plus, nous verrons que nous pouvons envoyer des données par cette liaison en cours d'exécution du programme.

Nous présentons ici, les principales caractéristiques du MC9328MXL. Pour plus d'informations, se référer à la documentation constructeur (Freescale, 2007).

Le microprocesseur intégré est un processeur ARM MC9328L(i.MXL) ARM920T 200MHz (Freescale). Le MC9328MXL embarque un cœur processeur ARM920T dont les principales caractéristiques sont :

- Fréquence maximum d'exécution 200 MHz ;
- Processeur RISC 32-bit haute performance ARM9 ;
- 16 Ko de cache d'instructions et 16 Ko de cache de données (algorithme de remplacement : pseudo-random ou round-robin) ;
- 5 niveaux de pipeline.

Le MC9328MXL contient deux timers 32-bit identiques qui permettront de mettre en place le compteur général ainsi que le mécanisme de protection temporelle.

8.2 Présentation du système d'exploitation Trampoline

Trampoline (Bechennec *et al.*, 2006) est à l'origine un système d'exploitation temps-réel (ou exécutif) à source ouvert développé par l'équipe.

Trampoline 2.x est composé des éléments suivants :
- le noyau du système, qui est responsable de l'ordonnancement des tâches et ISR, de la gestion des interruptions et des synchronisations entre processus (ressources, événements) ;
- une personnalité OSEK, qui propose une API syntaxiquement et sémantiquement alignée sur le standard OSEK/VDX OS ;
- une personnalité AUTOSAR, qui propose une API syntaxiquement et sémantiquement alignée sur le standard AUTOSAR OS ;
- des BSPs (Board Support Package) pour différentes cartes, qui fournissent les composants logiciels spécifiques à une carte.

Trampoline est distribué sous la licence LGPL v2. Trampoline fonctionne maintenant sur 7 plate-formes : système POSIX, système POSIX (en para-virtualisation) avec Viper 2, Infineon C166 (chaîne KEIL), Freescale HCS12, Freescale/IBM PowerPC, ARM (olimex-lpc-e2294 ; simtec-eb675001 ; Lego Mindstorm NXT2.0 ; AFP9328) et Atmel 8 bits AVR.

La documentation complète ainsi que la version actuelle sont disponibles sur le serveur de l'équipe [1]. Un forum est également ouvert à toutes questions sur les différentes fonctionnalités de Trampoline. Nous allons ici résumer brièvement les points qui nous intéressent plus particulièrement.

8.2.1 Un système d'exploitation compatible OSEK/VDX

Les tâches sont les éléments actifs de l'application. Deux catégories de tâches peuvent être différenciées dans OSEK/VDX : les tâches basiques et étendues. Nous nous concentrons sur les tâches basiques puisque nous considérons dans notre étude des tâches indépendantes. Une tâche basique est composée d'un code séquentiel (procédure écrite en langage C). Une tâche basique est à tout instant dans un des états *SUSPENDED*, *READY* ou *RUNNING* (figure 8.2).

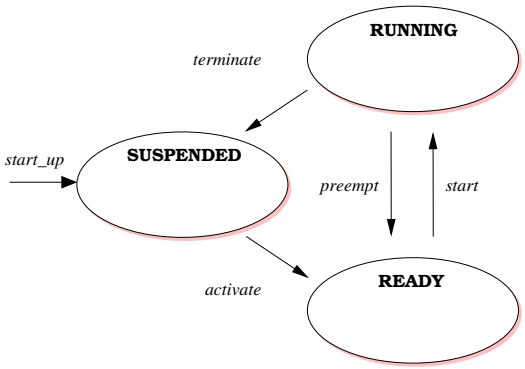


Figure 8.2 – Modèle d'exécution des tâches basiques dans OSEK/VDX OS

Les différents services pour une tâche basique sont :

1. Disponible à l'adresse http://trampoline.rts-software.org/

- *TerminateTask* : la tâche termine son exécution (c'est forcément la tâche en cours d'exécution qui appelle ce service) ;
- *ActivateTask* : une tâche peut activer une autre tâche ou elle-même ;
- *ChainTask* : dans un même appel (action atomique), la tâche appelante se termine et la tâche désignée est activée.

Un exemple de deux tâches basiques est présenté figure 8.3. La tâche `T1` active la tâche `T2`. La tâche `T2` se termine en activant la tâche `T1`.

```
Task(T1)
{
  --- ;
  ActivateTask(T2) ;
  --- ;
  TerminateTask() ;
};
```

```
Task(T2)
{
  --- ;
  --- ;
  --- ;
  ChainTask(T1) ;
};
```

Figure 8.3 – Exemple d'utilisation des services d'une tâche basique

L'activation des tâches périodiques est réalisée via un mécanisme d'alarme. Une alarme est associée à un compteur et à une tâche. Elle est déclenchée quand le compteur atteint une valeur de référence. Une action est alors exécutée qui peut être l'activation d'une tâche, la signalisation d'une occurrence d'événement ou l'exécution d'une routine de "callback". L'ensemble des alarmes cycliques permettant de générer l'activation des tâches périodiques est généralement relié à un compteur unique ("general_counter" ou "system_counter").

Une application se compose de deux types de fichiers : un fichier de description de l'architecture de l'application (écrite en langage OIL, OSEK Implementation Language), ainsi qu'un fichier contenant les algorithmes des différentes tâches et procédures d'interruption (écrit en langage C). Un exemple d'implémentation d'une tâche périodique est présenté figure 8.4 (fichier de description + algorithme).

Dans notre étude, une application est composée d'une ensemble de tâches périodiques. Le fichier de configuration comprendra donc un ensemble de paires tâche/alarme, ainsi qu'un compteur général.

8.2.2 L'extension AUTOSAR

Dans notre étude, nous nous intéressons plus particulièrement à la mise en place du mécanisme de protection temporelle. L'implantation du mécanisme dans trampoline a nécessité la réalisation des développements suivants :

- extension du compilateur goil ;
- création d'un module « protection temporelle » ;
- intégration de ce module dans le noyau de trampoline.

Extension de goil

Goil est le compilateur de fichiers de configuration OIL du projet trampoline. Il génère la configuration statique du noyau en fonction des besoins spécifiques de l'application, décrits

```
TASK task1 {
  PRIORITY = 2 ;              // Priorité de la tâche
  AUTOSTART = FALSE ;
  SCHEDULE = FULL ;           // Tâche préemptive
  ACTIVATION = 1 ;            // Compteur d'activation
};

ALARM activate_task1 {
  COUNTER = SytemCounter;     // Compteur lié à l'alarme
  ACTION = ACTIVATETASK {
    TASK = task1 ;            // Tâche activée par l'alarme
  } ;
  AUTOSTART = TRUE {
    ALARMTIME = 50 ;
    CYCLETIME = 100 ;         // Période de la tâche (100ms)
    APPMODE = std ;
  } ;
} ;

COUNTER SytemCounter {        // Période du compteur fixée à 1ms
  SOURCE = TIMER1_INT ;
  MAXALLOWEDVALUE = 2000 ;
  TICKPERBASE = 10 ;
  MINCYCLE = 1 ;
} ;
```

Fichier de description de l'architecture (OIL)

```
Task(T1)
{
  --- ;
  Algorithme de la tâche
  --- ;
  TerminateTask() ;
};
```

Fichier d'algorithme (C)

Figure 8.4 – Exemple d'implémentation d'une tâche périodique

dans un ensemble de fichiers OIL.

Goil a été étendu pour prendre en considération les informations de configuration du mécanisme de protection temporelle. Les extensions réalisées sont illustrées par les exemples présentés figure 8.5.

Création d'un module « protection temporelle »

Le module de protection temporelle développé trouve naturellement sa place dans la personnalité AUTOSAR de trampoline.

La partie du module qui contrôle les activations et le temps d'exécution des processus comporte quatre fonctions principales, qui correspondent aux transitions des diagrammes états / transitions présentés section 2.3.2, figure 2.5 et 2.6 :

tpl_bool tpl_tp_on_activate_or_release (**tpl_proc** proc_id)
tpl_bool tpl_tp_on_terminate_or_wait (**tpl_proc** proc_id)
tpl_bool tpl_tp_on_start (**tpl_proc** proc_id)
tpl_bool tpl_tp_on_preempt (**tpl_proc** proc_id)

Chaque fonction contrôle la validité de la transition pour le processus identifié en paramètre (seule une activation peut être refusée) puis met à jour les variables d'états de ce processus.

```
TASK t1 {
  ...
  LOCKINGTIME = TRUE {
    MAXRESOURCELOCKTIME = 0.001;
    RESOURCE = Res1;
  }
  TIMING_PROTECTION = TRUE {
    EXECUTIONBUDGET = 0.01;
    TIMEFRAME = 0.05;
    MAXOSINTERRUPTLOCKTIME = 0;
    MAXALLINTERRUPTLOCKTIME = 0;
  };
};
```

Configuration d'une tâche

```
ISR isr1 {
  ...
  TIMING_PROTECTION = TRUE {
   TIMEFRAME = 0.01;
   EXECUTIONTIME = 0.005;
   MAXOSINTERRUPTLOCKTIME = 0.0025;
   MAXALLINTERRUPTLOCKTIME = 0;
  };
};
```

Configuration d'une ISR

Figure 8.5 – Configuration de la protection temporelle dans le fichier OIL

Surveillance de la fréquence maximale d'activation : La date de la dernière activation réussie est mémorisée Lorsqu'une requête d'activation arrive, la date courante est comparée avec la dernière date mémorisée pour décider si cette requête peut être acceptée ou si l'erreur `E_OS_PROTECTION_ARRIVAL` doit être positionnée. Tous ces traitements sont réalisés dans la fonction *tpl_ tp_ on_ activate_ or_ release*. Si une erreur est positionnée, le noyau se charge d'invoquer la fonction de recouvrement de l'application fautive (c'est-à-dire l'application propriétaire de la tâche activée).

Surveillance de la durée maximale d'exécution : Le budget d'exécution restant à un processus est mémorisé. Il est initialisé à sa valeur maximale, et réinitialisé à chaque terminaison d'un module du processus (fonction *tpl_ tp_ on_ terminate_ or_ wait*).

Lorsque le processus devient élu, la date d'élection est mémorisée et un événement est programmé pour alerter le mécanisme à la date égale à la date courante plus le budget restant (fonction *tpl_ tp_ on_ start*).

Lorsque le processus est préempté, le budget restant est mis à jour et l'événement programmé est annulé (fonction *tpl_ tp_ on_ preempt*).

Si le processus n'est pas préempté alors que l'alerte est signalée, la fonction *tpl_ tp_ on_ time_ error* est appelée. Elle positionne l'erreur `E_OS_PROTECTION_TIME` et invoque la fonction de recouvrement de l'application fautive.

Surveillance de la durée maximale des sections critiques : Le mécanisme est similaire à celui utilisé pour surveiller la durée maximale d'exécution des processus. Les budgets (un par ressource, un pour le verrouillage des ISR2, un pour le verrouillage de toutes les interruptions) sont initialisés au moment de l'entrée en section critique (fonctions *tpl_ tp_ on_ enter_ cs_ res*, *tpl_ tp_ on_ enter_ cs_ osisr*, *tpl_ tp_ on_ enter_ cs_ allisr*).

La fonction *tpl_ tp_ on_ time_ error* positionne l'erreur `E_OS_PROTECTION_LOCKED` et invoque la fonction de recouvrement de l'application fautive.

Validation de la protection temporelle

Des tests du mécanisme de protection temporelle ont été développés et intégrés à la suite de tests de Trampoline. Ils sont décrits dans les documents (Pavin, 2010a) et (Pavin, 2010b).

Ils concernent la surveillance de la fréquence maximale d'activation et de la durée maximale d'exécution. Ces tests doivent être complétés pour la surveillance de la durée maximale des sections critiques.

8.2.3 Module de traces

Nous désirons connaître les différents événements caractéristiques de l'application afin de pouvoir analyser l'impact de la protection temporelle. Le mécanisme de gestion des traces, *TRACE*[2] permet de détecter et d'enregistrer les différents événements qui nous intéressent (activation, démarrage, terminaison, préemption d'une tâche).

Un petit outil de représentation graphique écrit en Java, *T3*[3], permet de tracer les chronogrammes correspondants aux traces enregistrées.

Fonctionnement de TRACE

Un attribut TRACE a été ajouté au langage OIL. Les sous-attributs sont METHOD et FORMAT. METHOD permet de spécifier comment la trace est envoyée (actuellement, seule la méthode FILE est supportée avec comme sous attribut le nom du fichier NAME). FORMAT permet de spécifier le format de la trace. Le FORMAT est libre : le fichier *tpl_ target_ trace.c*, fichier de portage de TRACE spécifique à la plate-forme, contient un ensemble de fonctions considérant divers formats (texte, xml, binaire pour la version POSIX). Le partie configuration de l'OS du fichier OIL est donc complétée des attributs présentés figure 8.6.

```
TRACE = TRUE {
  METHOD = FILE {
    NAME = "toto.[bin, xml, txt]";
  };
  FORMAT = [bin, xml, txt];
};
```

Figure 8.6 – Modification du fichier OIL pour utiliser TRACE

Une trace se compose d'une date de début d'événement, une date de fin d'événement, un type d'événement et d'un ensemble de paramètres (au nombre de 10 maximum). La figure 8.7 donne la structure de trace utilisée.

```
struct STR_TRACE
{
  int32 begin_date ;
  int32 trace_type ;      // À chaque type d'événement est associé un identifiant
  int32 value[10] ;       // Ensemble des paramètres
  int32 end_date ;
}
```

Figure 8.7 – Structure d'une trace d'événement

La trace est actuellement enregistrée sous forme d'un fichier. L'écriture du fichier est réalisée actuellement à chaque apparition d'un événement. Pour le format texte, la procédure d'écriture ajoute une nouvelle ligne au fichier ; le format xml, enregistre une nouvelle entrée ;

2. Documentation disponible à l'adresse http://trampoline.rts-software.org/trac/wiki/Trace
3. Programme et documentation disponible à l'adresse http://jttrace.rts-software.org/

le format bin ajoute de nouvelles données à la suite. Les données décrivent la nature de l'événement (activation, démarrage, préemption, terminaison) et la date d'occurrence (exprimée par rapport à la date de démarrage du système).

8.3 Portage sur cible AFP9328

Nous avons réalisé le portage de la protection temporelle ainsi qu'un module de génération de trace sur la plate-forme matérielle AFP9328.

8.3.1 Portage de la protection temporelle

Actuellement, la protection temporelle est uniquement disponible pour la plate-forme POSIX. La cible utilisée étant la plate-forme AFP9328 dont le processeur est un ARM9, nous réalisons le portage des fonctions principales nécessaires à la protection temporelle pour notre cible. La mise en place du mécanisme passe par l'implémentation de trois fonctions :

- *tpl_get_local_current_date* qui permet de fournir la date actuelle par rapport à la date de démarrage de l'application. Cette fonction a été réalisée à partir du premier timer disponible sur la cible ;
- *tpl_set_watchdog* qui permet de démarrer le contrôle du budget d'exécution. Le contrôle du budget est réalisé par le deuxième timer disponible sur la cible est du branchement de l'interruption TIMER1_INT sur la procédure de détection d'erreur ;
- *tpl_stop_watchdog* qui permet de stopper le contrôle du budget d'exécution.

8.3.2 Portage du module de traces

Actuellement, la protection temporelle est uniquement disponible pour la plate-forme POSIX. Nous réalisons de même qu pour la protection temporelle, le portage des fonctions principales nécessaires au module de traces pour notre cible.

Le module de trace nécessite de connaître la date courante à laquelle survient un événement. La fonction *tpl_get_local_current_date*, déjà implémentée pour la protection temporelle, peut être réutilisée à cet effet.

L'enregistrement proposé dans la version actuelle par fichier de données n'est pas envisageable dans notre cas d'étude. Nous choisissons de conserver un ensemble maximum de traces en mémoire dans un tableau de « structure de trace » afin de perturber au minimum l'exécution de l'application. La transmission de l'ensemble des données enregistrées est réalisée par liaison série et l'utilisation d'un programme de contrôle de communication (de type minicom) sur l'ordinateur base. Le format choisi pour l'écriture des données est un format texte, comparable au format réalisé pour le portage POSIX.

Une nouvelle trace est également ajouté afin d'enregistrer la terminaison d'une tâche après détection d'une erreur par le mécanisme de protection temporelle.

Un exemple de chronogramme obtenu par l'outil T3 après exécution d'un système comportant deux tâches, une tâche périodique de période 100 millisecondes et une tâche activée

par la première toutes les dix activations, est présenté figure 8.8.

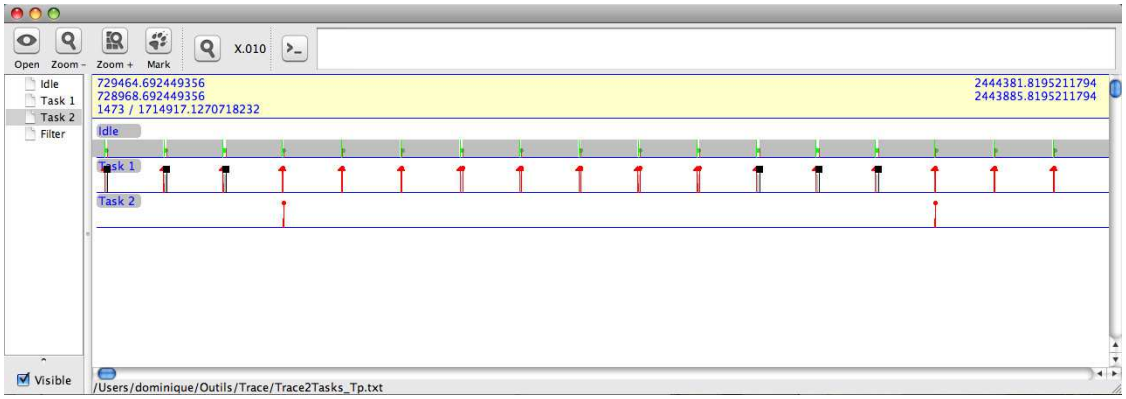


Figure 8.8 – Exemple de chronogramme obtenu par T3

Une mesure sur le logiciel T3 à permis de déterminer la période de la tâche 1, période mesurée à 100 millisecondes. Nous pouvons remarquer également que 10 activations de la tâche 1 séparent deux activations de la tâche 2. Le chronogramme reflète donc parfaitement le comportement du système considéré. Un carré noir représente la détection d'une erreur temporelle (non respect du budget d'exécution). La tâche est alors tué par le mécanisme de protection.

8.4 Précision du mécanisme de protection temporelle

Nous avons mesuré la précision du mécanisme de protection temporelle en testant différentes valeurs de budget d'exécution et en mesurant le temps réel sur la plate-forme d'exécution. Les résultats sont présentés tableau 8.1.

Budget d'exécution	Valeur moyenne mesurée	Pourcentage d'erreur
100 μs	231 μs	+131%
1000 μs	1130 μs	+13%
10000 μs	10121 μs	+1.2%

Tableau 8.1 – Précision du mécanisme de protection temporelle

Les temps mesurés sont largement supérieurs à la valeur désirée. Cette différence est dû aux temps systèmes non négligeables dans le cas de faibles budgets. Les temps systèmes sont en moyenne de l'ordre de 130 μs. Un budget d'exécution trop petit (inférieur à quelques millisecondes) ne peut donc pas contrôler de façon sûr le système. Il est donc nécessaire de conserver une marge de sécurité lors e la configuration du mécanisme de protection temporelle.

« Estimer correctement son degré d'ignorance est une étape saine et nécessaire. »

Patience dans l'azur - Hubert Reeves

Chapitre 9

Estimation du pire temps d'exécution

Sommaire

9.1 Méthodes d'analyse des temps d'exécution 124
9.1.1 Méthodes par analyse statique 124
9.1.2 Méthodes par simulations 124
9.2 Les différentes tâches et comportements temporels 130
9.2.1 Détermination des différents paramètres 130
9.2.2 Observateur de Luenberger 131
9.2.3 Filtre de Kalman 133
9.2.4 Tâche dont le temps d'exécution suit une loi de Gumbel 135
9.2.5 Tâche de type « machine à états » 138
9.3 Discussion 140

Une étape préliminaire à l'étude d'ordonnançabilité des systèmes et au dimensionnement des budgets d'exécution est l'estimation des temps d'exécution (ou du pire-temps d'exécution dans le cas d'une étude déterministe).

Dans cette partie, nous allons essayer de mettre en place des programmes types représentant des comportements possibles de programmes réels.

Le chapitre s'articule comme suit. La section 1 permet de faire un rappel sur les différentes méthodes d'analyse des temps d'exécution (méthodes par analyse statique ou par simulation). Les méthodes par simulation sont encore très largement utilisées par le domaine industriel. Celles-ci comportent quelques contraintes (sous-estimation du pire temps d'exécution possible, données importantes à traiter) que nous essaierons de surmonter. Puis, nous introduirons en section 2 quelques comportements temporels de tâches implémentés puis analysés. Enfin nous discuterons, section 3, les résultats obtenus et les perspectives possibles.

9.1 Méthodes d'analyse des temps d'exécution

Comme énoncé précédemment (section 3.2.1), l'analyse du temps d'exécution d'une tâche peut être réalisée par calcul (à l'aide d'outil d'analyse statique) ou par mesure (tests du programme). Le choix de la méthode dépend généralement du caractère critique de la tâche (Vestal, 2007).

9.1.1 Méthodes par analyse statique

Les méthodes statiques analysent le binaire d'une tâche afin d'en déterminer un pire temps d'exécution. Ces méthodes peuvent être divisées en deux étapes. La première étape consiste en une analyse de haut niveau qui permet de déterminer le chemin d'exécution le plus long. Le chemin d'exécution est composé d'un ensemble de blocs séquentiels de codes. La seconde étape consiste en une analyse de bas niveau qui permet de déterminer le temps maximum d'exécution de ces blocs de code. Celle-ci est réalisée en modélisant la plate-forme matérielle d'exécution (cache, pipeline, ...).

Comme nous l'avons fait remarquer précédemment, l'étape de bas niveau peut être réalisée par simulation ou exécution sur cible (Bernat *et al.*, 2002; Colin et Petters, 2003). Dans ce cas, l'analyse des temps d'exécution est dite mixe.

9.1.2 Méthodes par simulations

Les méthodes par simulations sont simples puisqu'elles reposent sur un ensemble de mesures de durées d'exécution d'une tâche en fonction des différentes configurations du logiciel et de l'architecture matérielle. L'ensemble des mesures permet de tracer un profil d'exécution pouvant être représenté sous la forme d'une fonction de densité de probabilité discrète.

Cependant, ces mesures ne représentent pas l'ensemble des situations possibles car il est très difficile de réaliser un ensemble de tests exhaustifs. L'estimation du pire temps d'exécution peut donc être biaisée et sous-estimée. La figure 9.1 représente un exemple de profil d'exécution et la mesure obtenue par simulation.

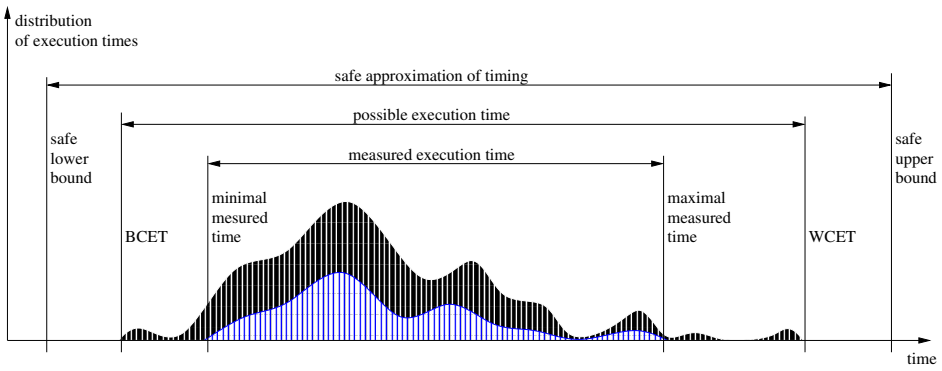


Figure 9.1 – Estimation des temps d'exécution (Gustafsson, 2008)

Dans le cas présenté, les mesures ne sont pas suffisantes pour déterminer sûrement le pire

temps d'exécution (WCET) de la tâche. En effet, la valeur maximale d'exécution mesurée est inférieure à la valeur du WCET. Un facteur de sécurité peut être appliqué afin de s'assurer de la valeur retenue du WCET pour les tests d'ordonnançabilité ("safe upper bound").

Les distributions discrètes obtenues peuvent contenir un nombre de valeurs d'exécution très important. Les études menées précédemment utilisant des méthodes probabilistes ont été considérées (en simulation) comme ayant en entrées des distributions discrètes réduites (5 valeurs d'exécution). Le temps de calcul de ces méthodes croît rapidement avec la taille des distributions considérées. Une étape intermédiaire semble donc nécessaire afin de réduire ces distributions. Dans ce cadre, deux méthodes peuvent être mises en œuvre. La première solution est de modéliser les temps d'exécution par une distribution particulière (discrète). La seconde solution est de réduire directement la distribution par échantillonnage de la distribution obtenue par mesure.

Modélisation des temps d'exécution

Une modélisation des temps d'exécution sous forme d'une distribution de Gumbel (Gumbel, 1958) a été proposée par Burns et Edgar (2000). Par rapport au comportement asymptotique des résultats de simulation, la distribution considérée peut être uniquement de trois types (référence à Fisher et Tippett (1928)).

Définition 9.1. *La fonction de distribution de Gumbel approximant un ensemble de données de la forme* $\{x_1, \ldots, x_n\}$*, de moyenne* μ *et d'écart type* σ *possède la fonction de densité de probabilité suivante*

$$G_{(\mu,\sigma)}(x) = exp\left(-exp\left(-\frac{x-\mu}{\sigma}\right)\right)$$

Selon Burns et Wellings (2001), un modèle est construit à l'aide d'un échantillon aléatoire de mesures de temps d'exécution. Un échantillon est composé de n mesures $\{x_1, \ldots, x_n\}$. À partir de ces mesures, une distribution de Gumbel, $G_{(\mu,\sigma)}(t)$, modélisant les temps d'exécution est calculée (t représente le temps). Les valeurs des estimateurs non-biaisés des paramètres μ et σ sont exprimées en fonction de la moyenne $\overline{x}$ et la variance $\hat{\sigma}^2$:

$$\mu = \overline{x} - \lambda.\sigma, \quad \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \lambda \approx 0.5772 \text{ (constante d'Euler)}$$

$$\sigma = \frac{\sqrt{6}}{\pi}\hat{\sigma}, \quad \hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

Il est donc rapide de déterminer une modélisation des temps d'exécution en fonction des mesures effectuées. À partir de cette distribution $G_{(\mu,\sigma)}(t)$, les auteurs construisent une fonction de distribution de l'estimation du WCET, θ_i exprimée par :

$$\theta_i(x) = \frac{G(x + \max_k\{x_k\}) - G(\max_k\{x_k\})}{1 - G(\max_k\{x_k\})}$$

Cette fonction permet de déterminer la confiance que nous pouvons accorder à la valeur du WCET choisie. Plus le pire temps d'exécution considéré est important, plus la confiance comme borne supérieure sûre est élevée. Cependant, une valeur trop importante n'est pas

intéressante pour une étude d'ordonnançabilité. La fonction permet de préciser la garantie que l'on peut avoir dans la valeur du WCET. La figure 9.2 représente un exemple de modèle de temps d'exécution ainsi que la fonction de confiance du WCET.

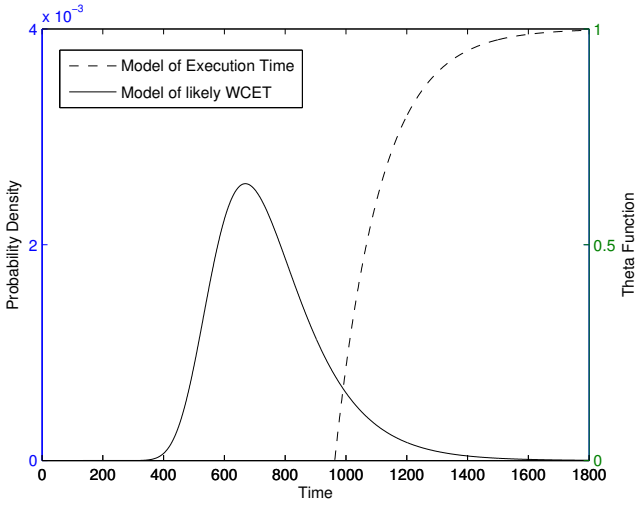


Figure 9.2 – Tracé des densités de probabilité et de l'estimation du WCET par rapport à des données hypothétiques (Burns et Wellings, 2001)

Échantillonnage

Refaat et Hladik (2010) proposent une méthode permettant d'échantillonner une distribution donnée sous la forme d'une distribution réduite telle que celle-ci puisse être utilisée pour une étude probabiliste d'ordonnançabilité (échantillonnage pessimiste).

L'étude se base sur les concepts de pessimisme en analyse probabiliste exposés par Diaz *et al.* (2004).

Définition 9.2. *Soit $\mathcal{X}'$ et $\mathcal{X}$ deux variables aléatoires. La variable aléatoire $\mathcal{X}'$ est plus grande que $\mathcal{X}$, notée $\mathcal{X}' \succ \mathcal{X}$, si $\forall X$, $\mathbb{P}(\mathcal{X}' \leq X) \leq \mathbb{P}(\mathcal{X} \leq X)$ et les deux variables ne sont pas identiques.*

L'échantillonnage considéré doit prendre en considération la probabilité d'occurrence de chaque temps d'exécution. En effet, un temps d'exécution ayant une forte probabilité d'occurrence doit avoir une plus forte chance d'apparaître dans l'échantillonnage. L'échantillonnage proposé permet de sélectionner k échantillons parmi les N valeurs possibles. La distribution réduite $f_{e'}$ extraite de la distribution originale f_e s'exprime par :

$$f_{e'}(x) = \sum_{i=1}^{k} f_e(x_i)\delta(x - x_i) + \left(1 - \sum_{i=1}^{k} f_e(x_i)\right)\delta(x - x_w) \tag{9.1}$$

où $\{x_i\}_{i=1,\ldots,k}$ est l'ensemble des k points d'échantillonnage prélevé dans $\{x_j\}_{j=1,\ldots,n}$, x_w est le pire temps d'exécution de la distribution d'origine et δ est la fonction définie par :

$$\delta(x) = \begin{cases} 1 & \text{si } x = 0 \\ 0 & \text{si } x \neq 0 \end{cases} \tag{9.2}$$

Le premier terme de l'équation (9.1) représente l'ensemble des valeurs d'échantillonnage, le second terme représente le reste des probabilités accumulées sur le temps d'exécution maximal afin de préserver le pessimisme.

Le choix des k valeurs d'échantillonnage $\{x_i\}_{i=1,\ldots,k}$ est réalisé par rapport à la probabilité d'occurrence. Soit la distribution d'origine de la variable aléatoire $\mathcal{X}$ caractérisée par :

$$\mathbb{P}(\mathcal{X} = x_j) = p_j \tag{9.3}$$

Un nombre aléatoire U est tiré uniformément dans l'intervalle $[0, 1[$. D'après la valeur de U on choisit un échantillon x'_i comme suit

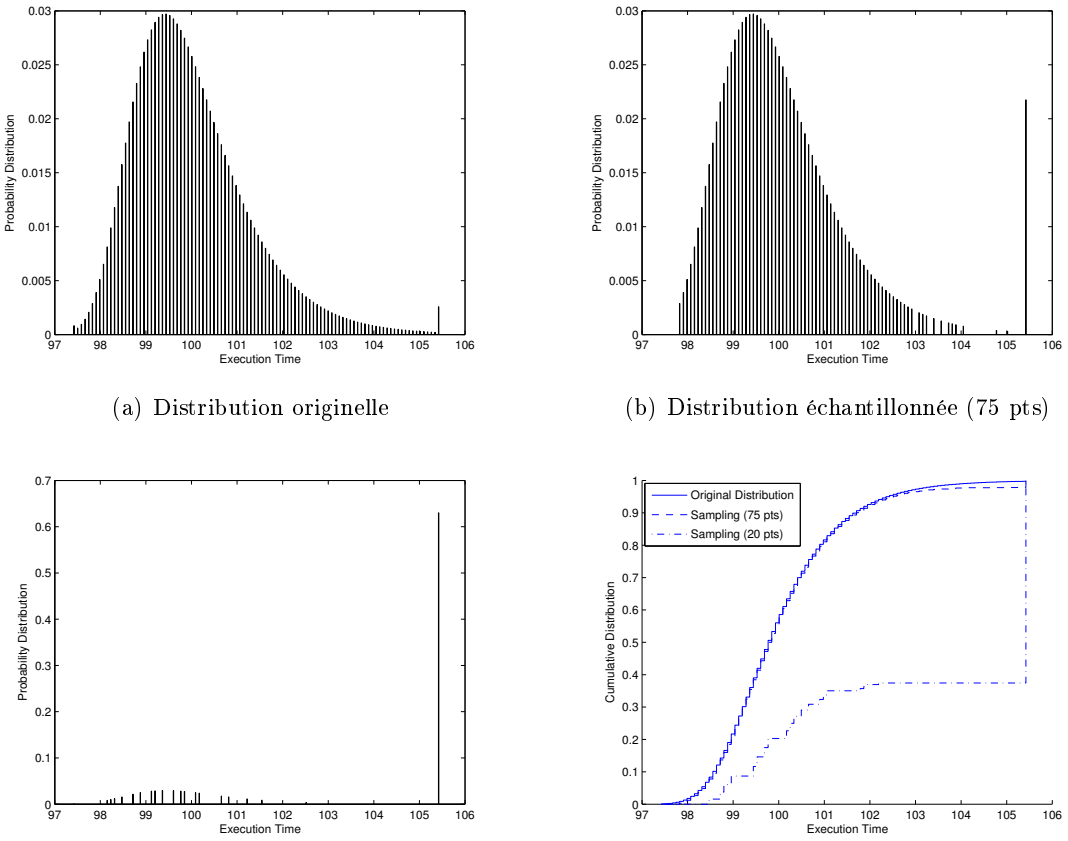
$$x'_i = \begin{cases} x_0 & \text{si } U < p_0 \\ x_1 & \text{si } p_0 \leq U < p_0 + p_1 \\ \vdots & \\ x_j & \text{si } \sum_{k=0}^{j-1} p_k \leq U < \sum_{k=0}^{j} p_k \\ \vdots & \\ x_n & \text{si } \sum_{k=0}^{n-1} p_k \leq U < 1 \end{cases} \tag{9.4}$$

Cette procédure est répétée k fois afin d'obtenir l'ensemble des points d'échantillonnage. Si un échantillon est déjà présent, une nouvelle valeur est tirée afin d'obtenir k valeurs distinctes.

La figure 9.4 présente un exemple d'échantillonnage d'une distribution comportant 100 valeurs distinctes (k est fixé à 75 puis à 20 valeurs).

Dans le premier cas ($k = 75$, figure 9.3(b)), la distribution obtenue est assez proche de la distribution originelle. Cependant, plus la valeur de k diminue ($k = 20$, figure 9.3(c)), plus les valeurs d'échantillonnage semblent inutiles par rapport à la probabilité affectée à la valeur maximale d'exécution. La fonction de répartition, présentée figure 9.3(d), permet de montrer les écarts existant entre la répartition originelle et les deux échantillonnages. Néanmoins, l'effet illustré figure 9.3(c) est « limité » puisque l'échantillonnage et réalisé à chaque itération de calcul du steady-state backlog.

Cette méthode d'échantillonnage permet de minimiser la taille des calculs du steady-state backlog (Diaz *et al.*, 2002) en réduisant la taille des distributions mises en jeu dans les produits de convolution. En effet, le produit de convolution de deux variables aléatoires discrètes $\mathcal{X}$ et $\mathcal{Y}$ de tailles respectivement $n_\mathcal{X}$ et $n_\mathcal{Y}$ produit une variable aléatoire $\mathcal{Z}$ de taille $n_\mathcal{X} * n_\mathcal{Y}$ (en général). Le calcul du steady-state backlog faisant intervenir fréquemment un nombre important de produits de convolution, la taille de sa distribution devient très vite assez importante. Une réduction des variables par échantillonnage est une solution pour en diminuer la taille. Un échantillonnage des distributions est réalisé à chaque itération de calcul (convolution) et peut permettre de conserver une taille constante des données manipulées.



(a) Distribution originelle

(b) Distribution échantillonnée (75 pts)

(c) Distribution échantillonnée (20 pts)

(d) Fonctions de répartition

Figure 9.3 – Exemple d'échantillonnage d'une distribution

Utilisation pour notre étude

Dans notre cas, nous nous limitons à des mesures de temps d'exécution sur un ensemble de tests (méthode encore très largement utilisée dans le milieu industriel).

Pour notre étude, nous pouvons utiliser la méthode proposée par Burns et Edgar (2000) afin de définir l'estimation du WCET d'une tâche critique avec une certaine confiance (ces estimations pourront être de plus affectées d'un coefficient de sécurité afin de garantir une borne supérieure).

Le temps d'exécution d'une tâche non critique peut être modélisé par une variable aléatoire définie à partir d'une distribution issue des mesures. Le calcul du steady-state backlog fera appel à un échantillonnage (Refaat et Hladik, 2010) réalisé à chaque itération (échantillonnage différent à chaque nouvelle convolution) afin de minimiser la taille des distributions mises en jeu.

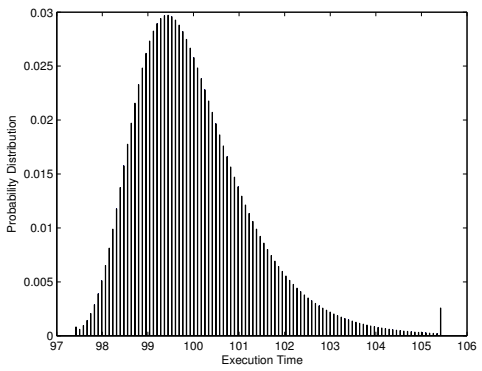
Nous apportons une légère modification à la méthode d'échantillonnage proposée. Le choix

des échantillons reste le même mais nous modifions l'équation (9.1) par l'équation suivante :

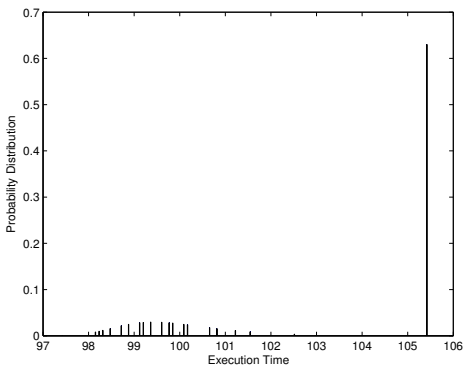
$$\begin{aligned} f_{e'}(x) =& F_e(x_0)\delta(x - x_0) + \sum_{i=2}^{k} \left(F_e(x_i) - F_e(x_{i-1})\right) \delta(x - x_i) \\ &+ \left(1 - F_e(x_k)\right) \delta(x - x_w) \end{aligned} \tag{9.5}$$

La preuve de la propriété pessimiste de l'échantillonnage obtenu est identique à la preuve formulée par Refaat et Hladik (2010). En effet, puisque les probabilités sont projetées sur une valeur d'exécution supérieure, la fonction de répartition obtenue sera toujours inférieure à la fonction de répartition non échantillonnée. Nous garantissons ainsi le pessimisme $\mathcal{E}' \succ \mathcal{E}$.

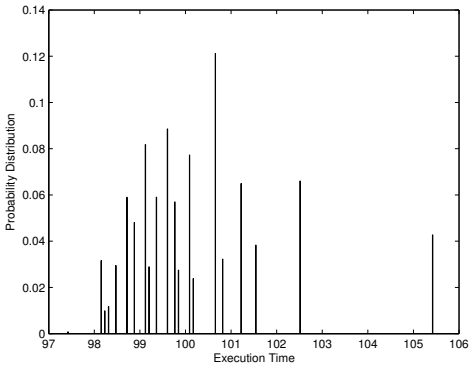
La figure 9.4 présente un exemple d'échantillonnage d'une distribution comportant 100 valeurs distinctes par la méthode précédente proposée par Refaat et Hladik (2010) et la version modifiée.



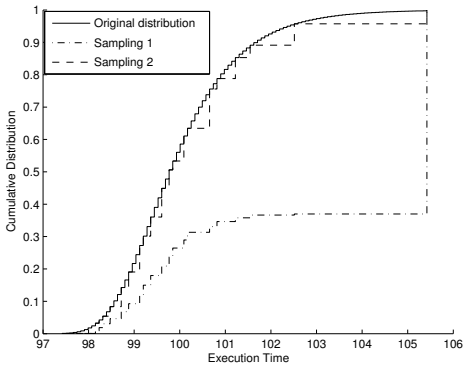
(a) Distribution originelle



(b) Distribution échantillonnée (Refaat et Hladik, 2010)



(c) Distribution échantillonnée (version modifiée)



(d) Fonctions de répartition

Figure 9.4 – Comparaison des méthodes d'échantillonnage

La distribution obtenue avec la version modifiée (figure 9.4(c)) comporte des valeurs significatives pour d'autres valeurs que la valeur maximale d'exécution. La fonction de répartition

présentée figure 9.4(d) permet de conclure que la répartition obtenue par échantillonnage modifié (sampling 2) est plus proche que celle obtenue par l'échantillonnage proposé par Refaat et Hladik (2010) (sampling 1) tout en restant pessimiste (la courbe échantillonnée reste en tout point inférieure à la courbe originelle).

9.2 Les différentes tâches et comportements temporels

Dans cette partie, nous considérons un ensemble de programmes caractéristiques. Ces programmes sont exécutés sur cibles afin de réaliser des mesures de temps d'exécution. Nous considérons l'application composée d'une tâche unique ne pouvant donc pas être préemptée. Les mesures sont réalisées via le mécanisme de trace qui permet de transmettre les dates d'activation et de terminaison de la tâche considérée.

9.2.1 Détermination des différents paramètres

Les mesures obtenues par expérimentation sont analysées afin de déterminer les différents paramètres utiles aux études d'ordonnançabilité décrites à travers les précédents chapitres (analyse déterministe et probabiliste).

Modèle probabiliste

Le modèle probabiliste présenté section 6.2 caractérise chaque tâche par un 7-uplet :

$$\left((\mathcal{C}_i, \mathfrak{C}_i), \phi_i, D_i, T_i, L_i, \pi_i \right)$$

Dans notre étude, l'offset ϕ_i, l'échéance D_i, la période D_i et la criticité L_i sont supposés fixés. Les priorités π_i sont déterminées à l'aide de l'algorithme proposé par Vestal (2007) (voir section 6.2.2). Les seuls paramètres à déterminés sont $(\mathcal{C}_i, \mathfrak{C}_i)$ caractérisant le temps d'exécution de chaque tâche.

Le premier paramètre $\mathcal{C}_i$ est la variable aléatoire caractérisant le temps d'exécution de la tâche τ_i. Pour une tâche critique, une valeur unique d'exécution est considérée représentant une borne supérieure sûre d'exécution (C_{sup}). Dans notre cas, nous considérons la valeur obtenue par les travaux de Burns et Wellings (2001) décrit ci-dessus. Pour une tâche non critique, $\mathcal{C}_i$ correspond à la distribution mesurée des temps d'exécution. Le calcul de l'ordonnançabilité d'une tâche explosant lorsque les distributions considérés ont une dimension importante, il est nécessaire d'échantillonner celles-ci. La méthode proposée par Refaat et Hladik (2010) considère un échantillonnage à chaque itération. Actuelle, cette méthode n'a pas été implémentée, nous considérons un échantillonnage préalable réalisé manuellement en utilisant l'équation (9.5), identique pour chaque itération.

Le second paramètre $\mathfrak{C}_i$ est une fonction dépendant du niveau de criticité. Dans le cas d'une tâche critique, nous avons proposé de fixer la fonction telle que

$$\mathfrak{C}_i(l=0) = \mathfrak{C}_i(l=1) = C_{sup}$$

Dans le cas d'une tâche non critique, nous proposons de fixer la fonction telle que :

$$\begin{cases} \mathfrak{C}_i(l=0) = \overline{C},\ \mathbb{P}(\mathcal{C}_i \leq \overline{C}) \geq p_i \\ \mathfrak{C}_i(l=1) = C_{sup} \end{cases}$$

Modèle déterministe

Dans le cas d'une étude déterministe d'ordonnançabilité, si la tâche est considérée non-critique, la valeur retenue pour le pire temps d'exécution est la valeur $\overline{C}$. Si la tâche est critique, la valeur considérée est la valeur C_{sup}.

9.2.2 Observateur de Luenberger

Présentation du programme

Nous allons coder un capteur logiciel (ou observateur) selon le modèle proposé par Luenberger (1964). Ce capteur permet, après identification d'un système, d'estimer son état interne afin de récupérer certaines informations non disponibles directement. Soit le système linéaire discret suivant :

$$\begin{cases} X_{k+1} &= AX_k + BU_k \\ Y_k &= CX_k \end{cases} \tag{9.6}$$

Un observateur dynamique a la forme suivante :

$$\hat{X}_{k+1} = A\hat{X}_k + BU_k + L(Y_k - C\hat{X}_k) \tag{9.7}$$

L'observateur permet de reconstruire une estimation $\hat{X}$ de l'état interne X à chaque instant k, à partir de la sortie du système Y, de l'entrée U et de la valeur du vecteur interne à l'instant précédent. La figure 9.5 représente un exemple d'utilisation de l'observateur dans une bouvle de commande.

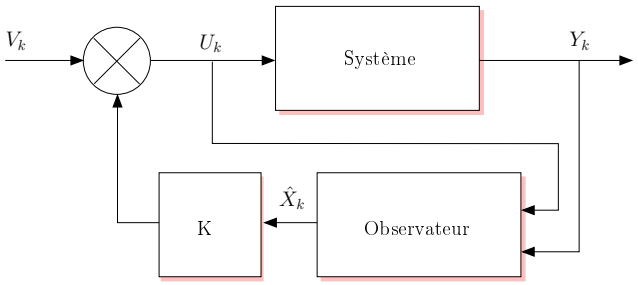


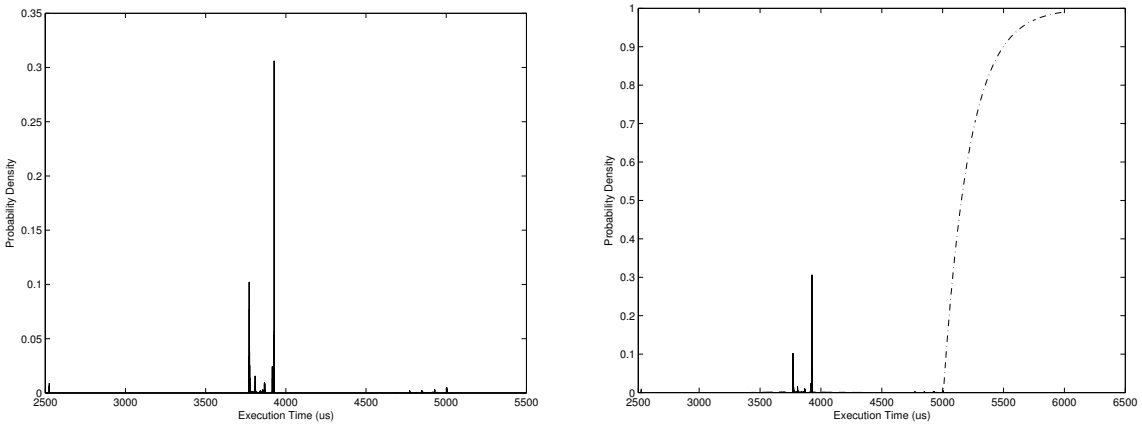
Figure 9.5 – Exemple de commande basée sur un observateur

Dans notre cas, nous allons considérer un système tel que la dimension du vecteur interne est 15, comportant une seule entrée et une seule sortie mesurée. Nous avons ainsi, A matrice 15×15, B vecteur colonne 1×15 et C vecteur ligne 15×1. Le but est de calculer une estimation $\hat{X}$ (vecteur colonne 1×15) en fonction de la valeur d'entrée u_k et de la mesure y_k (tous les deux scalaires). Le calcul ne fait pas intervenir de multiplication de matrices (uniquement des multiplications « matrice/vecteur colonne »). Cependant, les multiplications sont réalisées entre nombres flottants.

Deux ensembles de n valeurs, pour variable X_k et Y_k, ont été générés (génération des valeurs de sortie par simulation du système sous Matlab/Simulink afin d'obtenir des sorties cohérentes) et intégrées au programme.

Mesures du temps d'exécution

L'implémentation de l'observateur a été réalisée puis exécutée sur cible. Les résultats obtenus sont présentés figure 9.6.Nous pouvons remarquer que le nombre de valeurs différentes relevées n'est pas très important. La valeur moyenne des temps d'exécution mesurés est d'environ 4 ms, avec un écart type de l'ordre de 0.3 ms, une valeur minimale de 2.5 ms et maximale de 5 ms. Ce comportement est dû a une relative simplicité des calculs de l'algorithme de l'observateur étudié.



(a) Mesures (b) Estimation WCET

Figure 9.6 – Résultats des mesures des valeurs d'exécution (observateur de Luenberger)

Analyse des résultats

La fonction θ a été calculée et tracée (figure 9.6(b)). Le tableau 9.1 résume les valeurs d'estimation du WCET en fonction de quelques indices de confiance.

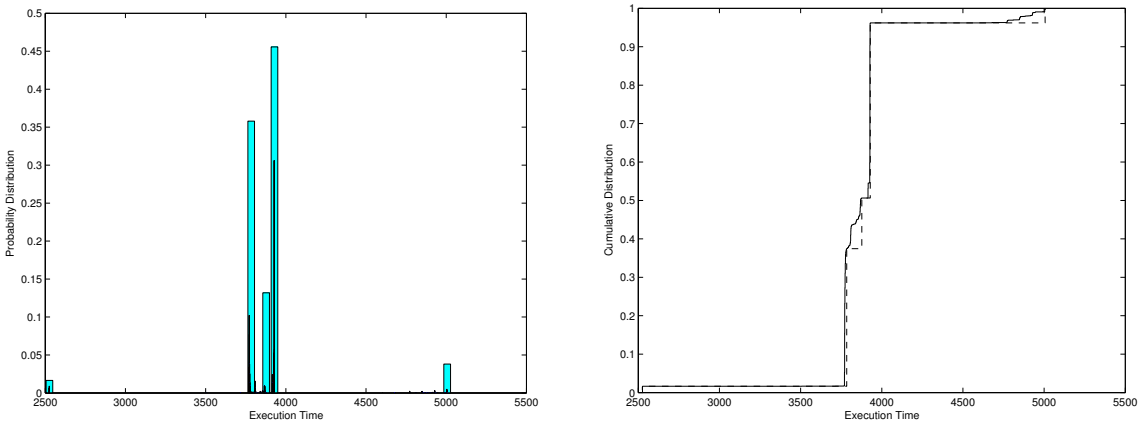
Niveau de confiance	Estimation du WCET
90%	5.5 ms
99%	6.0 ms
99.9%	6.5 ms
99.99%	7.0 ms

Tableau 9.1 – Estimation du WCET en fonction de l'indice de confiance (observateur de Luenberger)

Un échantillonnage de la distribution obtenue par mesure a été réalisé. La figure 9.7 représente l'échantillonnage appliqué aux mesures.

L'échantillonnage des valeurs donne la distribution suivante :

$$\mathcal{C}_i = \begin{pmatrix} 2.53 & 3.79 & 3.88 & 3.93 & 5.00 \\ 0.016 & 0.358 & 0.132 & 0.456 & 0.038 \end{pmatrix}$$



(a) Densité de probabilité (b) Fonction de répartition

Figure 9.7 – Résultats de l'échantillonnage des mesures d'exécution (observateur de Luenberger)

Nous donnons tableau 9.2, les valeurs d'exécution possibles pour certaines valeurs de p_i.

Niveau de confiance	Estimation du WCET
80%	3.9 ms
90%	3.9 ms
100%	5.0 ms

Tableau 9.2 – Valeur d'exécution pour le niveau non-critique selon p_i (observateur de Luenberger)

9.2.3 Filtre de Kalman

Présentation du programme

Nous allons modéliser un deuxième type de capteur logiciel, appelé filtre de Kalman, selon le modèle exposé dans Kalman (1960). De même que l'observateur précédent, le filtre de Kalman est un estimateur récursif. Cela signifie que pour estimer l'état courant, seuls l'état précédent et les mesures actuelles sont nécessaires. Soit le système linéaire discret bruité suivant (les bruits W et V sont considérés comme des bruits blancs de matrices de covariance Q et R) :

$$\begin{cases} X_{k+1} &= AX_k + BU_k + W_k \\ Y_k &= CX_k + V_k \end{cases} \tag{9.8}$$

Le filtre de Kalman comporte deux phases distinctes : la phase de prédiction et la phase de mise à jour. La phase de prédiction utilise l'état estimé de l'instant précédent pour produire

une estimation de l'état courant.

$$\begin{aligned} \hat{X}_{k|k-1} &= A\hat{X}_{k-1|k-1} + BU_{k-1} \\ P_{k|k-1} &= AP_{k-1|k-1}A^T + Q \end{aligned}$$

Dans l'étape de mise à jour, les observations de l'instant courant sont utilisées pour corriger l'état prédit dans le but d'obtenir une estimation plus précise.

$$\begin{aligned} K_k &= P_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1} \\ \hat{X}_{k|k} &= X_{k|k-1} + K_k\ (y_k - C\hat{x}_{k|k-1}) \\ P_{k|k} &= (I - K_kC)P_{k|k-1} \end{aligned}$$

Dans notre cas, nous allons considérer un système tel que la dimension du vecteur interne est 15, comportant une seule entrée et une seule sortie mesurées. Le calcul fait maintenant intervenir des multiplications de matrices (mais pas de calcul d'inverse, calcul de l'inverse d'un scalaire).

De même que pour l'exemple précédent, un ensemble de n valeurs pour X_k et y_k ont été générées et intégrées au programme.

Mesures du temps d'exécution

Une première implémentation du filtre de Kalman a été réalisée comportant une multiplication matricielle « classique ». Cependant, des problèmes de cache ont été rencontrés lors de la mise en place d'une application et des préemptions que cette tâche pouvait subir. La taille du cache étant seulement de 16Ko et la multiplication matricielle comporte de mauvaises localités spatiale et temporelle, des défauts de cache sont largement présents lors de préemptions.

Une méthode pour augmenter les localités est de réaliser les multiplications matricielles par blocs. Nous choisissons une taille par bloc de 3×3. L'implémentation du filtre de Kalman avec cette méthode de calcul a été réalisée puis exécutée sur cible. Les résultats obtenus sont présentés figure 9.8. Nous pouvons remarquer que le nombre de valeurs différentes relevées est beaucoup plus important que dans le cas précédent. Ce phénomène est dû à la complexité plus importante des calculs en nombres flottants. La valeur moyenne des temps d'exécution mesurés est d'environ 139 ms, avec un écart type de l'ordre de 3 ms, une valeur minimale d'exécution de 128 ms et maximale de 140 ms.

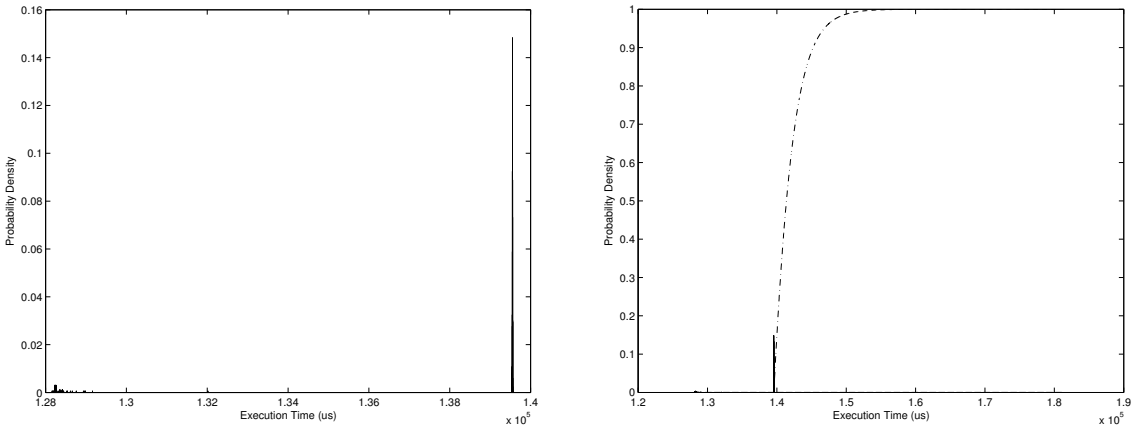
Analyse des résultats

La fonction θ proposée par Burns et Wellings (2001) est calculée et tracée. Le tableau 9.3 résume les valeurs d'estimation du WCET en fonction de quelques indices de confiance.

Un échantillonnage de la distribution obtenue par mesure a été réalisé. La figure 9.9 représente l'échantillonnage appliqué aux mesures.

L'échantillonnage des valeurs donne la distribution suivante :

$$\mathcal{C}_i = \begin{pmatrix} 128 & 129 & 139 & 140 \\ 0.052 & 0.020 & 0.011 & 0.917 \end{pmatrix}$$



(a) Mesures (b) Estimation WCET

Figure 9.8 – Résultats des mesures des valeurs d'exécution (filtre de Kalman)

Niveau de confiance	Estimation du WCET
90%	146 ms
99%	151 ms
99.9%	156 ms
99.99%	161 ms

Tableau 9.3 – Estimation du WCET en fonction de l'indice de confiance (filtre de Kalman)

Le tableau 9.4 résume les valeurs d'exécution considérées dans le choix des priorités au niveau non-critique pour certaines valeurs de p_i.

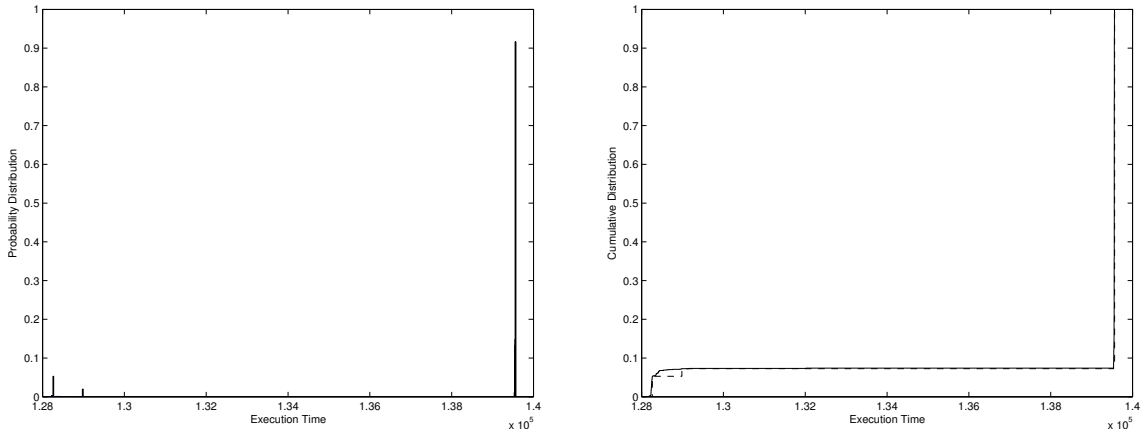
Niveau de confiance	Estimation du WCET
80%	140 ms
90%	140 ms
100%	140 ms

Tableau 9.4 – Valeur d'exécution pour le niveau non-critique selon p_i (filtre de Kalman)

9.2.4 Tâche dont le temps d'exécution suit une loi de Gumbel

Présentation du programme

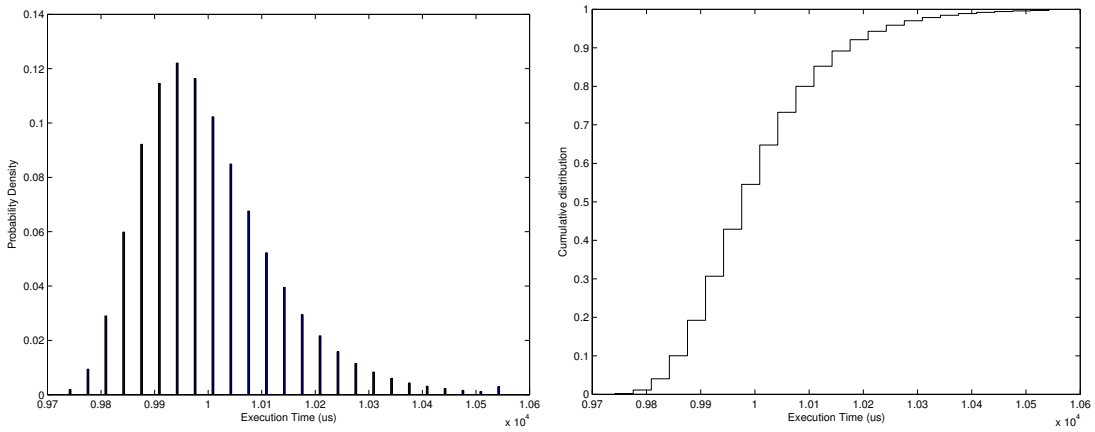
L'étude menée par Burns et Wellings (2001) suggère que le temps d'exécution d'une tâche peut être représenté par une distribution de Gumbel. Nous allons donc générer une tâche à plusieurs chemins d'exécution, chacun de ces chemins ayant une probabilité d'occurrence. Dans un premier temps, nous discrétisons la distribution de Gumbel afin de générer une distribution de référence pour le choix du chemin. Cette discrétisation est réalisée sur 5 points d'échantillonnage équi-répartis. La distribution de référence est présentée figure 9.10 (la moyenne de la



(a) Densité de probabilité (b) Fonction de répartition

Figure 9.9 – Résultats de l'échantillonnage des mesures d'exécution (filtre de Kalman)

distribution de référence est de 10000 unité de temps et son écart type de 230 unité de temps).



(a) Densité de probabilité (b) Fonction de répartition

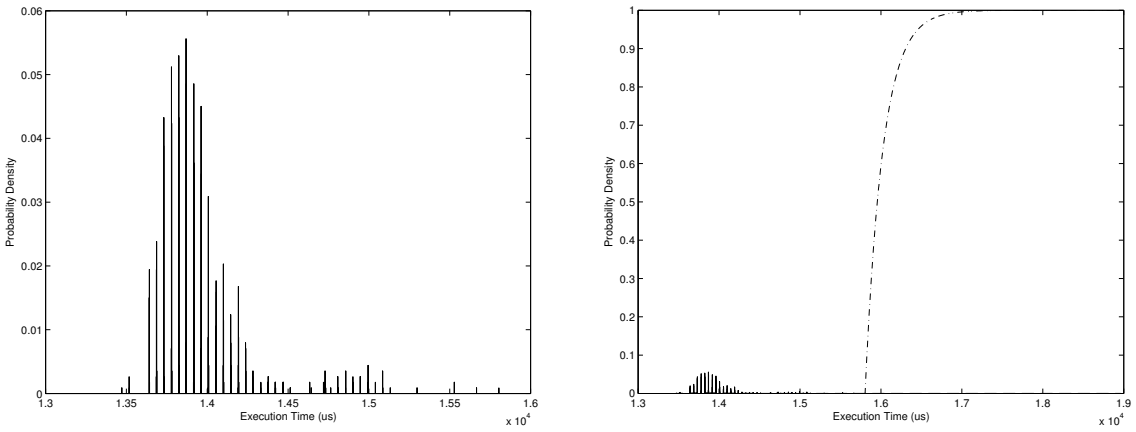
Figure 9.10 – Distribution de référence (Gumbel discrétisée)

À partir de cette distribution et d'un tirage aléatoire d'une valeur U (loi uniforme), une valeur d'exécution est choisie (à l'image du choix de l'échantillon présenté précédemment).

Un ensemble de n valeurs U ont été générées et intégrées au programme afin de choisir le chemin que doit suivre le programme à chaque exécution. Chaque chemin est modélisé par une boucle finie dont le nombre d'itération dépendant de la valeur du chemin (nous avons réglé la boucle telle que la valeur de chemin 1000 entraîne un temps d'exécution d'environ 1 milliseconde).

Mesures du temps d'exécution

Les résultats obtenus sont présentés figure 9.11. Nous pouvons remarquer que le nombre de valeurs différentes relevées est plus important que le nombre de chemins générés. Ceci est dû aux aléas générés par la plate-forme matérielle (dispersion autour de chaque chemin). La valeur moyenne des temps d'exécution mesurés est d'environ 14 ms, avec un écart type de l'ordre de 0.3 ms, une valeur minimale d'exécution de 13.5 ms et maximale de 15.8 ms.



(a) Mesures (b) Estimation WCET

Figure 9.11 – Résultats des mesures des valeurs d'exécution (loi de Gumbel discrétisée)

Analyse des résultats

La fonction θ proposée par Burns et Wellings (2001) est calculée et tracée. Le tableau 9.5 résume les valeurs d'estimation du WCET en fonction de quelques indices de confiance.

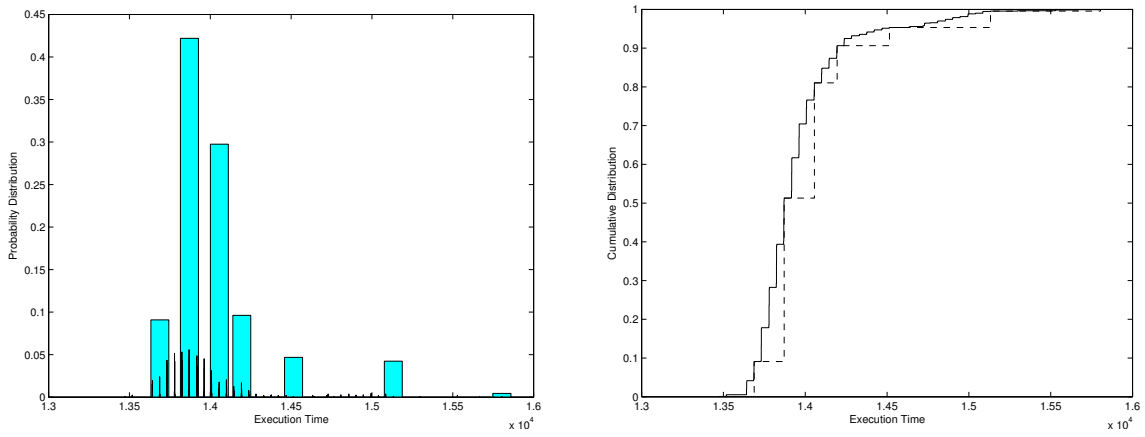
Niveau de confiance	Estimation du WCET
90%	16.3 ms
99%	16.8 ms
99.9%	17.3 ms
99.99%	17.8 ms

Tableau 9.5 – Estimation du WCET en fonction de l'indice de confiance (loi de Gumbel discrétisée)

La distribution de référence reste une bonne approximation pour l'échantillonnage. Cependant, le nombre de valeur sera réduit à 7 afin d'augmenter la rapidité des calculs d'analyse probabiliste. Nous ne gardons ainsi qu'un point sur deux de la distribution de référence. La figure 9.12 représente l'échantillonnage appliqué aux mesures.

L'échantillonnage des valeurs donne la distribution suivante :

$$\mathcal{C}_i = \begin{pmatrix} 13.7 & 13.9 & 14.0 & 14.2 & 14.5 & 15.1 & 15.8 \\ 0.091 & 0.422 & 0.297 & 0.096 & 0.047 & 0.042 & 0.005 \end{pmatrix}$$



(a) Densité de probabilité (b) Fonction de répartition

Figure 9.12 – Résultats de l'échantillonnage des mesures d'exécution (loi de Gumbel discrétisée)

Le tableau 9.6 résume les valeurs d'exécution considérées dans le choix des priorités au niveau non-critique pour certaines valeurs de p_i.

Niveau de confiance	Estimation du WCET
80%	14.0 ms
90%	14.2 ms
100%	15.8 ms

Tableau 9.6 – Valeur d'exécution pour le niveau non-critique selon p_i (loi de Gumbel discrétisée)

9.2.5 Tâche de type « machine à états »

Présentation du programme

La type de tâches modélisées dans cette section est une tâche comportant différents chemins d'exécution qui ont chacun une probabilité d'occurrence (type « machine à états »). Nous pouvons générer des distributions semblables à celles considérées dans simulations des études probabilistes chapitres 6 et 7.

Nous rappelons que l'algorithme de choix des valeurs d'exécution peut être choisi tels que :

$$\mathcal{C}_i = \begin{pmatrix} 17000 & 21000 & 22000 & 23000 & 26000 \\ p_1 & p_2 & p_3 & p_4 & p_5 \end{pmatrix}$$

avec

$$\begin{cases} \sum_{i=1}^{5} p_i = 1.00 \\ p_1 + p_5 = 0.20 \end{cases}$$

Ce choix permet de ne pas avoir de probabilité trop importante sur les valeurs extrêmes d'exécution (cas le plus fréquent). Un exemple de distribution générée par cette méthode est présentée figure 9.13.

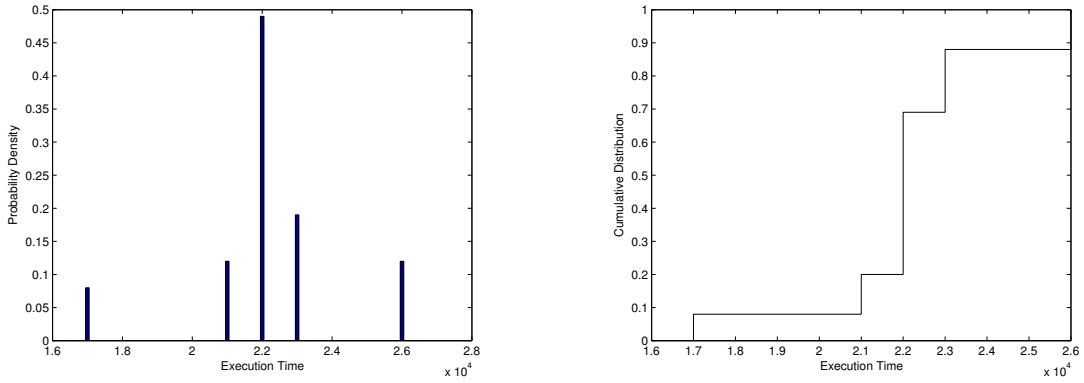


Figure 9.13 – Exemple d'une distribution de référence

La distribution générée pour l'exemple a pour moyenne 1107 unités de temps et pour écart type 108 unités de temps.

Mesures du temps d'exécution

Les résultats obtenus pour les temps d'exécution sont présentés figure 9.14. La valeur moyenne des temps d'exécution mesurés est d'environ 34 ms, avec un écart type de l'ordre de 3 ms, une valeur minimale d'exécution de 26 ms et maximale de 41 ms.

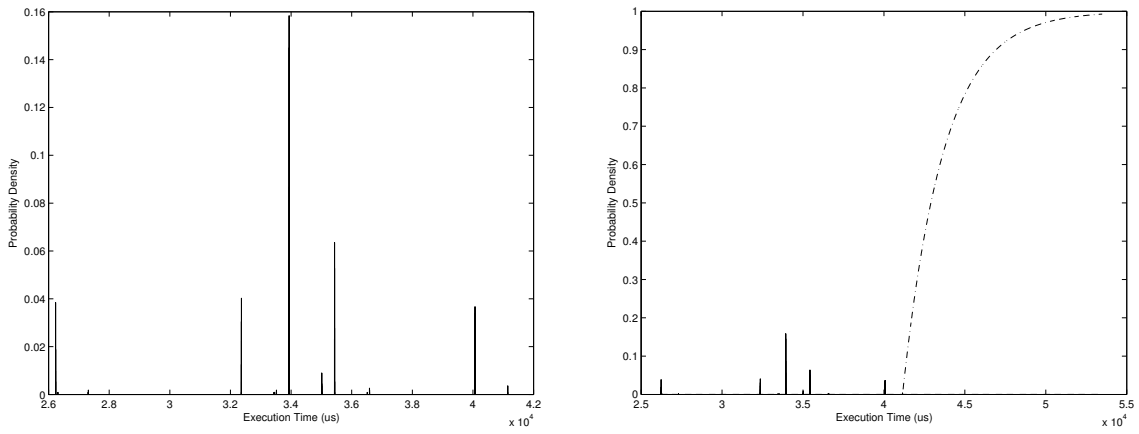
Analyse des résultats

La fonction θ proposée par Burns et Wellings (2001) est calculée et tracée. Le tableau 9.7 résume les valeurs d'estimation du WCET en fonction de quelques indices de confiance.

De même que pour le cas précédent, la distribution de référence reste une bonne approximation pour l'échantillonnage. La figure 9.15 représente l'échantillonnage appliqué aux mesures.

L'échantillonnage des valeurs donne la distribution suivante :

$$\mathcal{C}_i = \begin{pmatrix} 27.3 & 32.4 & 33.9 & 36.6 & 41.1 \\ 0.078 & 0.112 & 0.478 & 0.200 & 0.132 \end{pmatrix}$$



(a) Mesures (b) Estimation WCET

Figure 9.14 – Résultats des mesures des valeurs d'exécution (distribution aléatoire)

Niveau de confiance	Estimation du WCET
90%	47 ms
99%	53 ms
99.9%	59 ms
99.99%	64 ms

Tableau 9.7 – Estimation du WCET en fonction de l'indice de confiance (distribution aléatoire)

Le tableau 9.8 résume les valeurs d'exécution considérées dans le choix des priorités au niveau non-critique pour certaines valeurs de p_i.

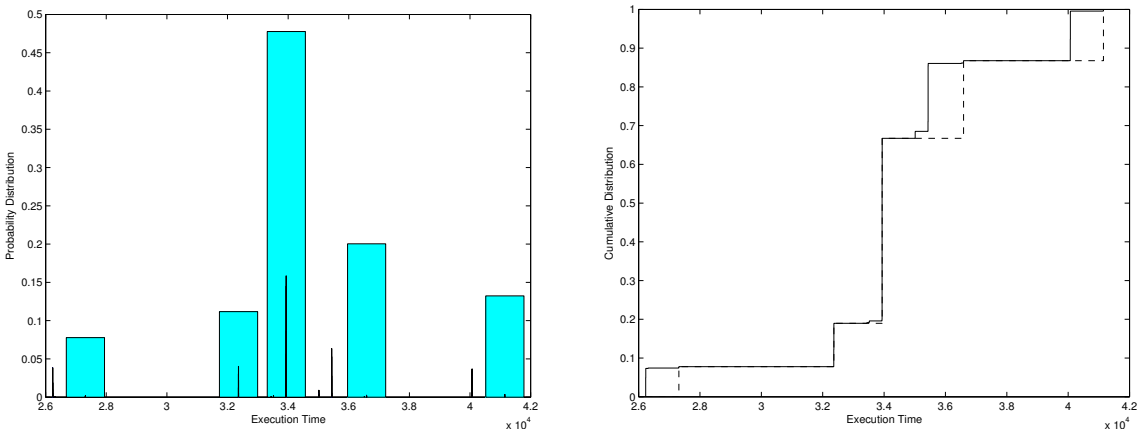
Niveau de confiance	Estimation du WCET
80%	35.4 ms
90%	40.1 ms
100%	41.1 ms

Tableau 9.8 – Valeur d'exécution pour le niveau non-critique selon p_i (distribution aléatoire)

Dans notre étude, nous nous limitons à des distributions de référence comportant uniquement 5 valeurs afin de faciliter l'étude et de réduire le temps de calcul des analyses probabilistes d'ordonnançabilité. Cependant, l'échantillonnage proposé section précédente réduit largement celui-ci et permet de résoudre des systèmes comportant des comportements complexes.

9.3 Discussion

Cette étude permet d'illustrer le comportement de quelques algorithmes (observateur de Luenberger, filtre de Kalman) et de définir d'autres comportements caractéristiques (distribution selon une loi de Gumbel ou de type « machine à états »).



(a) Densité de probabilité (b) Fonction de répartition

Figure 9.15 – Résultats de l'échantillonnage des mesures d'exécution (distribution aléatoire)

Les comportements obtenus permettent de nous renseigner sur les aléas matériels qui peuvent survenir notamment sur le calcul avec des nombres réels (observateur de Luenberger et filtre de Kalman). En effet, le processeur i.MXL (MC9328MXL) utilisé ne comporte pas de coprocesseur de calcul de nombres flottants. Les opérations sur des nombres réels sont donc réalisées de façon logicielle, ce qui augmente la dispersion du temps de calcul.

La modélisation par échantillonnage de la distribution obtenue par mesure dans le cas d'une étude probabiliste d'ordonnançabilité a été réalisée en n'utilisant que partiellement les résultats exposés par Refaat et Hladik (2010). En effet, nous exploitons l'équation 9.5 (version modifiée de l'équation 9.1) afin d'obtenir la distribution échantillonnée, mais la partie choix des échantillons a été réalisée manuellement. Nous n'avons pas implémenté l'échantillonnage dans le processus de calcul du temps de réponse d'une instance et nous avons donc dû restreindre notre étude à des distributions de temps d'exécution par tâche réduites. Refaat et Hladik (2010) considèrent un échantillonnage différent à chaque nouvelle instance convoluée dans le processus de calcul, ce qui permet de conserver une taille raisonnable. Pour notre étude, nous considérons un échantillonnage préalable global, identique pour chaque itération.

Nous avons donc à notre disposition les briques élémentaires pour construire une application composée d'un ensemble de tâches dont chaque paramètre est connu ou peut être estimé. Le prochain chapitre consiste à développer cette application.

« Le discours traduisant une expérience est souvent plus important que l'expérience elle-même. »

Les Thanatonautes - Bernard Werber

Chapitre 10

Étude d'une application

Sommaire

10.1 Présentation de l'application 143
10.2 Configurations du mécanisme de protection 146
10.2.1 Première configuration 146
10.2.2 Étude de sensibilité 148
10.2.3 Étude probabiliste 149
10.3 Discussion 151
10.3.1 Effets des préemptions sur le temps d'exécution 151
10.3.2 Sous-estimations et mécanisme de protection temporelle 152

Ce chapitre est consacré à l'expérimentation du mécanisme de protection temporelle sur une application exécutée sur la plate-forme matérielle d'exécution AFP9328.

Le chapitre s'articule comme suit. La section 1 permet de faire une présentation de l'application. Nous expérimentons, section 2, l'ensemble des configurations proposées dans le mémoire. Enfin, nous discutons en section 3, les résultats obtenus

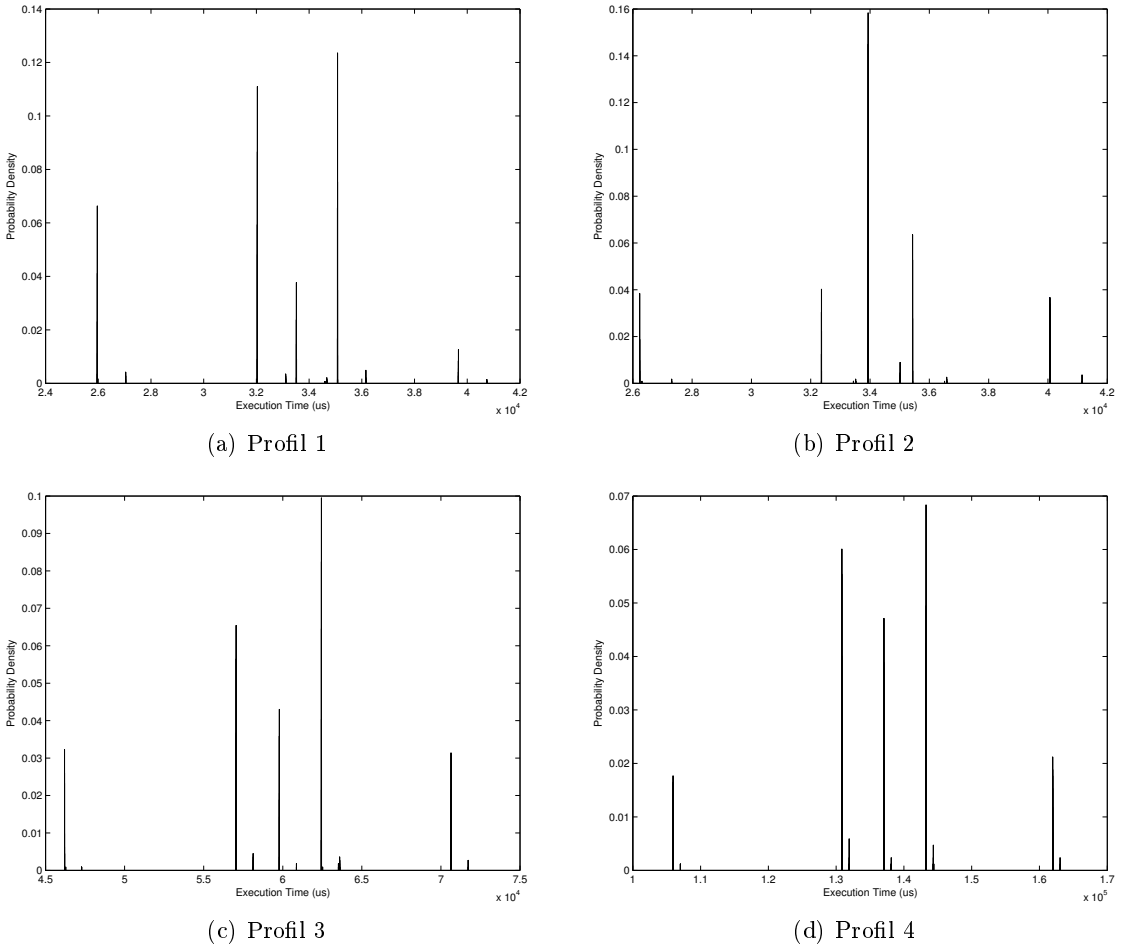
10.1 Présentation de l'application

Nous construisons une application multi-critique composée de 10 tâches périodiques indépendantes $\{\tau_1, \ldots, \tau_{10}\}$. Cette application comporte :

- 2 tâches de type « observateur de Luenberger » présenté section 9.2.2. Une des tâches est considérée critique, l'autre non critique ;
- 2 tâches de type « filtre de Kalman » présenté section 9.2.3. Une des tâches est considérée critique, l'autre non critique ;
- 2 tâches dont le temps d'exécution suit une loi Gumbel (section 9.2.4). Une des tâches est considérée critique, l'autre non critique ;

– 4 tâches de type « machine à états » dont le profil d'exécution est présenté section 9.2.5. Une des tâches est considéré critique, les 3 autres sont non critiques.

Les comportements temporels des six premières tâches ont été étudiés au chapitre précédent. Quatre profils d'exécutions de type « machine à états » (correspondant aux quatre dernières tâches) ont été générés selon les critères définit au chapitre précédent. Les mesures des temps d'exécution effectuées sur ces quatre profils après implémentation et exécution sur cible sont présentées figure 10.1. Nous rappelons que ces mesures sont réalisées en considérant la tâche isolée (pas de préemption possible). À partir de ces mesures, nous déterminons, de



(a) Profil 1 (b) Profil 2

(c) Profil 3 (d) Profil 4

Figure 10.1 – Mesures des temps d'exécution des profils générés

même qu'au chapitre précédent, pour chaque tâche les différents paramètres utiles à l'analyse. Le tableau 10.1 résume l'ensemble de ces paramètres.

Nous fixons les périodes T_i de chaque tâche afin d'obtenir une utilisation moyenne $\overline{U} = \bigotimes_{i=1}^{n} \mathcal{C}_i/T_i$ d'environs 90% ($\mathcal{C}_i$ représente la distribution échantillonnée dans le cas de tâches non critiques) et une utilisation maximale $U^{max} = \sum_{i=1}^{10} \widetilde{WCET}_i/T_i$ supérieure à 1. Les tâches sont considérées à échéances sur requêtes ($D_i = T_i$). Les priorités sont fixées selon la politique

	Valeurs mesurées (ms)				$\widetilde{WCET}_{x\%}$ (ms)			$C_{x\%}$ (ms)		
	min	max	moy	σ	90%	99%	99.9%	80%	90%	100%
Profil 1	26	41	33	3	43	53	59	35	35	41
Profil 2	26	41	33	3	43	53	59	35	40	41
Profil 3	46	72	60	6	83	94	105	62	71	72
Profil 4	106	163	137	13	188	212	236	143	162	163

Tableau 10.1 – Résumé des paramètres des 4 tâches de type « machine à états »

« Deadline Monotonic », politique optimale dans le cas déterministe (voir section 4.1.2) et selon l'algorithme proposé par Vestal (2007) (voir section 6.2.2, chapitre 6) dans le cas d'une étude probabiliste.

L'ensemble des paramètres de chaque tâche composant l'application est présenté tableau 10.2.

	Paramètres fixés			Paramètres issus de mesures			Priorités	
	Type	L_i	$T_i = D_i$	$\mathfrak{C}_i(l=0)$	$\mathfrak{C}_i(l=1)$	$\mathcal{C}_i$	DM	Vestal
τ_1	Observateur	1	50 ms	6 ms	6 ms	$\mathcal{C}_1$	10	10
τ_2	Observateur	0	100 ms	3.9 ms	6 ms	$\mathcal{C}_2$	8	7
τ_3	Kalman	1	1.5 s	156 ms	156 ms	$\mathcal{C}_3$	3	4
τ_4	Kalman	0	2 s	140 ms	156 ms	$\mathcal{C}_4$	1	2
τ_5	Gumbel	1	100 ms	16.8 ms	16.8 ms	$\mathcal{C}_5$	9	9
τ_6	Gumbel	0	200 ms	14 ms	16.8 ms	$\mathcal{C}_6$	7	6
τ_7	Profil 1	0	400 ms	35 ms	53 ms	$\mathcal{C}_7$	6	5
τ_8	Profil 2	1	500 ms	53 ms	53 ms	$\mathcal{C}_8$	5	8
τ_9	Profil 3	0	800 ms	62 ms	94 ms	$\mathcal{C}_9$	4	3
τ_{10}	Profil 4	0	1.5 s	143 ms	212 ms	$\mathcal{C}_{10}$	2	1

Tableau 10.2 – Les différents paramètres fixés ou estimés de l'application

Les distributions échantillonnées correspondantes pour chaque tâche τ_i sont :

$$\mathcal{C}_1 = \begin{pmatrix} 6 \\ 1.000 \end{pmatrix} \qquad \mathcal{C}_2 = \begin{pmatrix} 2.53 & 3.79 & 3.88 & 3.93 & 5.00 \\ 0.016 & 0.358 & 0.132 & 0.456 & 0.038 \end{pmatrix}$$

$$\mathcal{C}_3 = \begin{pmatrix} 178 \\ 1.000 \end{pmatrix} \qquad \mathcal{C}_4 = \begin{pmatrix} 161 & 166 & 170 & 171 \\ 0.046 & 0.003 & 0.919 & 0.032 \end{pmatrix}$$

$$\mathcal{C}_5 = \begin{pmatrix} 17 \\ 1.000 \end{pmatrix} \qquad \mathcal{C}_6 = \begin{pmatrix} 13.7 & 13.9 & 14.0 & 14.2 & 14.5 & 15.1 & 15.8 \\ 0.091 & 0.422 & 0.297 & 0.096 & 0.047 & 0.042 & 0.005 \end{pmatrix}$$

$$\mathcal{C}_7 = \begin{pmatrix} 27 & 32 & 33.5 & 35.1 & 40.7 \\ 0.157 & 0.323 & 0.135 & 0.337 & 0.048 \end{pmatrix} \qquad \mathcal{C}_8 = \begin{pmatrix} 53 \\ 1.000 \end{pmatrix}$$

$$\mathcal{C}_9 = \begin{pmatrix} 47.3 & 57.1 & 59.8 & 62.5 & 71.7 \\ 0.107 & 0.228 & 0.188 & 0.358 & 0.119 \end{pmatrix} \qquad \mathcal{C}_{10} = \begin{pmatrix} 107 & 132 & 138 & 144 & 163 \\ 0.093 & 0.271 & 0.230 & 0.295 & 0.111 \end{pmatrix}$$

Dans le cas d'une tâche critique, les valeurs de $\mathfrak{C}_i(l=0)$ et $\mathfrak{C}_i(l=1)$ sont définies telles que :

$$\mathfrak{C}_i(l=0) = \mathfrak{C}_i(l=1) = \widetilde{WCET}_{99\%}$$

La valeur $\widetilde{WCET}$ est obtenue en utilisant l'approche de Burns et Edgar (2000) avec une confiance sur l'estimation de 99%. Dans le cas d'une tâche non critique, les valeurs de $\mathfrak{C}_i(l=0)$

et $\mathfrak{C}_i(l=1)$ sont définies telles que :

$$\begin{cases} \mathfrak{C}_i(l=0) = C_{80\%},\ \mathbb{P}(\mathcal{C}_i \leq \overline{C}) \geq 0.80 \\ \mathfrak{C}_i(l=1) = \widetilde{WCET}_{99\%} \end{cases}$$

L'utilisation moyenne processeur $\overline{U}$ de cette application calculée à l'aide des distributions échantillonnées est de 94.3%. L'utilisation maximale U^{max} en considérant les valeurs $\mathfrak{C}_i(l=1)$ est de 111%.

Dans le cas d'une étude déterministe, le système considéré est le système $S = \{\mathfrak{C}_i(l = L_i), T_i, D_i, \pi_i\}$. Dans le cas d'une allocation des priorités selon l'algorithme « Deadline Monotonic », le système est ordonnançable. Dans le cas d'une étude déterministe, l'algorithme proposé par Vestal (2007) permet d'allouer les priorités du système tel qu'il soit ordonnançable (selon les critères multi-critiques proposés). Le choix particuliers des priorités proposé section 6.2.2 permet d'allouer les plus hautes priorités aux tâches les plus critiques. Le choix est donc différent de celui obtenu par la première méthode.

L'hyperpériode de l'application est égale au plus petit commun multiple des périodes (le système étant considéré synchrone). L'hyperpériode est donc égale à 12 secondes.

10.2 Configurations du mécanisme de protection

10.2.1 Première configuration

Nous allons dans un premier temps configuré le mécanisme de protection temporelle selon le schéma proposé au chapitre 4 :

$$\begin{cases} TIMEFRAME & = T_i \\ EXECUTION\_BUDGET & = C_i \end{cases}$$

Nous avons démontré que cette configuration n'offre pas une bonne précision du mécanisme dans le cas de système sous-dimensionné (ici $C_i = C_{80\%}$). Nous appliquons cette configuration et exécutons l'application sur la plate-forme selon les deux choix de priorités proposés.

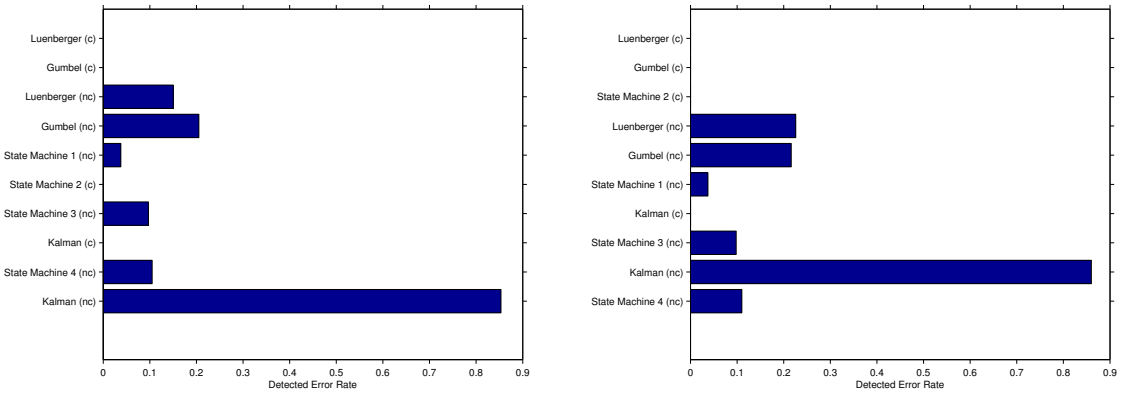
Le système a été exécuté sur une durée d'environ 20 minutes ce qui garantit un nombre significatif d'activation pour chaque tâche (100 hyperpériodes environ et plus de 500 activations pour la tâche de plus grande période). Le tableau 10.3 recense le nombre d'instances activées(ligne Activations), le nombre d'instances arrêtées (ligne erreurs détectées) et le nombre de défaillances (ligne échéances ratées).

La figure 10.2 représente les taux d'erreurs détectées (nombre d'erreurs détectées / nombre d'activations) pour les deux configurations de choix des priorités. Les tâches sont classées par ordre de priorités croissante (les tâches les plus prioritaires sont en haut).

Nous remarquons qu'aucune tâche critique ne dépasse son budget d'exécution. Les budgets sont donc correctement dimensionnés afin de ne pas interrompre leur exécution. Une tâche non

Tâche	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}
Activations	23967	11984	799	600	11984	5992	2996	2397	1498	799
Choix des priorités « Deadline Monotonic »										
Priorité	10	8	3	1	9	7	6	5	4	2
Erreurs détectées	0	1803	0	512	0	1230	113	0	145	84
Échéances ratées	0	0	0	0	0	0	0	0	0	0
Choix des priorités « Algorithme Vestal »										
Priorité	10	7	4	2	9	6	5	8	3	1
Erreurs détectées	0	2706	0	516	0	1296	112	0	147	88
Échéances ratées	0	0	0	0	0	0	0	0	0	0

Tableau 10.3 – Nombre d'activations et d'erreurs détectées (configuration « naïve »)



(a) Deadline Monotonic (b) Algorithme de Vestal

Figure 10.2 – Taux d'erreurs détectées (configuration « naïve »)

critique peut en théorie subir un taux d'erreurs détectées d'environs 20% (valeur de budget choisi telle que $C_i = C_{80\%}$). Toutes les tâches respectent ce critère excepté la tâche « Kalman (nc) » dont un nombre important d'instances ne respectent pas leur budget d'exécution (85% des instances environs sont tuées pour les deux choix de priorités).

La sous-estimation cette tâche apparaît non négligeable. La tâche « Kalman (nc) » fait partie des tâches le moins prioritaires. Elle est donc préemptée par un nombre important d'instances lors de son exécution. La figure 10.3 représente l'exécution d'une instance de la tâche « Kalman (nc) ».

La tâche « Kalman (nc) » est une tâche mettant en jeu des calculs matriciels. Même si les localités spatiales et temporelles des données dans le cache ont été améliorées par la mise en place d'une multiplication matricielle par blocs, à chaque préemption, une partie des données contenues dans le cache peut être effacée par l'exécution d'une tâche plus prioritaire (par exemple une des tâches de type « observateur de Luenberger » ou la tâche « Kalman (c) » plus prioritaire). Les temps d'exécution sont donc augmentés ce qui explique le nombre d'erreurs détectées. Une tâche faisant peu appel au cache comme les tâches de type « machine à états »

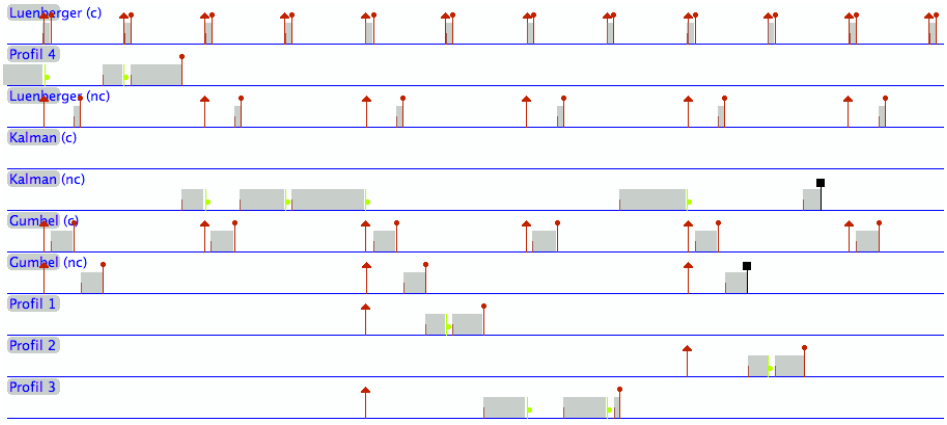


Figure 10.3 – Préemptions subies par une instance de la tâche « Kalman (nc) »

sont, elles, peu impactées par les préemption.

Une préemption par une tâche plus prioritaire peut supprimer des valeurs du cache utilisée avant préemption par la tâche préemptée. Lorsque la tâche préemptée s'exécute à nouveau, les valeurs doivent être rechargées dans le cache ce qui augmente considérablement le temps d'exécution. Les temps d'exécution mesurés lors d'une estimation en considérant la tâche isolée peut donc être ainsi très largement sous-estimés.

10.2.2 Étude de sensibilité

Nous allons dans cette section configurer le mécanisme de protection temporelle selon le schéma proposé au chapitre 5 :

$$\begin{cases} TIMEFRAME & = T_i \\ EXECUTION\_BUDGET & = (1 + \lambda.w_i).C_i \end{cases}$$

De même que précédemment, nous faisons les hypothèse qu'une tâche critique a une estimation sûre de son temps d'exécution (ce qui semble confirmé par les mesures effectuées ci-dessus). Une relaxation de budget n'est pas nécessaire : $w = 0$. Une tâche non critique peut avoir un pire temps d'exécution sous-estimé. Une relaxation des budgets est possible : $w = 1$.

L'étude de sensibilité appliquée aux deux systèmes (choix des priorités selon « Deadline Monotonic » et l'algorithme de Vestal) donne les mêmes résultats (cas particulier pour notre système). La relaxation des budgets des tâches non critique est d'environs 5%. Les budgets d'exécution alloués pour chaque tâche sont données tableau 10.4.

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}
C_i (ms)	6	3.9	156	140	16.8	14.0	35.0	53	62	143
B_i (ms)	6	4.1	156	147	16.8	14.7	36.7	53	65	150

Tableau 10.4 – Allocation des budgets (étude de sensibilité)

Nous appliquons cette configuration et exécutons l'application sur la plate-forme selon les deux choix de priorité proposés.

De même que pour la configuration précédente, le système est exécuté pendant environ 20 minutes. Le tableau 10.5 recense le nombre d'instances activées, le nombre d'instances arrêtées et le nombre de défaillances.

Tâche	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}
Activations	23967	11984	799	600	11984	5992	2996	2397	1498	799
Choix des priorités « Deadline Monotonic »										
Priorité	10	8	3	1	9	7	6	5	4	2
Erreurs détectées	0	0	0	0	0	0	37	0	56	7
Échéances ratées	0	0	0	0	0	0	0	0	0	0
Choix des priorités « Algorithme Vestal »										
Priorité	10	7	4	2	9	6	5	8	3	1
Erreurs détectées	0	933	0	1	0	31	33	0	56	2
Échéances ratées	0	0	0	0	0	0	0	0	0	0

Tableau 10.5 – Nombre d'activations et d'erreurs détectées (étude de sensibilité)

Le nombre d'erreurs détectées a largement diminué tout en garantissant qu'aucune instance ne s'exécute au delà de son échéance. Le choix d'allocation des priorités selon l'algorithme proposé par Vestal (2007) semble moins approprié puisqu'il ne permet pas de réduire aussi largement le nombre d'erreurs détectée sur la tâche « Luenberger (nc) ». Ce comportement est dû à la priorité moins importante de cette tâche dans cette configuration (la tâche « Profil 2 » plus critique est plus prioritaire) et donc un nombre de préemptions plus importants. Cependant, le but de l'algorithme dérivé de Vestal (2007) est de protéger les tâche critiques. Il peut donc aggraver la situation des pour les tâches non critiques.

La figure 10.4 représente les taux d'erreurs détectées (nombre d'erreurs détectées / nombre d'activations) pour les deux configurations de choix des priorités. Les tâches sont classées par ordre de priorités croissantes (les tâches les plus prioritaires sont en haut).

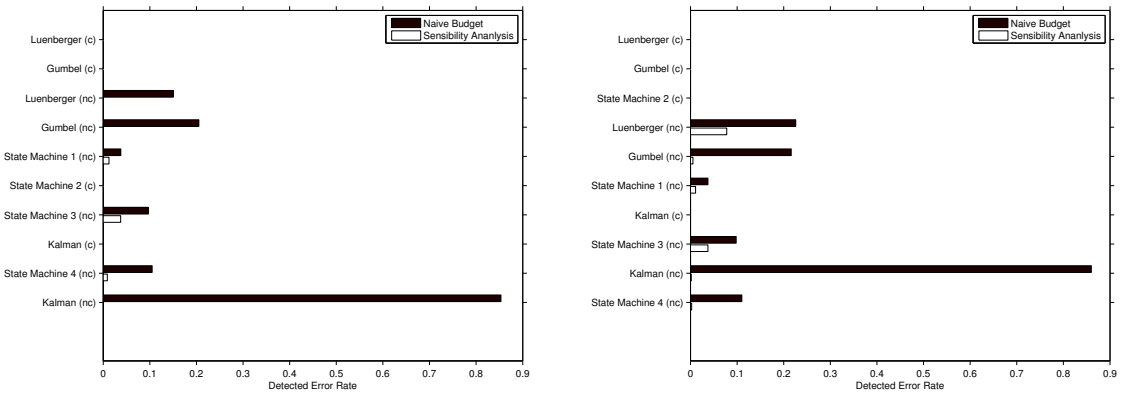
10.2.3 Étude probabiliste

Nous allons dans cette section configuré le mécanisme de protection temporelle selon le schéma proposé au chapitre 6 :

$$\begin{cases} TIMEFRAME & = T_i \\ EXECUTION\_BUDGET & = \mathfrak{C}_i(l=0) + \lambda.\big(\mathfrak{C}_i(l=1) - \mathfrak{C}_i(l=0)\big) \end{cases}$$

telle que

$$\underset{\lambda\in[0,1]}{\text{Argmin}} \, \|g(\lambda)\|, \quad g(\lambda) = f(\mathbf{B}), \quad \mathbf{B} = \big\lfloor \mathfrak{C}_i(l=0) + \lambda.\big(\mathfrak{C}_i(l=1) - \mathfrak{C}_i(l=0)\big) \big\rfloor$$



(a) Deadline Monotonic (b) Algorithme de Vestal

Figure 10.4 – Taux d'erreurs détectées (étude de sensibilité)

Les études réalisées au chapitre 6 ne supposait pas connu d'estimation du WCET. Afin de déterminer une valeur de $\mathfrak{C}_i(l=1)$ un facteur de sécurité de 1.25 était appliqué. La recherche de l'optimum était donc réalisé dans l'intervalle $[1, 1.25].\mathfrak{C}_i(l=0)$. Pour l'application considérée, nous supposons connu une estimation du WCET de chaque tâche, celle-ci étant obtenue par la méthode proposée par Burns et Edgar (2000). L'intervalle de recherche est donc modifié et devient $[\mathfrak{C}_i(l=0),\ \mathfrak{C}_i(l=1)] = [C_{80\%}, \widetilde{WCET}_{99\%}]$ pour les tâches non critiques. Les budgets des tâches critiques ne sont pas relaxés ($\mathfrak{C}_i(l=0) = \mathfrak{C}_i(l=1)$).

Le système considérée pour l'étude probabiliste est le système $S = \{(\mathcal{C}_i, \mathfrak{C}_i), T_i, D_i, \pi_i\}$. Nous considérons, de même qu'au chapitre 6 que la probabilité d'ordonnançabilité à respecter est de 100% pour les tâches critiques et de 99% pour les tâches non critiques. Dans le cas d'allocation selon « Deadline Monitoring », l'évaluation de l'ordonnançabilité du système par rapport au critère n'est pas respectée. Une optimisation du budget est donc à réalisé (B_i inférieur C_i^{max} mesuré). Dans le cas d'une allocation selon l'algorithme de Vestal modifié, l'évaluation par rapport au critère d'ordonnançabilité est respecté. Aucune optimisation n'est à réaliser et les budgets sont fixés aux valeurs maximales $\mathfrak{C}_i(l=1)$. Les budgets d'exécution alloués pour chaque tâche sont données tableau 10.6.

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}
C_i (ms)	6	3.9	156	140	16.8	14.0	35.0	53	62.0	143
Choix des priorités « Deadline Monotonic »										
B_i (ms)	6	4.8	156	147	16.8	15.3	43.2	53	76.6	174
Choix des priorités « Algorithme Vestal »										
B_i (ms)	6	6	156	156	16.8	16.8	53.0	53	94.0	212

Tableau 10.6 – Allocation des budgets (étude probabiliste)

Nous appliquons cette configuration et exécutons l'application sur la plate-forme selon les deux choix de priorités proposés.

De même que pour la configuration précédente, le système est exécuté sur une durée d'environ 20 minutes. Le tableau 10.7 recense le nombre d'instances activées, le nombre d'instances tuées et le nombre de défaillances.

Tâche	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}
Activations	23967	11984	799	600	11984	5992	2996	2397	1498	799
Choix des priorités « Deadline Monotonic »										
Priorité	10	8	3	1	9	7	6	5	4	2
Erreurs détectées	0	0	0	0	0	0	0	0	0	0
Échéances ratées	0	0	0	0	0	0	0	0	0	0
Choix des priorités « Algorithme Vestal »										
Priorité	10	7	4	2	9	6	5	8	3	1
Erreurs détectées	0	0	0	0	0	0	0	0	0	0
Échéances ratées	0	0	0	0	0	0	0	0	0	0

Tableau 10.7 – Nombre d'activations et d'erreurs détectées (étude probabiliste)

Dans les deux cas, le nombre d'erreurs détectées est nul pour chaque tâche composant l'application. Le nombre d'échéances ratées est nul, soulignons que c'est pourtant théoriquement possible. Cependant, la probabilité d'ordonnançabilité que doit respecter chaque instance est élevée et la période d'étude est restreinte. Le cas hypothétique dans lequel une instance de tâche n'est pas ordonnançable n'est donc pas survenu.

De même, le nombre d'erreurs détectées dans le cas d'une allocation « Deadline Monotonic » est nul. Le choix du budget d'exécution dans le cas de tâches non critiques est inférieur à la valeur maximale mesurée d'exécution. De plus, des sous-estimations dues aux préemptions peut être présentes. Cependant, toutes les valeurs mesurées ne sont pas présentes au cours de la simulation et ne représente qu'un échantillon des valeurs. Le cas hypothétique dans lequel une instance de tâche ne satisfait pas son budget n'est donc pas survenu.

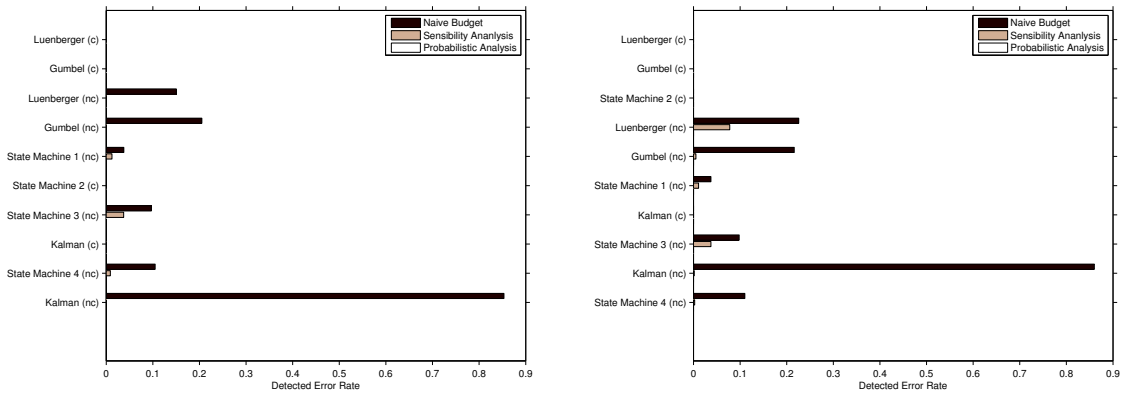
La figure 10.5 résume l'ensemble des taux d'erreurs détectées pour les deux configurations de choix des priorités et les différentes configurations de budgets étudiées.

10.3 Discussion

10.3.1 Effets des préemptions sur le temps d'exécution

Nous profitons des résultats obtenus sur la dernières configuration afin d'étudier les temps d'exécution des tâches à l'intérieure de l'application, sans perturbation du mécanisme de protection (aucune erreurs détectés).

Comme nous l'avons fait remarquer, les préemptions peuvent augmenter le temps d'exécution. Cette modification est due principalement aux délais des préemptions relatifs au cache (Cache Related Preemption Delay). Dans notre cas, ce phénomène est visible sur les tâches de type « observateur de Luenberger » et de type « filtre de Kalman » qui manipulent un grand



(a) Deadline Monotonic (b) Algorithme de Vestal

Figure 10.5 – Taux d'erreurs détectées en fonction des configurations de budgets d'exécution

nombre de données. Moins une tâche est prioritaire, plus elle est préemptée et plus elle peut être soumise à des modifications de son temps d'exécution.

Au cours de l'exécution de l'application, les différents événements d'ordonnancement (activation, préemption, terminaison) sont enregistrés. Nous pouvons donc déterminer les temps d'exécution des instances de chaque tâche dans l'application. Les résultats de mesures sont fournis dans le cas d'une allocation des priorités selon l'algorithme « Deadline monotonic », figures 10.6,10.7 et 10.8.

Nous pouvons remarquer que les tâches de types « filtre de Kalman » sont particulièrement touchée par l'effet des préemptions. Le temps maximum d'exécution fourni au chapitre précédent de 140 ms est dépassé et l'estimation de la valeur maximale serait aux alentours de 148 ms soit une sous-estimation d'environ 5%. Les tâches de type « observateur de Luenberger » sont peu modifiées puisqu'elles occupent des priorités élevées.

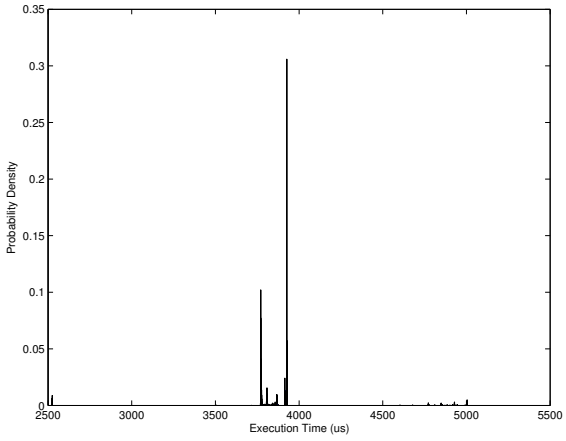
Les autres tâches sont beaucoup moins modifiées. Par contre, nous pouvons observer que plus une tâche est préemptée, plus la répartition des temps d'exécution est dispersée.

10.3.2 Sous-estimations et mécanisme de protection temporelle

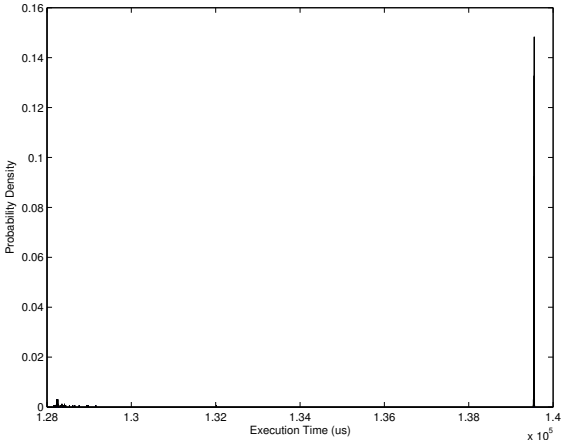
L'étude sur une plate-forme réelle permet de confirmer les résultats obtenus par simulation au cours des chapitres précédents.

Premièrement, nous vérifions qu'une sous-estimation des temps d'exécution est possible. Cette sous-estimation est due, dans le cas de notre étude, à une non prise en compte des interactions indirectes avec les autres tâches du système lors de la phase de modélisation.

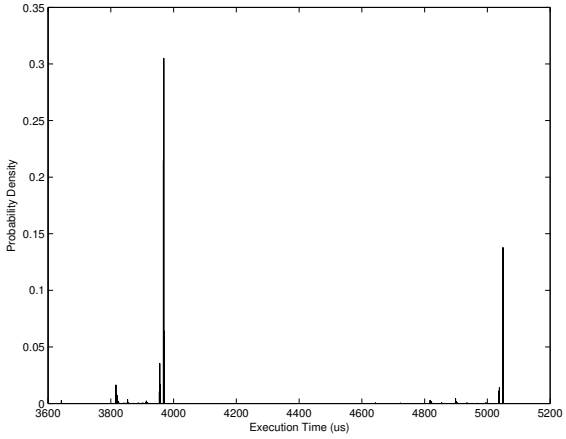
Deuxièmement, nous vérifions que les techniques de configurations proposées dans ce mémoire permettent de dimensionner au mieux les budgets en fonction de l'effort investi dans l'estimation du WCET. Nous remarquons à travers cette étude de cas que les relaxations proposées permettent d'améliorer la précision du mécanisme de protection en conservant une



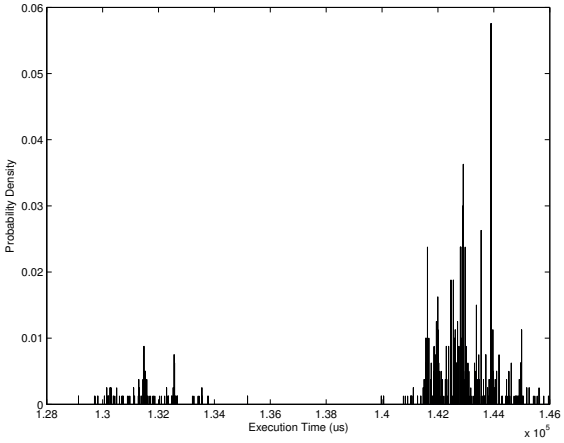
(a) Luenberger seul



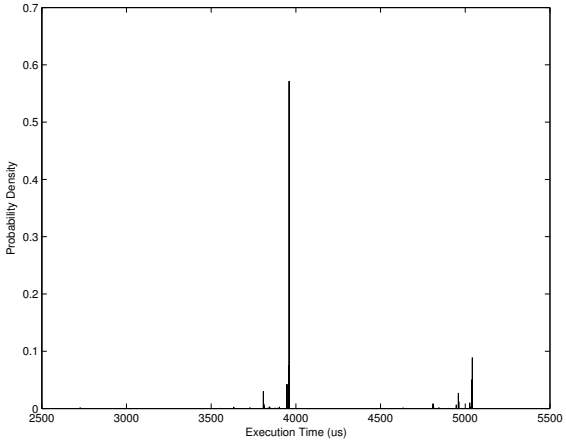
(b) Kalman seul



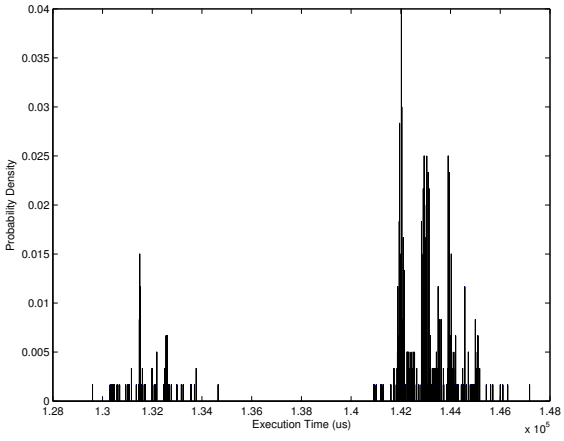
(c) Luenberger (c) - $\pi_i = 10$



(d) Kalman (c) - $\pi_i = 3$

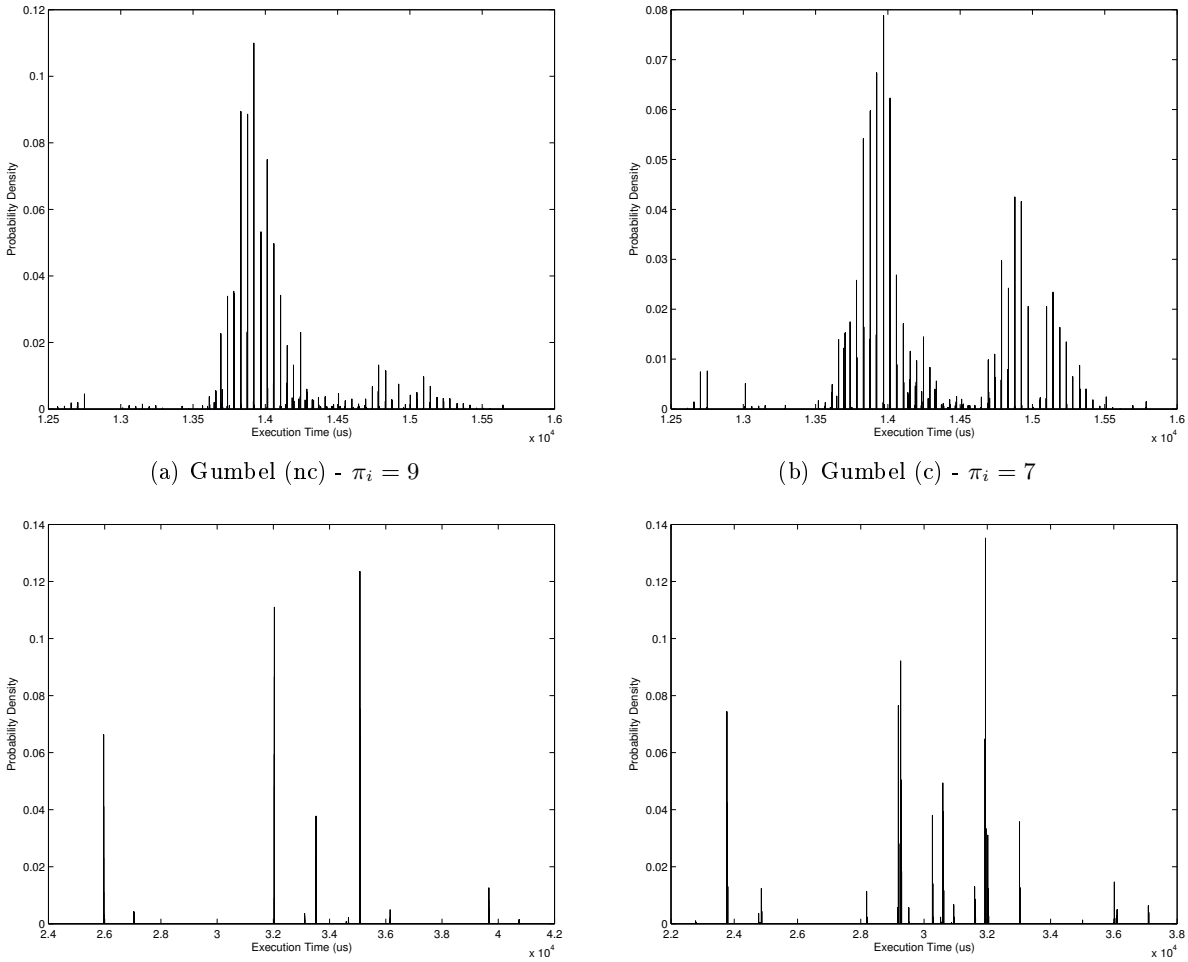


(e) Luenberger (nc) - $\pi_i = 8$



(f) Kalman (nc) - $\pi_i = 1$

Figure 10.6 – Mesures des temps d'exécution (Observateur + filtre de Kalman)



(a) Gumbel (nc) - $\pi_i = 9$

(b) Gumbel (c) - $\pi_i = 7$

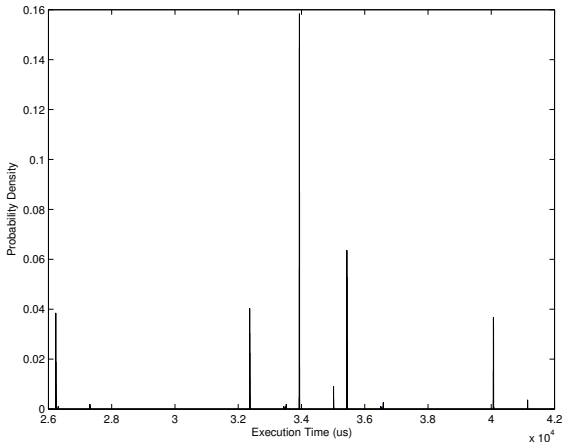
(c) Profil 1 - seul

(d) Profil 1 (nc) - $\pi_i = 6$

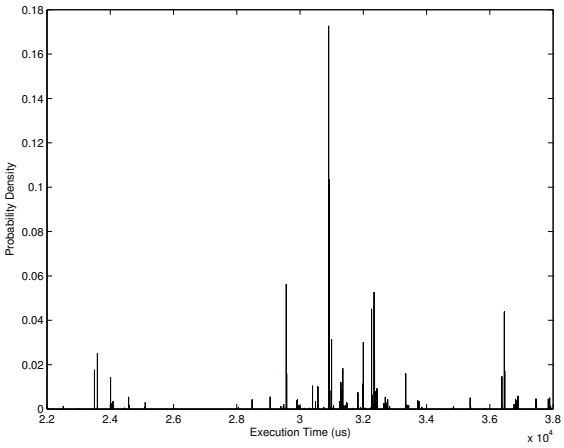
Figure 10.7 – Mesures des temps d'exécution (Gumbel + Machine à États 1)

garantie sûre de fonctionnement.

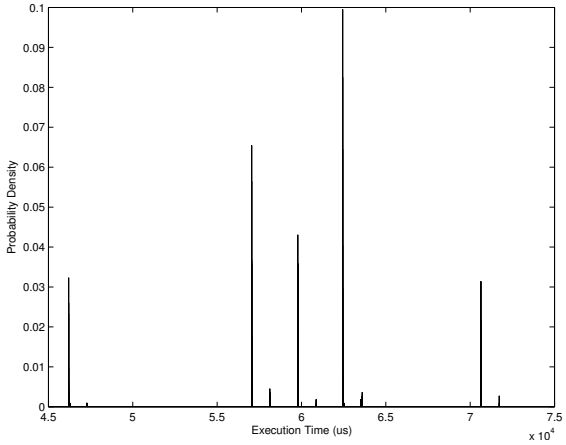
Cependant, il faut faire attention à la précision du mécanisme à l'exécution. En effet, les temps système ne sont pas nuls (de l'ordre de 150 μs, voir chapitre 8) et des marges de sécurité doivent être prises afin d'empêcher toute défaillance. Dans notre cas d'étude, les temps d'exécution des tâches sont importants et les budgets ont été arrondis à la milliseconde inférieure. Nous n'observons aucune défaillance au cours de l'exécution. Cependant, dans la pratique, le concepteur devra prendre en compte les temps système et se prémunir en conséquence.



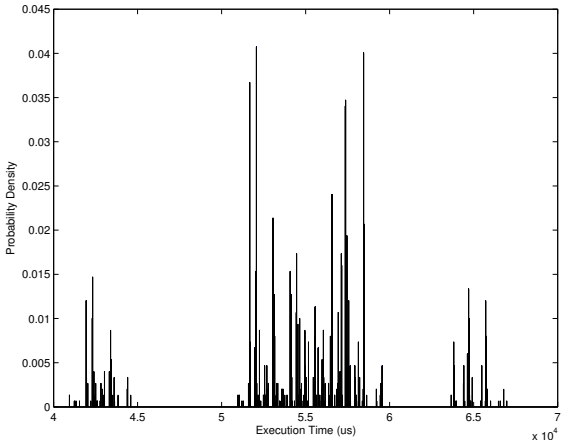
(a) Profil 2 - seul



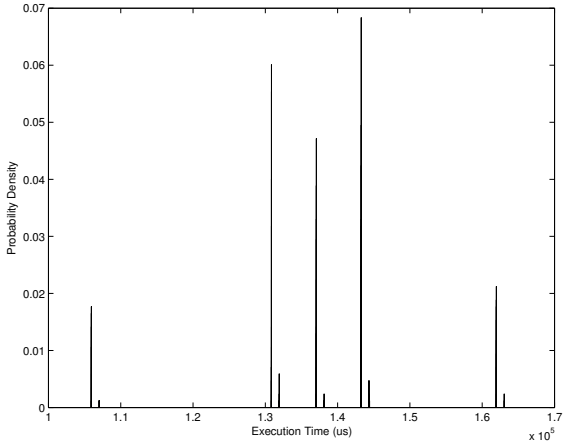
(b) Profil 2 (c) - $\pi_i = 5$



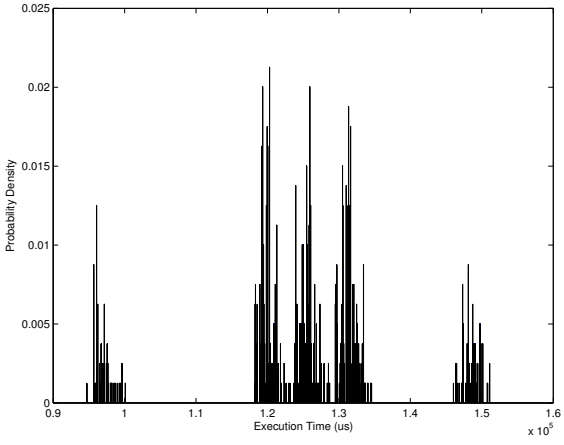
(c) Profil 3 - seul



(d) Profil 3 (nc) - $\pi_i = 4$



(e) Profil 4 - seul



(f) Profil 4 (nc) - $\pi_i = 2$

Figure 10.8 – Mesures des temps d'exécution (Machines à États 2-4)

« Le danger qui menace les chercheurs aujourd'hui serait de conclure qu'il n'y a plus rien à découvrir. »

La Recherche passionnément - Pierre Joliot

Conclusion

Les systèmes informatiques embarqués ont percé dans des domaines très variés. Depuis quelques années, le domaine de l'automobile connaît de grands changements en utilisant ces technologies afin d'améliorer la sécurité, le confort ou la consommation. Aujourd'hui, de multiples composants informatisés sont développés et intégrés aux automobiles. Ces composants doivent offrir un niveau de sûreté de fonctionnement en rapport avec la criticité des services qu'ils délivrent. De plus, des composants peu critiques ne doivent pas engendrer de défaillances par propagation sur des composants plus critiques. Une isolation des composants est donc nécessaire. Dans ce contexte, notre travail porte sur la robustesse (sûreté de fonctionnement vis-à-vis des fautes externes) temporelle des systèmes temps réel embarqués.

Le domaine automobile est relativement standardisé et un des standards émergents, le standard AUTOSAR, spécifie l'architecture des systèmes informatiques embarqués dans les automobiles afin de faciliter le développement de systèmes multi-concepteurs, de permettre la réutilisation de composants (standards) sur étagères et d'accroître la flexibilité de tout le processus de développement. Dans notre étude, nous nous intéressons plus particulièrement au mécanisme de protection temporelle proposée par le composant standard AUTOSAR OS. Ce mécanisme permet d'isoler chaque fonction (tâche) de l'application en détectant une erreur avant qu'une défaillance (échéance ratée) ne se produise.

Notre étude porte sur le dimensionnement du mécanisme et plus particulièrement la partie contrôlant le temps maximal d'exécution de chaque fonction. Le modèle d'application considéré est un ensemble de n tâches périodiques indépendantes, ordonnancées sur un unique processeur. La politique d'ordonnancement est préemptive à priorités fixes. Les fautes considérées sont les sous-estimations des temps d'exécution lors de la phase de conception. Cette sous-estimation peut être involontaire (cas d'un outil d'estimation défaillant), ou délibérée (cas des tâches non critiques pour lesquelles une analyse sûre peut s'avérer trop coûteuse en pratique).

Quatre propositions de configuration du mécanisme de protection temporelle ont été proposées. La première consiste à fixer le budget d'exécution à la valeur maximale d'exécution estimée d'une tâche. Nous avons montré qu'une sous-estimation des temps d'exécution peut entraîner un comportement non acceptable de l'application avec cette configuration. En effet, le nombre d'erreurs détectées par le mécanisme, et donc le nombre d'instance avortées, peut être très important comparé au nombre d'échéances ratées sans recours à un mécanisme de

protection. Nous avons alors proposé une seconde approche permettant de relaxer les budgets d'exécution tout en garantissant la robustesse de l'ordonnancement. Cette approche est basée sur une étude de sensibilité (Bini *et al.*, 2006) qui permet de placer le système à ses limites d'ordonnançabilité. L'approche proposée permet également de prendre en compte le niveau de criticité des tâches en contrôlant la direction de relaxation des budgets (les tâches critiques ne sont pas relaxées puisque nous supposons que nous connaissons une valeur sûre de leur pire temps d'exécution). La précision du mécanisme de protection est largement améliorée par cette approche.

Nous avons proposé, dans un second temps, d'appuyer nos recherches sur un modèle d'application probabiliste où les temps d'exécution sont représentés par des variables aléatoires. Cette modélisation permet de mieux prendre en considération les méthodes d'estimation des temps d'exécution des tâches, notamment des tâches non critiques, dont les estimations sont généralement réalisées par des campagne de tests. La méthode proposée s'appuie sur une garantie d'ordonnançabilité probabiliste des tâches non critiques (Diaz *et al.*, 2002, 2004) tout en conservant une ordonnançabilité sûre des tâches critiques. À partir de ce nouveau modèle, des budgets d'exécution plus relaxés peuvent être obtenus pour les tâches non critiques. Pour cela, nous proposons d'utiliser une technique classique d'optimisation mathématique. L'analyse probabiliste sur laquelle nous nous basons pour ces travaux souffre de différents soucis sur le plan calculatoire (complexité et sûreté des calculs) et conceptuel (possibilité d'occurrence de défaillance mal contrôlée dans le système). Nous proposons donc une quatrième méthode permettant de palier ces différents problèmes en utilisant en ligne un mécanisme de surveillance d'échéance. Ces différentes configurations ont été testées par simulation sur un certain nombre de systèmes. Les résultats montrent que, pour des systèmes surchargés, les techniques basées sur une analyse probabiliste permettent de relaxer les budgets trois à quatre fois plus que les techniques déterministes.

Une étude de cas vient compléter le mémoire. Celle-ci a été réalisée sur la plate-forme matérielle AFP9328 d'Armadeus System (micro-contrôleur de type ARM9), en utilisant le système d'exploitation temps réel Trampoline. Un premier travail a été de réaliser l'implémentation des mécanismes pour la cible utilisée. Nous avons ensuite créer des programmes représentatifs de différents profils d'exécution. Ces programmes ont alors été utilisés afin de construire une application de démonstration. Enfin, une évaluation des techniques proposées a été réalisée. Les résultats obtenus sont conforme aux résultats observés par simulation.

Perspectives. Dans toute notre étude, nous considérons un modèle d'application composée uniquement de tâches périodiques indépendantes. Une première évolution serait d'étendre le modèle à des tâches faisant appel à des ressources partagées. Il faudrait, dans ce cas, prendre en compte le temps de blocage maximal qu'une tâche ou instance de tâche peut subir. Des étude d'ordonnançabilité prenant en compte ce type de modèle ont été proposées, aussi bien pour une étude déterministe (Audsley *et al.*, 1993) que probabiliste (López *et al.*, 2008). Dans le cas probabiliste, le temps de blocage peut être représenté par une unique valeur correspondante à la valeur maximale de blocage ou à une variable aléatoire $\mathcal{B}$, ce qui est une extension peu compliquée de nos travaux.

Dans le cas d'une étude probabiliste d'ordonnançabilité (Diaz *et al.*, 2004; López *et al.*, 2008), nous avons montré que le calcul du steady-state backlog souffre de différents soucis sur le plan calculatoire et conceptuel. Nous avons proposé un élément de réponse par la mise en place d'un mécanisme de surveillance des échéances. Cependant, nous avons montré que le

calcul ne fournit plus dans ce cas une valeur exacte d'ordonnançabilité. Même si cette méthode permet d'obtenir une borne inférieure d'ordonnançabilité, une étude de la précision de la borne obtenue semble indispensable. Dans un premier temps, celle-ci peut être réalisée en effectuant une étude statistique du comportement des système utilisant le mécanisme. Il est envisageable de chercher à obtenir une quantification analytique de cette borne.

L'étude de cas a illustré le rôle des temps systèmes dans la mise en œuvre concrète du mécanisme de protection temporelle. Nous avons montré que la précision atteignable en pratique est limitée et que les temps systèmes modifient le comportement du système. Si l'on ne tient pas compte de ces facteurs pratiques dans la mise en place des budgets sur un système réel, il est possible que des défaillances non contrôlées se produisent. Une solution envisageable est de prendre en compte les temps systèmes dans la phase modélisation de l'application. Les études menées par Meumeu Yomsi et Sorel (2007) peuvent apportées des éléments de réponse.

Enfin, le dernier point concerne la résolution des difficultés de mise en œuvre de l'approche reposant sur une analyse probabiliste. La motivation de cette approche est de permettre la prise en compte d'évaluations non sûres mais précises des temps d'exécution, dans le but de maximiser l'utilisation des ressources matérielles. Étant donnée un système dont les temps d'exécution sont donnés sous forme de variables aléatoires obtenues par tests sur cible, notre approche permet de calculer des priorités et des budgets pour permettre l'exécution robuste du système. En exécutant le système ainsi configuré, on obtient de nouvelles distributions des temps d'exécution, tenant compte des effets des temps systèmes et surtout des délais liés aux conséquences des préemptions sur le contenu des mémoires caches. Il peut alors être nécessaire de chercher une nouvelle configuration du système, à partir de ces nouvelles distributions. En théorie, il n'est pas garanti que ce cycle se stabilise rapidement. En pratique, nous ne disposons pas de suffisamment de résultats pour généraliser ce que nous avons observé sur notre étude de cas. Quoi qu'il en soit, il est souhaitable de mettre en œuvre des moyens pour accélérer cette stabilisation. Il nous semble en particulier nécessaire de se diriger vers la prise en compte des délais liés aux effets des préemptions sur les mémoires caches, à l'image de ce qui est évoqué ci-dessus concernant les temps système. Des études ont été réalisées sur la détermination de ces effets pour certains types de cache (Burguière *et al.*, 2009). Elles pourraient être utilisées pour avancer dans cette direction.

Annexe A

Simulateur de système temps-réel

Afin de simuler le comportement d'un système temps réel, nous avons choisi d'utiliser et de modifier le simulateur TrueTime (version 1.5), boîte à outil de Matlab/Simulink.

A.1 Le simulateur TrueTime

TrueTime est une boîte à outil de Matlab/Simulink qui facilite la simulation commune d'un noyau d'ordonnancement temps réel, des transmissions par réseaux et des dynamiques continues d'appareil. Il a été développé à l'université de Lund (Suède) par le département "Automatik Control". Le manuel de référence (Ohlin *et al.*, 2007) peut être téléchargé sur le site Web de l'université.

Dans notre étude, nous nous focalisons sur l'implémentation du noyau d'une application. La principale structure de données du noyau d'application proposée par TrueTime est une classe C++, appelée *RTsys*. Une instance de cette classe est créée à l'initialisation de la S-fonction Simulink. La figure A.1 présente les principaux attributs et fonctions de la classe *RTsys*.

Les listes *readyQ* et *timeQ* sont des listes ordonnées. Les éléments de la liste *timeQ* sont des tâches ou des horloges (timers) attendant leur activation. Ils sont ordonnés selon leur date d'activation. Une horloge dans la liste *timeQ* est représentée par son "handler" correspondant. Les tâches insérées dans la liste *readyQ* sont ordonnées par niveaux de priorité. La politique d'ordonnancement peut être à priorités fixes ou dynamiques et la fonction de priorité retourne la priorité courante selon la politique choisie.

Les tâches peuvent être périodiques ou apériodiques. Les tâches apériodiques sont activées par la création d'une instance. Toutes les instances en attente sont insérées dans une file d'attente d'une tâche par ordre d'instant d'activation. Pour une tâche périodique, une horloge interne est créée qui permet d'activer la tâche à intervalles réguliers. La figure A.1 donne les principaux attributs et fonction de la classe tâche (*UserTask*).

Une tâche peut être caractérisée par son échéance relative (D), sa priorité (π), son pire-temps d'exécution (C) et sa période (T). Ces attributs sont considérés constants durant la

```
classRTsys {
 public:
  double time;   // Current time in simulation

  Task* running; // Currently running task
  List* readyQ;  // usertasks and handlers ready for execution
  List* timeQ;   // usertasks and timers waiting for release

  List* taskList;  // A list containing all created tasks

  double (*prioFcn)(Task*); // Priority function
};
```

Figure A.1 – Classe *RTsys*

simulation. À ces attributs s'ajoute tous les attributs dynamiques de la tâche : son échéance absolue (d), sa prochaine date d'activation (r), son temps d'exécution (b) et le temps restant d'exécution (f). Il est possible d'associer deux procédures d'interruption (Interrupt Handler) à chaque tâche : une procédure de violation d'échéance (activée si l'exécution d'une instance d'une tâche dépasse son échéance) et la procédure de violation de temps d'exécution (activée si l'exécution d'une instance est supérieure à la valeur maximale donnée par le pire-temps d'exécution). Nous reviendrons plus tard sur ces deux procédures d'interruption.

Il est possible d'attacher des programmes à chaque activation de l'OS (hooks) pour chaque tâche. Ces programmes sont exécutés à chaque étape durant la simulation. la figure A.3 représente l'ensemble des points d'appel possible à ces programmes durant l'exécution d'une instance.

La date d'arrivée et d'activation d'une instance sont les mêmes sauf dans le cas où un résidu d'exécution est présent lors de la prochaine activation. Dans ce cas, le programme "arrival hook" est exécuté immédiatement et le programme "release hook" est appelé lorsque l'instance est réellement relâchée de la liste d'attente.

La fonction du noyau est de manipuler les différentes structures de données (*readyQ* et *timeQ*) et est appelée à chaque date appropriée pendant la simulation. Cette fonction peut être modélisée comme suit :

- Mettre à jour le temps d'exécution restant de l'instance en cours et analyser si celle-ci a terminée son exécution ;
- Analyser la liste *timeQ* afin d'activer les éléments possibles (tâches ou horloges) ;
- Déterminer la tâche ayant la plus haute priorité et faire de celle-ci la tâche prochainement en exécution ;
- Déterminer la prochain instant d'appel à l'OS (fonction du noyau).

Prenons un exemple pour illustrer le fonctionnement de la fonction du noyau et des évolutions des diférentes listes. Le système considéré est le système présenté tableau A.1.

Lors de l'initialisation, trois instances de tâche sont créées. Comme ces trois tâches sont

```
class UserTask : public Task {
public:
  double priority;
  double wcExecTime;   // Worst-Case Execution Time of the task
  double deadline;
  double absDeadline;
  double release;      // task release time if in timeQ
  double budget;

  int state; // Task state (IDLE;WAITING;SLEEPING;READY;RUNNING)

  double tempPrio;  // temporarily raised prio value

  List *pending; // list of pending jobs

  InterruptHandler* deadlineORhandler; // deadline overrun handler
  InterruptHandler* exectimeORhandler; // exec-time overrun handler

  void (*arrival_hook)(UserTask*);     // hooks
  void (*release_hook)(UserTask*);
  void (*start_hook)(UserTask*);
  void (*suspend_hook)(UserTask*);
  void (*resume_hook)(UserTask*);
  void (*finish_hook)(UserTask*);
};
```

Figure A.2 – Classe *UserTask*

périodiques, trois "handler" attachés à trois horloges sont également créés. Pour chaque tâche, une instance est créée et la procédure d'interruption correspondante est insérée dans la liste *timeQ*. Considérons aucun offset pour toutes les tâches (système synchrone), toutes les dates de réveil des procédures d'interruption sont égales à zéro. À la date zéro, la fonction du noyau est donc appelée et trois procédures d'interruption de la liste *timeQ* sont analysées pour activation, de même que celles-ci ainsi que leur tâche correspondante sont insérées dans la liste *readyQ*. De plus, trois nouvelles procédures d'interruptions sont insérées dans la liste *timeQ* correspondant aux prochaines instances (dates de réveil respectivement égales à T_A, T_B et T_C). les trois procédures d'interruption sont exécutées et la tâche taskA devient la tâche en cours d'exécution. L'évolution des listes *timeQ* et *readyQ* est représentée figure A.4.

Tâche ID	Priorité	Temps d'exécution	Échéance = Période
TaskA	Haute	1	5
TaskB	Moyenne	3	10
TaskC	Basse	5	15

Tableau A.1 – Système considéré

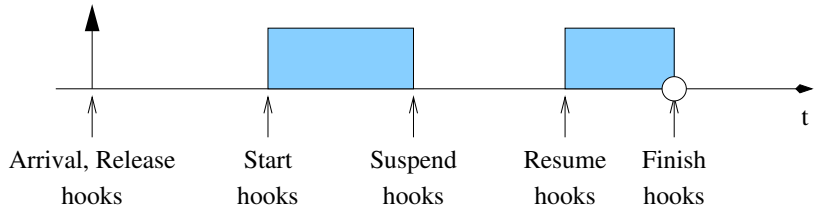


Figure A.3 – Appels de programmes (hook)

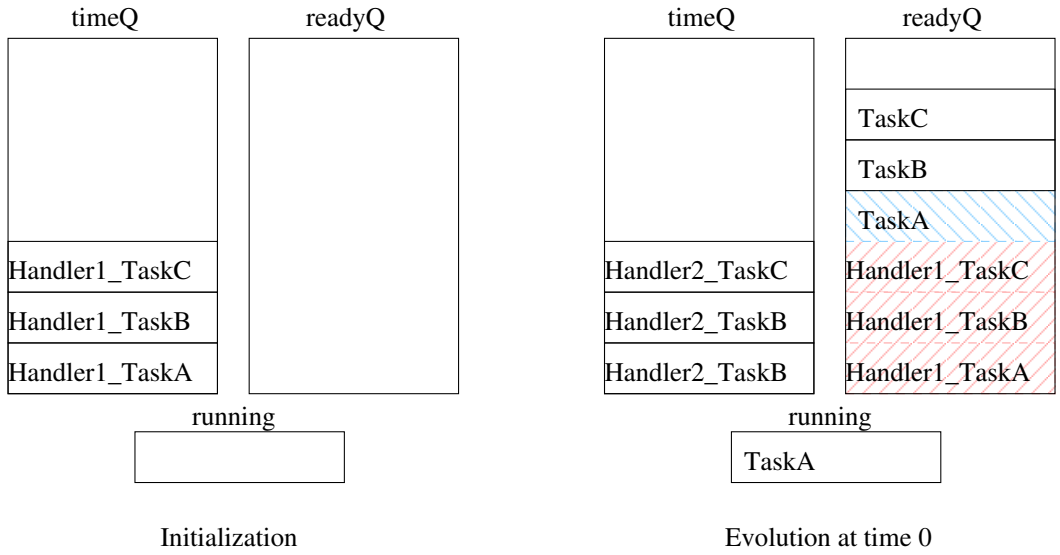


Figure A.4 – Évolution des listes *readyQ* et *timeQ* à l'instant 0

Le prochain appel au noyau est à la date $t = 1$ lorsque la tâche TaskA finit son exécution. La tâche TaskB devient alors la tâche en cours d'exécution (*readyQ* ne contient plus que TaskC). À l'instant $t = 4$, la tâche TaskB finit son exécution et la tâche TaskC est choisie pour être exécutée. À l'instant $t = 5$, la date de réveil de la procédure d'interruption associée à la tâche TaskA est atteinte, le noyau la traîte et la tâche TaskA est insérée dans la liste *readyQ*. La tâche TaskA a la plus haute priorité, elle préempte la tâche TaskC. La tâche TaskC est insérée dans la liste *readyQ*. L'évolution des listes *timeQ* et *readyQ* est représentée figure A.5.

A.2 Modifications apportées au simulateur TrueTime

Dans le cadre de notre étude du mécanisme de protection temporelle proposé par AUTOSAR OS (voir chapitre 5), nous avons voulu implémenter le mécanisme de budget d'exécution dans le simulateur. Pour les différentes simulations, nous gardons une version avec aucun mécanisme de protection implémenté (version système libre) que nous comparerons avec les différentes versions proposées (relatif aux différentes études déterministes, section 5.3). Nous avons donc implémenté quatre versions :

1. **Version 0 :** Pas de budget implémenté ;

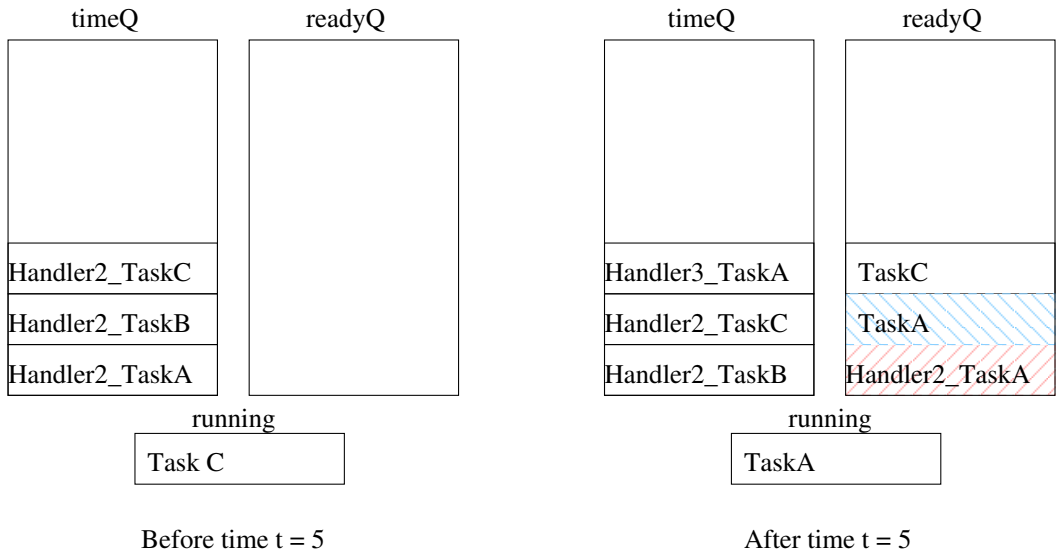


Figure A.5 – Évolution des listes *readyQ* et *timeQ* à l'instant 5

2. **Version 1.1 :** Budget simple AUTOSAR implémenté (budget = pire-temps d'exécution) ;
3. **Version 1.2 :** Budget avec relaxation proportionnelle ;
4. **Version 1.3 :** Budget avec relaxation pondérée.

Nous avons également besoin de prises de mesures. Celles-ci sont réalisées à partir des deux procédures d'interruption présentées ci-dessus :

1. Procédure d'interruption de détection de dépassement d'échéance ;
2. Procédure d'interruption de détection de dépassement de budget d'exécution.

A.2.1 Version basique (pas de mécanisme de protection)

Pour la version 0, aucun budget n'est implémenté. Nous utilisons la détection de dépassement d'échéance afin de détecter la propagation des fautes à travers le système. La détection de dépassement ne veut pas dire que la tâche défaillante est stoppée (pas de "Deadline Monitoring"), l'instance continue à s'exécuter jusqu'à sa terminaison et peut donc retarder l'ensemble du système (tâches moins prioritaires). Nous modifions le fichier *createtask.cpp* afin d'intégrer par défaut le détecteur pour chaque tâche. La détection d'une erreur est enregistrée dans une variable (modification du fichier *codefunctions.cpp* + écriture au niveau du fichier *ttkernel.cpp*).

A.2.2 Version Protection par budget d'exécution

A.2.2.1 Version 1.1

Pour la version 1.1, le budget d'exécution est égal au pire temps d'exécution. Ainsi nous pouvons utiliser le détecteur de dépassement de budget proposé dans la version de base. Le détecteur de dépassement d'échéance est gardé afin de contrôler le bon fonctionnement (recouvre-

ment complet) du mécanisme de protection. Nous modifions pour cela le fichier *createtask.cpp*.

De plus, au niveau du noyau (fichier *ttkernel.cpp*), nous modifions le comportement de l'ordonnanceur lors de la détection de dépassement d'un budget. Lorsqu'un dépassement est détecté, le noyau reprend la main. À ce moment, la tâche actuellement en exécution fautive est stoppée (appel à sa terminaison, suppression dans la liste *readyQ*).

A.2.2.2 Version 1.2

Pour la version 1.2, la relaxation des budgets est proportionnelle $B = \lambda.C$. Nous ajoutons un argument à la classe *RTsys* qui correspond au facteur de relaxation λ. Nous créons également une nouvelle fonction matlab *ttSetBudgets* permettant d'affecter ce facteur (le programme correspondant à la fonction est défini dans le fichier C++ *setbudgets.cpp*).

Ayant accès au budget de chaque tâche (mise à disposition de λ et C), le mécanisme de protection peut être mis en place. La procédure de détection ne doit plus être appelée lorsque $c > C$, mais lorsque $c > B$. Le fichier de mise à jour du mécanisme de détection *defaulthooks.cpp* doit être mis à jour afin de prendre en compte cette modification. Lors du démarrage de la tâche, la procédure *start hook* initialise le budget tel que $B = B_{init} = \lambda.C$. La valeur initiale du budget est un nouvel argument d'une tâche (modification du fichier *usertask.h*).

A.2.2.3 Version 1.3

Pour la version 3, le seul changement par rapport à la version précédente est la présence d'un poids pour chaque tâche pour la relaxation du budget $B = \lambda.w.C$.

Il suffit d'ajouter un nouvel argument à la tâche qui est le poids w. Celui-ci est affecté par la fonction matlab *ttSetWeights* (le programme correspondant à la fonction est défini dans le fichier C++ *setweights.cpp*).

A.2.3 Résumé des modifications

Les différentes modifications sur les fichiers du simulateur TrueTime sont résumées tableau A.2.

Fichier	Version 0	Version 1.1	Version 1.2	Version 1.3
createtask.cpp	x	x	x	x
codefunctions.cpp	x	x	x	x
usertask.h	-	-	-	x
defaulthooks.cpp	-	-	x	x
ttkernel.h	-	-	x	x
ttkernel.cpp	-	x	x	x
setbudgets.cpp	-	-	x	x
setweights.cpp	-	-	-	x

Tableau A.2 – Modifications apportées au simulateur TrueTime

Annexe B

Variables aléatoires et probabilités

Dans cette partie, Ω désigne un univers. On note $\mathcal{P}(\Omega)$ l'ensemble des parties de Ω et $\omega \in \Omega$ un événement élémentaire de Ω.

B.1 Variables aléatoires réelles

B.1.1 Axiomatique de Kolmogorov

Définition B.1. *Une famille $\mathcal{F}$ de parties de Ω est une tribu (ou σ-algèbre) si*

1. $\Omega \in \mathcal{F}$
2. *Si* $\Lambda \in \mathcal{F}$, $\Omega - \Lambda \in \mathcal{F}$
3. $(\Lambda_n, n \in \mathbb{N})$ *est une suite d'éléments dans* $\mathcal{F}$, *alors* $\cup \Lambda_n \in \mathcal{F}$

Définition B.2. *On appelle mesure de probabilité (ou probabilité) sur l'univers Ω une application $\mathbb{P} : \mathcal{P}(\Omega) \to [0,1]$ qui vérifie les axiomes suivants*

1. $\mathbb{P}(\Omega) = 1$
2. *Si* Λ *et* Λ' *sont deux parties disjointes de* Ω, *alors* $\mathbb{P}(\Lambda \cup \Lambda') = P(\Lambda) + (\Lambda')$
3. *Si* $(\Lambda_n, n \in \mathbb{N})$ *est une suite croissante d'événements, alors* $\mathbb{P}(\bigcup \Lambda_n) = \lim_{n \to \infty} \mathbb{P}(\Lambda_n)$

Définition B.3. *Un espace probabilisé (ou espace de probabilité) est un triplet $(\Omega, \mathcal{F}, P)$ formé d'un ensemble Ω, d'une tribu $\mathcal{F}$ sur Ω et d'une mesure de probabilité $\mathbb{P}$ sur cette σ-algèbre. Pour un évènement ω de $\mathcal{F}$, $\mathbb{P}(\omega)$ s'appelle la probabilité de l'évènement e.*

Définition B.4. *On appelle espace mesurable un couple $(E, \mathcal{E})$ où E est un ensemble et $\mathcal{E}$ une tribu sur E. Les éléments de $\mathcal{E}$ sont appelés ensembles mesurables.*

Définition B.5. *Soient (Ω, $\mathcal{F}$, $\mathbb{P}$) un espace probabilisé (ou espace de probabilité) et (E,$\mathcal{E}$) un espace mesurable. On appelle variable aléatoire de Ω vers E, toute fonction mesurable $\mathcal{X}$*

de Ω vers E telle que

$$\begin{array}{rccc} \mathcal{X}: & \Omega & \to & E \\ & \omega & \mapsto & \mathcal{X}(\omega) \end{array}$$

Si $\mathcal{X}$ est une variable aléatoire réelle, on restreint l'ensemble d'arrivé E à $\mathbb{R}$.

Dans la suite, le mot "variable aléatoire" désigne par défaut des variables aléatoires réelles.

B.1.2 Caractérisation d'une variable aléatoire

Définition B.6. *Soit $\mathcal{X} : (\Omega, \mathcal{F}, \mathbb{P}) \to (E, \mathcal{E})$ une variable aléatoire. On appelle loi de $\mathcal{X}$ la mesure de probabilité de $P_{\mathcal{X}}$ sur $(E, \mathcal{E})$ donnée par*

$$P_{\mathcal{X}}(A) = \mathbb{P}(\mathcal{X} \in A) = \mathbb{P}(\{\omega \in \Omega : \mathcal{X}(\omega) \in A\}),\ A \in \mathcal{E}$$

Une variable aléatoire $\mathcal{X}$ est caractérisée par sa fonction de répartition $F_{\mathcal{X}}$.

Définition B.7. *La fonction de répartition d'une variable aléatoire $\mathcal{X}$ est la fonction $F_{\mathcal{X}}$ telle que*

$$\begin{array}{rccc} F_{\mathcal{X}}: & \mathbb{R} & \to & [0,1] \\ & x & \mapsto & F_{\mathcal{X}}(x) = P_{\mathcal{X}}(x \in]-\infty, x]) = \mathbb{P}(\mathcal{X} \leq x) \end{array}$$

Cette fonction de répartition de $\mathcal{X}$ est une fonction croissante, continue à droite.

Définition B.8. *La fonction de densité de probabilité d'une variable aléatoire $\mathcal{X}$ est la fonction $f_{\mathcal{X}}$ définie par*

$$\begin{array}{rccc} f_{\mathcal{X}}: & \mathbb{R} & \to & \mathbb{R} \\ & x & \mapsto & f_{\mathcal{X}}(x) = F'_{\mathcal{X}}(x) \end{array}$$

La dérivée $F'_{\mathcal{X}}$ est prise au sens généralisé. On a : $F_{\mathcal{X}}(x) = \displaystyle\int_{-\infty}^{x} f_{\mathcal{X}}(t)\ \mathrm{d}t$

B.1.3 Moments d'une variable aléatoire

Définition B.9. *Une variable aléatoire $\mathcal{X}$ de densité de probabilité $f_{\mathcal{X}}$ admet un moment d'ordre k si*

$$m_k(\mathcal{X}) = \int_{-\infty}^{+\infty} x^k . f_{\mathcal{X}}(x)\ \mathrm{d}x$$

est convergente

Définition B.10. *Si $\mathcal{X}$ est une variable aléatoire de densité de probabilité $f_{\mathcal{X}}$ dont le moment d'ordre 1 ($m_1(\mathcal{X})$) existe, on appelle espérance (ou moyenne) de $\mathcal{X}$, le réel $\mathbb{E}(\mathcal{X})$, ce moment d'ordre 1*

$$\mathbb{E}(\mathcal{X}) = \int_{-\infty}^{+\infty} x . f_{\mathcal{X}}(x)\ \mathrm{d}x$$

si cette intégrale est convergente

Définition B.11. *Soit $\mathcal{X}$ une variable aléatoire dont le moment d'ordre 2, ($m_2(\mathcal{X}) = \mathbb{E}\left(\mathcal{X}^2\right)$, existe. On appelle variance de $\mathcal{X}$, le réel $Var(\mathcal{X})$, défini par :*

$$Var(\mathcal{X}) = \mathbb{E}\left(\mathcal{X}^2\right) - (\mathbb{E}(\mathcal{X}))^2 = \mathbb{E}\left[(\mathcal{X} - \mathbb{E}(\mathcal{X}))^2\right]$$

On appelle l'écart type, le réel $\sigma(\mathcal{X})$ tel que $\sigma(\mathcal{X}) = \sqrt{Var(\mathcal{X})}$

B.2 Variables aléatoires indépendantes et somme de variables aléatoires

Définition B.12. *On dit que les variables aléatoires $\mathcal{X}_1, \ldots, \mathcal{X}_n$ sont indépendantes si*

$$\mathbb{P}(\mathcal{X}_1 \in A_1, \ldots, \mathcal{X}_n \in A_n) = \prod_{i=1}^{n} \mathbb{P}(\mathcal{X}_i \in A_i)$$

pour tout $A_1 \in E_1, \ldots, A_n \in E_n$

Définition B.13. *Soient $\mathcal{X}$ et $\mathcal{Y}$ deux variables aléatoires de densités de probabilité respectives $f_{\mathcal{X}}$ et $f_{\mathcal{Y}}$, la densité de probabilité conditionnelle de $\mathcal{X}$ sachant $\mathcal{Y}$ notée $f_{\mathcal{X}|\mathcal{Y}}$ est telle que*

$$f_{\mathcal{X}|\mathcal{Y}}(x, y) = \frac{f_{\mathcal{X},\mathcal{Y}}(x, y)}{f_{\mathcal{Y}}(y)}$$

Définition B.14. *La somme de deux variables aléatoires $\mathcal{X}$ et $\mathcal{Y}$ de densités de probabilité respectives $f_{\mathcal{X}}$ et $f_{\mathcal{Y}}$ est la variable aléatoire $\mathcal{Z}$ définie par sa densité de probabilité $f_{\mathcal{Z}}$ telle que*

$$f_{\mathcal{Z}}(z) = \int_{-\infty}^{+\infty} f_{\mathcal{X}|\mathcal{Y}}(z - y|y).f_{\mathcal{Y}}(y)\ \mathrm{d}y$$

Dans le cas où les deux variables aléatoires $\mathcal{X}$ et $\mathcal{Y}$ sont indépendantes, la somme est la variable aléatoire $\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y}$, produit de convolution de $\mathcal{X}$ par $\mathcal{Y}$, telle que sa fonction de densité de probabilité

$$f_{\mathcal{Z}}(z) = \int_{-\infty}^{+\infty} f_{\mathcal{X}}(z - y).f_{\mathcal{Y}}(y)\ \mathrm{d}y$$

B.3 Variables aléatoires discrètes

Définition B.15. *On dit qu'une variable aléatoire est discrète si elle ne prend qu'un nombre fini ou dénombrable de valeurs :*

$$\mathcal{X} \in \{x_k\,,\ k \in K \subset \mathbb{N}\}$$

Une variable aléatoire discrète $\mathcal{X}$ est caractérisée par l'ensemble des valeurs qu'elle peut prendre et par la probabilité d'occurrence de ces valeurs.

Définition B.16. *La fonction de probabilité d'une variable aléatoire discrète $\mathcal{X}$ est la fonction $f_{\mathcal{X}}$ telle que*

$$\begin{array}{rccl} f_{\mathcal{X}}: & \Omega & \to & \mathbb{R} \\ & x & \mapsto & f_{\mathcal{X}}(x) = \mathbb{P}(\mathcal{X} = x) \end{array}$$

Définition B.17. *La fonction de répartition d'une variable aléatoire discrète $\mathcal{X}$ est la fonction $F_{\mathcal{X}}$ telle que*

$$\begin{array}{rccl} F_{\mathcal{X}}: & \Omega & \to & [0,1] \\ & x & \mapsto & F_{\mathcal{X}}(x) = \mathbb{P}(\mathcal{X} \leq x) = \displaystyle\sum_{i=-\infty}^{c} f_{\mathcal{X}}(x) \end{array}$$

Les moments d'une variable aléatoire discrète sont définis comme suit

Définition B.18. *Soit $\mathcal{X}$ une variable aléatoire discrète. L'espérance de $\mathcal{X}$ est le réel $\mathbb{E}(\mathcal{X})$ tel que*

$$\mathbb{E}(\mathcal{X}) = \sum_{k=-\infty}^{+\infty} x.\mathbb{P}(\mathcal{X} = k)$$

Définition B.19. *Soit $\mathcal{X}$ une variable aléatoire discrète On appelle variance de $\mathcal{X}$, le réel $Var(\mathcal{X})$, défini par :*

$$Var(\mathcal{X}) = \mathbb{E}\left(\mathcal{X}^2\right) - (\mathbb{E}(\mathcal{X}))^2 = \mathbb{E}\left[(\mathcal{X} - \mathbb{E}(\mathcal{X}))^2\right]$$

On appelle l'écart type, le réel $\sigma(\mathcal{X})$ tel que $\sigma(\mathcal{X}) = \sqrt{Var(\mathcal{X})}$

Définition B.20. *La somme de deux variables aléatoires indépendantes discrètes $\mathcal{X}$ et $\mathcal{Y}$ est la variable indépendante $\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y}$ telle que*

$$\mathbb{P}(\mathcal{Z} = z) = \sum_{k=-\infty}^{+\infty} \mathbb{P}(\mathcal{X} = k).\mathbb{P}(\mathcal{Y} = z - k)$$

Annexe C

Exemple de résolution par différentes méthodes d'un cas d'étude simple

C.1 Présentation de l'exemple

Le système $\mathcal{S}$ étudié est composé de 3 tâches $\{\tau_1, \tau_2, \tau_3\}$. Chaque tâche est caractérisée par son temps d'exécution $\mathcal{C}$ considéré comme une variable aléatoire, sa période T et son échéance relative D considérés parfaitement connus. Dans notre cas, nous avons

$$\tau_1 = (\begin{pmatrix} 1 & 2 \\ 0.50 & 0.50 \end{pmatrix}, 3, 3),\ \tau_2 = (\begin{pmatrix} 1 & 3 \\ 0.50 & 0.50 \end{pmatrix}, 8, 8),\ \tau_3 = (\begin{pmatrix} 2 \\ 1.00 \end{pmatrix}, 12, 12)$$

Nous pouvons déterminer l'utilisation globale de ce système. Celle-ci est également une variable aléatoire $\mathcal{U}$ telle que

$$\begin{aligned} \mathcal{U} &= \mathcal{U}_1 \otimes \mathcal{U}_2 \otimes \mathcal{U}_3 \\ &= \begin{pmatrix} 1/3 & 2/3 \\ 0.50 & 0.50 \end{pmatrix} \otimes \begin{pmatrix} 1/8 & 3/8 \\ 0.50 & 0.50 \end{pmatrix} \otimes \begin{pmatrix} 2/12 \\ 1.00 \end{pmatrix} \\ &= \begin{pmatrix} 0.625 & 0.875 & 0.958 & 1.208 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix} \end{aligned}$$

L'utilisation globale moyenne est donc de :

$$\overline{\mathcal{U}} = 0.9165$$

On peut également spécifier que l'utilisation maximale U^{max} est supérieure à 1. En effet, nous avons

$$U^{max} = \sum_{i=1}^{3} \frac{C_i^{max}}{T_i} = 1.208$$

C.2 Exemple de calcul de la probabilité d'ordonnançabilité

Nous développons ici les calculs par la méthode de Diaz *et al.* (2002) (sans le calcul du steady-state backlog) du temps de réponse du job $\tau_{3,0}$. Nous faisons donc l'hypothèse de résidus nuls après une hyperpériode. De plus, le système est considéré synchrone.

On a donc :

$$\begin{aligned} \mathcal{R}_{3,0} &= \mathcal{B}_{\pi}^{stat} \otimes \mathcal{W}_{\pi}(a_{3,0}) \otimes \mathcal{C}_3 \otimes \mathcal{J}_{3,0} \\ &= \mathcal{C}_3 \otimes \mathcal{J}_{3,0} \end{aligned}$$

Le calcul de $R_{3,0}$ est itératif afin d'obtenir les interférences des instances de tâche plus prioritaires. Nous développons les résultats pour chaque itération (la partie de la fonction de probabilité à droite de la barre verticale est la partie qui ne varie plus au cours des prochaines itérations).

– Itération 1 :

$$\mathcal{R}_{3,0}^{[0,3]} = \mathcal{C}_1 \otimes \mathcal{C}_2 \otimes \mathcal{C}_3 = \begin{bmatrix} 4 & 5 & 6 & 7 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

– Itération 2 :

$$\mathcal{R}_{3,0}^{[0,6]} = \mathrm{AF}\left(\mathcal{R}_{3,0}^{[0,3]}, 6, \mathcal{C}_1\right) = \begin{bmatrix} 5 & 6 & 7 & 8 & 9 \\ 0.125 & 0.25 & 0.25 & 0.25 & 0.125 \end{bmatrix}$$

– Itération 3 :

$$\mathcal{R}_{3,0}^{[0,8]} = \mathrm{AF}\left(\mathcal{R}_{3,0}^{[0,6]}, 8, \mathcal{C}_1\right) = \left[\begin{array}{cc|cccc} 5 & 6 & 8 & 9 & 10 & 11 \\ 0.125 & 0.25 & 0.125 & 0.25 & 0.1875 & 0.0625 \end{array}\right]$$

– Itération 4 :

$$\mathcal{R}_{3,0}^{[0,9]} = \left[\begin{array}{ccc|ccccc} 5 & 6 & 8 & 10 & 11 & 12 & 13 & 14 \\ 0.125 & 0.25 & 0.125 & 0.125 & 0.09375 & 0.15625 & 0.09375 & 0.03125 \end{array}\right]$$

– Itération 5 :

$$\mathcal{R}_{3,0}^{[0,12]} = \left[\begin{array}{ccc|cccccc} 5 & 6 & 8 & 11 & 12 & 13 & 14 & 15 & 16 \\ 0.125 & 0.25 & 0.125 & 0.0625 & 0.109375 & 0.125 & 0.125 & 0.0625 & 0.015625 \end{array}\right]$$

L'échéance de la tâche τ_3 est égale à 12. Nous obtenons donc la probabilité d'ordonnançabilité pour l'instance $\tau_{3,0}$:

$$S_{\tau_{3,0}} = \mathbb{P}(\mathcal{R}_{3,0} \leq D_3) = 0.67185$$

Cette valeur est celle qui serait obtenue dans le cas du calcul de la borne inférieure sur la probabilité d'ordonnançabilité de l'instance $\tau_{3,0}$ lors de l'utilisation du "Deadline Monitoring".

Le calcul du steady-state backlog est réalisé de façon numérique (itérations successives jusqu'à convergence). La figure C.1 représente les différentes itérations (après 1,2,... hyper-périodes) de calcul. Le critère de convergence permet de réduire le nombre d'itérations à 9.

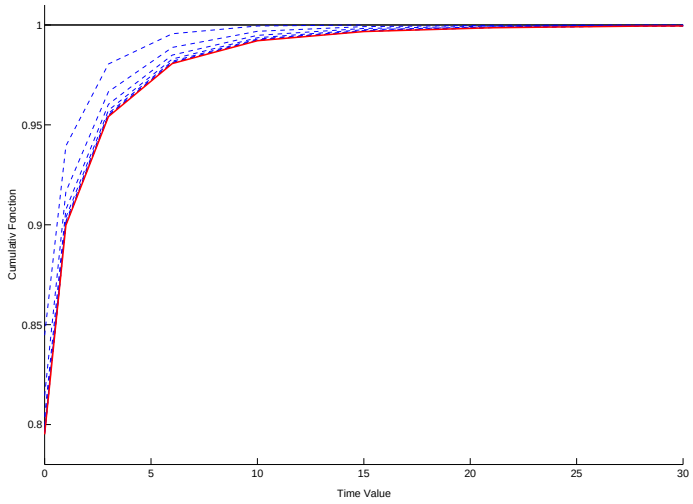


Figure C.1 – Itérations de calcul de l'état stationnaire

Le calcul de la probabilité d'ordonnançabilité $S_{3,0}$ en considérant le steady-state backlog est réalisé à l'aide de la formule

$$\begin{aligned} \mathcal{R}_{3,0} &= \mathcal{B}_{\pi}^{stat} \otimes \mathcal{W}_{\pi}(a_{3,0}) \otimes \mathcal{C}_3 \otimes \mathcal{J}_{3,0} \\ &= \mathcal{B}_{\pi}^{stat} \otimes \mathcal{C}_3 \otimes \mathcal{J}_{3,0} \end{aligned}$$

Le calcul est donc le même que précédemment mais en initialisant la première itération à l'aide du steady-state backlog calculé ci-dessus. On obtient alors

$$S_{\tau_{3,0}} = \mathbb{P}(\mathcal{R}_{3,0} \leq D_3) = 0.40586$$

Annexe D

Présentation de quelques méthodes d'optimisation

D.1 Optimisation unidirectionnelle (variable scalaire)

Le problème d'optimisation considéré est le suivant

$$\underset{x\in\mathbb{E}}{\text{Argmin}} f(x) \tag{D.1}$$

avec la fonction f considérée telle que

$$\begin{array}{rccc} f: & \mathbb{X} & \rightarrow & \mathbb{R} \\ & x & \mapsto & f(x) \end{array}$$

D.1.1 Méthode de Newton

Conditions d'utilisation : f deux fois dérivables (connaissance de $\frac{df}{dx}$ et $\frac{d^2f}{dx^2}$)

Méthode itérative telle que

$$\begin{cases} x_{k+1} = x_k - \frac{df/dx_k}{d^2f/dx_k^2} \\ x_0 = a \end{cases} \tag{D.2}$$

a est le point de départ de l'algorithme et df/dx_k et d^2f/dx_k^2 représentent respectivement les dérivées première et seconde en $x = x_k$.

Défaut de la méthode : pas de convergence globale garantie (dépend du point de départ a).

Qualités : convergence en une itération pour les fonctions quadratiques et rapide pour les fonctions bien approximées (par des fonctions quadratiques ou par son développement au deuxième ordre)

D.1.2 Méthode par réduction d'intervalle

Plusieurs méthodes par réduction d'intervalle ont été proposées. Elles consistent, à partir d'un intervalle de départ $\Delta_0 = [\lambda_p^{min}, \lambda_p^{max}]$, à réduire celui-ci jusqu'à obtenir une précision voulue. Elles consiste généralement à calculer des points à l'intérieur de l'intervalle, à calculer la valeur de la fonction en ces points et de définir à partir des valeurs calculées un nouvel intervalle (plus petit).

Le premier choix est d'optimiser la réduction de l'intervalle. Il suffit alors d'utiliser une méthode de dichotomie où les deux points λ_1 et λ_2 sont tels que

$$\lambda_1 = \frac{\lambda_p^{min} + \lambda_p^{max}}{2} - \epsilon \qquad \lambda_2 = \frac{\lambda_p^{min} + \lambda_p^{max}}{2} + \epsilon$$

La valeur de ϵ est alors à choisir par l'utilisateur (généralement, ϵ : précision machine). Le nombre de calculs de la fonction par itération est de deux. La réduction de l'intervalle est optimale mais le nombre de calculs de la fonction peut être réduit.

L'algorithme de Kiefer (1953) permet d'aller dans ce sens en réutilisant un des points déjà calculés. Une présentation de l'évolution de la réduction d'intervalle est donnée figure D.1.

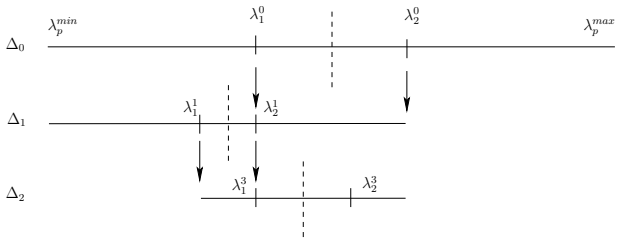


Figure D.1 – Réduction par algorithme de Kiefer

L'algorithme permet de réduire l'intervalle tel que $\Delta_k = \Delta_{k+1} + \Delta_{k+2}$. Pour garder cette propriété, le choix de λ_1 à l'initialisation doit être tel que :

$$\frac{1}{3} < \frac{\Delta_1}{\Delta_0} < \frac{2}{5}$$

Une solution est de choisir λ_1^0 tel que :

$$\frac{\Delta_1}{\Delta_0} = -\frac{1-\sqrt{5}}{2} = 0.618\,^{1} \quad i.e. \quad \lambda_1 = 0.618.\Delta_0$$

Dans ce cas la méthode d'optimisation s'appelle la méthode du nombre d'or.

D.1.3 Méthode de Brent

Le but de la méthode de Brent (Brent, 1972) est de se rapprocher des propriétés de l'algorithme de Newton sans connaissance des fonctions dérivées de f et en ne calculant qu'un seul

1. Le coefficient choisi est l'inverse du nombre d'or.

point par itération.

Dans un premier temps, une approximation de la fonction f par une parabole permet de se rapprocher de la méthode de Newton et de réduire rapidement l'intervalle de recherche. Cette méthode devenant instable lorsque l'intervalle est trop petit (test de cohérence), la deuxième phase consiste à passer sur la méthode du nombre d'or pour obtenir l'optimum.

D.2 Optimisation multi-directionnelle (variable vectorielle)

Dans cette partie, le problème d'optimisation considéré est le suivant

$$\underset{\mathbf{X}\in\mathbb{E}^n}{\text{Argmin}} f(x) \tag{D.3}$$

avec la fonction f considérée telle que

$$\begin{array}{rccc} f: & \mathbb{X}^n & \rightarrow & \mathbb{R} \\ & x & \mapsto & f(x) \end{array}$$

D.2.1 Méthodes analytiques

Plusieurs méthodes analytiques peuvent être utilisées toutes basées sur des méthodes de descente. Une méthode de descente optimise la fonction dans une ou plusieurs directions itérativement. Nous présentons ici deux méthodes celle de Fletcher-Reeves et celle de Powell (BFGS).

La méthode du gradient conjugué de Fletcher-Reeves est définie comme suit

$$\left\{ \begin{array}{l} x_0, \quad d_0 = -\frac{\partial f}{\partial x_0} \\ x_{k+1} = x_k + \lambda_k.d_k, \quad \lambda_k = \underset{\lambda\in\mathbb{R}}{\text{Argmin}}\ f(x_k + \lambda.d_k) \\ d_{k+1} = -\frac{\partial f}{\partial x_{k+1}} + \beta_k.d_k \end{array} \right.$$

avec

$$\beta_k = \frac{\left(\frac{\partial f}{\partial x_{k+1}}\right)^T \left(\frac{\partial f}{\partial x_{k+1}}\right)}{\left(\frac{\partial f}{\partial x_k}\right)^T \left(\frac{\partial f}{\partial x_k}\right)}$$

La méthode de Powell (BFGS) est définie comme suit

$$\left\{ \begin{array}{l} x_0, \quad H_0 = I \\ x_{k+1} = x_k - \lambda_k.H_k.\frac{\partial f}{\partial x_k}, \quad \lambda_k = \underset{\lambda\in\mathbb{R}}{\text{Argmin}}\ f\left(x_k + \lambda.H_k.\frac{\partial f}{\partial x_k}\right) \\ H_{k+1} = H_k + \Delta_k \end{array} \right.$$

avec

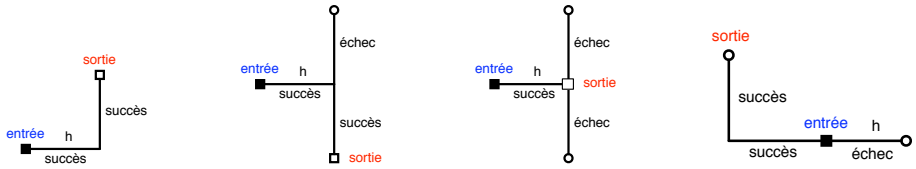
$$\Delta_k = \frac{(\delta_k - H_k.\gamma_k)^T (\delta_k - H_k.\gamma_k)}{\gamma_k^T (\delta_k - H_k.\gamma_k)}, \qquad \left\{ \begin{array}{l} \delta_k = x_{k+1} - x_k \\ \gamma_k = \frac{\partial f}{\partial x_{k+1}} - \frac{\partial f}{\partial x_k} \end{array} \right.$$

D.2.2 Méthodes heuristiques

Si la fonction f est n'est pas dérivable ou difficilement, des méthodes heuristiques peuvent être utilisées. Dans ce cadre plusieurs méthodes d'optimisation existent comme la méthode de Hooke et Jeeves (1961), la méthode de Rosenbrock (1960) ou la méthode de Nelder et Mead (1965) (simplexe).

Nous allons nous concentrer sur la méthode proposée par Hooke et Jeeves qui possède une progression lente mais sûre. Elle peut se décomposer en deux étapes : une étape locale et une étape de progression.

L'étape locale consiste à scruter les environs d'un point d'entrée dans le but d'améliorer la fonction. Cette scrutation est réalisée avec un pas h successivement selon les différentes dimensions B_1, B_2, ..., B_n (voir figure D.2). Si aucune amélioration autour du point d'entrée n'est possible avec le pas h, le pas est divisé par deux ($h/2$) et une nouvelle scrutation est effectuée.



(a) Cas 1 (b) Cas 2 (c) Cas 3 (d) Cas 4

Figure D.2 – Exemple de cas possibles pour l'étape locale

L'étape de progression permet de construire le point d'entrée suivant. La construction du point d'entrée e_{k+1} se fait par extrapolation linéaire des deux sorties s_k et s_{k+1} (pour le point d'entrée e_1, on utilise les points e_0 et s_0) (voir figure D.3). On vérifie cependant que le point d'entrée e_{k+1} est meilleur que s_{k+1} sinon $e_{k+1} = s_{k+1}$.

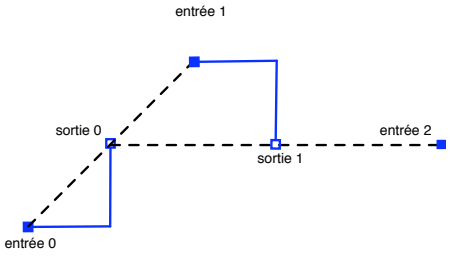


Figure D.3 – Exemple de progression

D.2.3 Méta-heuristiques et algorithme évolutionnaire

La fonction f peut admettre plusieurs minimums locaux et l'utilisateur veut dans certains cas obtenir plusieurs réponses (les techniques présentées précédemment permettent de déter-

miner un seul point d'optimisation).

Dans ce cadre les algorithmes évolutionnaires (Dréo *et al.*, 2003) permettent de traiter des problèmes vectoriels ayant plusieurs minimums et peuvent fournir plusieurs points de sortie. Le principe global de ces algorihmes est présenté figure D.4. Il s'agit de choisir aléatoirement un ensemble de points initiaux (population initiale), d'évaluer ces points (calculer la norme de f) et de sélectionner un ensemble de points (par un choix à définir) pour la reproduction.

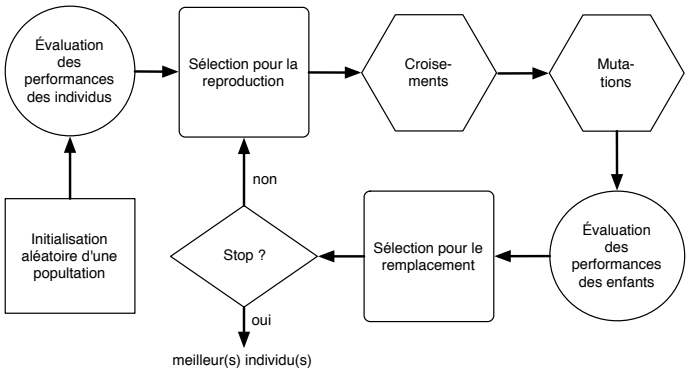


Figure D.4 – Principe d'un algorithme évolutionnaire

Publications

D. Bertrand, A.M. Déplanche, S. Faucou et O. Roux : A Study of the AADL Mode Change Protocol, *13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'08)* : p. 288-293, April 2008

D. Bertrand : Étude de systèmes temps réel avec des temps d'exécution incertains, *École d'été temps réel (ETR'09)*, Sept. 2009

D. Bertrand, S. Faucou et Y. Trinquet : An analysis of the AUTOSAR OS timing protection mechanism, *IEEE Conference on Emerging Technologies Factory Automation (ETFA'09)*, p. 1-8, Sept. 2009

D. Bertrand, S. Faucou et Y. Trinquet : How to configure AUTOSAR OS timing protection ? *Proceedings of the Junior Researcher Workshop on Real-Time Computing (JRWRTC'09)*, p. 43-46, Oct. 2009

D. Bertrand, S. Faucou et Y. Trinquet : Temporal isolation for the cohabitation of applications in automotive embedded software, *Proceedings of the 1st Workshop on Critical Automotive applications (EDCC-CARS '10)*, p. 25-28, April 2010

Bibliographie

Abeni, L. et Buttazzo, G. : Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, 2004.

Abeni, L., Buttazzo, G., Superiore, S. et Anna, S. : Integrating multimedia applications in hard real-time systems. *In Proceedings of the 19th IEEE Real-time Systems Symposium*, p. 4 –13, dec 1998.

Alur, R., Courcoubetis, C., Henzinger, T. et Ho, P. : Hybrid automata : An algorithmic approach to the specification and verification of hybrid systems. *In* Grossman, R., Nerode, A., Ravn, A. et Rischel, H., éds : *Hybrid Systems*, vol. 736 de *Lecture Notes in Computer Science*, p. 209–229. Springer Berlin / Heidelberg, 1993.

ARINC : Avionics application software standard interface. Rap. tech., ARINC Specification 653, 1997.

Arlat, J., Crouzet, Y., Deswarte, Y., Fabre, J.-C., Laprie, J.-C. et Powell, D. : *Tolérance aux fautes dans "Encyclopédie de l'informatique et des systèmes d'information"*. Vuibert, 2006.

Åsberg, M., Behnam, M., Nemati, F. et Nolte, T. : Towards hierarchical scheduling in autosar. *In Proceedings of 14th IEEE International Conference on Emerging Techonologies and Factory (ETFA'09)*, September 2009.

Audsley, N., Burns, A., Richardson, M., Tindell, K. et Wellings, A. : Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284 –292, Sep 1993.

Audsley, N. C. : Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Tech. rep. 164, University of York, November 1991.

Audsley, N., Burns, A., Richardson, M. F. et Wellings, A. J. : Hard real-time scheduling : The deadline-monotonic approach. *In Proceedings of IEEE Workshop on Real-Time Operating Systems and Software*, p. 133–137, 1991.

AUTOSAR : AUTOSAR - Specification of operating system. Rap. tech. v3.0.1, AUTOSAR GbR, 2008.

Aviznienis, A., Randell, B. et Landwehr, C. : Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE transactions on dependable and secure computing*, 1(1), 2004.

Baruah, S. et Vestal, S. : Schedulability analysis of sporadic tasks with multiple criticality specifications. *Euromicro Conference on Real-Time Systems (ECRTS '08)*, p. 147–155, July 2008.

Bechennec, J.-L., Briday, M., Faucou, S. et Trinquet, Y. : Trampoline an open source implementation of the osek/vdx rtos specification. *In IEEE Conference on Emerging Technologies and Factory Automation (ETFA '06)*, p. 62 –69, sep. 2006.

Bernat, G., Colin, A. et Petters, S. : Wcet analysis of probabilistic hard real-time systems. *In Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, p. 279 – 288, 2002.

Bertrand, D., Faucou, S. et Trinquet, Y. : An analysis of the autosar os timing protection mechanism. *In IEEE Conference on Emerging Technologies Factory Automation (ETFA'09)*, p. 1 –8, 22-25 2009.

Bini, E. et Buttazzo, G. : Biasing effects in schedulability measures. *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04)*, p. 196–203, June-2 July 2004.

Bini, E. : Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.

Bini, E., Natale, M. D. et Buttazzo, G. : Sensitivity analysis for fixed-priority real-time systems. *In Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS '06)*, p. 13–22, Washington, DC, USA, 2006. IEEE Computer Society.

Bougueroua, L., George, L. et Midonnet, S. : Dealing with execution-overruns to improve the temporal robustness of real-time systems scheduled fp and edf. *In Proceedings of the Second International Conference on Systems (ICONS '07)*, p. 52–52, April 2007.

Brent, P. : *Algorithms for minimization without derivatives.* Prentice-Hall, 1972.

Burguière, C., Reineke, J. et Altmeyer, S. : Cache-related preemption delay computation for set-associative caches—pitfalls and solutions. *In Proceedings of 9th International Workshop on Worst-Case Execution Time (WCET) Analysis*, June 2009.

Burns, A. et Edgar, S. : Predicting computation time for advanced processor architectures. *In Proceedings of the 12th Euromicro Conference on Real-Time Systems (Euromicro-RTS'00)*, p. 89 –96, 2000.

Burns, A. et Wellings, A. : *Real-Time Systems and Programming Languages : Ada 95, Real-Time Java and Real-Time C/POSIX (3rd Edition).* Addison Wesley, third edition édn, April 2001.

Caccamo, M., Buttazzo, G. et Thomas, D. : Efficient reclaiming in reservation-based real-time systems with variable execution times. *Computers, IEEE Transactions on*, 54(2):198 – 213, feb. 2005.

Caccamo, M., Buttazzo, G. et Sha, L. : Capacity sharing for overrun control. *In Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS'00)*, p. 295–304, 2000.

Cassé, H., Sainrat, P., Ballabriga, C. et de Michiel, M. : Experimentation of wcet computation on both ends of automotive processor range. *In Proceedings of the 1st Workshop on Critical Automotive applications (CARS '10)*, p. 67–70, New York, NY, USA, 2010. ACM.

Cassez, F. : Timed Games for Computing Worst-Case Execution-Times. Rap. tech., National ICT Australia, Sidney, Australia, 2010.

CEI 61508 : International standard IEC 61508 : Functional safety of electrical/electronic/programmable electronic safety-related systems (E/E/PES). Rap. tech., International Electrotechnical Commission, 1999.

Chabrol, D., Aussaguès, C. et David, V. : A spatial and temporal partitioning approach for dependable automotive systems. *In IEEE Conference on Emerging Technologies Factory Automation (ETFA'09)*, p. 1125–1132, 22-25 2009.

Colin, A. et Petters, S. : Experimental evaluation of code properties for wcet analysis. *In Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS'03)*, p. 190 – 199, dec. 2003.

Cucu, L. et Tovar, E. : A framework for the response time analysis of fixed-priority tasks with stochastic inter-arrival times. Rap. tech. 1, RR-INRIA, New York, NY, USA, 2006.

David, L. et Puaut, I. : Static determination of probabilistic execution times. *In Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04)*, p. 223 – 230, jun. 2004.

David, V., Delcoigne, J., Leret, E., Ourghanlian, A., Hilsenkopf, P. et Paris, P. : Safety properties ensured by the oasis model for safety critical real-time systems. *In Proceedings of the 17th International Conference on Computer Safety, Reliability and Security (SAFECOMP '98)*, p. 45–59, London, UK, 1998. Springer-Verlag.

Davis, R., Tindell, K. et Burns, A. : Scheduling slack time in fixed priority pre-emptive systems. *In Proceedings of Real-Time Systems Symposium (RTSS'93)*, p. 222–231, Dec 1993.

Diaz, J., Garcia, D., Kim, K., Lee, C.-G., Lo Bello, L., Lopez, J., Min, S. L. et Mirabella, O. : Stochastic analysis of periodic real-time systems. *In Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, p. 289–300, 2002.

Diaz, J., Lopez, J., Garcia, M., Campos, A., Kim, K. et Bello, L. : Pessimism in the stochastic analysis of real-time systems : concept and applications. *In Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS'04)*, p. 197 – 207, dec. 2004.

DO-178B : Software considerations in airborne systems and equipment certification (rtca do-178b). Rap. tech., RTCA Inc., 1992.

Dorin, F., Richard, P., Richard, M. et Goossens, J. : Uniprocessor Schedulability and Sensitivity Analysis of Multiple Criticality Tasks with Fixed-Priorities. *In* Laurent George et

Maryline Chetto andMikael Sjodin, éds : *Proceedings of the 17th International Conference on Real-Time and Network Systems*, p. 13–22, Paris France, 2009.

Dréo, J., Pétrowski, A., Siarry, P. et Taillard, E. : *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.

ED-12B : ED-12B - software considerations in airborne systems and equipment certification. Rap. tech., EUROCAE, 1999.

EN 50128 : EN 50128 : Software for railway control and protection systems. Rap. tech., CENELEC, 2001.

Fisher, R. A. et Tippett, L. H. C. : Limiting forms of the frequency distribution of the largest or smallest member of a sample. *In Proceedings of the Cambridge Philosophical Society*, vol. 24, p. 180–+, 1928.

Freescale : *MC9328MXL i.MX Integrated Portable System Processor - Reference Manual*. Freescale Semiconductor, june 2007.

Goossens, J. et Macq, C. : Limitation of the hyper-period in real-time periodic task set generation. *In Proceedings of the RTS Embedded System (RTSÕ01)*, p. 133–147, 2001.

Gumbel, E. J. : *Statistics of Extremes*. Columbia University Press, 1958.

Gustafsson, J. : Usability aspects of wcet analysis. *In Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC'2008)*, p. 346 –352, may. 2008.

Hooke, R. et Jeeves, T. A. : "direct search" solution of numerical and statistical problems. *Journal of the ACM*, 8(2):212–229, 1961.

ISO 26262 : ISO 26262 : Road vehicles – functional safety. Rap. tech., Technical Comity ISO TC22 SC3 WG16, 2010.

Joseph, M. et Pandya, P. : Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

Kalman, R. E. : A new approach to linear filtering and prediction problems. *Transactions of the ASME Ð Journal of Basic Engineering*, (82 (Series D)):35–45, 1960.

Kiefer, J. : Sequential minimax search algorithm. *In Proceedings of The American Society*, 1953.

Kirner, R. et Puschner, P. : Classification of wcet analysis techniques. *In Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, p. 190 – 199, may. 2005.

Kopetz, H., Laprie, J. C., Randell, B. et Littlewood, B. : *Predictably Dependable Computing Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1995.

Kopetz, H. : Event-triggered versus time-triggered real-time systems. *In Proceedings of the International Workshop on Operating Systems of the 90s and Beyond*, p. 87–101, London, UK, 1991. Springer-Verlag.

Laprie, J.-C., Arlat, J., Blanquart, J.-P., Costes, A., Crouzet, Y., Deswarte, Y., Fabre, J.-C., Guillermain, H., Kaâniche, M., Kanoun, K., Mazet, C., Powell, D., Rabéjac, C. et Thévenod, P. : *Guide de la sureté de fonctionnement*. Cépaduès Editions, 1995.

Lehoczky, J., Sha, L. et Ding, Y. : The rate monotonic scheduling algorithm : exact characterization and average case behavior. *In Proceedings of the 10th IEEE Real Time Systems Symposium (RTSS'89)*, p. 166–171, Dec 1989.

Lehoczky, J. : Fixed priority scheduling of periodic task sets with arbitrary deadlines. *In Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS'90)*, p. 201 –209, 5-7 1990.

Liu, C. L. et Layland, J. W. : Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

López, J. M., Díaz, J. L., Entrialgo, J. et García, D. : Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-Time Systems*, 40(2):180–207, 2008.

Luenberger, D. G. : Observing the state of a linear system. *Military Electronics, IEEE Transactions on*, 8(2):74 –80, apr. 1964.

Mancina, A., Faggioli, D., Lipari, G., Herder, J. N., Gras, B. et Tanenbaum, A. S. : Enhancing a dependable multiserver operating system with temporal protection via resource reservations. *Real-Time Systems*, 43(2):177–210, 2009.

Mercer, C., Rajkumar, R. et Zelenka, J. : Temporal protection in real-time operating systems. *In Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software (RTOSS '94)*, p. 79 –83, may. 1994.

Meumeu Yomsi, P. et Sorel, Y. : Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. *In Proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS'07)*, p. 280–290, Washington, DC, USA, 2007. IEEE Computer Society.

Muller, G., Banatre, M., Peyrouze, N. et Rochat, B. : Lessons from ftm : an experiment in design and implementation of a low-cost fault tolerant system. *IEEE Transactions on Reliability*, 45(2):332 –340, jun 1996.

Nassor, E. et Bres, G. : Hard real-time sporadic task scheduling for fixed priority schedulers. *In Proceedings of the International Workshop on Responsive Systems*, 1991.

Nelder, J. A. et Mead, R. : A simplex method for function minimization. *The computer journal*, 7:308–313, 1965.

Ohlin, M., Henriksson, D. et Cervin, A. : Truetime 1.5 - reference manual. Rap. tech., Department of Automatic Control, Lund University, Sweden, January 2007.

OSEKtime : OSEK/VDX - Time Triggered Operating System Specification. Rap. tech. v1.0, OSEK Group, 2001.

OSEK/VDX : OSEK/VDX - Operating system. Rap. tech. v2.2.3, OSEK Group, 2005.

Pavin, F. : Trampoline (OSEK/VDX OS) Test Plan - Version 1.0. June 2010a.

Pavin, F. : Trampoline (OSEK/VDX OS) Test Procedure - Version 1.0. June 2010b.

Refaat, K. S. et Hladik, P.-E. : Efficient stochastic analysis of real-time systems via random sampling. *In Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS'10)*, p. 175–183, 2010.

Rosenbrock, H. : An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3:175–184, 1960.

Schlager, M., Herzner, W., Wolf, A., Gründonner, O., Rosenblattl, M. et Erkinger, E. : Encapsulating application subsystems using the DECOS Core OS. *In Proceedings of Conference on Computer Safety, Reliability and Security (SAFECOMP)*, vol. 4166 de *LNCS*, p. 386–397, Gdansk, Poland, September 2006. Springer Berlin / Heidelberg.

Sehlberg, D. : Static WCET Analysis of Task-Oriented Code for Construction Vehicles. Mémoire de D.E.A., Mälardalen Univerity, Mälardalen, Sweden, 2005.

Shaw, A. : Reasoning about time in higher-level language software. *IEEE Transactions on Software Engineering*, 15(7):875 –889, jul. 1989.

Shin, I. et Lee, I. : Periodic resource model for compositional real-time guarantees. *In Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS'03)*, p. 2 – 13, dec. 2003.

Tia, T.-S., Deng, Z., Shankar, M., Storch, M., Sun, J., Wu, L.-C. et Liu, J.-S. : Probabilistic performance guarantee for real-time tasks with varying computation times. *In Proceedings of Real-Time Technology and Applications Symposium (RTAS'95)*, p. 164–173, May 1995.

Tindell, K. W., Burns, A. et Wellings, A. J. : An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6:133–151, 1994.

TTP OS : Real time operating system. Rap. tech., TTTech, 2010.

Vestal, S. : Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. *In Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS'07)*, p. 239 –243, dec. 2007.