Synchronisation et Autonomie énergétique des Systèmes temps réel embarqués

Audrey Queudet

UNIVERSITÉ DE NANTES

LS2N LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

Synchronisation et Autonomie énergétique des Systèmes temps réel embarqués

Habilitation à diriger des recherches

Audrey Queudet
Maître de conférences à l'UFR Sciences et Techniques de Nantes

07 juin 2021

Composition du jury :

Présidente : Maryline CHETTO, Professeur des Universités à l'Université de Nantes / LS2N

Rapporteurs : Daniel CHILLET, Professeur des Universités à l'ENSSAT / IRISA Rennes
Laurent GEORGE, Professeur à l'ESIEE Paris, Université Gustave Eiffel / LIGM,
Pascal RICHARD, Professeur des Universités à l'IUT de Poitiers / LIAS

Examinateurs : Alfons CRESPO, Professeur des Universités à l'ETSINF Valencia, UPV, Espagne
Liliana CUCU-GROSJEAN, Chargée de Recherche INRIA Paris

Table des matières

Introduction

Ce document présente la synthèse de mes travaux de recherche depuis ma nomination en septembre 2007 aux fonctions de Maître de Conférences au sein du Département Informatique de l'UFR Sciences et Techniques de l'Université de Nantes. J'ai intégré pour la partie recherche le LINA (UMR 6241 : Laboratoire d'Informatique de Nantes Atlantique), qui a fusionné en janvier 2017 avec le laboratoire IRCCyN (UMR 6597 : Institut de Recherche en Communications et Cybernétique de Nantes) pour former le laboratoire LS2N (UMR 6004 : Laboratoire des Sciences du Numérique de Nantes).

Ma thèse de doctorat a été préparée au sein de l'équipe "Ordonnancement temps réel" du LINA, sous la direction de Maryline Chetto, sur la problématique de *l'ordonnancement temps réel sous contraintes de Qualité de Service (QoS)*. Elle a donné lieu d'une part à des résultats théoriques portant sur la gestion des cas de surcharge dans les systèmes centralisés monoprocesseur, et d'autre part à une intégration des politiques d'ordonnancement proposées sous forme de composants logiciels à code source ouvert sous Linux temps réel. Ces travaux ont été menés en synergie avec les partenaires du projet CLEOPATRE (*Composants Logiciels sur Etagères Ouverts Pour les Applications Temps Réel Embarquées*, projet RNTL – Réseau National de recherche et d'innovation en Technologies Logicielles, 2002-2006), porté par le LINA.

Par la suite, j'ai effectué un post-doctorat d'un an à l'Université polytechnique de Valencia en Espagne au cours duquel j'ai particulièrement porté mon attention sur *la gestion dynamique de mémoire pour les systèmes temps réel* et sur le développement d'intergiciels sous Linux temps réel. Mes activités étaient soutenues par le projet européen FRESCOR (*Framework for Real time Embedded Systems based on COntRacts*, projet IST – Information Society Technologies, 2006-2009) coordonné par l'Université de Cantabria (Espagne).

En tant que MCF en 2007, j'ai intégré l'équipe de recherche ATLAS-GDD (Gestion de Données Distribuées) du LINA (l'axe "Ordonnancement temps réel" ayant rejoint l'IRCCyN, et plus précisément l'équipe "Systèmes Temps Réel" en 2005). L'équipe développait alors deux axes de recherche : d'une part *l'ingénierie des modèles*, et d'autre part, *la gestion transparente des données en environnement distribué*. J'ai ouvert une nouvelle thématique qui m'a fait opérer une reconversion : *la gestion de données partagées à base de mémoires transactionnelles pour des systèmes temps réel multicoeur*. Ce thème de recherche s'est tout naturellement rattaché au deuxième axe de l'équipe. Cependant, suite à 4 départs de membres permanents travaillant dans le deuxième axe et 3 nouveaux membres recrutés sur la période 2009-2012 sur le premier axe, l'équipe a subi une restructuration profonde de ses thématiques et je me suis donc tournée vers l'IRCCyN afin de retrouver un cadre d'échanges scientifiques plus en phase avec mes thématiques. J'ai ainsi intégré l'IRCCyN au 1er janvier 2012.

Depuis lors, c'est au sein de l'équipe STR (Systèmes Temps Réel) que je développe mes activités de recherche, principalement autour de deux thèmes :

— *la synchronisation et la parallélisation d'applications temps réel multicoeur* via la fourniture de mécanismes de synchronisation adaptés aux architectures temps réel multicoeur. Dans ce cadre, j'ai co-encadré deux thèses sur la proposition de protocoles de concurrence à base de mémoire transactionnelle : celle de Toufik Sarni soutenue

en 2012 et celle de Sylvain Cotard soutenue en 2013. Par ailleurs, cette thématique a bénéficié du soutien de l'Agence Nationale de la Recherche (ANR) avec le financement du projet RESPECTED (ANR ARPEGE, Programme "Systèmes Embarqués et Grandes Infrastructures", 2010-2014) porté par l'Ecole Centrale de Nantes et auquel j'ai contribué. Concernant le problème de la parallélisation d'applications sur des systèmes temps réel multicoeur, j'ai co-encadré une thèse sur le sujet dans le domaine de la cryptographie (travaux de thèse de Mohammad Abu Taha soutenus en 2017).

— *l'autonomie énergétique des systèmes temps réel embarqués.* Garantir à la fois le respect de contraintes temporelles et d'énergie nécessite d'adresser un certain nombre de questions liées à la récolte, au stockage, et à la consommation de l'énergie. Mes travaux dans ce cadre consiste à proposer des solutions d'ordonnancement temps réel spécifiquement adaptées aux systèmes temps réel autonomes du point de vue énergétique. Les thèses de Maïssa Abdallah et Nadine Abdallah soutenues en 2014, ont contribué à l'avancée de cette thématique. Plus récemment, la thèse CIFRE de Mohamed Irfanulla Mohamed Abdulla, initiée en 2020, a pour objectif de poursuivre les travaux dans cette thématique. Cet axe de recherche a également reçu un soutien financier pour 2020-2021 au travers du projet OTREC, projet NExT (Nantes Excellence Trajectory) de l'I-SITE (Initiatives-Science Innovation Territoires Economie) financé par le Programme d'Investissements d'Avenir.

Première partie

Résumé de carrière

Chapitre 1

Présentation synthétique des activités de recherche

Sommaire

1.1 Problématiques de recherche

La terminologie *temps réel* est directement liée à l'aptitude d'un système informatique à présenter des temps de réponse (appelés aussi temps de réaction) en adéquation avec les dynamiques de l'environnement contrôlé et/ou commandé. Ces temps de réponse doivent donc être bornés et en adéquation avec la vitesse d'évolution des sollicitations émises par l'environnement extérieur. Un système *embarqué* est quant à lui une infrastructure informatique plus ou moins complexe (microcontrôleur, noeud de capteur, réseau de capteurs, etc.) qui assure une fonction dédiée conçue pour être utilisée avec une application logicielle spécifique. Ses ressources sont généralement limitées en termes de puissance de calcul, d'empreinte mémoire, ou de taille des réservoirs de stockage d'énergie (batteries ou supercondensateurs). Il présente un certain nombre de contraintes non fonctionnelles (encombrement, dissipation thermique, autonomie de fonctionnement, sécurité, portabilité, etc.). Nombre de systèmes embarqués présentent également des contraintes temporelles : on parle alors de *systèmes temps réel embarqués*.

La gestion de l'énergie électrique est un défi crucial aujourd'hui pour les systèmes temps réel embarqués. Jusqu'alors, la problématique à adresser reposait sur la *minimisation de l'énergie consommée pour maximiser la durée d'autonomie*. Cependant, l'avènement d'une nouvelle technologie, celle du *energy harvesting*, a révolutionné le marché de l'embarqué. L'idée de cette technologie est de collecter l'énergie présente dans l'environnement pour assurer son alimentation et ainsi permettre au système de fonctionner en mode de neutralité énergétique sur une durée infinie. Il existe une grande variété de sources : lumière, vibration, variation de temperature, etc. Dans ce cadre, la problèmatique à traiter n'est plus celle de l'économie d'énergie mais bien celle qui vise à *assurer une autonomie énergétique de fonctionnement* au système.

Les applications sont multiples et s'étendent du simple dispositif embarqué isolé au réseau plus complexe de capteurs sans fil. La recherche dans ce domaine pourra ainsi se voir intégrée au sein d'applications temps réel environnementales (détection de feux de forêts, enregistrements des mouvements des animaux, monitoring des niveaux de pollution), militaires (coordination des positions des fantassins, analyses de terrain, surveillance de zones

difficilement accessibles et/ou dangereuses), médicales (surveillance de fonctions humaines vitales) ou commerciales (suivi de fabrication/livraison de produits manufacturés).

Dans le contexte des systèmes temps réel embarqués à base de processeurs multicoeur, une autre problématique ouverte repose sur la manière dont sont gérés les accès concurrents aux données partagées. Jusqu'à présent, des solutions à base de verrous (ex : sémaphores, spinlocks, etc.) étaient communément adoptées dans les supports d'exécution temps réel pour garantir la cohérence des données partagées. Ces mécanismes simples de synchronisation basés sur l'attente active, peuvent cependant provoquer des interblocages par utilisation incorrecte. Si la conception de l'application l'impose néanmoins, alors un ordre unique de prises des verrous doit être défini, ce qui représente une tâche difficile pour le programmeur. Avec l'avènement du multicoeur, les solutions à base de verrouillage sont d'autant plus remises en cause qu'elles viennent réduire le parallélisme des processeurs multicoeur. Par conséquent, il est nécessaire de considérer de nouveaux mécanismes de synchronisation *non bloquants* (ex : mémoires transactionnelles), dans le but *d'exploiter au maximum le parallélisme des processeurs multicoeur.*

La recherche dans ce domaine vise non seulement les applications scientifiques hautes performances, les serveurs d'entreprise, les téléphones portables, etc., mais également dans un futur proche les services embarqués au sein des véhicules motorisés qui reposeront de plus en plus sur des calculateurs (ou Electronic Control Units – ECUs) multicoeur.

Tout au long de mon parcours recherche j'ai abordé les problématiques décrites précédemment du point de vue de *l'ordonnancement temps réel*. A partir des contraintes de ressources, de synchronisation, d'exécution, d'énergie, de temps, et selon le critères de performance de l'application, l'ordonnanceur doit déterminer à tout instant l'identité de la tâche à exécuter. Pour ce faire, il peut mettre en oeuvre un ou plusieurs *algorithmes d'ordonnancement* qui spécifient une politique d'allocation du processeur aux tâches. Les questions scientifiques liées à ces algorithmes (typologie, propriétés, etc.) représentent le coeur de mes recherches.

1.2 Vue d'ensemble des travaux de recherche

Mon domaine de recherche est centré sur des problèmes d'ordonnancement et de synchronisation dans les systèmes *temps réel* et/ou *embarqués* sur des architectures matérielles *monoprocesseur* ou *multicoeur*. Pour ces systèmes, mes travaux de recherche consistent à apporter des solutions innovantes en ce qui concernent :

— des mécanimes de synchronisation des accès concurrents mémoire adaptés aux architectures multicoeur,
— des politiques d'ordonnancement temps réel garantissant une Qualité de Service prédéfinie et une autonomie énergétique de fonctionnement.

Synchronisation et parallélisation d'applications temps réel multicoeur

Dans la cadre des travaux de thèse de *Toufik Sarni* (taux d'encadrement : 70%), nous nous sommes intéressés à l'adéquation des mémoires transactionnelles aux systèmes temps réel multicoeur, en évaluant la variabilité du temps d'exécution des transactions lors de l'accès aux ressources partagées. En se basant sur un nouveau modèle transactionnel temps réel, nous avons démontré la faisabilité d'une mémoire transactionnelle temps réel. En particulier, nous

avons fait évoluer les heuristiques du contrôleur de concurrence [Sarni 2009a, Sarni 2009b, Sarni 2009c] afin qu'elles intègrent des contraintes temps réel lors de la résolution des conflits.

L'encadrement de la thèse de *Sylvain Cotard* (taux d'encadrement : 30%) a ensuite abouti à la proposition d'un nouveau protocole de concurrence *non bloquant* offrant une garantie de progression de type *wait-free* (i.e. le progrès de *toutes* les transactions est garanti). Nous avons démontré formellement les résultats suivants [Cotard 2015] : (i) toutes les transactions se terminent systématiquement en un temps borné, (ii) le pire temps d'exécution d'une transaction est indépendant de la politique d'ordonnancement choisie, et (iii) le pire temps d'exécution d'une transaction peut être inclus dans le pire temps d'exécution de la tâche comprenant la transaction, ceci permettant de mener une analyse d'ordonnançabilité hors-ligne de l'application.

Par ailleurs, j'ai également adressé le problème de la parallélisation d'algorithmes de cryptographie pour des systèmes temps réel multicoeur d'encryptage d'images et de flux vidéos [Abu Taha 2015, Jallouli 2016, Abu Taha 2017, Abu Taha 2018, Hammoudi 2020] (travaux de thèse de *Mohammad Abu Taha*, taux d'encadrement : 30%).

Autonomie énergétique des systèmes temps réel embarqués

Dans le cadre de ma collaboration avec *Maryline Chetto* (Professeur des Universités, IUT de Nantes), j'ai travaillé sur la théorie de l'ordonnancement temps réel pour les systèmes énergétiquement autonomes [Chetto 2016]. Nous avons notamment étudié les limitations de l'algorithme *EDF (Earliest Deadline First)* dans le contexte du *energy harvesting* [Chetto 2013, Chetto 2014b] et nous avons contribué à l'analyse de faisabilité de tâches temps réel périodiques sous contraintes d'énergie [Chetto 2019].

Dans le cadre des travaux de thèse de *Maïssa Abdallah* (taux d'encadrement : 30%), nous avons considéré un système monoprocesseur dans lequel l'autonomie énergétique doit être garantie dynamiquement par un ordonnancement adéquat des tâches temps réel. Nous avons proposé de nouvelles politiques d'ordonnancement en-ligne de tâches temps réel [Abdallah 2011a, Abdallah 2012a, Abdallah 2012b, Abdallah 2013] présentant des contraintes de QoS (Qualité de Service) et d'énergie. Ces travaux permettent d'apporter des solutions à la gestion des cas de surcharge de traitement et/ou de pénurie d'énergie auxquels un système temps réel autonome énergétiquement peut avoir à faire face.

Dans la thèse de *Nadine Abdallah* (taux d'encadrement : 40%), nous avons proposé des solutions de partitionnement pour des systèmes temps réel multicoeur énergétiquement autonomes autorisant une dégradation contrôlée de la QoS [Abdallah 2011b, Abdallah 2014c, Queudet 2017], en particulier dans des phases de famine énergétique et/ou de surcharge de traitement. L'enjeu consistait à répartir les tâches sur les différents processeurs en respectant à la fois leurs contraintes temporelles et leurs contraintes énergétiques, tout en assurant l'exécution des tâches les plus critiques permettant de garantir la QoS minimale requise pour le système. Nous avons également proposé un guide de choix en vue d'aider le concepteur dans la difficile tâche du dimensionnement des réservoirs de stockage d'énergie.

Dans la thèse de *Mohamed Irfanulla Mohamed Abdulla* (taux d'encadrement : 50%) qui a démarré en juin 2020, l'objectif est d'effectuer un transfert des compétences universitaires en matière d'ordonnancement temps réel sous contraintes de qualité de service et d'énergie vers le monde industriel par l'intégration et le test de nouveaux ordonnanceurs sur une plate-forme cobotique de démonstration dotée de la technologie *energy-harvesting*. Il s'agit de montrer, sur une application réelle, l'aptitude de Linux temps réel muni de ces composants innovants, à satisfaire les spécifications applicatives en termes de temps de réponse, de qualité de service et de consommation énergétique.

Chapitre 2

Curriculum Vitæ

Sommaire

2.1 Formation et expérience Professionnelle

Audrey QUEUDET

Age : 41 ans
Mariée, 1 enfant
Grade : Maître de Conférences CN
Établissement : UFR Sciences et Techniques, Université de Nantes
Section de CNU 27
Unité de recherche d'appartenance : Laboratoire des Sciences du Numérique de Nantes (LS2N - UMR 6004)
Page personnelle : http://pagesperso.ls2n.fr/~queudet-a

2.1.1 Titres et diplômes

2007 Qualification aux fonctions de Maître de conférences
N° : MCF_2007_27_07227175277
Sections de qualification : CNU 27 et 61

2006 Doctorat de l'Université de Nantes
Spécialité : Automatique et Informatique Appliquée
Directrice de thèse : Prof. Maryline Chetto
Titre : *Ordonnancement Temps Réel avec Contraintes de Qualité de Service*
Date et lieu de soutenance : 02 octobre 2006 à l'Ecole Centrale de Nantes.

2003 DEA de l'Ecole Centrale de Nantes
Spécialité : Automatique et Informatique Appliquée
Titre : *Simulation et évaluation d'algorithmes de résolution de surcharge dans les systèmes temps réel*

2002 Diplôme d'Ingénieur de l'Ecole polytechnique de l'Université de Nantes
Spécialité : Systèmes Electroniques et Informatique Industrielle

2.1.2 Expérience professionnelle

Depuis septembre 2007 : Maître de Conférences CN section 27
Université de Nantes, Département Informatique de l'UFR Sciences et Techniques.

2006 - 2007 : Post-doctorante
Université Polytechnique de Valencia (Espagne), Departamento de Informática de Sistemas y ComputAres (DISCA), équipe GII (Grupo de Informática Industrial).
Superviseur : Prof. Alfons Crespo

2003 - 2006 : Monitrice de l'enseignement supérieur section 61
Polytech'Nantes, Département SEII (Systèmes Electroniques et Informatique Industrielle).

2.2 Informations significatives sur le déroulement de la carrière

Congés maternité

Ma première fille, Elsa, est née le 26 juin 2015. Arrêtée au début du 6ème mois de grossesse, j'ai été absente du travail durant 5 mois (07/04/2015 - 06/09/2015).

Ma seconde fille, Anaïs, est née sans vie le 14 décembre 2018. Arrêtée au début du 6ème mois de grossesse, j'ai été absente du travail durant 5 mois (07/09/2018 - 26/02/2019).

Sur les 5 dernières années, cela représente donc 10 mois d'arrêt de travail, soit l'équivalent d'une année universitaire.

Temps partiel

En décembre 2017, j'ai choisi d'exercer mon activité à temps partiel (80%) pour consacrer un jour par semaine à ma fille. Je travaille de nouveau à temps plein depuis le 01/05/2019.

2.3 Publications et production scientifique

Dans cette section, l'ensemble de ma production scientifique est listé. La Table 1 résume mes publications les plus significatives depuis ma nomination en tant que MCF.

La liste complète de mes publications scientifiques est fournie dans ce qui suit. Les doctorants associés aux différentes productions apparaissent soulignés.

1. Source : CORE Journal Rankings
2. Journal Citation Reports
3. Facteur d'impact

Type	Année	Nom	Rang [1]	Indexation
Revue	2019	SUSCOM	B	JCR [2], IF [3]=1.80
Revue	2017	J. SC	B	JCR, IF=2.157
Revue	2015	ACM TECS	A	JCR, IF=1.156
Revue	2014	IEEE TC	A+	JCR, IF=1.263
Revue	2014	J. RTS	A	JCR, IF=1.717
Type	Année	Nom	Taux d'acceptation	Distinction
Conférence	2019	ICICT	21%	*Best paper award*
Conférence	2016	IEEE ICACCI	15.5%	
Conférence	2016	ECSA	15.4%	
Conférence	2015	IEEE ICITST	24.2%	
Conférence	2012	IEEE HPCC-ICESS	26.2%	

Table 2.1 – *Récapitulatif des publications les plus significatives depuis ma nomination en tant que MCF*

2.3.1 Revues internationales avec comité de rédaction

[1] Maryline Chetto and **Audrey Queudet** : Feasibility Analysis of Periodic Real-Time Systems with Energy Harvesting Capabilities, *Sustainable Computing : Informatics and Systems*, 22 : 84-95, June 2019.

[2] Mohammed Abutaha, Safwan El Assad, **Audrey Queudet** and Olivier Deforges : Design and Efficient Implementation of a Chaos-based Stream Cipher, *International Journal of Internet Technology and Secured Transactions*, 7(2), October 2017.

[3] **Audrey Queudet**, Nadine Abdallah, and Maryline Chetto : KTS : a real-time mapping algorithm for NoC-based many-cores, *The Journal of Supercomputing*, 73(8) : 3635-3651, August 2017.

[4] Sylvain Cotard, **Audrey Queudet**, Jean-Luc Béchennec, Sébastien Faucou, and Yvon Trinquet : STM-HRT : A Robust and Wait-Free STM for Hard Real-Time Multicore Embedded Systems. *ACM Transactions on Embedded Computing Systems*, 14(4) : 1-25 December 2015.

[5] Maryline Chetto and **Audrey Queudet** : A Note on EDF Scheduling for Real-Time Energy Harvesting Systems, *IEEE Transactions on Computers*, 63(4) : 1037-1040, April 2014.

[6] Maryline Chetto and **Audrey Queudet** : Clairvoyance and Online Scheduling in Real-Time Energy Harvesting Systems, *Real-Time Systems Journal*, March 2014, 50(2) : 179-184, March 2014.

[7] **Audrey Marchand** and Maryline Chetto : Quality of Service Scheduling in Real-Time Systems, *International Journal of Computers, Communication and Control (ISSN 1841-9836)*, 3(4) : 354-366, December 2008.

[8] Maryline Chetto and **Audrey Marchand** : Dynamic Scheduling of Skippable Periodic Tasks : Issues and Proposals. *Journal of Software*, 2(5) : 44-51, November 2007.

[9] Maryline Chetto, Thibault Garcia-Fernandez, and **Audrey Marchand** : Cléopatre : Open-source Operating System Facilities for Real-Time Embedded Applications. *Journal of Computing and Information Technology*, 15(2) : 131-142, 2007.

[10] **Audrey Marchand** and Maryline Chetto : Dynamic Real-time Scheduling of Firm Periodic Tasks with Hard and Soft Aperiodic Tasks. *Journal of Real-Time Systems*, 32(1-2) : 21-47, February 2006.

2.3.2 Livres

[11] Maryline Chetto, **Audrey Queudet** : *Sistemas de tiempo real autónomos en energía*, série "Energía, Gestión de la energía en sistemas empotrados", éditions ISTE International, 132 pages, 978-1-80028-024-3, 2020.

[12] Maryline Chetto, **Audrey Queudet** : *Systèmes temps réel autonomes en énergie*, série "Gestion de l'énergie dans les systèmes embarqués", éditions ISTE Press Ltd, 140 pages, 2017.

[13] Maryline Chetto, **Audrey Queudet**, *Energy Autonomy of Real-Time Systems*, éditions ISTE Press Ltd and Elsevier Ltd, 142 pages, 2016.

2.3.3 Chapitres de livres

[14] **Audrey Queudet**, Maryline Chetto : Quality of Service Scheduling in the Firm Real-Time Systems. In *Real-Time Systems, Architecture, Scheduling, and Application, edited by INTECH*, pp. 191, 978-953-51-0510-7, April 2012.

2.3.4 Actes de conférences internationales avec comité de sélection

[15] **Audrey Queudet** and Maryline Chetto, Energy-aware Aperiodic Task Servers for Firm Real-time Energy harvesting Systems. *Proceedings of the IEEE International Conference on Green Computing and Communications (GreenCom'2020)*, Rhode Island, Greece. November 2020.

[16] Maryline Chetto and **Audrey Queudet**, EDF-based real-time scheduling for self-powered sensors : a survey of main theoretical results. *Proceedings of the IEEE International Conference on Intelligent Systems and Computer Vision (ISCV'20)*, Fez, Marroco. June 2020.

[17] Adrien Quillet, **Audrey Queudet** and Didier Lime, Analysis of Polka Contention Manager for use in Multicore Hard Real-Time Systems. *Proceedings of the 28th International Conference on Real-Time Networks and Systems (RTNS'20)*, Paris, France. June 2020.

[18] Karim Hammoudi, Mohammed Abu Taha, Halim Benhabiles, Mahmoud Melkemi, Feryal Windal, Safwan El Assad, and **Audrey Queudet**, Image-based Ciphering of Video Streams and Object Recognition for Urban and Vehicular Surveillance Services. *Proceedings of the 4th International Congress on Information and Communication Technology (ICICT'19)*, London, UK. Springer "Advances in Intelligent Systems and Computing" serie, February 2019. **Best paper award**.

[19] Mohammed Abu Taha, Safwan El Assad, and **Audrey Queudet**, Parallel Generator of Discrete Chaotic Sequences Using Multi-threading Approach. *Proceedings of the International Conference on Future Information Technology (FutureTech'2018)*, pp. 161-167, Salerno, Italy, April 2018.

[20] Ons Jallouli, Mohammed Abu Taha, Safwan El Assad, Maryline Chetto, **Audrey Queudet**, and Olivier Deforge, Comparative Study of two Pseudo Chaotic Number Generators for Securing the IoT, *Proceedings of the IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI'2016)*, Symposium on Signal Processing for Wireless and Multimedia Communications, Jaipur, India, 2016.

[21] Adel Hassan, **Audrey Queudet**, and Mourad Oussalah : Evolution style : framework for dynamic evolution of real-time software architecture, *Proceedings of*

the 10th European Conference on Software Architecture (ECSA'2016), Copenhagen, Denmark, 2016.

[22] Mohammad Abu Taha, Safwan El Assad, Mousa Farajallah, **Audrey Queudet**, and Olivier Deforges : Chaos-based Cryptosystems using Dependent Diffusion : An Overview. *Proceedings of the 10th IEEE International Conference for Internet Technology and Secured Transactions (ICITST'15)*, pp. 44-49, London, UK, December 2015.

[23] Nadine Abdallah, **Audrey Queudet**, and Maryline Chetto : Task Partitioning Strategies for Multicore Real-Time Energy Harvesting Systems. *Proceedings of the 17th IEEE Computer Society symposium on object/component/service-oriented real-time distributed computing (ISORC'14)*, Reno, Nevada (USA), June 2014.

[24] Mohamed Ould Sass, Maryline Chetto, and **Audrey Queudet** : The BGW model for QoS aware scheduling of real-time embedded systems. *Proceedings of the 11th ACM international symposium on Mobility management and wireless access (MobiWac'13)*, pp. 93-100, Barcelona, Spain, November 2013.

[25] Maissa Abdallah, Maryline Chetto, **Audrey Queudet**, and Rafic Hage Chehade : Stability and Robustness Issues in Real-time Sustainable Wireless Sensors. *Proceedings of the Green Computing and Communications conference (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom'13), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pp. 86-93, Beijing, China, August 2013.

[26] Sylvain Cotard, Sébastien Faucou, Jean-Luc Béchennec, **Audrey Queudet**, and Yvon Trinquet : A Data Flow Monitoring Service Based on Runtime Verification for AUTOSAR. *Proceedings of the 14th IEEE International Conference on High Performance Computing and Communication & 9th IEEE International Conference on Embedded Software and Systems (HPCC-ICESS'12)*, pp. 1508-1515, Liverpool, United Kingdom, June 2012.

[27] Maissa Abdallah, Maryline Chetto, and **Audrey Queudet** : Energy-aware schedulers for Real-Time Energy Harvesting systems with Quality of Service requirements. *Proceedings of the 2nd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA'12)*, Beirut, Lebanon, 2012.

[28] Maissa Abdallah, Maryline Chetto, and **Audrey Queudet**. Scheduling with Quality of Service Requirements in Real-Time Energy Harvesting Sensors. *Proceedings of the IEEE International Conference on Green Computing and Communications (GreenCom'12)*, pp. 644-646, 2012.

[29] Nadine Abdallah, **Audrey Queudet**, Maryline Chetto, and Rafic Hage Chehade : Partitioned EDF scheduling in multicore systems with quality of service constraints. *Proceedings of the 18th IEEE International Conference on Electronics, Circuits and Systems (ICECS'11)*, pp. 764-767, Beirut, Lebanon, December 2011.

[30] Maissa Abdallah, Maryline Chetto, **Audrey Queudet**, and Rafic Hage Chehade : Quality of service facilities for firm real-time energy harvesting systems. *Proceedings of the 18th IEEE International Conference on Electronics, Circuits and Systems (ICECS'11)*, pp. 768-771, Beirut, Lebanon, December 2011.

[31] Toufik Sarni, **Audrey Queudet**, and Patrick Valduriez : Real-Time Support for Software Transactional Memory. *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'09)*, pp. 477-485, Beijing, China, August 2009.

[32] Toufik Sarni, **Audrey Queudet**, and Patrick Valduriez, Software Transactional Memory : Worst Case Execution Time Analysis, *Proceedings of the 17th IEEE International Conference on Real-Time and Network Systems (RTNS'09)*, Paris, France, October 2009.

[33] **Audrey Marchand**, Patricia Balbastre, Ismael Ripoll, and Alfons Crespo : Providing Memory QoS Guarantees for Real-Time Applications. *Proceedings of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'08)*, pp. 312-317, Kaohsiung, Taiwan, August 2008.

[34] **Audrey Marchand** and Maryline Chetto, Dynamic Scheduling Periodic Skippable Tasks in an Overloaded Real- Time System, *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'08)*, Doha, Qatar, April 2008.

[35] Maryline Chetto and **Audrey Marchand** : Dynamic Scheduling of Skippable Periodic Tasks in Weakly-Hard Real-Time Systems. *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, pp. 171-177, Tucson, USA, March 2007.

[36] **Audrey Marchand**, Patricia Balbastre, Ismaël Ripoll, Miguel Masmano, and Alfons Crespo : Memory Resource Management for Real-Time Systems. *Proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS'07)*, pp. 201-210, Dresden, Germany, August 2007.

[37] **Audrey Marchand** and Maryline Chetto : QoS scheduling components based on firm real-time requirements. *Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'05)*, Cairo, Egypt, January 2005.

[38] **Audrey Marchand** and Maryline Chetto : RLP : Enhanced QoS Support for Real-Time Applications. *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pp. 241-246, Hong-Kong, China, August 2005.

2.3.5 Actes de conférences nationales avec comité de sélection

[39] Maryline Chetto and **Audrey Queudet**, Scheduling and Power Management Facilities in Linux/RTAI for Real-time Systems powered by Ambient Energy Source, *Conférence Francophone en Systèmes d'Exploitation, The 1st Embed With Linux (EWiLi) Workshop*, Saint-Malo, France, February 2011.

2.3.6 Workshops internationaux

[40] Toufik Sarni, **Audrey Queudet**, and Patrick Valduriez, Real-Time Scheduling of Transactions in Multicore Systems, *Workshop on Massively Multiprocessor and Multicore Computers*, Rocquencourt, France, February 2009.

2.4 Activités d'encadrement

2.4.1 Encadrement doctoral

Depuis mon recrutement, j'ai co-encadré 6 thèses de doctorat avec un taux d'encadrement compris entre 30% et 70%.

2020- Mohamed Irfanulla MOHAMED ABDULLA
Thèse de doctorat de l'Université de Nantes, Ecole doctorale MathSTIC
Titre : Systèmes cobotiques temps réel sous contraintes d'énergie
Encadrement : **50%** avec Maryline Chetto (PU, Univ. Nantes – 50%)
Financement : Bourse CIFRE (E-Cobot)

2014-2017 Mohammad ABU TAHA
Thèse de doctorat de l'Université de Nantes, Ecole doctorale STIM
Titre : Real-Time and Portable Chaos-based Crypto-Compression Systems for Efficient Embedded Architectures
Soutenance : 12 juillet 2017
Jury : Présidente : Danièle Fournier (PU, Univ. Toulouse), Rapporteurs : Danièle Fournier (PU, Univ. Toulouse), Laurent Nana (PU, UBO), Examinateur : Damien Sauveron (MCF HDR, Univ. Limoges).
Encadrement : **30%** avec Safwan El Assad (MCF HDR, Polytech Nantes – 40%) et Olivier Desforges (PU, INSA Rennes – 30%)
Financement : Gouvernement étranger (Palestine)
Publications : 1 revue internationale, 4 conférences internationales (dont un best paper award)

2010-2014 Nadine ABDALLAH
Thèse de doctorat de l'Université de Nantes, Ecole doctorale STIM
Titre : Partitionnement temps réel multiprocesseur sous contraintes de qualité de service et d'énergie
Soutenance : 21 février 2014
Jury : Président : Ye-Qiong Song (PU, Univ. Lorraine), Rapporteurs : Cécille Belleudy (PU, Univ. Nice), Frank Singhoff (PU, Univ. Brest), Examinateur : Claire Pagetti (IR, ONERA-ENSEEIHT Toulouse).
Encadrement : **40%** avec Maryline Chetto (PU, Univ. Nantes – 40%) et Rafic Hage Chehade (PU, IUT-Saida, Univ. libanaise, Liban – 20%)
Financement : Gouvernement étranger (Liban)
Publications : 1 revue internationale, 2 conférences internationales

2010-2014 Maïssa ABDALLAH
Thèse de doctorat de l'Université de Nantes, Ecole doctorale STIM
Titre : Ordonnancement temps réel pour l'optimisation de la Qualité de Service dans les systèmes autonomes en énergie
Soutenance : 18 juin 2014
Jury : Présidente : Annie Geniet (PU, Univ. Poitiers), Rapporteurs : Annie Geniet (PU, Univ. Poitiers), Zoubir Mammeri (PU, Univ. Toulouse III), Examinateurs : Samia Bouzefrane (MCF HDR, CNAM Paris), Daniel Simon (CR HDR, INRIA, Montpellier).
Encadrement : **30%** avec Maryline Chetto (PU, Univ. Nantes – 50%) et Rafic HAGE CHEHADE (PU, IUT-Saida, Univ. libanaise, Liban – 20%)

Financement : Gouvernement étranger (Liban)
Publications : 4 conférences internationales

2011-2013 Sylvain COTARD
Thèse de doctorat de l'Université de Nantes, Ecole doctorale STIM
Titre : Contribution à la robustesse des systèmes temps réel embarqués multicoeur automobile
Soutenance : 12 décembre 2013
Jury : Présidente : Isabelle Puaut (PU, Univ. Rennes), Rapporteurs : Jean-Charles Fabre (PU, INP Toulouse), Jean-Luc Béchennec (CR CNRS, Nantes), Examinateurs : Gaël Thomas (MCF HDR, Univ. Pierre), Invités : Rémy Brugnon (Ingénieur logiciel, Renault Guyancourt), Sébastien Le Nours (MCF, Univ. Nantes).
Encadrement : **30%** avec Yvon Trinquet (PU, Univ. Nantes – 40%) et Sébastien Faucou (MCF, IUT Nantes – 30%)
Financement : Bourse CIFRE (Renault)
Publications : 1 revue internationale, 1 conférence internationale

2008-2012 Toufik SARNI
Thèse de doctorat de l'Université de Nantes, Ecole doctorale STIM
Titre : Vers une Mémoire Transactionnelle Temps Réel
Soutenance : 16 octobre 2012
Jury : Président : Luc Bouganim (DR, INRIA Rocquencourt), Rapporteurs : Luc Bouganim (DR, INRIA Rocquencourt), Michel Banatre (DR INRIA Rennes), Examinateurs : Gaël Thomas (MCF HDR, Univ. Pierre), Maryline Chetto (PU, Univ. Nantes).
Encadrement : **70%** avec Patrick Valduriez (INRIA, LIRMM, Montpellier– 30%)
Financement : Bourse MENRT
Publications : 2 conférences internationales, 1 workshop international

2.4.2 Encadrement de stages recherche de L3 et de M2

A l'UFR Sciences et Techniques de Nantes, les étudiants de L3 informatique doivent réaliser un stage obligatoire de 8 à 10 semaines. Certains étudiants choisissent d'effectuer leur stage en laboratoire de recherche. J'ai encadré les 2 étudiants suivants :

[1] X. Heraud, *FreeRTOS : Modification de l'ordonnanceur temps réel*, L3 informatique, UFR Sciences et Techniques de Nantes, 2016 (durée : 8 semaines).

[2] P.-A. Jahan, *Etude et mise en oeuvre d'une application mobile sous l'OS Android*, L3 informatique, UFR Sciences et Techniques de Nantes, 2012 (durée : 8 semaines).

En M2 EMARO (European Master on Advanced RObotics), M2 ARIA (Automatique, Robotique et Informatique Appliquée), et et M2 CORO (COntrol and Robotics) à l'Ecole Centrale de Nantes, les stages ont une durée de 6 mois. Cette durée associée à un niveau de 3ème cycle permet de proposer des travaux de recherche allant de la théorie à l'intégration. C'est le cas des 2 stages suivants que j'ai supervisés :

[1] T. Asfouri, *Efficient storage and retrieval of motion primitives for humanoid robots*, M2 EMARO, Ecole Centrale de Nantes, 2015 (durée : 6 mois).

[2] A. Naji, *Implémentation de l'ordonnanceur EDF multiprocesseur au sein d'un RTOS conforme au standard AUTOSAR*, M2 ARIA, Ecole Centrale de Nantes, 2013 (durée : 6 mois).

2.4.3 Encadrement de séminaires bibliographiques de M2

Durant le premier semestre de leur formation, les étudiants du Master CORO (anciennement Master ARIA) doivent mener un travail de recherche bibliographique. Le tuteur accompagne l'étudiant dans la découverte de la méthodologie de la recherche en le guidant via la lecture d'articles scientifiques en lien avec le sujet proposé.

[1] R. Guillemot et S. Mudaliar , *Improving the efficiency frequency scaling mechanism for the Linux SCHED_ DEADLINE scheduler*, Master CORO (COntrol and Robotics), Ecole Centrale de Nantes, 2020 (durée 16 semaines).

[2] V.-P. Diwakar et V. Prasanna Kumar, *Towards autonomous energy +for smart objects in the IoT*, Master CORO (COntrol and Robotics), Ecole Centrale de Nantes, 2020 (durée 16 semaines).

[3] H. Khamila, *Etat de l'art des solutions d'ordonnancement dédiées aux systèmes temps réel à criticité mixte*, Master ARIA (Automatique, Robotique et Informatique Appliquée), Ecole Centrale de Nantes, 2018 (durée 16 semaines).

[4] M. Et-Tayal, *Étude des algorithmes d'ordonnancement multicoeur existants pour les systèmes temps réel à criticité mixte*, Master ARIA (Automatique, Robotique et Informatique Appliquée), Ecole Centrale de Nantes, 2018 (durée 16 semaines).

2.4.4 Encadrement de Travaux d'Etudes et de Recherche de M1 et M2

Ces projets de Travaux d'Etudes et de Recherche (TER) en M2 ALMA, M1 CORO, et M1 EEA-Automatique sont constitués de 2 à 6 étudiants qui travaillent sur un sujet un jour par semaine durant le premier semestre de leur formation.

[1] V.-P. Diwakar et V. Prasanna Kumar, *Operating Systems for Real-time Processing*, TER du Master CORO (COntrol and Robotics), Ecole Centrale de Nantes, 2020.

[2] N. Kalsi et R. Guillemot, *Measuring Power Consumption on the Linux operating system*, TER du Master CORO (COntrol and Robotics), Ecole Centrale de Nantes, 2020.

[3] E. Lejeune et S. Sarkar, *Bringing real-time capabilities to ROS*, TER du Master CORO (COntrol and Robotics), Ecole Centrale de Nantes, 2020.

[4] C. Bauchet, J. Lenormand, S. Harfouche, H. Hamrouni, H. Benayed et M-T. Le Treguer, *Mise en oeuvre et test de performance de Linux temps réel sur une carte embarquée Raspberry Pi 3*, TER du Master ALMA (Architectures Logicielles Distribuées), UFR Sciences et Techniques de Nantes, 2017.

[5] M. Bobin, T. Jarry, D. Maussion, Y. Noblet, Y. Ouchala, et T. Pichaud, *Etude et mise en oeuvre de RTOS sur carte Rapberry Pi 1 type B*, M2 ALMA (Architectures Logicielles Distribuées), UFR Sciences et Techniques de Nantes, 2016.

[6] D. Bordet, A. Giraudet, et J.-C. Guérin, *FreeRTOS : Modification de l'ordonnanceur temps réel*, TER du Master ALMA (Architectures Logicielles Distribuées), UFR Sciences et Techniques de Nantes, 2016.

[7] C. Alaoui et M. Moussaoui, *Développement d'applications temps réel embarquées sous Raspberry Pi*, TER du Master EEA-Automatique, UFR Sciences et Techniques de Nantes, 2016.

[8] C. Claus et D. Combes, *Temps réel sur un système embarqué Raspberry Pi*, TER du Master EEA-Automatique, UFR Sciences et Techniques de Nantes, 2016.

[9] A. Diouf et T. Ciceron, *ChibiOS/RT : Modification de l'ordonnanceur temps réel*, TER du Master EEA-Automatique, UFR Sciences et Techniques de Nantes, 2016.

[10] N. Catin, M. El Marzgioui, F. El Yakoubi, B. Grouhan, et A. Guilbaud, *Etude d'un mécanisme alternatif aux sémaphores pour la gestion des variables partagées sous Linux temps réel multiprocesseur*, M2 ALMA (Architectures Logicielles Distribuées), UFR Sciences et Techniques de Nantes, 2015.

[11] G.-R. Mukhtar et M. Mizi, *Development and test of a real time scheduler for autonomous energy harvesting systems*, M1 ARIA (Architectures Logicielles Distribuées), Ecole Centrale de Nantes, 2014.

[12] F. Menguy et D. Solet *Etude d'un mécanisme alternatif aux sémaphores pour la gestion des variables partagées sous Linux temps réel multiprocesseur*, TER du Master EEA-Automatique, UFR Sciences et Techniques de Nantes, 2014.

[13] P. Lecointre et K. Xue, *Parallélisation de code sous Linux temps réel multiprocesseur*, TER du Master EEA-Automatique, UFR Sciences et Techniques de Nantes, 2014.

[14] G. Leray et A. Proust, *Mise en oeuvre et exploitation sous Linux temps réel d'une carte embarquée de type "ultra low-power"*, ETN 5ème année, Polytech'Nantes, 2012.

[15] A. Bartholomé, F. Coulon, P. Moueza et F. Treguer, *Applications temps réel sur smartphones : Etude de l'OS Android*, TER du Master ALMA (Architectures Logicielles Distribuées), UFR S&T Nantes, 2011.

[16] F. Btissam et B. Cruz, *Solutions Temps Réel de Gestion de la QoS des Services Webs*, TER du Master ALMA (Architectures Logicielles Distribuées), UFR Sciences et Techniques de Nantes, 2011.

[17] B. Castaing, *Évaluation de performance d'une bibliothèque de composants logiciels fonctionnant sous Linux/RTAI*, TER du Master EEA-Automatique, UFR Sciences et Techniques de Nantes, 2011.

[18] F. Dumont et M. Altuntas, *Solutions Temps Réel de Gestion de la QoS des Services Webs*, TER du Master ALMA (Architectures Logicielles Distribuées), UFR Sciences et Techniques de Nantes, 2010.

[19] Y. Morice et G. Soudine, *Etude de l'implémentation d'un ordonnanceur temps réel sous Linux*, TER du Master EEA-Automatique, UFR Sciences et Techniques de Nantes, 2010.

[20] Y. Coutard et T. Prouin, *Mise en oeuvre du RTOS Xenomai - Etude de portabilité des applications Temps réel*, TER du Master EEA-Automatique, UFR Sciences et Techniques de Nantes, 2009.

[21] W. Maldonado, *Concurrent Programming using Transactional Memory (TM)*, Master européen EMOOSE (European Master in Object-, component-, aspect-, Oriented Software Engineering technologies), Ecole des Mines de Nantes, 2008.

2.5 Diffusion, Rayonnement, Visibilité

2.5.1 Distinctions

26 février 2019 : Best-paper award à la conférence internationale ICICT'19 (4th International Congress on Information and Communication Technology, London, UK – *Acceptance rate : 21%*).

2016-2020 : Titulaire de la Prime d'encadrement doctoral et de recherche (PEDR).

2.5.2 Activités de relecture

Je suis relectrice pour des conférences et revues d'audience internationale dont :

— International Conference for Internet Technology and Secured Transactions.
— International Conference on Emerging Technologies and Factory Automation.
— Journal of Real-Time Systems.
— IEEE Transactions on Industrial Informatics.
— IEEE Transactions on Computers.

2.5.3 Valorisation de la recherche

2013 – Paquetage logiciel (code source + plans de test) développé dans le cadre des travaux de thèse de Sylvain Cotard.

Logiciel : Protocole de concurrence en langage Promela (Process or Protocol Meta Language) permettant de décrire des modèles de systèmes concurrents afin de vérifier formellement leur comportement avec l'outil de model-checking SPIN

Site web : `http://stmhrt.rts-software.org`

2006 – Bibliothèque de composants logiciels développée dans le cadre du projet RNTL CLEOPATRE

Logiciel : Modules d'ordonnancement, mécanismes de synchronisation et de tolérance aux fautes de niveau noyau, compatibles avec l'OS temps réel Linux/RTAI.

Site web : `http://cleopatre.rts-software.org`

2.5.4 Jurys et invitations

Jurys de thèse (hors étudiants encadrés)

24 septembre 2018 – Membre invité dans le jury de thèse de **Adel Hassan**
Etablissement : Université de Nantes / LS2N
Titre : Style and Meta-Style : Another way to reuse Software Architecture Evolution
Directeur : Mourad Oussalah (PU, Univ. Nantes)

29 mai 2018 – Rapporteur dans le jury de thèse de **Antonio Barros**
Etablissement : Faculdade de Engenharia da Universidade do Porto, Portugal
Titre : Real-Time Software Transactional Memory
Directeur : Luis Miguel Pinho (Research Associate, CISTER, ISEP/IPP, Portugal)

28 avril 2016 – Examinatrice dans le jury de thèse de **Mariem Abid**
Etablissement : INSA de Rennes / IETR
Titre : System-Level Hardware Synthesis of Dataflow Programs with HEVC as Study Use Case
Directeurs : Olivier Déforges (PU, INSA de Rennes), Mohamed Abid (PU, Ecole Nationale d'Ingénieur de Sfax, Tunisie)

4 juin 2012 – Examinatrice dans le jury de thèse de **Hui Zhang**
Etablissement : Université de Nantes / IRCCyN
Titre : Gestion de l'énergie renouvelable et ordonnancement temps réel dans les systèmes embarqués
Directrice : Maryline Chetto (PU, Université de Nantes)

Médiation scientifique
En décembre 2020, j'ai accepté d'être **Ambassadrice Declics**. Mon rôle était de participer à une action de vulgarisation dans les lycées nantais. L'idée était de rencontrer des groupes de lycéens pendant une dizaine de minutes chacun, tout cela pendant 1h30, afin de leur parler de mes activités de recherche, dans une rencontre de type speed dating. Il y a eu par la suite un temps de discussion avec leurs professeurs.

Conférence invitée
J'ai été invitée en 2012 à donner une conférence dans le cadre des rencontres organisées par la Cantine Numérique de Nantes (`http://cantine-nantes.org`). Des rencontres entre le monde industriel et le monde de la recherche académique y sont régulièrement organisées. Typiquement un exposé d'une heure accessible à tous suivi d'une heure de discussion avec les participants permet de découvrir un sujet de vulgarisation ou autre. Dans ce cadre-ci, j'ai animé l'une des réunions, en présentant mes propres activités de recherche.

Journées thématiques

17 janvier 2012 – Journée thématique "Temps réel et énergie"
Organisateur : Groupe ACTRISS (Action Temps Réel : Infrastructures et Services Systèmes) soutenu par le GDR ASR (Architecture Systèmes Réseaux)
Présentation : Qualité de Service des Systèmes Temps réel Embarqués sous Contraintes d'Energie

1-2 février 2007 – Jornadas de Tiempo Real
Organisateur : Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial (ESAII). Universitat Politècnica de Catalunya (UPC), Espagne
Présentation : Gestión de Memoria Dinámica en Sistemas de Tiempo Real

30 septembre 2004 – Journée "Logiciel libre et temps réel embarqué, exemples de réussite en robotique"
Organisateur : Département Organisation et Gestion de la Production (OGP) - IUT de Nantes, site de La Chanterie, Nantes
Présentation : OS temps-réel libres : le point

4 juillet 2002 – Journée "Temps réel et production automatisée – Applications et perspectives"
Organisateur : Laboratoire d'Informatique de Nantes Atlantique (LINA), IUT de Nantes, site de la Fleuriaye, Carquefou
Présentation : Contrôle-commande d'un robot mobile avec Linux temps réel embarqué

2.5.5 Activités d'expertise

2.5.5.1 Comités de sélection

J'ai été membre de 7 comités de sélection MCF et PRAG depuis 2012 :
2021 : Poste MCF, 27ème section, Université de Lille,
2019, 2014 : Poste MCF 0964 et MCF 0914, 27ème section, ENSMA à Poitiers,
2018 : Poste MCF 1843, 27ème et 61ème section, Polytech'Nantes,
2017, 2012 : Poste MCF 4477 et MCF 1060, 27ème section, ENSIP à Poitiers,
2017 : Poste PRAG 4424, Mathématiques pour l'informatique, IUT de Nantes,

2.5.5.2 Comités de suivi de thèse

2010-2013 – Rémy Pottier
Etablissement : Ecole des Mines de Nantes-INRIA / LINA
Titre : Administration de grilles virtualisées : Une approche langage pour la gestion de grilles virtualisées
Directeurs : Jean-Claude Royer (PU, EMN), Jean-Marc Menaud (Maître Assistant, EMN)

2009-2012 – Hussein El Ghor
Etablissement : Université de Nantes / IRCCyN
Titre : Load Distribution in Distributed Real-Time Systems with Energy Constraints
Directeurs : Maryline Chetto (PU, Université de Nantes), Rafic Rafic Hage Chehade (PU, IUT-Saida, Univ. libanaise, Liban)

2009-2012 – Hui Zhang
Etablissement : Université de Nantes / IRCCyN
Titre : Gestion de l'énergie renouvelable et ordonnancement temps réel dans les systèmes embarqués
Directrice : Maryline Chetto (PU, Université de Nantes)

2.5.5.3 Expertise de demandes de promotion

J'ai été évaluatrice dans le cadre des campagnes 2012 et 2018 d'avancement de grade des enseignants-chercheurs de l'Université de Nantes. J'ai émis un rapport et un avis sur 2 dossiers de demande d'accès à la hors classe des maîtres de conférences pour le secteur Sciences-Santé.

2.5.6 Activités d'animation

2021 – General chair de la conférence internationale Real-Time Networks and Systems (RTNS – `https://rtns2021.univ-nantes.fr/call-for-papers`). Je suis en charge de l'organisation de cette conférence internationale qui aura lieu en 100% virtuel (mise en place du site web, établissement du budget, demande de subventions, demande d'ouverture de proceedings auprès d'ACM, choix de la plate-forme et des outils, gestion des inscriptions, etc.).

2021 – Membre du comité de programme de la conférence internationale Real-Time Systems Symposium (RTSS – `http://2021.rtss.org/organizers`).

2020 – Membre du comité de programme de la conférence internationale RTNS (`https://rtns2020.inria.fr/committees`).

Depuis 2019 – Membre du groupe de travail égalité femmes/hommes du LS2N. Ce groupe de travail s'intéresse à 3 sujets différents : (i) l'égalité professionnelle entre femmes/hommes, (ii)la féminisation des métiers de "l'informatique", et (iii) le lien "Genre et recherche". Pour ma part, j'interviens principalement sur le 2ème sujet en participant à un certain nombre d'actions comme la présentation de mon métier à des lycéen(ne)s, pour montrer que l'informatique n'est pas réservée aux hommes.

2012 – Responsable du montage et de l'animation d'une exposition de vieux matériels informatique dans le cadre de la Fête de la Science : micro-ordinateurs, PCs, Macs, serveurs, supports de stockage. J'ai dû gérer le prêt et la restitution de ce matériel

auprès de la Mission de sauvegarde du patrimoine scientifique et technique contemporain de l'Université de Nantes.

2003 – Membre du comité d'organisation de la journée de rentrée de l'Ecole Doctorale STIM (Sciences et Technologies de l'Information et des Matériaux). Lors de cette journée qui s'est déroulée le 23 octobre 2003, j'ai participé activement à l'animation de la table ronde réunissant les témoignages de jeunes docteurs issus de l'ED STIM.

2.5.7 Collaborations extérieures au laboratoire

- **Luis Miguel Pinho** (Research Associate) et **Patrick Meumeu Yomsi**, (Research Scientist), CISTER (Research Centre in Real-Time and Embedded Computing Systems), ISEP/IPP (Engineering School of the Polytechnic Institute of Porto), Portugal.

2.6 Responsabilités scientifiques

2.6.1 Projets partenariaux

2020-2021 Projet NExT AAP INNOVEZ Nouveaux Partenariats (*participant*).

Titre : OTREC : Optimisation et gestion en Temps Réel de la consommation Energétique d'un système Cobotique

Consortium : LS2N/Université de Nantes, E-Cobot.

Champ thématique : Industrie du futur.

Objectifs : Le projet propose un ensemble de travaux formant un continuum depuis la spécification jusqu'à l'intégration au sein d'un système d'exploitation libre de fonctionnalités de gestion intelligente de la consommation d'énergie et d'ordonnancement temps réel sous contraintes énergétiques. Le démonstrateur est composé d'un cobot (robot collaboratif de transport et de manutention de charges). L'évaluation en fin de projet portera sur le fait d'avoir pu démontrer la faisabilité de la technologie du "energy harvesting" appliquée à une plateforme cobotique.

Mon rôle : Spécification, conception et implémentation sous le système d'exploitation temps réel Linux/Xenomai des nouveaux composants. Mise en oeuvre et test des composants sur le démonstrateur. Dissémination via publications et activités de valorisation.

Montant total du projet : 175 015 € (dont 46 481 € financés)

2010-2014 Projet ANR ARPEGE RESPECTED (*participant*)

Titre : RESPECTED : Real-time Executive Support with scheduling Policies for thErmally-Constrained mulTicore EmbeddeD systems

Consortium : IRCCyN/Ecole Centrale de Nantes, LAAS rattaché au CNRS, LAAS (Laboratoire d'Analyse et d'Architecture des Systèmes)/CNRS, LEAT Laboratoire d'Electronique, Antennes et Télécommunications/Université de Nice Sophia Antipolis, SEE4SYS.

Objectifs : Implémentations efficaces d'algorithmes d'ordonnancement temps réel et gestion efficace du partage de données pour des architectures multicoeur embarquées dans les véhicules.

Mon rôle : Evaluation de la faisabilité de l'adaptation d'algorithmes d'ordonnancement multiprocesseur existants à un partage de données à base de mémoire transactionnelle. Etablissement d'un modèle de tâches temps réel transactionnel.

Montant total du projet : 555 905 €.

2011-2013 Projet PHC CEDRE GreenEmbedded (*participant*)

Titre : Green Embedded : gestion optimale de l'énergie renouvelable pour une informatique verte dans les systèmes embarqués

Consortium : Fédération AtlanSTIC (France), Université Libanaise/IUT Saida (Liban)

Objectifs : Proposition et validation (par la théorie, la simulation numérique et le prototypage) d'un gestionnaire de l'énergie consommée par une architecture matérielle de type mono-processeur dédié à un système embarqué connecté à une source d'énergie renouvelable.

Mon rôle : Implémentation et validation par la simulation numérique de nouveaux algorithmes d'ordonnancement intégrant une gestion dynamique de l'énergie.

Montant total du projet : 20 000 €.

2007-2008 Projet AtlanSTIC (*coordinatrice*)

Titre : Etude comparative des architectures logicielles des systèmes d'exploitation temps réel

Consortium : LINA (équipe ATLAntic data Systems - Gestion de Données Distribuées – ATLAS-GDD), IRCCyN (équipe Systèmes temps réel – STR), IREENA (équipe Modélisation et Conception des Systèmes Embarqués - MCSE)

Objectifs : Réalisation d'un état de l'art des principales architectures fonctionnelles et structurelles des systèmes d'exploitation temps réel du marché avec synthèse globale sous forme de modèle objet.

Mon rôle : Caractérisation des principaux systèmes d'exploitation utilisés dans les environnements temps réel et embarqués. Développement d'un outil web permettant de guider l'utilisateur dans le choix d'un système d'exploitation temps réel adapté au développement et à l'évaluation d'une application temps réel donnée. Gestion humaine et financière du projet (conduite et compte-rendus des réunions, rédaction du rapport final).

Montant total du projet : 10 000 €.

2006-2009 Projet européen IST FRESCOR (*participant*)

Titre : FRESCOR : Framework for Real time Embedded Systems based on COntRacts

Consortium : Universidad de Cantabria (Espagne), University of York (Grande-Bretagne), Scuola Superiore di Studi Universitari e Perfezionamento Sant'Anna (Italie), Kaiserslautern University of Technology (Allemagne), Universidad Politécnica de Valencia (Espagne), Czech Technical University in Prague (Republique Tchèque), Visual Tools S.A. (Espagne), Thales Communications (France), Rapida Systems Ltd (Grande-Bretagne), ENEA AB (Suède), EVIDENCE (Italie).

Objectifs : Développement d'un framework intégrant des techniques d'ordonnancement avancées directement au sein d'une méthodologie de conception des systèmes embarqués ; ce framework adressant tous les niveaux d'implémentation depuis les primitives du système d'exploitation jusqu'au niveau applicatif, en passant par le middleware.

Mon rôle : Développement d'un middleware pour gérer les interactions entre l'application (des tâches qui expriment des besoins en ressources) et le système d'exploitation temps réel sous-jacent qui fournit les ressources. Proposition de techniques innovantes en matière de gestion de la mémoire dynamique pour les applications temps réel.

Montant total du projet : 4 261 939 €.

2002-2006 Projet RNTL CLEOPATRE (*participant*)

Titre : CLEOPATRE : Composants Logiciels sur Etagères Ouverts Pour les Applications Temps Réel Embarquées

Consortium : LINA (Laboratoire d'Informatique de Nantes Atlantique)/Université de Nantes, LRV (Laboratoire de Robotique de Versailles)/CNRS, L2TI (Laboratoire de Traitement et de Transport de l'Information)/Université de Paris Nord, CEA (Commissariat à l'Energie Atomique), CRTTI (Centre de Recherche et de Transfert Technologique Industriel)/Université de Nantes, ROBOSOFT S.A.

Objectifs : Développement de composants logiciels (ordonnanceurs, mécanismes de synchronisation et de tolérance aux fautes) inter-changeables et inter-opérables pour construire à la carte un RTOS, compatible avec l'OS généraliste Linux. Fourniture d'une librairie de fonctions applicatives (vision et robotique).

Mon rôle : Développement et intégration d'ordonnanceurs avec contraintes de qualité de service sous Linux temps réel. Evaluation des performances de l'ensemble des composants logiciels intégrés.

Montant total du projet : 992 257 € (dont 236 827 € de subvention pour le LINA) .

2.6.2 Membre de conseils d'établissement

Conseil d'administration du Département d'informatique de l'UFR Sciences et Techniques
J'ai été élue pour 3 mandats successifs (2010-2012, 2014-2016, 2016-2018) de deux ans au conseil du Département d'informatique de l'UFR Sciences et Techniques de l'Université de Nantes. Le conseil de département est décisionnaire. Il seconde et oriente l'action du directeur du département élu lui aussi pour deux ans. Le conseil se réunit toutes les trois à quatre semaines.

Conseil scientifique de l'UFR Sciences et Techniques
Sur la période 2011-2013, j'ai été membre suppléante du conseil scientifique de l'UFR Sciences et Techniques. Je suis membre titulaire du conseil scientifique de l'UFR Sciences et Techniques pour le mandat 2018-2020.

2.7 Activités d'enseignement

2.7.1 Enseignements à l'UFR Sciences et Techniques

Le tableau suivant récapitule le volume d'enseignements réalisés à l'UFR Sciences et Techniques ces cinq dernières années, principalement au Département Informatique :

Année	Heures équivalent TD
2015-2016	324
2016-2017	286
2017-2018	173 *(Temps partiel 80%)*
2018-2019	73 *(Temps partiel 80% + Congés maternité et maladie)*
2019-2020	196

Table 2.2 – *Récapitulatif du service d'enseignement réalisés ces cinq dernières années*

2.7.1.1 Descriptif des enseignements réalisés

Système d'exploitation (L2). Je suis responsable depuis 2008 de ce module de L2 (entre 80 et 120 étudiants). L'objectif de cet enseignement est de présenter les concepts de base d'un système d'exploitation : gestion des processus, gestion de la mémoire, gestion de la synchronisation, gestion de E/S. Les étudiants doivent être capables d'utiliser les fonctionnalités de base d'un système Linux.

Systèmes temps réel embarqués (M1). Je suis responsable depuis 2009 de ce module de M1 ALMA (entre 15 et 30 étudiants). A l'issue de ce cours, les étudiants doivent connaître les spécificités des systèmes temps réel : leur finalité, leurs domaines d'application. Ils doivent également appréhender les enjeux liés aux systèmes embarqués : ressources matérielles limitées, faible empreinte mémoire, contraintes d'énergie, contraintes d'encombrement.

Programmation objet pour les biologistes (M2, *Présentiels+Apprentis*). J'assure les TDs et les TPs de cet enseignement de M2 Bio-informatique depuis mon recrutement en 2007 (36 étudiants). Cette formation a ouvert en alternance en 2017. Les notions introduites vont d'un niveau d'abstraction élevé (modélisation objet) à un niveau d'implémentation des concepts présentés (classes, héritage, polymorphisme et virtualité, surcharge des opérateurs, conteneurs et tri) en C++ ou Java.

Programmation objet (L3 pro, *Présentiels+Apprentis*). Depuis 2010, j'assure environ 2*22h en Licence professionnelle MiAR (Métiers de l'informatique : conception, développement et test de logiciels, parcours Applications Réparties – anciennement L3 pro SIL) à l'IUT de Nantes. Ces interventions me permettent en tant que responsable adjointe de cette licence de rencontrer tous les étudiants (groupe des présentiels et groupe des alternants) et le cas échéant, d'assurer mon rôle d'interlocutrice pour une éventuelle ré-orientation des étudiants vers des formations de l'UFR Sciences et Techniques.

DIU "Enseigner l'informatique au lycée' (*Professeurs du secondaire*). Ouverte en 2018-2019, cette **formation continue** vient répondre aux besoins de formation d'enseignants du secondaire à l'informatique dans le cadre de la mise en place de la spécialité NSI (Numérique et Sciences Informatiques) au lycée. J'ai accepté l'an dernier d'intervenir à hauteur de 33h auprès de 72 enseignants du secondaire pour les former sur le volet "Système d'exploitation". Pédagogiquement parlant, le montage de cet enseignement était particulier dans le choix des outils utilisés en TP, afin de permettre aux futurs enseignants de la spécialité NSI de réaliser leurs enseignements avec les moyens informatiques disponibles au sein de leur lycée.

M1 MEEF Informatique. J'assure environ 32h dans le Master 1 MEEF (Métiers de l'Enseignement, de l'Education et de la Formation) parcours Informatique qui a ouvert l'an dernier (une dizaine d'étudiants) et qui a pour objectif principal la préparation du CAPES "Numérique et Sciences Informatiques". La formation est dispensée au sein de l'INSPE (Institut Supérieur du Professorat et de l'Education) et est réalisée en partenariat avec l'UFR Sciences et Techniques. J'ai défini le déroulé et créé le contenu d'un nouveau module intitulé "Architectures matérielles et Systèmes".

2.7.2 Enseignements en dehors de l'UFR Sciences et Techniques

J'assure un certain nombre d'enseignements en lien direct avec ma thématique de recherche au sein de différents établissements de la place nantaise, à savoir Polytech'Nantes, l'IMT Atlantique et l'Ecole Centrale de Nantes.

Vacations à Polytech'Nantes. De 2009 à 2012, j'ai assuré chaque année 3h CM et 6h TP dans un module intitulé "Ingénierie du logiciel". Le public était des étudiants de 5ème année ETN (Electronique et Technologies du Numérique". Mes interventions concernaient une présentation des notions et outils autour de l'UML pour le temps réel.

Vacations à l'IMT Atlantique. De 2009 à 2013, je suis intervenue auprès des 5ème année du cursus ingénieur GSI (Génie des Systèmes Informatiques) en dispensant 24h eq. TD sur les systèmes embarqués au sein d'une UV intitulée "Informatique nomade". Depuis 2009, je dispense également chaque année 18h45 eq. TD d'enseignements auprès des 5ème année du cursus ingénieur AII (Automatique et Informatique Industrielle) au sein d'une UV intitulée "Informatique temps réel". Là encore, j'apporte mon expertise dans le domaine qui est directement en lien avec mes activités de recherche.

Vacations à l'Ecole Centrale de Nantes. De 2012 à 2018, j'ai effectué 6h CM et 6h TP chaque année auprès d'étudiants de M2 ARIA (Automatique, Robotique et Informatique Appliquée). Ces enseignements ciblés sur la conception des systèmes embarqués étaient en lien direct avec ma thématique de recherche. L'an dernier, j'ai participé à l'encadrement de projets de TER *en anglais* d'étudiants de M1 CORO (COntrol and RObotics). Cette année, j'ai effectué 32h eq. TD d'enseignements *en anglais* auprès des étudiants de M2 CORO.

2.7.3 Activités de tutorat

2.7.3.1 Suivi de stages en entreprise

[1] H. Khamila, *Etude de l'implémentation d'un serveur web embarqué sous RTOS*, M2 ARIA (Automatique, Robotique et Informatique Appliquée), Ecole centrale de Nantes, 2017 (durée : 6 mois).

[2] S. Falhun, *Réalisation d'une application mobile pour la gestion de notes de frais*, L3 informatique, UFR Sciences et Techniques de Nantes, 2017 (durée : 8 semaines).

[3] G. Gautreau, *Development of an R package for visualization, analysis and integration of high-dimensional cytometry data*, M2 Bio-informatique, UFR Sciences et Techniques de Nantes, 2016 (durée : 5 mois et demi).

[4] M. Feyeux, *Détection automatique de marqueurs morphocinétiques sur des vidéos embryonnaires dans le cadre d'une FIV*, M2 Bio-informatique, UFR Sciences et Techniques de Nantes, 2016 (durée : 5 mois et demi).

[5] R. Bernard, *Réalisation d'une application mobile hybride Android / iOS pour une manifestation sportive*, L3 informatique, UFR Sciences et Techniques de Nantes, 2017 (durée : 8 semaines).

[6] E. Martinet, *Application de suivi du temps de travail pour Nantes Métropole*, L3 informatique, UFR Sciences et Techniques de Nantes, 2017 (durée : 8 semaines).

[7] F. Fagniez, *Développement d'applications web à l'aide des technologies Microsoft*, M2 ALMA (Architectures Logicielles Distribuées), UFR Sciences et Techniques de Nantes, 2014 (durée : 6 mois).

[8] P.-A. Jahan, *Etude et mise en oeuvre d'une application mobile et développement de services dans le domaine de l'imagerie médicale*, M2 ALMA (Architectures Logicielles Distribuées), UFR Sciences et Techniques de Nantes, 2014 (durée : 6 mois).

[9] N. Proust, *Développement d'évolutions d'applications Java JEE dans le cadre de TMA*, M2 ALMA (Architectures Logicielles Distribuées), UFR Sciences et Techniques de Nantes, 2014 (durée : 6 mois).

[10] W. Zenad, *La migration d'une application de provisionnement en JEE*, M2 ALMA (Architectures Logicielles Distribuées), UFR Sciences et Techniques de Nantes, 2014 (durée : 6 mois).

[11] N. Rullier, *Développement d'une application de réalité augmentée*, M2 ALMA (Architectures Logicielles Distribuées), UFR Sciences et Techniques de Nantes, 2014 (durée : 6 mois).

[12] S. Viaud, *Réalisation d'une application pour mobile*, L3 informatique, UFR Sciences et Techniques de Nantes, 2010 (durée : 8 semaines).

[13] A. Lefray, *Modélisation et validation d'un simulateur de jeu d'instructions*, L3 informatique, UFR Sciences et Techniques de Nantes, 2010 (durée : 8 semaines).

[14] A. Le Govic, *Réalisation d'une base de travaux au sein d'un extranet*, L3 informatique, UFR Sciences et Techniques de Nantes, 2010 (durée : 8 semaines).

2.7.3.2 Suivi d'apprentis de L3 pro

Les apprentis sont en mission en entreprise durant 31 semaines au total (dont 11 semaines consécutives en fin de formation). Le tuteur pédagogique suit l'apprenti durant toute l'année de formation et assure les missions suivantes : rencontres régulières avec l'apprenti, visites en entreprise (au nombre de 3) réparties sur l'année universitaire, accompagnement à la réalisation du mémoire et participation aux présentations des travaux (à mi-parcours et en fin de formation).

[15] J. Martin, *Création de sites web*, L3 pro SIL, IUT de Nantes, 2017.

[16] W. Flambard, *Développement Java et tests sur plate-formes Androïd*, L3 pro SIL, IUT de Nantes, 2017.

[17] A. Cadeau, *Développement web et mobile*, L3 pro SIL, IUT de Nantes, 2016.

[18] S. Ugho, *Développements en TMA (Tierce Maintenance Appplicative)*, L3 pro SIL, IUT de Nantes, 2016.

[19] M. Zafiriou, *Développement PhP et Javascript sous DRUPAL*, L3 pro SIL, IUT de Nantes, 2014.

[20] C. Gaucher, *Analyse et mise en place d'un module de gestion de stock*, L3 pro SIL, IUT de Nantes, 2013.

2.7.4 Responsabilités pédagogiques

Responsabilité de diplôme
Depuis 2010, je suis co-responsable avec Arnaud Lanoix (MC, IUT de Nantes) de la Licence professionnelle MiAR (Métiers de l'informatique : conception, développement et test de logiciels, parcours Applications Réparties – anciennement L3 pro SIL) côté UFR Sciences et Techniques (la formation est organisée en collaboration entre le Département Informatique de l'IUT de Nantes et le Département Informatique de l'UFR Sciences et Techniques). L'ambition est de fournir aux diplômés les compétences pour intégrer les

équipes de développement et de maintenance des systèmes d'information des grands comptes et des grandes administrations. La formation est ouverte en alternance, en contrat de professionnalisation ou d'apprentissage. Les promotions sont d'environ 30 étudiants pour la formation en apprentissage et 20 étudiants pour la formation initiale.

Commissions de recrutement

Depuis 2017 : Membre de la commission de recrutement du Master Bio-informatique, UFR S&T de Nantes,

Depuis 2014 : Membre de la commission de recrutement de la L3 pro MiAR, IUT de Nantes.

Responsabilité de pôle pédagogique

Depuis 2012, je suis responsable du pôle pédagogique "Système, Réseaux et Architecture" au sein du Département informatique de l'UFR Sciences et Techniques. Chaque responsable de pôle est chargé de réunir une équipe pédagogique ayant comme objectifs de : (1) s'assurer d'une cohérence des thématiques sur les 4 années de formation L1, L2, L3, et M1, et (2) veiller à ce que la transition ancienne/nouvelle accréditation soit organisée de manière efficace dans chacune des thématiques. Lors de la mise en place de la dernière accréditation, j'ai collecté les résultats d'apprentissage attendus dans chacun des modules de formation concernés par le pôle.

Commission Lycées-Université

Depuis mon recrutement en 2007, je suis correspondante Lycées-Université pour l'UFR Sciences et Techniques. Voici les actions menées tout au long de l'année :

- Présentation de l'UFR Sciences dans les lycées (dans ou hors agglomération nantaise).
- Participation à des forums sur les formations du supérieur : salon Formathèque, salon de l'Etudiant.
- Accueil de classes du secondaire sur le campus Sciences.
- Université à l'essai : les élèves lycéens de Terminale suivent un ou deux "cours" à l'UFR Sciences et Techniques.
- Journée Portes Ouvertes : animation de conférences.
- Conférences ponctuelles et thématiques dans les lycées environnants.

Responsabilités de jurys

Depuis 2008 : Membre du jury de Master 2 Bio-informatique, UFR Sciences et Techniques, Nantes,

Depuis 2010 : Membre du jury de L2, UFR Sciences et Techniques, Nantes,

Depuis 2010 : Membre du jury de L3 pro MiAR (anciennement SIL), IUT de Nantes,

2019 : Président du jury de Baccalauréat, série S, lycée J. De Lattre de Tassigny, La Roche-sur-Yon,

2009-2017 : Membre du jury de Master 2 ALMA, UFR Sciences et Techniques, Nantes,

2013 : Président du jury de Baccalauréat, série S, lycée Nicolas Appert, Orvault,

2010-2012 : Membre du jury de Master 1 Bio-informatique, UFR Sciences et Techniques, Nantes.

Deuxième partie

Contributions à la synchronisation d'applications temps réel multicoeur

Chapitre 3

Problématique de la synchronisation en temps réel multiprocesseur

3.1 L'intérêt de la synchronisation non-bloquante

Les solutions communément rencontrées pour assurer la cohérence des ressources partagées en mémoire reposent sur des mécanismes à base de *verrouillage*. Ils permettent de garantir un accès mutuellement exclusif à une donnée (ou une structure de données). La zone de code protégée par un verrou de protection (ou *mutex*) est qualifiée de "section critique". Ce mécanisme de verrouillage garantit que toutes les tâches voient toujours une version consistante de la donnée. Malheureusement, ces mécanismes sont connus pour être sujets aux inversions de priorités, aux interblocages (*deadlocks*), aux situations de famine (*live-locks*), et ils peuvent également entraîner des dégradations de performances (ex : *lock convoy*).

Dans un contexte multicoeur, une autre problème inhérent à l'utilisation de ces mécanismes bloquants est le fait que le parallélisme de la plate-forme d'exécution peut être sévèrement impacté. Dès lors que plusieurs tâches concurrentes concourant pour un même verrou peuvent être bloquées, cela réintroduit de l'exécution séquentielle dans l'application, pourtant parallèle par construction. Une difficulté réside par ailleurs dans le choix de la granularité de verrouillage. Dans le cas d'un verrouillage *à gros grain* (*coarse-grain locking* en anglais), le verrou protège une grande portion de code. Simple à implémenter, il tend cependant à réduire le parallélisme. Lorsqu'un verrou protège une portion réduite du programme, on parle au contraire de verrouillage à *grain fin* (*fine-grain locking* en anglais). Dans ce cas, le parallélisme est moins réduit mais l'approche a tout de même des inconvénients : le codage est complexe à réaliser et la multiplication du nombre de prises de verrous peut avoir un côut non négligeable.

Pour pallier les inconvénients des verrous, des techniques de programmation non-bloquantes, ont été proposées [Fraser 2004]. Elles reposent sur des protocoles qui autorisent l'accès parallèle aux ressources tout en garantissant la cohérence de celles-ci. Ils permettent la pleine réentrance au niveau des sections critiques de code. Aucun verrou n'est explicitement utilisé par le programmeur qui se trouve totalement libéré de la difficile tâche de déclarer et placer explicitement les mécanismes de contrôle de la concurrence d'accès aux différentes ressources partagées. Ces approches reposent sur des tests de validation pour vérifier la présence ou non de conflits. Les problèmes d'inversion de priorité ou d'interblocage sont écartés grâce à la mise en oeuvre d'heuristiques de résolution des conflits permettant de choisir les tâches à privilégier pour les accès aux ressources. Ces protocoles se basent généralement sur des instructions machine de type *CAS* (*Compare-And-Swap*) ou *LL/SC* (*Load-Link/Store-Conditional*) qui permettent de comparer une valeur stockée à une adresse

mémoire donnée avec une valeur argument et, en cas d'égalité, de la modifier de manière *atomique*.

On recense trois types de stratégies non-bloquantes selon le type de garantie de progression qu'elles offrent aux tâches [Fraser 2004] :

— *obstruction-free* : cette garantie de progression assure qu'une taĉhe exécutée *en totale isolation* peut progresser. Elle représente la plus faible garantie de progression. Cette approche écarte les situations d'interblocage et d'inversion de priorité, mais n'élimine pas les phénomènes de famine (*livelock*) car deux opérations en compétition peuvent s'annuler mutuellement. Cette faible garantie est inappropriée pour des systèmes temps réel.

— *lock-free* : cette garantie de progression assure la progression *d'au moins une* tâche. Pour les autres, il n'est pas possible de définir une borne maximale sur leur durée d'exécution. Cette garantie permet d'assurer des contraintes temps réel *soft* et ces implémentations sont évaluées en termes de qualité de service (i.e. ratio du nombre de tâches exécutées dans le respect de leur échéance sur le nombre total de tâches exécutées). Le paradigme lock-free est incontestablement le plus répandu.

— *wait-free* : cette garantie de progression assure la progression de *toutes* les tâches. En pratique, l'implémentation d'algorithmes wait-free est très complexe et assez rare. Seule cette garantie peut se révéler adaptée aux systèmes temps réel *hard*.

Parmi les implémentations non bloquantes existantes dans la littérature, on distingue les algorithmes de synchronisation élaborés spécifiquement pour les accès à des structures de données simples comme les files (FIFO), les piles (LIFO), les buffers et les listes chaînées [Zhang 2019], des approches à usage plus général qui gèrent les accès concurrents en mémoire indépendamment des structures de données utilisées, telles que *les mémoires transactionnelles*.

Le concept de *mécanisme transactionnel* apparaît dès la fin des années 1970 [Eswaran 1976]. Ces mécanismes sont initialement utilisés pour gérer la cohérence des accès aux données partagées dans une base de données. Les accès aux ressources partagées sont réalisés à l'intérieur d'une *transaction*. Le terme *mémoire transactionnelle* est quant à lui introduit par Herlihy et Moss en 1993 [Herlihy 1993] et désigne une manipulation des transactions *en mémoire*. Le protocole transactionnel est indépendant de l'application et peut se présenter sous la forme d'une bibliothèque ou d'une API du système d'exploitation. Par construction, il est capable de détecter les conflits, d'abandonner puis de redémarrer une transaction, jusqu'à ce que l'on soit certain de la cohérence des données. Une fois clairement identifiées, les manipulations des ressources critiques sont encapsulées à l'intérieur de transactions, au sein de sections atomiques (étiquetées `atomic{}`) comme représenté sur la Figure 3.1. On se rend alors bien compte de l'attrait indéniable des mémoires transactionnelles pour le programmeur qui n'a plus à se soucier des problèmes liés à la synchronisation des verrous, de la granularité de prise des verrous, et qui peut s'affranchir de la difficulté à composer du code de programmation modulaire.

Le modèle de programmation et d'abstraction offert par les mémoires transactionnelles étant très séduisant, nous nous sommes intéressés à ce concept autour de deux questions clefs :

```
lock(a)
  lock(b)
    a <= f(a,b)
  unlock(b)
  lock(c)
    a <= g(a,c)
  unlock(c)
unlock(a)
```

(a) Lock-modify-unlock

```
atomic {
  a <= f(a,b)
  a <= g(a,c)
}
```

(b) Bloc transactionnel

Figure 3.1 – *Interfaces de programmation : verrous vs. blocs transactionnels*

— *les mémoires transactionnelles peuvent-elles représenter une alternative aux solutions à base de verrouillage dans un contexte temps réel* soft *et/ou* hard *?*

— *sur une plate-forme multicoeur avec gestion des ressources partagées via une mémoire transactionnelle, comment garantir pour les tâches temps réel un accès en temps borné aux ressources partagées ?*

Cette problématique est nouvelle non seulement d'un point de vue théorique mais aussi pratique car la mise en oeuvre des mécanismes de synchronisation inter-tâches demeure encore un défi et une source de difficultés pour les développeurs.

3.2 Les mémoires transactionnelles

3.2.1 La notion de "Transaction"

Le terme "transaction" a été établi par la communauté des bases de données dès 1976 [Eswaran 1976]. Il désigne les opérations apportant des modifications aux données. La définition de "transaction" est centrée sur les propriétés dites ACID [Haerder 1983] :

— *Atomicité.* Les transactions répondent à un comportement de validation de type *tout-ou-rien* : une transaction est validée en totalité (*commit*) ou bien elle est entièrement abandonnée (*abort*).

— *Cohérence.* La propriété de cohérence garantit le fait que toute transaction qui commence avec un ensemble de données cohérent (respectant les contraintes d'intégrité) doit, à l'issue de son exécution, laisser le système avec un ensemble de données également cohérent.

— *Isolation.* Cette propriété garantit que l'exécution d'une transaction semble totalement indépendante des autres transactions. Les résultats intermédiaires de chaque transaction ne doivent pas être visibles depuis les autres transactions concurrentes.

— *Durabilité.* Le système garantit que toute interruption du système survenant après la validation des transactions ne remettra pas en cause ces mises à jour.

Au sein d'une mémoire transactionnelle, une transaction est considérée comme une séquence finie d'instructions machine exécutée par une tâche donnée et satisfaisant les propriétés d'*atomicité*, de *cohérence* et d'*isolation* (la *durabilité* ne fait pas partie des propriétés associées aux transactions puisque les ressources que la mémoire transactionnelle met à jour sont des données partagées présentes en mémoire vive, mémoire volatile par nature). Afin de pouvoir garantir ces propriétés, des mécanismes de *gestion des versions de*

données et de *détection et de résolution de conflits* doivent être mis en oeuvre. Les mémoires transactionnelles se distinguent les unes des autres en fonction des implémentations qu'elles réalisent de ces différents mécanismes.

Au cours de son cycle de vie, une transaction peut manipulate plusieurs données communément classifiées comme suit :

— *read-set* : l'ensemble des données accédées en lecture par une transaction.
— *write-set* : l'ensemble des données accédées en écriture par une transaction.
— *data-set* : l'union des *read-set* et *write-set* d'une transaction.

Par ailleurs, une transaction peut passer par quatre états distincts et bien définis : *en exécution* (les opérations de la transaction sont en train de s'éxécuter), *en cours de validation* (la transaction a terminé l'exécution de ses opérations), *validée* (la transaction a été validée) et *abandonnée* (la transaction a été abandonnée). Les états *en exécution* et *en cours de validation* sont des états temporaires tandis que les états *validée* et *abandonnée* sont des états permanents.

3.2.2 Les éléments fondamentaux des mémoires transactionnelles

Gestion des versions de données

Les mémoires transactionnelles doivent implémenter une *politique de gestion des versions de données* de manière à maintenir à la fois les anciennes valeurs des données (valeurs lues au début de la transaction, nécessaires dans le cas où la transaction serait abandonnée), et les nouvelles valeurs des données (valeurs des données non validées, écrites localement par la transaction et requises si la transaction est validée). Deux approches sont communément rencontrées : *eager* et *lazy*. Dans une gestion de version *eager*, aussi appelée *directe*, les transactions modifient directement les données contenues en mémoire. Dans ce cas, un log dit de *roll-back* doit exister et contenir les valeurs qui ont été écrasées dans le cas où la transaction serait abandonnée et que le système devrait retourner dans un état cohérent. Dans le cas d'une gestion de versions *lazy*, aussi appelée *différée*, la mémoire n'est pas mise à jour tant que la transaction n'est pas validée. Dans ce cas, comme précédemment, un log est nécessaire. Qualifié alors de *redo-log*, il permet de mémoriser les mises à jour des données. Lorsque la transaction est abandonnée, le log peut être effacé. L'approche *eager* est considérée comme optimiste tandis que l'approche *lazy* est plutôt pessimiste.

Le contrôle de concurrence

Le contrôle de concurrence se rapporte à la politique de détection des conflits utilisée dans les mémoires transactionnelles. Il détermine le moment auquel les transactions sont abandonnées. Les conflits d'accès à des mêmes données peuvent être de type lecture-écriture, écriture-lecture, ou écriture-écriture. De manière similaire à la gestion des versions de données, on distingue deux approches : *lazy* et *eager* pour la détection et la résolution des conflits. Dans le cas d'une détection et d'une résolution de type *lazy*, toutes les transactions s'exécutent jusqu'à ce qu'une transaction tente de valider. Dans le cas d'une détection et d'une résolution de type *eager*, les conflits sont détectés au fur et à mesure de leur apparition. Des travaux de recherche ont montré l'intérêt de l'approche de détection de conflits *lazy* dans le sens où celle-ci favorise le parallélisme d'exécution au sein du système [Bobba 2007, Shriraman 2009]. Retarder la résolution des conflits jusqu'au moment de la validation permet de plus d'éviter la question difficile de la "meilleure" transaction à abandonner à un instant t.

Contention manager vs. Helping

Les mémoires transactionnelles délèguent la plupart du temps la tâche de *résolution des conflits* à un *gestionnaire de contention* (*contention manager* ou *CM* en anglais)[Herlihy 2003]. Le gestionnaire de contention a pour rôle de résoudre les conflits entre transactions dès lors qu'ils ont été détectés. Dans les faits, lorsqu'une trasaction détecte un conflit avec une autre transaction, elle consulte le gestionnaire de contention pour connaître l'issue du conflit. Le gestionnaire de contention détermine alors laquelle des deux transactions en conflit peut valider ses modifications, et à quel moment la transaction abandonnée peut redémarrer. Le choix de prioriser une transaction par rapport à une autre est dicté par une politique de gestion de la contention. Une plethora de politiques de gestion de la contention [Scherer III 2004] ont été proposées avec toujours le même objectif : réduire le nombre de conflits et favoriser la progression d'un maximum de transactions.

Une autre approche de gestion des conflits consiste à utiliser un *mécanime d'aide* (*Helping* en anglais). Dans ce cas, une transaction "aide" une transaction concurrente à valider [Anderson 1995]. Plus précisément, lorsqu'une transaction détecte un conflit avec une autre transaction, celle-ci réalise les mises à jour de l'autre transaction pour le compte de cette dernière. On distingue trois types d'aide : l'aide incrémentale, l'aide cyclique et l'aide basée sur les priorités [Anderson 1997]. Ce mécanisme est délicat à implémenter dans la mesure où il peut conduire à des comportements récursifs et à des situations de *livelocks*, en plus de générer un overhead potentiellement important [Harris 2014].

3.2.3 Les types d'implémentations

Il existe plusieurs techniques d'implémentation des mécanismes à mémoire transactionnelle. Un des objectifs majeurs consiste à gérer des transactions en concurrence, tout en garantissant les meilleures performances possibles (i.e. maximisation de l'utilisation des ressources matérielles offertes par les architectures multicoeur).

HTM – Hardware Transactional Memory

Les premières implémentations ayant vu le jour étaient matérielles (HTM – Hardware Transactional Memory). Le support de synchronisation transactionnel est implémenté au travers de modifications apportées aux protocoles de cohérence des caches de données afin de supporter la gestion des versions de données et la détection des conflits. L'étroite synergie du matériel entre processeur et cache permet à ces systèmes d'offrir de très bonnes performances. Néanmoins, ce type d'implémentation est limité par les capacités du matériel sous-jacent : les éléments matériels ne peuvent stocker qu'une taille fixe spécifique de données. La première implémentation matérielle [Herlihy 1993], de type lock-free, a été réalisée en 1993 par Herlihy et al.

STM – Software Transactional Memory

Entièrement implémentées de manière logicielle, les STMs offrent un cadre beaucoup plus flexible qui facilite le prototypage des protocoles de synchronisation. En revanche, les STMs ont la réputation d'être plus lentes que les HTMs. Cependant, si l'on considère les avantages conceptuels des STMs, la question de la performance est discutable dans de nombreux cas. Une première implémentation logicielle (lock-free) [Shavit 1997] a été proposée par Shavi et Touitou en 1997.

HyTM – Hybrid Transactional Memory

Les HyTMs combinent les avantages des implémentations précédentes. Dans le modèle d'implémentation HyTM, une HTM est utilisée pour offrir de bonnes performances, tandis d'une STM est là pour faire face aux cas où la HTM se trouve dans l'incapacité d'exécuter des transactions avec succès (ex : transactions de taille dépassant la capacité du cache, transactions pour lesquelles la HTM ne supporte pas les fonctionnalités (ex : préemption, opérations d'entrée/sortie, etc.). Ce schéma requiert par là même un protocole de coordination entre les transactions gérées en matériel et celles gérées en logiciel puisque potentiellement la HTM et la STM peuvent "surveiller" les mises à jour de données concernant de mêmes emplacements mémoire. Les premières implémentations HyTMs ont vu le jour en 2006 [Kumar 2006, Damron 2006].

3.3 Contributions

Dans les chapitres suivants, nous présentons nos contributions à la synchronisation à base de mémoire transactionnelle pour les applications temps réel multicoeur.

Le chapitre 4 présente nos résultats en considérant un contexte temps réel *soft*. Nous nous intéressons tout d'abord à l'identification des éléments clés de la plate-forme logicielle (système d'exploitation, ordonnanceur, allocateur mémoire) influant sur la performance des mémoires transactionnelles. Sur la base de ces observations, nous présentons une proposition de STM temps réel *soft*, la RT-STM. Nous décrivons la modélisation du système, l'architecture interne de la mémoire transactionnelle proposée, le fonctionnement de deux variantes de son protocole de concurrence, et rapportons les principaux résultats d'analyse de performance associés.

Le chapitre 5 présente nos contributions en considérant cette fois-ci un contexte temps réel *hard* appliqué au domaine de l'embarqué automobile. Après avoir listé les contraintes spécifiques à ce domaine, nous décrivons les mécanismes internes (structures de données, protocole de synchronisation) d'une nouvelle STM temps réel *hard*. Puis, nous présentons une analyse fonctionnelle et temporelle de celle-ci.

Chapitre 4

Mémoires transactionnelles pour les systèmes temps réel *soft*

Sommaire

4.1 Etude de l'adéquation des STMs aux systèmes temps réel *soft*

Dans un premier temps, nous nous sommes interrogés sur l'adéquation des mémoires transactionnelles logicielles *existantes* aux systèmes temps réel *soft*. Plus particulièrement, il s'agissait de déterminer si l'overhead d'exécution induit par les rejoues des transactions abandonnées du fait des conflits d'accès aux ressources partagées, s'avérait prohibitif à une utilisation dans un contexte temps réel *soft*.

Pour cette étude, nous avons sélectionné trois STMs différentes : la OSTM (Object-based STM) lock-free proposée par Fraser et Harris [Fraser 2007], la DSTM (Dynamic STM) lock-free introduite par Herlihy et al. [Herlihy 2003], et la STM d'Ennals [Ennals 2006] de type obstruction-free. Pour chacune d'entre elles, nous avons étudié expérimentalement leurs performances en mesurant l'influence que peuvent avoir trois éléments clés de la plate-forme d'exécution : le système d'exploitation, la politique d'ordonnancement et l'allocateur mémoire.

4.1.1 Contexte et métriques d'évaluation

La plate-forme matérielle considérée pour nos tests est dotée de deux processeurs multi-coeur Intel Core (TM) 2 Duo T7500 cadencés à 2.20 GHz, munis d'un cache L2 de 4Mo et d'une mémoire vive de 3.5Go. Dotée de la technologie *hyper-threading* permettant d'exécuter deux processeurs logiques sur une seule puce, cette plate-forme offre ainsi huit coeurs

d'exécution.

Pour évaluer la performance des STMs, des micro-benchmarks synthétiques sont couramment utilisés. Ils reposent la plupart du temps sur l'accès à des ensembles d'entiers stockés dans des structures de données plus ou moins complexes : listes à emjambements (*skip lists*), arbres rouge et noir (*red-black trees*), conteneurs (*hash sets*) ou listes chaînées (*linked lists*). Pour nos évaluations, nous avons considéré un ensemble d'entiers stockés dans un arbre rouge et noir auquel les transactions accèdent de manière concurrente. Plus concrètement, des transactions de lecture déterminent si un élément est présent dans l'ensemble (opération de *lookup*) et des transactions d'écriture ajoutent (opération *update*) ou suppriment (opération *remove*) un élément dans l'arbre rouge et noir. Le nombre de transactions générées est de l'ordre de 7x10^6 sur une durée fixée à 10s (durée considérée comme suffisante pour stabiliser les données dans le cache [Fraser 2004]). Les transactions réalisent 25% d'opérations d'écriture et 75% d'opérations de lecture. Fraser [Fraser 2004] a montré qu'une telle répartition des opérations était observée dans la grande majorité des applications. Le nombre de feuilles maximum pour l'arbre rouge et noir a été fixé à 2^4, ce qui implique une forte contention pour l'accès aux ressources partagées.

La première métrique à laquelle nous nous sommes intéressés est la gigue d'exécution maximale observée sur l'exécution de N transactions, notée $ACET_{jitter}$ et définie comme suit :

$$ACET_{jitter} = max_N\{ACET\} - min_N\{ACET\} \tag{4.1}$$

Le $ACET$ (*Average-case Execution Time*) correspond au temps d'exécution observé sur la cible matérielle.

Afin de juger plus finement de la dispersion des valeurs autour de la moyenne, nous avons également considéré le facteur de variation du $ACET$, défini de la manière suivante :

$$V = \frac{\bar{x}}{\sigma} \tag{4.2}$$

où $\bar{x}$ et σ représentent respectivement la moyenne et l'écart-type des ACET mesurées sur l'ensemble des transactions.

Nous avons fait varier le nombre de threads dans le système de 2 à 32. Chaque thread exécutant des transactions de manière continue, un nombre de threads élévé traduit une contention d'accès aux ressources partagées importante, et par conséquent, un nombre de conflits potentiellement grand au niveau des transactions.

4.1.2 Influence de l'OS

La différence principale entre un système d'exploitation temps réel (*Real-Time Operating System* ou *RTOS* en anglais) par rapport à un système d'exploitation généraliste (*General Purpose Operating System* ou *GPOS* en anglais) réside dans le fait que le RTOS doit être déterministe au niveau :

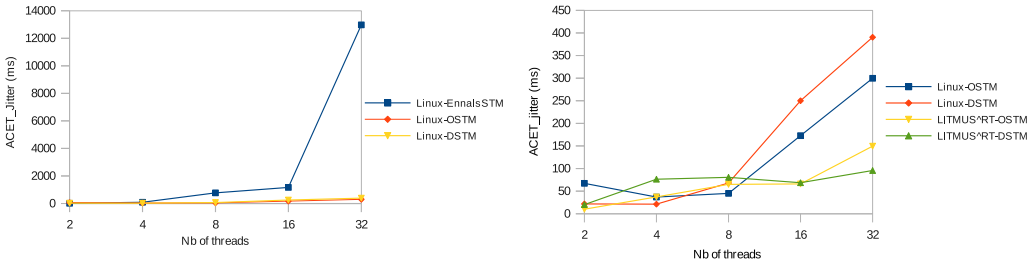
— du déclenchement des traitements ;
— des durées des traitements ;
— du service et de l'interférence du support d'exécution.

Le temps d'exécution des services d'un RTOS doit donc être fixe alors que dans le cas d'un GPOS, il peut être variable.

Nous nous sommes ici intéressés à démontrer que le système d'exploitation peut avoir un impact important sur le temps d'exécution des transactions.

La Figure 4.1(a) montre les résultats obtenus sous le système d'exploitation Linux. Nous observons que la STM d'Ennals ne passe pas à l'échelle. Ses performances se dégradent dès lors que la contention d'accès aux ressources est élevée. Une transaction peut attendre longtemps avant d'être validée. Ceci s'explique notamment par le fait qu'une transaction est systématiquement bloquée par le protocole de concurrence de la STM d'Ennals au moment de la validation dès lors que les objets accédés sont détenus par une autre transaction. Par ailleurs, comme la STM d'Ennals limite le nombre de transactions actives au nombre de coeurs de la plate-forme d'exécution, l'exécution des transactions s'en trouve retardée dès lors qu'il y a plus de transactions à exécuter que de coeurs d'exécution disponibles sur la cible matérielle. Ces résultats confirment ceux obtenus par [Dice 2006]. Par conséquent, la STM d'Ennals n'a pas été considérée pour les expérimentations suivantes.

La Figure 4.1(b) présente comparativement les résultats de gigue d'exécution sous les OS Linux et $LITMUS^{RT}$, et ce pour la OSTM et la DSTM. Les résultats nous montrent que ces deux STMs offrent des meilleures performances sous un système d'exploitation temps réel. Comme pressenti, la gigue d'exécution des transactions est moins importante sous $LITMUS^{RT}$ car par construction, les transactions souffrent moins de préemptions par les services de l'OS et/ou les autres tâches en cours d'exécution sur le système. En effet, dans un RTOS, il est capital qu'une tâche de plus haute priorité puisse préempter un appel système sous peine sinon de devoir attendre la terminaison de celui-ci et ce, même s'il avait été invoqué par une tâche de plus faible priorité. C'est pour cette raison que les RTOS intègre la préemption de ce type d'opérations dans leur design interne. Pour ce faire, la taille du noyau est généralement réduite au maximum et seules les requêtes de service courtes et/ou primordiales au fonctionnement du système y sont incluses. De cette manière, les services du noyau offrent des temps de latence réduits, ce qui contribue à assurer des temps de réponse prédictibles et bornés.



(a) *Linux* (b) *Linux* vs. $LITMUS^{RT}$

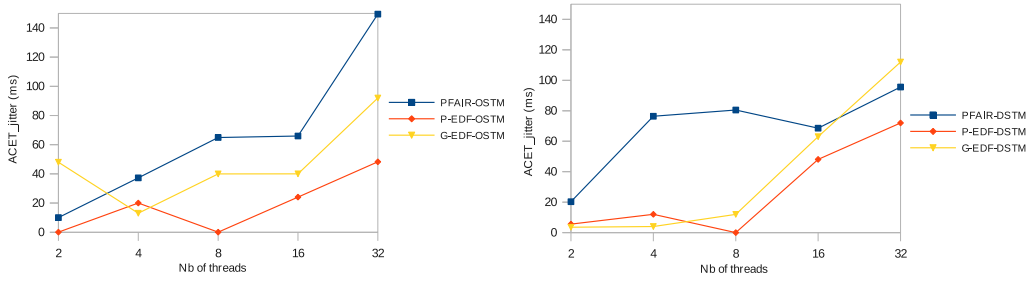
Figure 4.1 – *Influence de l'OS sur la gigue d'exécution des transactions*

4.1.3 Influence de la politique d'ordonnancement

Dans un système multi-tâches, plusieurs tâches peuvent concourir pour l'obtention du processeur. L'ordonnanceur (*scheduler* en anglais) est l'organe du système d'exploitation chargé d'allouer le processeur aux différentes tâches. C'est lui qui, suivant une politique d'ordonnancement donnée, choisit la tâche à exécuter parmi les tâches prêtes et détermine le temps durant lequel le processeur lui sera alloué. Les politiques d'ordonnancement des

GPOS sont généralement conçues de manière à optimiser les performances moyennes du système selon les objectifs suivants : une utilisation maximale du processeur, un débit maximum, un temps de rotation minimal, un temps d'attente minimal, un temps de réponse minimal. Afin d'optimiser au mieux ces critères, les GPOS mettent en oeuvre des politiques d'ordonnancement visant à une équité de service (*fairness* en anglais) entre les tâches. C'est le cas par exemple de l'ordonnanceur *Completely Fair Queuing (CFQ)* sous Linux qui maintient plusieurs files d'attente ordonnancées en round-robin [Axboe 2004]. Ce type d'ordonnancement généraliste prend en compte des notions de priorités tout en évitant les situations de famine pour les tâches. Il s'adapte parfaitement aux systèmes pour lesquels aucune information sur la charge du système n'est connue a priori. Le but d'un ordonnanceur au sein d'un RTOS est tout autre : il vise à garantir que toutes les tâches s'exécuteront dans le respect de leurs contraintes temporelles et ce, même dans le pire-cas. Les politiques d'ordonnnacement temps réel reposent le plus souvent sur des algorithmes à priorités (fixes ou dynamiques).

Nous avons considéré pour notre étude trois politiques d'ordonnancement temps réel communément rencontrées en multicoeur : *P-EDF* (*Partitioned Earliest Deadline First*), *G-EDF* (*Global Earliest Deadline First*), et *P-Fair* PD^2 (*Proportionate Fairness*) [Anderson 2000]. Pour chacune de ces politiques d'ordonnancement, nous avons à nouveau mesuré la gigue maximale $ACET_{jitter}$ des transactions sous les mémoires transactionnelles OSTM et DSTM. Les résultats obtenus sont présentés sur les Figures 4.2(a) et 4.2(b) respectivement.



(a) *OSTM* (b) *DSTM*

Figure 4.2 – *Influence de la politique d'ordonnancement sur la gigue d'exécution des transactions*

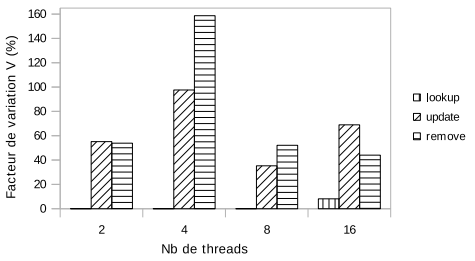
Ces résultats nous montrent que la politique *Pfair* offrent les moins bonnes performances à la fois sous la OSTM et la DSTM. Plus complexe que les autres politiques, son overhead d'exécution plus important explique ce résultat. La gigue observée est plus importante sous G-EDF que sous P-EDF, ce qui s'explique par les coûts de migration engendrés par G-EDF qui conduisent à des overheads d'exécution dans les STMs. On constate enfin sous P-EDF une inflexion naturelle des courbes dès lors que le nombre de threads dépasse le nombre de fils d'exécution offert par le plate-forme matérielle (c'est-à-dire huit). En résumé, P-EDF apparaît donc comme la politique d'ordonnancement la plus intéressante, ce qui rejoint les conclusions émises précédemment par Brandenburg et al. [Brandenburg 2008] sur les algorithmes de type *lock-free*.

4.1.4 Influence de l'allocateur mémoire

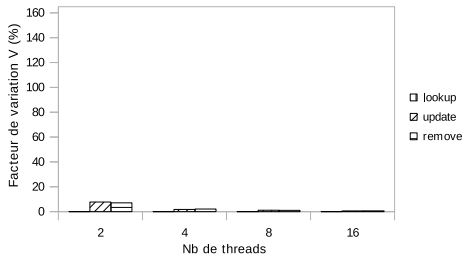
Les algorithmes d'allocation dynamique de mémoire sont des vecteurs d'indéterminisme dans le domaine du temps réel, en raison de leur durée d'exécution non bornée et de la

fragmentation du tas qu'ils génèrent. En ce qui concerne le problème de la fragmentation, celui-ci est communément résolu dans les GPOS grâce à un ramasse-miettes (*garbage collector* en anglais). Or, celui-ci n'est intrinsèquement pas déterministe donc inadapté aux applications temps réel. Certains RTOS choisissent donc de n'autoriser l'allocation dynamique de mémoire qu'en dehors du noyau. Ainsi, le noyau reste rapide et déterministe. Le seul problème est que l'utilisateur ne bénéficie plus des mécanismes de protection mémoire à l'extérieur du noyau. Une autre approche à l'allocation "dynamique" de mémoire consiste à allouer statiquement au moment de la compilation des partitions de taille fixe qui serviront à satisfaire les demandes d'allocation dynamique de mémoire à l'exécution. Cependant, les partitions doivent être soigneusement dimensionnées de manière à répondre aux besoins en mémoire dynamique au pire-cas. La dernière solution envisageable pour supporter l'allocation dynamique de mémoire pour des systèmes temps réel à contraintes strictes consiste recourir à un allocateur mémoire dont les opérations d'allocation et de désallocation s'effectuent en temps constant $O(1)$, tel l'algorithme *TLSF* (*Two-Level Segregate Fit*) [Masmano 2004] pour lequel la fragmentation maximale observée est inférieure à 25%.

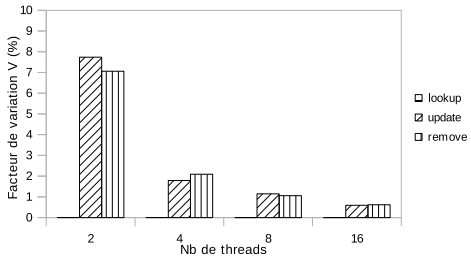
Nous avons mesuré ici facteur de variation V du $ACET$ des transactions sous la mémoire transactionnelle OSTM avec la politique d'ordonnancement P-EDF, en considérant d'une part l'allocateur de mémoire classique *malloc* (cf. Figure 4.3(a)), et d'autre part l'allocateur mémoire *TLSF* (cf. Figures 4.3(b) et 4.3(c)). Les mesures sont relevées pour les trois opérations qu'effectuent les transactions sur l'arbre rouge et noir, à savoir *lookup* pour les transactions de lecture, et *update* et *remove* pour les transactions d'écriture.



(a) Avec *malloc*



(b) Avec *TLSF*



(c) Avec *TLSF* (Zoom)

Figure 4.3 – *Influence de l'allocateur mémoire sur la variabilité du ACET*

Nous remarquons que la variabilité du temps d'exécution des transactions est beaucoup plus faible lorsque l'allocateur TLSF est utilisé. La variabilité maximale observée en utilisant malloc est de 160% contre 8% dans le cas de TLSF. Ces résultats soulignent l'importance de

sélectionner un allocateur de mémoire approprié au temps réel lorsque l'on envisage d'utiliser une mémoire transactionnelle pour la gestion de la cohérence des données partagées.

4.2 RT-STM : une STM temps réel *soft*

Sur la base de l'analyse menée précédemment, nous avons sélectionné la OSTM comme base du travail de recherche suivant consistant à étudier les adaptations nécessaires à apporter à une STM pour permettre son utilisation dans un contexte temps réel *soft*. L'objectif est d'étendre la OSTM de manière à ce que la résolution des conflits soit effectuée sur la base des paramètres temporels des tâches réalisant les transactions. La STM résultante est dénommée RT-STM (Real-Time STM).

4.2.1 Modèle du système

Le modèle du système est composé des 4 ensembles représentés sur la Figure 4.4.

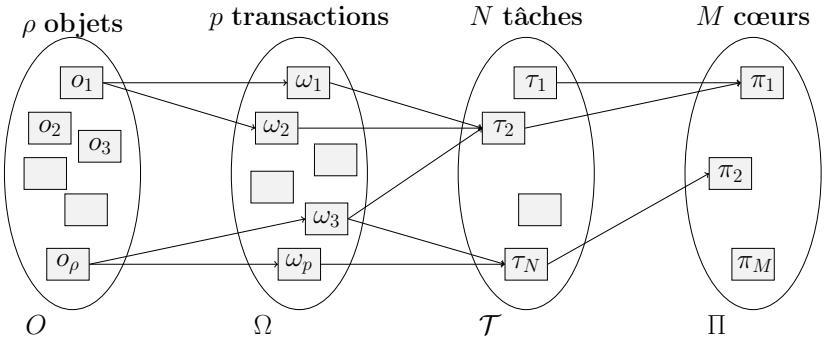


Figure 4.4 – *Modèle du système considéré*

Tous les accès aux données partagées sont effectués dans le contexte de transactions. Nous considérons que toute donnée est manipulée au travers d'un *objet* noté o_j. On considère un ensemble $\Omega = \{\omega_1, ..., \omega_p\}$ de p transactions dans le système. Une transaction ω_k est caractérisée par le tuple (e_k, O_k) où e_k est la durée maximale d'exécution de ω_k en totale isolation (i.e. pas de préemption, pas de conflits) et $O_k \in O$ est l'ensemble des objets manipulés par ω_k.

Les transactions sont allouées aux tâches. Dans le système considéré, chaque tâche est caractérisée par le tuple $(r_i, C_i, T_i, D_i, \Omega_{\tau_i})$, où r_i est la date de réveil de τ_i, C_i est son pire temps d'exécution (y compris les transactions en totale isolation), T_i est le temps d'inter-arrivée minimale entre deux activations successives de τ_i, D_i est son échéance relative et $\Omega_{\tau_i} \subseteq \Omega$ est l'ensemble des transactions qu'elle manipule. Ces paramètres sont illustrés sur la Figure 4.5. Chaque activation de la tâche τ_i donne lieu à l'exécution d'une instance (ou job) de la tâche.

4.2.2 Architecture de la RT-STM

Au sein de la OSTM (cf. Figure 4.6(a)), *l'objet* (*object*) est l'unité de base pour la concurrence. Chaque objet est pointé par un *entête d'objet* (*object header*) qui contient la version courante de l'objet. Cet entête est pointé par un gestionnaire d'objet (*object handle*) qui contient également des pointeurs vers l'ancienne (*old data*) et la nouvelle (*new data*)

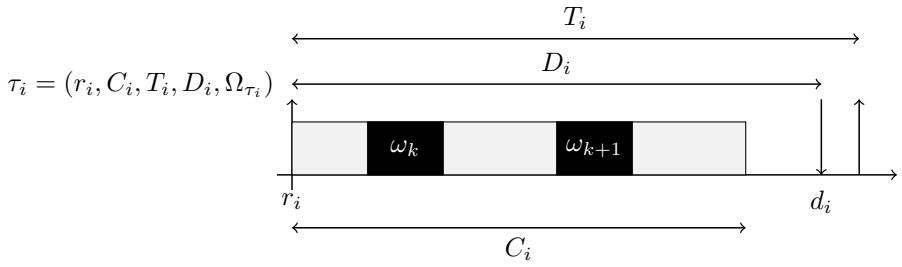
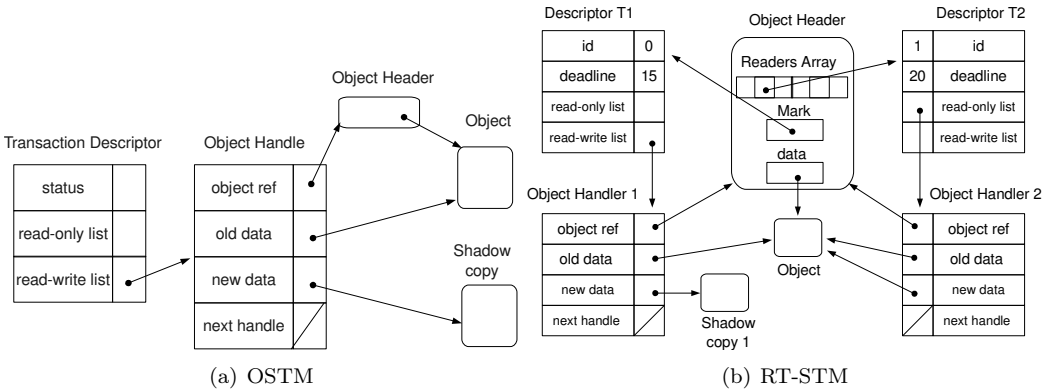


Figure 4.5 – *Modèle de tâche considéré*

référence de l'objet. Dans le cas où la transaction se termine avec succès, l'entête est mis à jour avec le nouveau bloc de données de l'objet. Le descripteur de la transaction contient à la fois la liste pour les objets manipulés en lecture seule (*read-only list*) et en lecture-écriture (*read-write list*).



(a) OSTM (b) RT-STM

Figure 4.6 – *Structures de données internes : OSTM vs. RT-STM*

Afin de résoudre les conflits sur la base des échéances des transactions, chaque descripteur de transaction est étendu avec l'ajout d'un identifiant unique *id* et d'une échéance *deadline*. Cette échéance correspond à l'échéance relative de la tâche temps réel réalisant la transaction. Nous introduisons également au sein de chaque entête d'objet : (i) un pointeur de marquage noté *Mark* et (ii) un tableau de lecteurs noté *Readers Array*. Nous décrivons ci-après le rôle de chacun de ses éléments.

Marquage d'objets

Etant donné que la RT-STM résoud les conflits entre transactions concurrentes selon leurs échéances relatives, il est nécessaire que les objets ouverts en écriture par les autres transactions soit *visibles*. A tout instant, le pointeur de marquage *Mark* de l'entête d'un objet pointe sur le descripteur de la transaction de plus grande priorité (i.e., la transaction ayant l'échéance la plus petite) manipulant l'objet en écriture. L'opération de marquage de l'objet est effectuée par les transactions d'écriture lors de l'ouverture de l'objet. Afin de réaliser ce marquage de manière atomique, nous avons implémenté une nouvelle procédure légère appelée *CAS if Greater priority* (CASG) basé sur l'instruction atomique CAS (cf. Algorithme 1). Le marquage est effectué uniquement si le contenu du pointeur de marquage

Mark est vide (ligne 3) ou bien si la transaction ω_{new} qui tente d'effectuer le marquage est plus prioritaire (lignes 5 et 9). Dans le cas où les transactions possèdent la même priorité, l'adresse mémoire du descripteur de transaction sur laquelle pointait déjà *Mark* est renvoyée afin de prévenir les situations d'interblocage (ligne 6).

Algorithm 1 *CASG*

Require: mot : *Mark, Transaction : ω_{old}, ω_{new}
Retourne : mot
1: **Init** mot :*p ⇐ MARK FAILED
2: **if** (ω_{old} = nil) **then**
3: **return** CAS(&Mark, nil, T_{new})
4: **end if**
5: **if** (ω_{new}.Deadline ≤ ω_{old}.Deadline) **then**
6: **if** ((ω_{new}.Deadline = ω_{old}.Deadline) **And** ($\omega_{old} < \omega_{new}$)) **then**
7: **return** p
8: **end if**
9: p ⇐ CAS(&Mark, ω_{old}, ω_{new})
10: **end if**
11: **return** p

Tableau de lecteurs

De manière à gérer les conflits de type lecture-écriture en fonction de l'échéance relative des transactions, nous avons choisi de rajouter un tableau statique de pointeurs au sein de chaque entête d'objet. Chaque transaction qui ouvre un objet en lecture positionne un pointeur vers son propre descripteur, à l'index correspondant à son identifiant. Le tableau n'est volontairement pas trié en-ligne selon les priorités des transactions car la dynamicité liée aux arrivées/terminaisons des transactions génèrerait un overhead trop important au sein de la RT-STM. Nous supposons cependant que les tâches partageant un objet en lecture ont des identifiants triés par ordre croissant de leur échéance relative de manière à minimiser le temps de recherche de la plus petite valeur d'échéance dans le tableau de lecteurs.

4.2.3 Protocole de concurrence de la RT-STM

RT-STM comporte trois procédures différentes : (i) une procédure pour ouvrir un objet en lecture seule, (ii) une procédure pour ouvrir un objet en lecture-écriture, et (iii) une procédure pour valider une transaction. Nous décrivons ci-après deux variantes temps réel que nous avons considérées pour l'implémentation du protocole de concurrence de la RT-STM : (i) *RT1W-STM (One-Writer)* basée sur une détection des conflits de type *eager* et (ii) *RTMW-STM (Multiple-Writers)* basée sur une détection des conflits de type *lazy*.

L'objectif des protocoles de concurrence de la RT-STM est de résoudre les conflits d'accès aux ressources partagées sur la base de l'évaluation des paramètres temporels des tâches, en garantissant prioritairement la progression des transactions possédant la plus petite échéance relative.

Le protocole mono-écrivain RT1W-STM

Le protocole mono-écrivain RT1W-STM implémente une détection de conflits de type *eager* (i.e., les conflits sont détectés dès l'ouverture des objets). Le protocole est *pessimiste*

dans le sens où il n'autorise pas l'existence d'écrivains concurrents dans le système. Seules des lectures et des écritures concurrentes sur un même objet peuvent avoir lieu. Par conséquent, c'est au moment où la transaction tente d'ouvrir un objet en écriture que le conflit est détecté. En cas de conflit (i.e. une transaction plus prioritaire a déjà marqué l'objet), la transaction devra réitérer sa tentative de marquage de l'objet. Les modifications apportées à la OSTM ont ciblé en particulier la sous-procédure de bas niveau chargée de renvoyer la version courante de l'objet, initialement communes aux deux procédures principales d'ouverture d'un objet en lecture ou en lecture-écriture. La procédure de validation a également été étendue pour résoudre les conflits de type lecture-écriture, en s'appuyant, pour chaque objet ouvert en écriture par la transaction, sur l'examen du tableau de lecteurs afin de déterminer si une transaction de plus haute priorité n'a pas également ouvert l'objet en lecture. Le cas échéant, un mécanisme d'aide entre les transactions est réalisé. A noter qu'une précaution doit être prise pour éviter la cyclicité d'aide entre transactions possédant des échéances identiques. Ceci est résolu en validant systématiquement la transaction la plus ancienne.

Le protocole multi-écrivain RTMW-STM

Le protocole multi-écrivain RTMW-STM met en oeuvre une détection des conflits de type *lazy* (i.e., les conflits sont détectés au moment de la validation des transactions). Etant donné que les conflits sont détectés à la fin de la manipulation des objets, le protocole autorise cette fois-ci la présence simultanée de plusieurs écrivains concurrents dans le système. De ce point de vue, son comportement est donc beaucoup plus *optimiste*. Le protocole RTMW-STM laisse s'exécuter les transactions sans que l'exécution de l'une n'influe sur l'exécution de l'autre. Les arbitrages entre transactions n'ont lieu qu'au moment du *commit*.

A la différence du protocole RT1W-STM, le protocole RTMW-STM ne réitère pas sur l'ouverture d'un objet en écriture si celui-ci a déjà été marqué par une transaction de plus haute priorité. Au moment de la validation, il est nécessaire de vérifier si l'objet a été marqué par une transaction d'écriture plus prioritaire. Si c'est le cas, celle-ci est "aidée". Un *Double CAS* (*DCAS*) a été introduit pour vérifier de manière atomique à la fois le pointeur de marquage et la valeur de l'objet au sein de son entête (cf. Algorithme 2). La vérification effectuée au niveau du tableau de lecteurs pour les conflits de type lecture-écriture est inchangée par rapport à celle réalisée par le protocole RT1W-STM.

4.2.4 Analyse de performance de la RT-STM

Dans cette section, nous décrivons les résultats des expérimentations menées afin d'évaluer la performance "temps réel" des protocoles de concurrence de la RT-STM par rapport au protocole non-bloquant de la OSTM originale et au protocole temps réel bloquant FMLP [Block 2007].

Contexte et métriques d'évaluation

La plate-forme matérielle utilisée repose sur un processeur à 8 coeurs Intel Xeon cadencé à 2.26GHz muni de 8MB de cache L3 et dotée de 3GB de mémoire vive. Le système d'exploitation considéré est LITMUS^{RT} (version temps réel 2.6.36, 32 bits). L'ordonnanceur P-EDF a été sélectionné. Les tâches ont été partitionnées sur les différents coeurs après avoir été triés par ordre croissant de leur échéance relative, supposée égale à leur période d'activation (i.e. tâches à échéances sur requêtes où $D_i = P_i$). Les périodes d'activation des tâches ont été générées aléatoirement dans l'intervalle $[20ms, 100ms]$ selon une distribution

Algorithm 2 *Double CASG*

Require: double mot : *pointer, double mot : $Double_{old}$, double mot : $Double_{new}$
Retourne double mot
1: **Init** *p ⇐ MARK FAILED (double mot), Transactions : ω_{old}, ω_{new}
2: **if** ($Double_{old}$.Mark = nil) **then**
3: **return** DCAS(&pointer, $Double_{old}$, $Double_{new}$)
4: **end if**
5: ω_{new} ⇐ ($Double_{new}$.Data & LSB($Double_{new}$.Data) ⇐ 0)
6: ω_{old} ⇐ ($Double_{old}$.Data)
7: **if** (ω_{new}.Deadline ≤ ω_{old}.Deadline) **then**
8: **if** ((ω_{old}.Deadline = ω_{new}.Deadline) **And** ($Double_{old}$ < $Double_{new}$)) **then**
9: **return** p
10: **end if**
11: p ⇐ DCAS(&pointer, $Double_{old}$, $Double_{new}$)
12: **end if**
13: **return** p

uniforme. Les durées d'exécution des tâches ont été calculées selon la formule $u_i = \frac{C_i}{P_i}$ où le facteur d'utilisation des tâches u_i est un paramètre d'entrée. Pour nos évaluations, nous avons à nouveau considéré un ensemble d'entiers stockés dans un arbre rouge et noir auquel les transactions accèdent de manière concurrente.

Les métriques considérées pour les évaluations sont les suivantes :
— *le temps de rejoue des transactions* par tâche (i.e. la proportion de temps gaspillé à ré-exécuter des transactions abandonnées vis-à-vis de la durée d'exécution totale de la tâche),
— *le progrès global du système* (i.e. le nombre de transactions réussies dans le respect de leur échéance sur le nombre total de transactions lancées),
— *la bande passante du système* (i.e. le nombre de transactions réussies dans le respect de leurs contraintes temporelles par unité de temps).

Les paramètres d'entrée que nous avons fait varier sont listés ci-dessous :
— *le taux de lecture* R qui définit la proportion des objets accédés en lecture seule. Ce paramètre impacte directement le degré de contention entre les transactions,
— *la durée de traitement transactionnel* e_i (i.e. la longueur ou durée d'exécution des transactions si exécutées en totale isolation). Plus les transactions sont longues, plus les conflits entre transactions peuvent être potentiellement plus nombreux.
— *le délai inter-transactions* a_i (i.e., le délai séparant l'arrivée de deux transactions consécutives au sein d'une même tâche). Plus les transactions sont temporellement distantes les unes des autres, moins les conflits sont fréquents.
— *le facteur d'utilisation* U_p du système.

Résultats de performance de la RT-STM

Temps de rejoue des transactions. Nous cherchons ici à quantifier les performances de la RT-STM du point de vue de la priorisation des accès aux ressources partagées sur la base des échéances relatives des tâches. Plus précisément, nous nous focalisons sur le ratio du temps de rejoue des transactions, individuellement pour les tâches. Nous avons choisi arbitrairement d'observer leur répartition dans le cas de contention normale (i.e. cas usuel

dans la littérature qui reflète souvent le cas pratique [Fraser 2004]) : $R = 75\%$, $U_p = 50\%$, and $a_i = 75\mu s$). Des résultats similaires ont été observés en cas de forte contention (i.e. pire-cas : $R = 0\%$, $U_p = 100\%$, et $a_i = 0\mu s$). Les tâches τ_i sont ordonnées de telle manière que $i < j$ implique que les échéances relatives des tâches sont telles que $D_i < D_j$. Les résultats obtenus pour la OSTM, la RT1W-STM et la RTMW-STM sont rapportés sur les Figures 4.7, 4.8 and 4.9 respectivement.

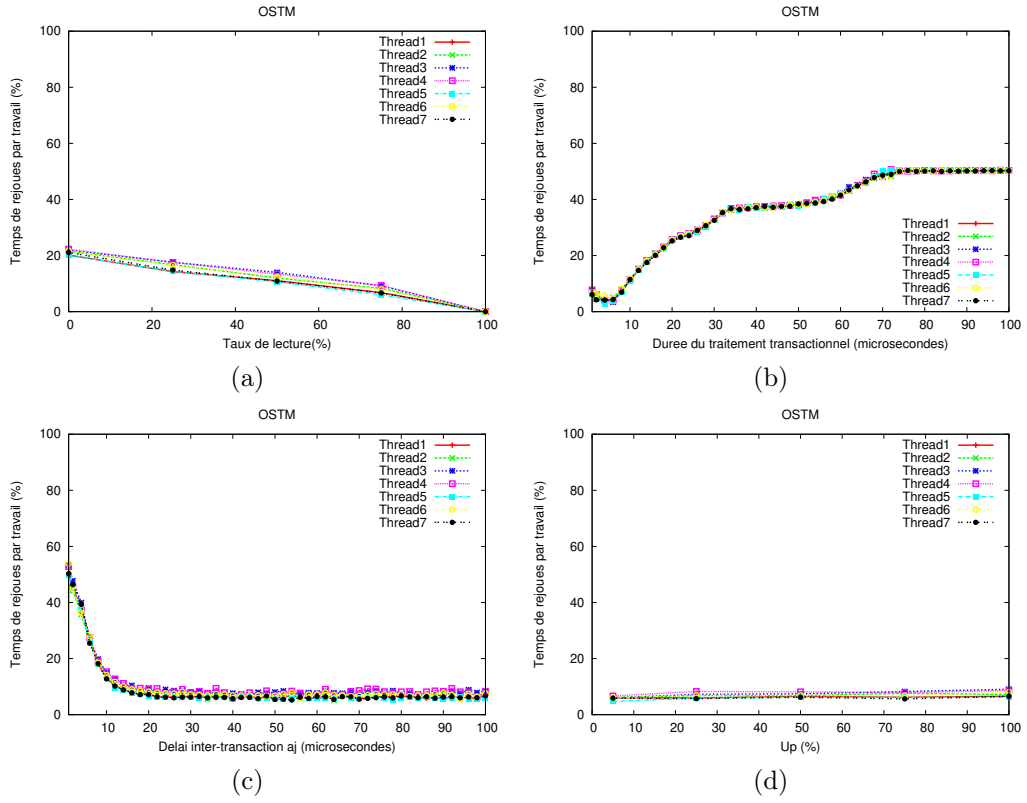


Figure 4.7 – Temps de rejoue *par tâche sous la OSTM*

Les résultats montrent que le protocole OSTM est équitable puiqu'il ne favorise pas les transactions de certaines tâches par rapport à d'autres. En revanche, pour les deux protocoles RT1W-STM et RTMW-STM, les temps de rejoue sont distribués en fonction des priorités des tâches. Les conflits sont donc bien résolus en fonction de l'échéance relative des tâches.

Progrès global des transactions. Nous nous intéressons à présent à la progression globale des transactions afin de vérifier que l'apport des fonctionnalités temps réel au sein de la RT-STM ne dégrade pas les performances globales. Les performances respectives des trois protocoles ont été étudiés pour deux configurations différentes : sous *contention normale* et sous *contention élevée*.

En cas de forte contention (cf. Figure 4.10), on observe que la RT1W-STM domine tous les autres protocoles en offrant une performance jusqu'à deux fois supérieure à celle observée pour la OSTM . La RTMW-STM quant à elle affiche la plus faible performance

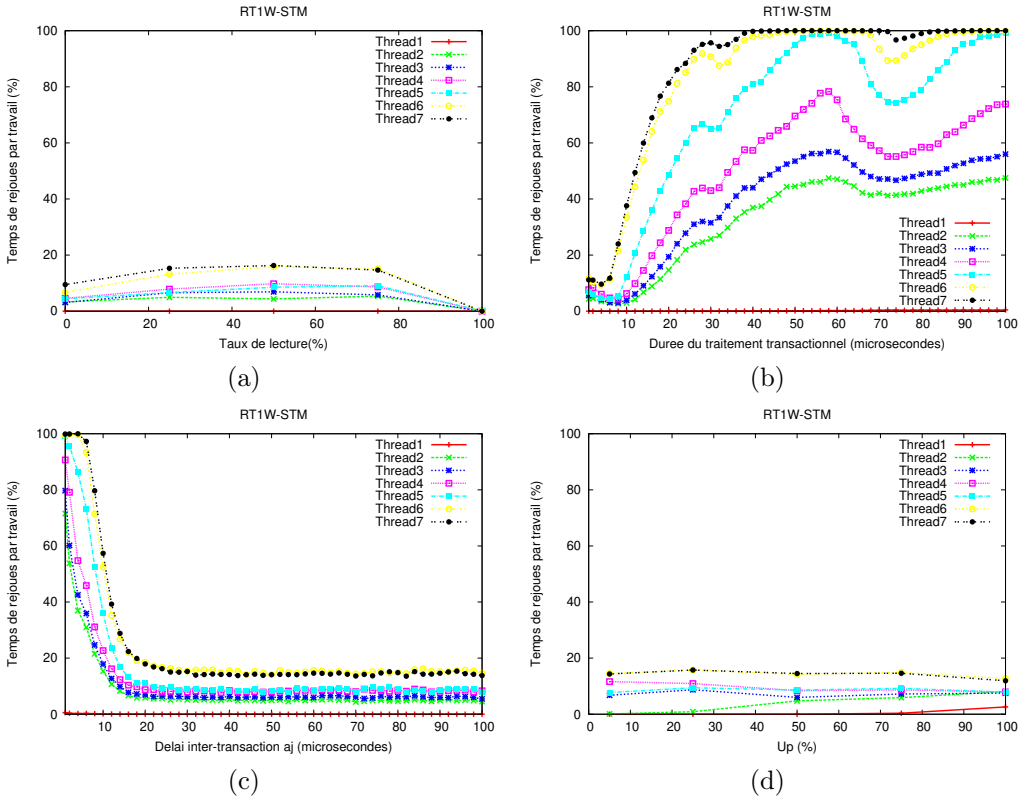


Figure 4.8 – Temps de rejoue *par tâche sous la RT1W-STM*

bien qu'offrant de meilleurs résultats que la OSTM dès que $a_i \geq 50\mu s$. Enfin, le graphique (d) nous permet de conclure que la variation du facteur d'utilisation du processeur U_p n'a pas d'impact sur le progrès global des transactions.

Dans le cas d'une contention normale (résultats rapportés dans la thèse de Toufik Sarni [Sarni 2012]), nous observons que les protocoles RT1W-STM and RTMW-STM offrent de meilleures performances que la OSTM pour des taux de lecture ne dépassant pas $R = 70\%$ et $R = 25\%$ respectivement. On observe par ailleurs que la OSTM garantit un meilleur progrès global quelles que soient les valeurs des paramètres e_i ou a_i.

Bande passante du système. La Figure 4.11 montre les résultats d'évaluation des protocoles RT1W-STM and RTMW-STM en termes de bande passante (i.e. nombre de transactions réussies dans le respect de leur échéance par unité de temps) comparativement avec le protocole non-bloquant de la OSTM et le protocole temps réel bloquant FMLP.

On remarque que plus la contention est forte, plus la performance de l'ensemble des protocoles devient médiocre. Par contre, sous plus faible contention (i.e., taux de lecture plus élevée), les protocoles non-bloquants offrent une meilleure bande passante que celle observée pour FMLP. La OSTM reste cependant toujours meilleure que la RT-STM, de part la plus faible complexité de son gestionnaire de contention.

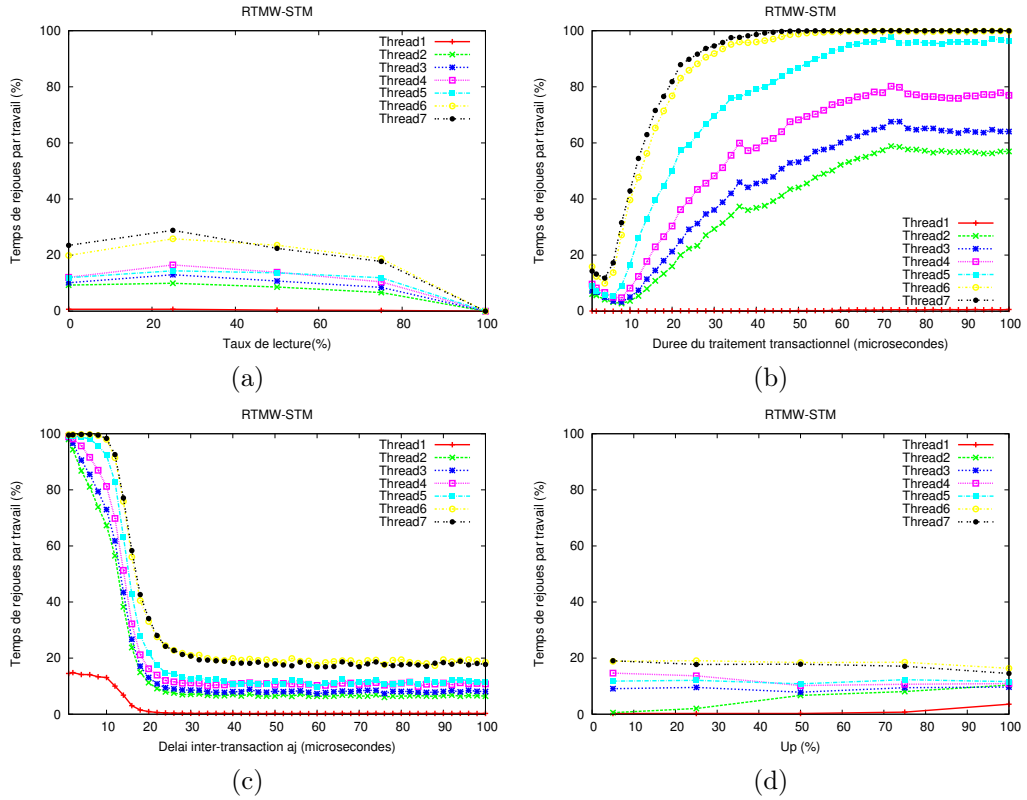


Figure 4.9 – Temps de rejoue *par tâche sous la RTMW-STM*

4.3 Bilan et diffusion des résultats

Avec l'émergence et la multiplication des systèmes multicoeur, le concept de mémoire transactionnelle a connu un regain d'intérêt. Les TMs facilitent en effet la programmation parallèle et évitent les problèmes liés aux verrous tels que les interblocages et l'inversion de priorité. Contrairement aux mécanismes basés sur du verrouillage d'accès qui abaissent le parallélisme du système, les TMs permettent à plusieurs transactions d'accéder en parallèle aux ressources partagées. L'objectif de ce chapitre était d'étudier l'adaptation des TMs à des systèmes temps réel *soft* au sein desquels les tâches doivent s'exécuter le plus souvent possible dans le respect de leurs contraintes temporelles. Jusqu'alors, l'impact de la variabilité du temps d'exécution des transactions lors de l'accès aux ressources partagées n'avait pas été étudié. Il est pourtant crucial d'évaluer l'overhead d'exécution induit par les rejoues des transactions abandonnées du fait des conflits d'accès aux ressources partagées afin d'évaluer la faisabilité de l'utilisation des TMs dans un contexte temps réel.

Dans un premier temps, nous avons proposé une étude expérimentale comparative nous permettant de statuer sur l'adéquation des TMs aux systèmes temps réel multicoeur. Nous avons mené une étude comparative de diverses mémoires transactionnelles (OSTM, DTSM et STM d'Ennals) pour lesquelles nous avons étudié expérimentalement leurs performances en mesurant l'influence du choix du système d'exploitation, de la politique d'ordonnancement et de l'allocateur mémoire.

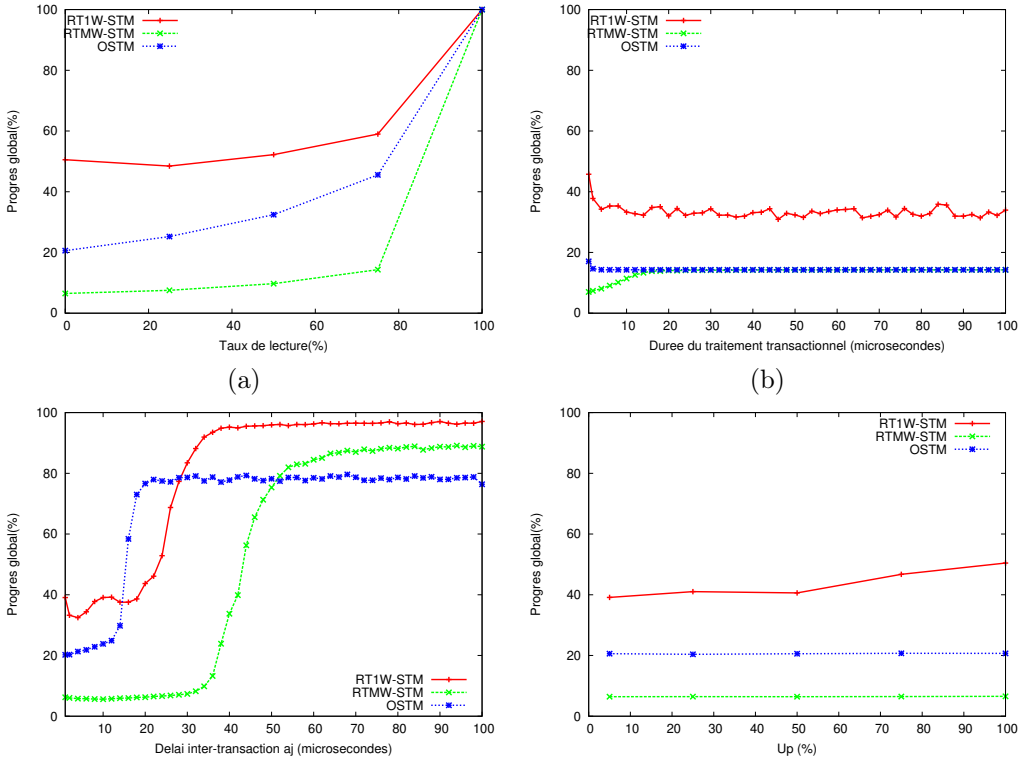


Figure 4.10 – Progrès global *sous* forte contention

Ces observations nous ont permis d'établir les conclusions suivantes :

— la gigue d'exécution des transactions est moins importante sous un RTOS tel que $LITMUS^{RT}$ étant donné que les services du noyau offrent des temps de latence réduits,

— l'algorithme *P-EDF* est la politique l'ordonnancement temps réel qui génère le moins de gigue d'exécution pour les transactions parmi *G-EDF* et *P-Fair* PD^2,

— il est préférable d'utiliser des allocateurs à temps d'allocation constant, tel que *TLSF*, pour lequel la variabilité du temps d'exécution des transactions observé est plus faible.

Dans un second temps, nous avons proposé un modèle transactionnel temps réel pour les STMs temps réel intégrant la notion d'urgence pour les transactions. Sur la base de ce modèle, nous avons proposé une extension de la OSTM, dénommée RT-STM, incluant une résolution des conflits prenant en compte les paramètres temporels des tâches réalisant les transactions. Nous avons proposé deux protocoles de concurrence pour la RT-STM. Le premier protocole, *RT1W-STM*, est un protocole mono-écrivain qui s'inspire du protocole *2PL-HP* [Abbott 1992] en intégrant en plus un mécanisme de *helping* entre transactions. Le second protocole, *RTMW-STM*, est un protocole multi-écrivain, plus optimiste que le précédent dans le sens où il autorise la présence simultanée de plusieurs écrivains concurrents dans le système. Ces deux protocoles sont à grain fin et utilisent l'instruction de base CAS pour implémenter les opérations atomiques de comparaison de données en mémoire, ainsi que des variantes que nous avons construites à savoir CASG et DCASG.

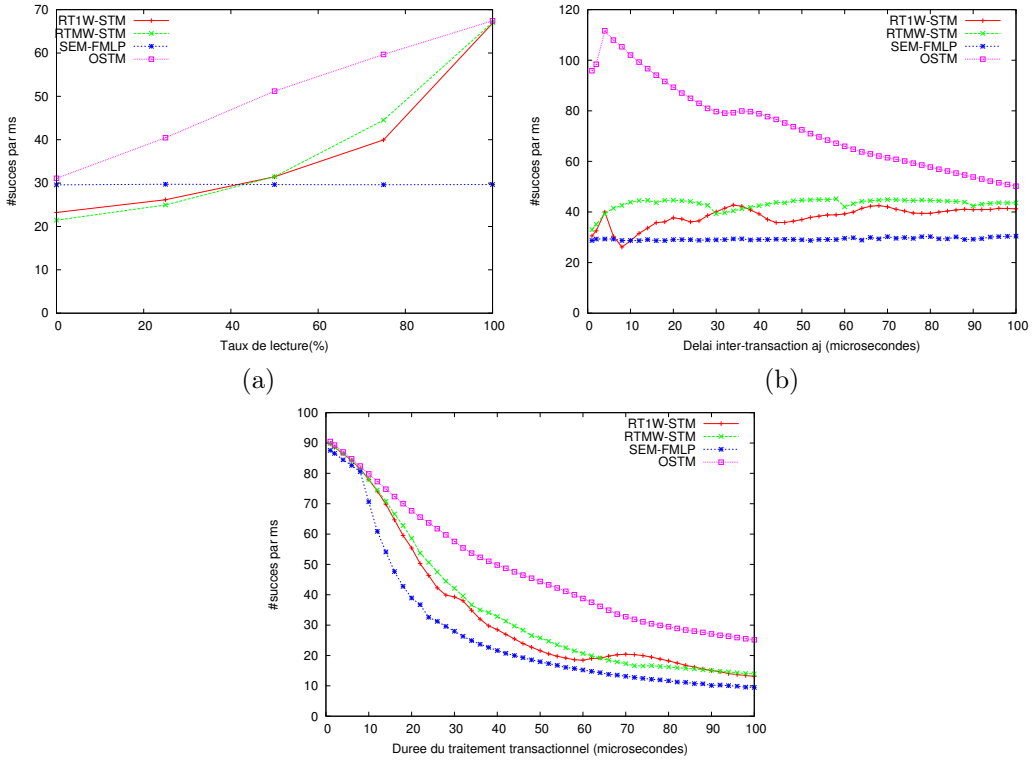


Figure 4.11 – Bande passante du système *pour* $U_p = 50\%$

Nous avons enfin mené un certain nombre de résultats expérimentaux afin de montrer le gain de performance apportée par la RT-STM par rapport à la OSTM, en évaluant notamment le nombre et la répartition par tâche des transactions exécutées dans le respect de leur échéance. Contrairement à la OSTM, nos deux protocoles *RT1W-STM* et *RTMW-STM* répartissent les temps de rejoue des transactions (et donc la bande passante) entre les tâches selon leur priorité. Les protocoles proposés ont également été comparés vis-à-vis du protocole temps réel bloquant FMLP. Sous une contention usuelle (75% de transactions de lecture), la bande passante mesurée pour la RT-STM, se situe dans une performance intermédiaire entre celles observées pour le protocole de la OSTM et le protocole FMLP.

La contribution sur la synchronisation non-bloquante en temps réel *soft* multiprocesseur à base de mémoire transactionnelle est une partie du travail de thèse de Toufik Sarni [Sarni 2012].

Les résultats de ces travaux ont été présentés lors des événements scientifiques suivants :
- *Workshop on Massively Multiprocessor and Multicore Computers* [Sarni 2009a],
- *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications* [Sarni 2009b],
- *IEEE International Conference on Real-Time and Network Systems* [Sarni 2009c].

Chapitre 5

Mémoires transactionnelles pour les systèmes temps réel *hard*

Sommaire

L'objectif ici est de fournir une solution robuste et efficace au problème de l'accès concurrent à des ressources partagées dans les contrôleurs multicoeur embarqués à contraintes temps réel dur. Les travaux menés précédemment dans la thèse de Toufik Sarni ont montré que lorsqu'une mémoire transactionnelle est utilisée pour le contrôle de l'accès aux ressources partagées en environnement multiprocesseur à contraintes temps réel dur, il est plus approprié pour des questions de performance, de faire le choix d'une politique d'ordonnancement partitionnée. C'est donc sous cette hypothèse que nous avons travaillé dans le cadre de la thèse de Sylvain Cotard (janvier 2010 - décembre 2013).

Nous présentons successivement dans cette partie : (i) les hypothèses retenues au niveau du modèle de communication en relation avec le domaine d'application, (ii) la mémoire transactionnelle logicielle proposée, et (iii) son analyse fonctionnelle et temporelle.

5.1 Contraintes du contexte embarqué automobile

Suite à des échanges avec le constructeur Renault, nous avons pu établir un certain nombre d'hypothèses réalistes à la fois sur le modèle de transactions et le modèle de communication. Les hypothèses retenues sont successivement exposées dans cette partie.

Hypothèse n° 1 : Transactions homogènes

Dans les applications temps réel embarquées communément rencontrées aujourd'hui dans le secteur automobile, la plupart des tâches temps réel manipulent les ressources partagées de la manière suivante :

1. un ensemble de ressources partagées est lu en début de tâche ; ces ressources correspondent souvent à des données issues de capteurs (ex : vitesse du véhicule, angle de direction, etc.) ;

2. un calcul associé à la fonctionnalité réalisée par la tâche est effectué sur la base des valeurs des ressources lues ;
3. un ensemble de ressources est écrit en fin de tâche ; ces ressources correspondent alors à des données de commande d'actionneurs (ex : consigne pour le régulateur de vitesse, allumage des warnings, etc.).

Ce constat nous a conduit à considérer uniquement deux types de transactions dites *homogènes* :

— *des transactions de lecture* (*read-set* en anglais) manipulant un ensemble de ressources seulement en lecture ;
— *des transactions d'écriture* (*write-set* en anglais) manipulant un ensemble de ressources seulement en écriture.

Cette hypothèse permet notamment de simplifier l'analyse temporelle de la STM.

Hypothèse n° 2 : Transactions non préemptives

Nous avons également supposé que lorsqu'une tâche exécute une transaction, elle devient non préemptive. En d'autres termes, elle ne peut pas être interrompue par d'autres tâches du même coeur. Cette hypothèse est intéressante car elle nous permet de garantir que le succès des transactions dépend uniquement de la politique de résolution de conflits de la STM. La STM proposée peut alors être utilisée avec n'importe quelle politique d'ordonnancement, le contrôle de l'accès aux ressources étant totalement indépendant des choix de l'ordonnanceur.

Par ailleurs, le fait d'exécuter les transactions en mode non préemptif nous permet de réduire l'empreinte mémoire de la STM, ce qui est un facteur important dans l'embarqué. En effet, sous cette hypothèse, seules M transactions peuvent être actives à la fois (M représentant le nombre de coeurs de la plate-forme matérielle).

Hypothèse n° 3 : Modèle de communication $1 : m$

L'hypothèse effectuée ici suppose que tous les écrivains d'une même ressource partagée soient localisés sur le même coeur. Fonctionnellement, cela signifie que les tâches qui modifient la valeur d'une ressource partagée sont associées au même coeur d'un calculateur. Il semble ainsi cohérent de mutualiser sur un même coeur des fonctionnalités qui sont proches vis-à-vis des ressources sur lequelles elles agissent.

Cette dernière hypothèse nous ramène à un schéma de communication $1 : m$. En d'autres termes, le système peut comporter plusieurs écrivains d'une même ressource mais ceux-ci seront obligatoirement localisés sur le même coeur. Etant donné que les transactions sont considérées comme non préemptives, ces écrivains ne pourront pas s'exécuter de manière concurrente. Par conséquent, aucun conflit *écriture-écriture* ne pourra survenir. Par ailleurs, on pourra observer l'exécution concurrente de plusieurs lecteurs d'une même ressource, répartis sur des coeurs différents, et pouvant s'exécuter en parallèle des écrivains de la ressource. En conséquence, des conflits *lecture-écriture* pourront surgir et devront être résolus par la STM.

5.2 STM-HRT : une STM temps réel *hard*

STM-HRT (*STM for Hard Real-Time Systems*) est une STM conçue pour les systèmes embarqués à contraintes temps réel dur. Nous exposons successivement les garanties offertes par cette STM, son architecture interne et les règles de fonctionnement de son protocole de concurrence.

5.2.1 Caractéristiques de la STM-HRT

Garanties de progression "wait-free"

Seule une STM de type *wait-free* est adaptée au contexte des systèmes temps réel dur. Le nombre d'abandons de transactions doit en effet être borné pour toutes les tâches, au risque sinon de ne pas respecter leurs échéances. La STM-HRT repose donc sur un protocole de synchronisation temps réel non bloquant qui assure des garanties de progression *wait-free*. Toutes les transactions réussissent en un temps fini. Plus précisément, la STM garantit que :

— les transactions d'écriture réussissent *systématiquement* ;

— les transactions de lecture peuvent échouer *au plus une fois*.

Ces garanties de progression *wait-free* sont atteintes sous la STM-HRT grâce à la mise en oeuvre d'une aide mutuelle entre transactions : le mécanisme de "Helping".

Absence de famines

Lorsque des tâches accèdent de manière concurrente à des ressources partagées, on parle de *famine* dans le cas où certaines tâches ne réussissent jamais à valider correctement les opérations qu'elles effectuent sur les ressources partagées. Ceci peut survenir notamment dans le cas où certaines tâches possèdent une période d'activation plus élevée que d'autres, et vont ainsi "monopoliser" l'accès aux ressources. Dans le cas de la STM-HRT, les garanties de progression *wait-free* préviennent par définition toute famine.

Garanties de robustesse

La *robustesse* vise à garantir la capacité d'un système à fonctionner de façon acceptable en présence de fautes ou de variations inhérentes à l'environnement du système. L'objectif ici est de garantir la progression des transactions des tâches même en cas de fautes. La STM-HRT est robuste vis-à-vis des pannes matérielles et/ou logicielles suivantes :

— *défaillance isolée d'un coeur* : la progression des transactions sur les autres coeurs est toujours garantie

— *défaillance isolée d'une tâche* : la progression des transactions des autres tâches est toujours garantie

5.2.2 Architecture de la STM-HRT

Toutes les structures de données de la STM-HRT sont allouées statiquement. Par conséquent, l'empreinte mémoire du protocole peut aisément être déterminée hors-ligne. Ses structures de données concernent d'une part les informations relatives au contexte d'exécution des transactions (regroupées dans des *descripteurs de transaction*) et d'autre part les informations concernant les accès concurrents aux ressources partagées (regroupées dans des *objets* manipulés par les transactions).

Les descripteurs de transaction

Du fait des hypothèses formulées (transactions non préemptives notamment), la STM-HRT requiert un seul descripteur de transaction par coeur. Ainsi au total, on comptera M descripteurs de transaction dans le système, chacun d'entre eux étant partagé entre toutes les transactions d'un même coeur. La Figure 5.1 présente la structuration d'un descripteur de transaction associé à une transaction de lecture homogène. Comme illustré, le descripteur de transaction est composé de 5 champs :

— *proc_ id* représente le numéro du coeur sur lequel la transaction s'exécute ;

— *status* est un champ contenant à la fois l'état courant et le numéro de l'instance courante de la transaction ;
— *read-set* est un pointeur vers l'ensemble des objets que la transaction peut manipuler en lecture seule (*read-set table*) ;
— *write-set* est un pointeur vers l'ensemble des objets que la transaction peut manipuler en écriture (*write-set table*) ;
— *access_ vector* est un champ représentant l'ensemble des objets actuellement ouverts par la transaction.

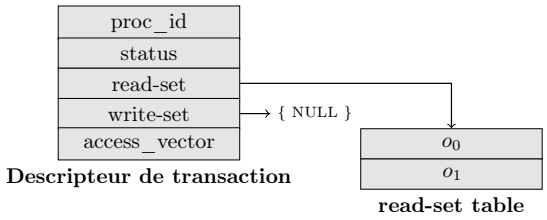


Figure 5.1 – *Descripteur d'une transaction de lecture manipulant 2 objets*

status est un champ de 8 bits. Les 6 bits de poids fort correspondent au *numéro de l'instance courante* de la transaction qui représente la n^{ieme} activation de transaction sur le coeur π_{proc_id}. Etant donné que le descripteur de transaction est partagé par toutes les transactions allouées à un même coeur, ce compteur permet de distinguer les utilisations successives du descripteur de transaction. Il est en particulier utilisé durant les phases de "Helping" entre transactions de manière à maintenir la cohérence des structures de données. Les 2 bits de poids faible représentent quant à eux l'*état courant* de la transaction (cf. diagramme des états possibles pour les transactions représentés sur la Figure 5.2).

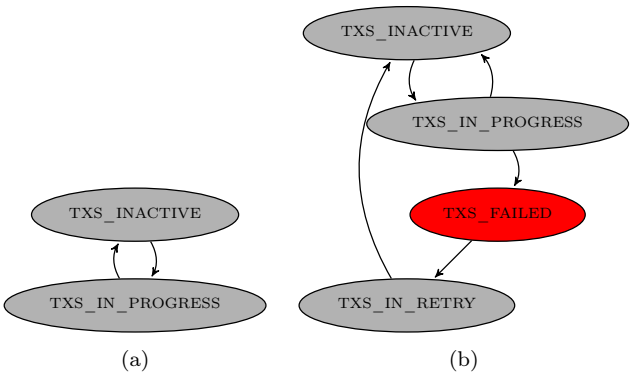


Figure 5.2 – Etats possibles des : *(a) transactions d'écriture, (b) transactions de lecture*

Le champ *access_ vector* est composé d'autant de bits que d'objets pouvant être manipulés sur le coeur $\pi_{proc\ id}$. Chaque fois qu'une transaction ouvre un objet, le bit correspondant est positionné à 1, ce qui permet de garder trace de tous les objets actuellement ouverts par la transaction. L'*access_ vector* est ensuite utilisé au moment du commit de la transaction. Il permet d'une part aux transactions de lecture de détecter des conflits lecture-écriture, et d'autre part aux transactions d'écriture d'identifier quels sont les objets à mettre à jour.

Les objets

A chaque ressource partagée est associé un *objet* (ou *data object*) manipulé par une ou plusieurs transaction(s). La structure de données d'un *objet* est présentée Figure 5.3. Elle comporte les 3 champs suivants :

— *data_ id* identifie la ressource associée à l'objet par son numéro ;
— *copy_ table* est un tableau contenant à la fois la ressource concurrente elle-même et l'ensemble des copies locales de celle-ci ;
— *concurrency_ vector* est un vecteur utilisé pour stocker l'ensemble des informations relatives aux accès concurrents en cours sur la ressource.

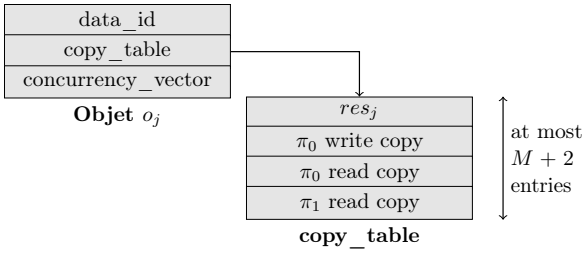


Figure 5.3 – *Objet associé à une ressource res_j manipulée par 2 coeurs π_0 et π_1 (l'écrivain de la ressource est localisé sur π_0)*

copy_ table permet de stocker les différentes versions de la ressource à un instant t. Les 2 premiers champs permettent de stocker la valeur de la ressource ainsi que la copie locale de l'écrivain. Lors du commit de l'écrivain, la copie locale devient la nouvelle valeur de la ressource tandis que le champ réservé à l'ancienne valeur sera utilisé pour la copie locale du prochain écrivain. Un bit du vecteur *concurrency_ vector* (cf. description ci-après), complémenté à chaque fois qu'une transaction d'écriture met à jour la ressource, permet de connaître la valeur de la ressource la plus "fraîche" (1ère ou 2ème position dans la table). Les autres champs de la *copy_ table* représentent les copies des lecteurs de la ressource. Etant donné qu'une tâche ne peut être préemptée lorsqu'elle exécute une transaction, on ne peut observer plus d'un lecteur par coeur à tout instant. Par conséquent, si l'on considère un système comportant M coeurs, la *copy_ table* comportera au pire-cas $M + 2$ entrées.

Le vecteur *concurrency_ vector* est un vecteur de taille $2M + 2 \leq 32$ bits. Il combine 4 informations à la base du protocole de concurrence de la STM-HRT (cf. Figure 5.4). Sa taille est bornée de manière à ce qu'il puisse être mis à jour de manière atomique. La version courante de la STM-HRT supporte jusqu'à 15 coeurs.

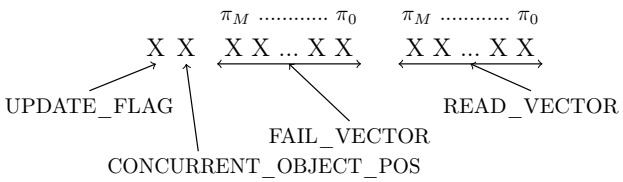


Figure 5.4 – *Organisation interne du vecteur* concurrency_vector

READ_ VECTOR est un vecteur de M bits dans lequel sont enregistrés tous les coeurs sur lequels une transaction a lu la ressource depuis sa dernière mise à jour. Par exemple, lorsque

qu'une transaction de lecture du coeur π_0 ouvre l'objet en lecture seule, elle positionne à 1 le bit correspondant (position π_0 dans le vecteur READ_VECTOR – $XX...X1$).

FAIL_ VECTOR est un vecteur de M bits qui consigne tous les coeurs sur lesquels une transaction de lecture a déjà échoué une fois. Son mode remplissage est identique à celui exposé précédemment pour le *READ_ VECTOR*.

Le bit *CONCURRENT_ OBJECT_ POS* représente la position actuelle de la ressource concurrente dans la *copy_ table* (0 = 1ère ligne, 1 = 2ème ligne).

Le flag *UPDATE_ FLAG* est quant à lui positionné pour indiquer qu'une transaction d'écriture est sur le point de mettre à jour la ressource.

5.2.3 Protocole de concurrence de la STM-HRT

Le protocole de contrôle de concurrence de la STM-HRT requiert ainsi une fonction distincte pour chacune des étapes d'une transaction (ouverture(s), commit). Ces fonctions varient également selon le type de transaction (lecture, écriture) :

- `open_read_stm_object()` : ouverture d'un objet en lecture seule ;
- `open_write_stm_object()` : ouverture d'un objet en écriture ;
- `commit_read_stm_tx()` : validation d'une transaction de lecture ;
- `commit_write_stm_tx()` : validation d'une transaction d'écriture.

La phase d'initialisation

La phase d'initialisation consiste à initialiser l'ensemble de structures de données de la STM-HRT (descripteurs de transaction, objets). Considérons par exemple un système caractérisé par les paramètres suivants :

- un ensemble de 2 processeurs $\Pi = \{\pi_0, \pi_1\}$. 2 descripteurs de transaction sont nécessaires : l'un dédié à π_0, l'autre à π_1,
- un ensemble de 2 ressources partagées $R = \{res_0, res_1\}$. 2 objets sont requis : o_0 pour res_0, o_1 pour res_1,
- un ensemble de 2 transactions $\Omega = \{\omega_0, \omega_1\}$. ω_0 et ω_1 sont respectivement des transactions d'écriture et de lecture manipulant chacune les deux ressources res_0 et res_1. Elles sont assignées aux coeurs π_0 et π_1.

L'ensemble des structures de données de la STM-HRT nécessaires au maintien de la cohérence des ressources partagées dans ce système est illustré sur la Figure 5.5. On remarquera qu'à l'initialisation, tous les éléments des *read-set tables* pointent sur le descripteur de transaction auquel ils sont rattachés, tandis que seuls les éléments des *write-set tables* pointent sur l'emplacement des copies locales de chacune des ressources.

La phase d'ouverture

La fonction `open_read_stm_object()`. Avant d'ouvrir un objet, une transaction de lecture qui n'a jamais échoué auparavant vérifie en premier lieu si cet objet est sur le point d'être mis à jour. Cette vérification est effectuée de manière à éviter un futur conflit potentiel et ainsi améliorer la performance de la STM-HRT. Elle consiste à vérifier le bit *UPDATE_ FLAG* du vecteur *concurrency_ vector* de l'objet. S'il est égal à 1, la transaction de lecture va "aider" la transaction d'écriture sur le point de mettre à jour l'objet sur lequel le conflit a été détecté. Dès que la transaction de lecture est prête à lire la donnée, le pointeur de l'entrée correspondante dans la *read-set table* est positionné sur l'objet (en cas d'échec de la lecture, cette étape permettra aux transactions qui éventuellement l'aideraient de lire la donnée).

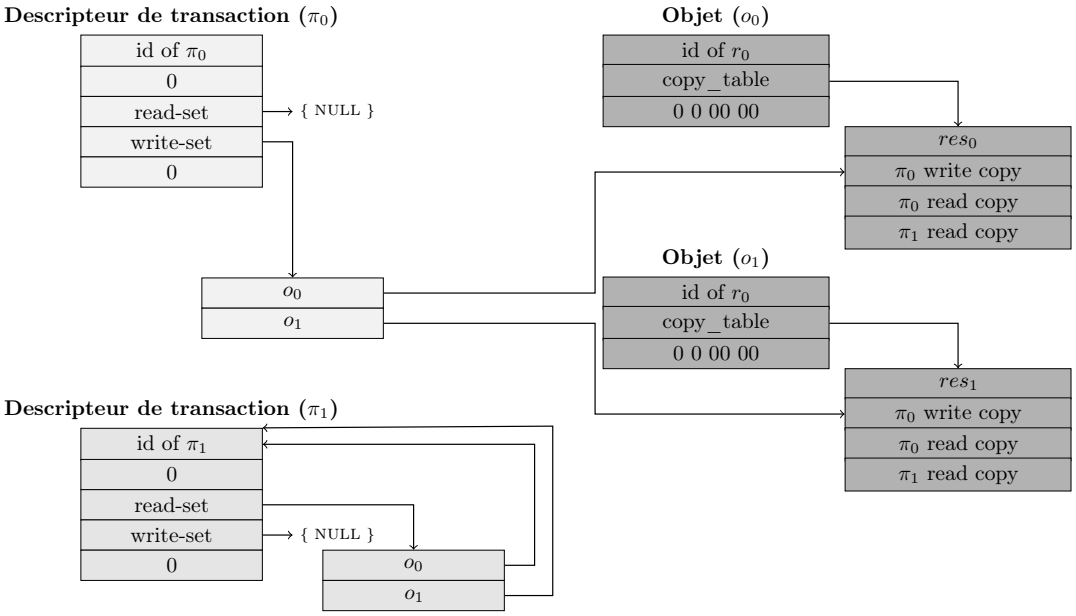


Figure 5.5 – *Phase d'initialisation pour 2 coeurs et 2 ressources*

La fonction `open_write_stm_object()`. Lorsqu'une transaction d'écriture ouvre un objet, la valeur à écrire est copiée dans l'entrée correspondante de la *copy_ table*. Par exemple, sur la Figure 5.5, lorsque ω_0 écrit res_0, elle copie la valeur dans le champ π_0 *write copy* de o_0. Une fois la copie effectuée, l'objet est prêt à être mis à jour. Cette mise à jour est annoncée aux autres transactions en basculant le flag *UPDATE_ FLAG* de l'objet à 1. Ensuite, la transaction d'écriture met à jour son *access_ vector* indiquant ainsi que l'objet a été ouvert par le coeur associé à cette transaction. A noter que la fonction `open_write_stm_object()` elle-même ne peut pas être appelée dans le cas de "Helping".

La phase de validation

La fonction `commit_read_stm_tx()`. La validation (commit) d'une transaction de lecture se déroule en 2 phases. La première phase consiste à vérifier que les objets manipulés sont consistants. Ceci est effectué en vérifiant le *READ_ VECTOR* de chaque objet marqué comme ouvert dans le vecteur *ACCESS_ VECTOR* de la transaction. La deuxième phase consiste à mettre à jour le statut de la transaction. Si la transaction est consistante, son statut est positionné à la valeur *TXS_ INACTIVE* et la fonction renvoie la valeur SUCCESS. Sinon, son statut prend la valeur *TXS_ FAILED* et la fonction renvoie la valeur FAILURE. En cas de "Helping", les différentes transactions (aidées et aidantes) peuvent exécuter la fonction `commit_read_stm_tx()` de manière concurrente. Dans ce cas, des précautions particulières doivent être prises quant à la mise à jour du statut.

La fonction `commit_write_stm_tx()`. Pour chacun des objets marqués comme ouverts dans l'*access_ vector* de la transaction, une sous-fonction nommée `update()` est appelée. Celle-ci consiste tout d'abord à réaliser une opération atomique sur le vecteur *concurrency_ vector* de l'objet qui vient le modifier à condition que :

1. le flag *UPDATE_ FLAG* du vecteur soit égal à 1 ;

2. le champ *FAIL_ VECTOR* du vecteur soit égal à 0.

Si tel est le cas, la modification réalisée sur le vecteur est la suivante :
— le flag *UPDATE_ FLAG* est remis à zéro ;
— le bit *CONCURRENT_ OBJECT_ POS* est complémenté ;
— le champ *READ_ VECTOR* est mis à zéro.

Par exemple, si le vecteur *concurrency_ vector* de l'objet avant la mise à jour est égal à 1 0 00..00 $XX..XX$, il sera modifié atomiquement pour prendre la valeur 0 1 00..00 00..00.

L'opération atomique de modification du vecteur *concurrency_ vector* peut échouer pour plusieurs raisons :
— le flag *UPDATE_ FLAG* est égal à 0. Cela signifie que la donnée et le vecteur *concurrency_ vector* ont déjà été mis à jour avec succès (cas de "Helping") ;
— le champ *FAIL_ VECTOR* n'est pas égal à 0. Dans ce cas, au moins une transaction de lecture a été abandonnée et n'a pas encore réussi à relire l'objet. Par conséquent, chacune de ces transactions de lecture apparaissant dans le *FAIL_ VECTOR* doit être aidée avant la mise à jour de l'objet. Une fois l'aide accomplie, le succès de la transaction de lecture est garanti et le bit du champ *FAIL_ VECTOR* correspondant à la transaction qui a été aidée est remis à zéro.

Le mécanisme de "Helping"

Comme souligné précédemment, les garanties de progression *wait-free* des transactions sous la STM-HRT sont assurées par un mécanisme de "Helping". Une transaction de lecture peut ainsi aider une transaction d'écriture au moment de l'ouverture d'un objet. Une transaction d'écriture peut aider une ou plusieurs transactions de lecture dès lors qu'elle échoue à mettre à jour un objet.

Le mécanisme de "Helping" doit cependant être utilisé avec précaution. En effet, dès lors que des aides concurrentes entre transactions ont lieu, des problèmes de cohérence des aides en termes d'instances de transactions peuvent survenir. Il convient de vérifier que la transaction qui en aide une autre, aide bien la "bonne" instance. Afin d'illustrer ce problème, considérons par exemple deux transactions ω_i (transaction de lecture sur π_0) et ω_j (transaction d'écriture sur π_1). Si ω_i a été abandonnée et ω_j est sur le point de mettre à jour un objet ouvert par ω_i, ω_j va aider ω_i. Supposons que la transaction ω_i réussisse d'elle-même lorsqu'elle est rejouée et soit ω_k, une nouvelle transaction qui démarre à sa suite sur π_0. Le protocole de concurrence doit dans ce cas empêcher ω_i de corrompre le descripteur de ω_k avec des opérations qui avaient pour but initialement d'aider ω_i (rappelons que le descripteur de transaction est partagé par toutes les transactions qui s'exécutent sur un même coeur).

Pour pallier à ce problème, le descripteur de transaction comporte un champ permettant d'identifier le numéro d'instance des transactions qui s'exécutent sur un même coeur. Lorsqu'une transaction réussit son commit, elle incrémente le numéro d'instance. Par ailleurs, dans toutes les fonctions qui peuvent être appelées en cas de "Helping", la valeur de l'instance est systématiquement vérifiée avant de modifier les champs d'un descripteur associé à une transaction qui est aidée par une autre. De cette manière, les structures de données de la STM-HRT sont maintenues consistantes.

5.3 Analyse fonctionnelle et temporelle de la STM-HRT

Cette section présente l'analyse fonctionnelle et temporelle de la STM-HRT. Nous énonçons en particulier différentes propriétés liées à la progression des transactions et établissons une borne supérieure sur les overheads associés aux tâches exécutant des transactions.

5.3.1 Analyse fonctionnelle de la STM-HRT

Tout au long du processus de conception de la STM-HRT, nous nous sommes appuyés sur une approche de model-checking afin de vérifier la validité du protocole de concurrence d'un point de vue fonctionnel. SPIN [1] est l'outil de model-checking avec lequel nous avons établi un modèle du protocole de concurrence de la STM-HRT. Les différents algorithmes (ouverture/commit des objets manipulés en lecture/écriture) ont été implémentés en Promela (langage proche du langage C utilisé pour décrire des modèles de systèmes concurrents sous SPIN) de manière à prouver la validité fonctionnelle de la STM-HRT. Le lecteur intéressé pourra se rapporter au code source Promela vérifié sous SPIN 6.2 et disponible en-ligne à l'adresse suivante : *http ://stmhrt.rts-software.org.*

Les deux lemmes qui suivent se rapportent à la manière dont la STM-HRT assure la progression des différentes transactions. Les transactions d'écriture réussissent *systématiquement* leur commit tandis que les transactions de lecture peuvent être abandonnées *au plus une fois* avant de réussir leur commit et ce, quelles que soient les transactions en conflit avec elles. La STM-HRT s'appuie sur un mécanisme de "Helping" entre écrivains et lecteurs pour garantir cette progression des transactions.

Lemme 5.3.1 *Aucune transaction d'écriture ne peut être abandonnée sous la STM-HRT.*

Le protocole de concurrence de la STM-HRT est par construction un protocole 1 écrivain, m lecteurs. Ainsi, une seule transaction à la fois est autorisée à mettre à jour une ressource. Par conséquent, aucun conflit écriture-écriture ne pouvant survenir, toute transaction d'écriture réussira systématiquement son commit.

Lemme 5.3.2 *Les transactions de lecture peuvent être abandonnées* au plus une fois *sous la STM-HRT.*

Un conflit lecture-écriture survient dès lors qu'une transaction d'écriture met à jour une ressource dans l'intervalle compris entre l'ouverture de l'objet associé à cettte ressource par une transaction de lecture et l'instant auquel elle effectue le commit. Dans le cas où son commit échoue, une transaction de lecture remplit le vecteur *FAIL_ VECTOR* de l'objet avant d'essayer de l'ouvrir une seconde fois. Ceci permet de "verrouiller" la ressource et d'éviter toute nouvelle mise à jour de la même ressource. En effet, une transaction d'écriture ne pourra procéder à la mise à jour de la ressource tant que le vecteur *FAIL_ VECTOR* ne sera pas égal à 0, c'est-à-dire, tant que la transaction de lecture n'aura pas réussi son commit. En pratique, soit la transaction de lecture réussit d'elle-même à ouvrir l'objet la seconde fois, soit elle sera aidée par une transaction d'écriture en phase de commit qui exécutera pour elle la fonction `open_read_stm_object` (et ce pour tous les objets ouverts par la transaction de lecture) ainsi que la fonction `commit_read_stm_tx`. Par conséquent, la transaction de lecture ne pourra être abandonnée une seconde fois.

1. G.-J. Holzmann. The model checker SPIN. *IEEE Transactins on Software Engineering – Special issue on Formal Methods in Software Practice*, 1997.

Les deux lemmes suivants se réfèrent au fait que la STM-HRT garantit l'absence de famine et empêche tout "Helping" cyclique.

Lemme 5.3.3 *La STM-HRT assure l'absence de famine.*

Les transactions d'écriture aident systématiquement les transactions de lecture concurrentes qui ont déjà été abandonnées une fois. De ce fait, une famine des transactions d'écriture est possible dans le cas où l'on observe l'abandon d'une infinité de transactions de lecture. Pour pallier à ce problème, avant d'ouvrir un objet, une transaction de lecture va aider toute transaction d'écriture en phase de commit qui pourrait conduire à son abandon dans le futur. Considérons l'exemple illustré sur la Figure 5.6 pour lequel les 3 transactions sont en conflit sur une même ressource. Nous observons qu'une transaction d'écriture qui s'exécute sur le coeur π_3 aide 2 transactions de lecture s'exécutant sur les coeurs π_2 et π_1 respectivement. Après avoir été aidée, on suppose qu'une nouvelle instance de transaction de lecture démarre sur le coeur π_2 et aide, à son tour, la transaction d'écriture avant d'ouvrir l'objet associé à la ressource partagée, lui permettant ainsi de réussir son commit. Ce phénomène d'aide mutuelle des transactions est mise en oeuvre pour l'ensemble des ressources partagées. Ainsi, aucune famine n'est possible pour les transactions d'écriture.

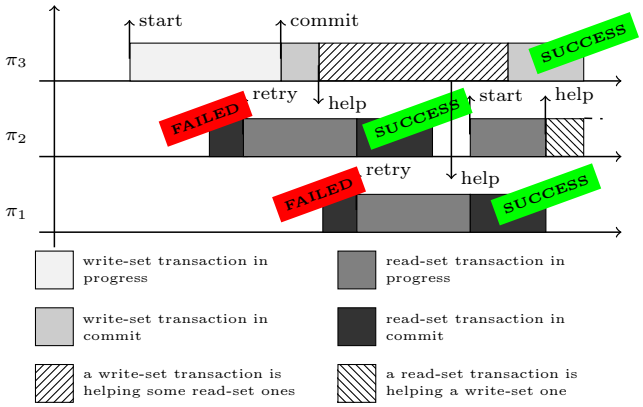


Figure 5.6 – *Illustration du "Helping" entre transactions permettant de garantir la progression wait-free et l'absence de famine*

Lemme 5.3.4 *Le Helping sous la STM-HRT n'est pas cyclique.*

La cyclicité du "Helping" survient par définition lorsqu'une transaction aide une autre transaction qui est déjà en train de l'aider. Considérons 2 transactions ω_i (transaction de lecture) et ω_j (transaction d'écriture). Supposons que ω_i échoue et recommence donc son exécution en indiquant son échec dans le vecteur *FAIL_ VECTOR* de l'objet juste avant de retenter l'ouverture de l'objet. ω_j va ensuite aider ω_i juste avant de mettre à jour la ressource car le vecteur *FAIL_ VECTOR* n'est pas nul. En parallèle, si ω_i est autorisée à aider elle-même des transactions d'écriture alors qu'elle rejoue, elle aidera alors ω_j, qui est elle-même déjà en train de l'aider. Ce cas de figure de "Helping" cyclique est écarté sous la STM-HRT en imposant la contrainte suivante : une transaction de lecture est autorisée à aider une transaction d'écriture uniquement dans le cas où elle s'exécute *pour la première fois* (c'est-à-dire qu'elle n'a jamais été abandonnée), comme illustré sur la Figure 5.6). Etant

donné qu'une transaction d'écriture ne peut aider que des transactions de lecture qui ont déjà échoué, le "Helping" ne peut donc pas être cyclique.

Théorème 5.3.1 *La STM-HRT assure une progression wait-free des transactions.*

Au travers des lemmes 5.3.3 et 5.3.4, nous avons montré que la STM-HRT empêchait toute famine et tout "Helping" cyclique. Les lemmes 5.3.1 et 5.3.2 garantissent par ailleurs que le nombre d'abandons subis par une transaction est borné. Ainsi, toute transaction gérée par la STM-HRT terminera nécessairement son exécution avec succès en un nombre fini d'opérations. Par conséquent, la STM-HRT est de type wait-free.

La STM-HRT est également robuste aux défaillances matérielles et/ou logicielles et garantit que le mécanisme de "Helping" est consistant, comme démontré ci-après.

Théorème 5.3.2 *La STM-HRT assure la progression des transactions même en cas de défaillance d'un coeur ou d'une tâche.*

Comme énoncé dans le Théorème 5.3.1, la progression wait-free des transactions est garantie. Cependant, afin d'illustrer la robustesse de la STM-HRT en cas de crash d'une tâche ou d'un coeur, considérons deux transactions ω_i (transaction de lecture) et ω_j (transaction d'écriture). Supposons que ω_i échoue et recommence donc son exécution. Juste avant d'ouvrir à nouveau un objet, ω_i remplit le vecteur *FAIL_ VECTOR* de l'objet de manière à empêcher toute future mise à jour de la ressource. ω_j va alors aider ω_i au moment de son commit. A ce moment-là, si une défaillance sur du coeur associé soit à ω_i, soit ω_j survient, alors ω_i ne réussira jamais son exécution et le vecteur *FAIL_ VECTOR* de l'objet manipulé par ω_j ne sera jamais remis à 0. Par conséquent, ω_j ne pourrait plus progresser. La STM-HRT résout ce problème en autorisant ω_j à remettre à zéro le vecteur *FAIL_ VECTOR* à la fin de chaque "Helping". Par conséquent, le crash d'un coeur ou d'une tâche n'est jamais propagé aux autres coeurs ou autres tâches, sous des réserves de mode de défaillance correct et de type *fail-silent*.

Théorème 5.3.3 *Le Helping de la STM-HRT est toujours consistant.*

La consistance du "Helping" est associée au fait que les structures de données de la STM-HRT restent toujours cohérentes même en cas d'aides mutuelles de transactions. Le problème survient lorsqu'une transaction de lecture qui est en train d'être aidée réussit à terminer avec succès par elle-même, et qu'une autre instance de transaction démarre éventuellement à sa suite sur le même coeur. Il y a alors ré-utilisation immédiate du même descripteur de transaction. Le risque de corruption des structures de données réside alors dans le fait qu'une (ou des) transaction(s) concurrente(s) continuent à aider en parallèle une transaction qui est associée à l'instance précédente du descripteur. De manière à prévenir ce cas de figure, le "Helping" mis en oeuvre inclut une vérification atomique qui repose sur la comparaison du numéro de l'instance de la transaction qui s'exécute actuellement sur le coeur avec celui de la transaction pour laquelle une aide avait était lancée. Ceci est implémenté avec des opérations atomiques de type LL/SC. Les modifications des structures de données n'étant appliquées par la transaction aidante seulement s'il y a correspondance des numéros d'instances, le "Helping" est donc toujours consistant.

5.3.2 Analyse temporelle de la STM-HRT

Afin de faciliter la compréhension des formules énoncées, l'ensemble des notations utilisées dans cette partie est résumé dans la Table 5.1.

Symbol	Description
τ_i	Tâche i de l'ensemble de tâches τ
ω_k	Transaction k de l'ensemble de transactions Ω
π_i	Coeur i
$\pi(\omega_k)$	Coeur sur lequel ω_k s'exécute
Ω_{π_i}	Ensemble des transactions s'exécutant sur le coeur π_i
Ω^R	Ensemble des transactions de lecture du système
$\Omega^R_{\pi_i}$	Ensemble des transactions de lecture du coeur π_i
$\Omega^R_{\tau_i}$	Ensemble des transactions de lecture de la tâche τ_i
Ω^W	Ensemble des transactions d'écriture du système
$\Omega^W_{\pi_i}$	Ensemble des transactions d'écriture du coeur π_i
$\Omega^W_{\tau_i}$	Ensemble des transactions d'écriture de la tâche τ_i
O^{ω_k}	Ensemble des objects manipulés par la transaction ω_k
o_j	Objet j de l'ensemble des objets O^{ω_k}
$\pi(o_j)$	Coeur autorisé à mettre à jour o_j.
$\Pi(o_j)$	Ensemble des coeurs autorisés à seulement lire o_j

Table 5.1 – *Notations utilisées*

Sans pertes de généralité, nous supposons que toutes les ressources partagées sont de taille identique et que le temps d'initialisation d'une transaction est négligeable. Nous introduisons tout d'abord des définitions relatives au calcul de la durée d'exécution des transactions lorsque celles-ci sont exécutées en totale isolation.

Définition 1 *La durée d'exécution d'une transaction d'écriture $\omega_k \in \Omega^W$ s'exécutant en totale isolation est égale à :*

$$e_k^W = |O^{\omega_k}| * (T\_open_{o_j}^W + T\_commit_{o_j}^W) \tag{5.1}$$

$T\_open_{o_j}^W$ et $T\_commit_{o_j}^W$ correspondent respectivement au temps requis pour ouvrir en écriture et mettre à jour un seul objet o_j, sous l'hypothèse que ω_k s'exécute en totale isolation.

Définition 2 *La durée d'exécution d'une transaction de lecture $\omega_k \in \Omega^R$ s'exécutant en totale isolation est égale à :*

$$e_k^R = |O^{\omega_k}| * T\_open_{o_j}^R + T\_commit_{\omega_k}^R \tag{5.2}$$

$T\_open_{o_j}^R$ représente le temps requis pour ouvrir en lecture seule un seul objet o_j, sous l'hypothèse que ω_k s'exécute en totale isolation. $T\_commit_{\omega_k}^R$ correspond au temps requis pour valider la transaction ω_k (c'est-à-dire le temps passé en phase de commit).

Le problème de la détermination d'une borne supérieure sur l'overhead subi par une tâche τ_i sous la STM-HRT du fait de ses accès aux ressources partagées, revient à déterminer

une borne supérieure sur l'overhead subi par ses transactions elles-mêmes. Dans les lemmes 7.5 and 7.6, nous quantifions donc l'overhead au pire-cas subi par les transactions du fait de conflits avec des transactions concurrentes. Celui-ci intègre à la fois le temps gaspillé du fait des éventuels abandons de transactions et le temps relatif aux aides mutuelles entre transactions.

Lemme 5.3.5 *Chaque transaction d'écriture $\omega_k \in \Omega^W$ subit au pire-cas un overhead égal à :*

$$T\_overhead^W_{\omega k} = \sum_{j=1}^{|O^{\omega_k}|} \left(T\_helping^R_{o_j} + T\_attempt\_commit^W_{o_j} \right) \tag{5.3}$$

$T\_helping^R_{o_j}$ correspond à l'overhead subi par ω_k lorsqu'elle aide des transactions de lecture d'autres coeurs qui ont déjà été abandonnées une fois et qui souhaitent ouvrir des objets $o_j \in O^{\omega_k}$:

$$T\_helping^R_{o_j} = \sum_{\pi_i \in \Pi(o_j) \setminus \{\pi(\omega_k)\}} \max_{\substack{\omega_m \in \Omega^R_{\pi_i} \\ s.t.\ o_j \in O^{\omega_m}}} e^R_m \tag{5.4}$$

$T\_attempt\_commit^W_{o_j}$ représente l'overhead subi par ω_k à chaque fois qu'elle tente de mettre à jour o_j :

$$T\_attempt\_commit^W_{o_j} = \sum_{\pi_i \in \Pi(o_j) \setminus \{\pi(\omega_k)\}} T\_commit^W_{o_j} \tag{5.5}$$

Lorsqu'une transaction d'écriture ω_k échoue à mettre à jour un objet donné en phase de commit, cela signifie qu'elle doit d'abord aider toute transaction de lecture concurrente qui s'exécute sur un coeur distant et qui souhaite ouvrir ce même objet (et qui a déjà échoué par le passé). Au pire-cas, cette aide concerne au plus $(M-1)$ coeurs. Comme montré dans l'équation 5.4, une seule transaction de lecture est aidée par coeur. Dans le pire-cas, cette transaction est celle qui possède la durée d'exécution la plus grande parmi celles qui peuvent être actives sur le coeur considéré. L'équation 5.5 représente quant à elle le nombre total de fois (au plus $(M-1)$ fois) où ω_k peut tenter de mettre à jour l'objet o_j avant de finalement réussir son commit. Enfin, étant donné que ω_k peut être en conflit avec des transactions de lecture pour chacun des objets qu'elle souhaite mettre à jour (c'est-à-dire au pire cas $vert O^{\omega_k}|$ objets), le mécanisme d'aide peut être répété pour les $|O^{\omega_k}|$ objets, ce qui correspond à la sommation de l'équation 5.3.

Lemme 5.3.6 *Chaque transaction de lecture $\omega_k \in \Omega^R$ subit au pire-cas un overhead égal à :*

$$T\_overhead^R_{\omega_k} = \underbrace{\sum_{j=1}^{|O^{\omega_k}|} T\_helping^W_{o_j}}_{first\ play} + \underbrace{e^R_{k,retry}}_{second\ play} \tag{5.6}$$

$T\_helping^W_{o_j}$ est l'overhead subi par ω_k du fait de l'aide procurée aux transactions d'écriture localisées sur le coeur $\pi(o_j) \neq \pi(\omega_k)$ (une transaction de lecture aide une transaction d'écriture uniquement pour les objets avec lesquels elle est en conflit). Il est défini comme suit :

$$T\_helping_{o_j}^{W} = \begin{cases} 0 & si\ \pi(o_j) = \pi(\omega_k), \\ \left(T\_helping_{o_j}^{R} + T\_attempt\_commit_{o_j}^{W}\right) & sinon. \end{cases} \tag{5.7}$$

$e_{k,retry}^{R}$ correspond au temps requis pour ré-exécuter la transaction suite à un abandon :

$$e_{k,retry}^{R} \begin{cases} 0 & si\ \forall o_j \in O^{\omega_k},\ \pi(o_j) = \pi(\omega_k) \\ e_k^R & sinon. \end{cases} \tag{5.8}$$

Une transaction de lecture ω_k peut subir deux types d'overhead (équation 5.6). La première fois qu'elle s'exécute, ω_k peut être amenée à aider une transaction d'écriture à mettre à jour un objet. Ce cas de figure survient uniquement dans le cas où au moins un des objets manipulés par ω_k a été mis à jour sur un autre coeur (équation 5.7). Dans le cas où ω_k est réjouée suite à un abandon, elle n'aide pas d'autres transactions. Par conséquent, seule la durée d'exécution liée à sa ré-exécution est à considérer (cf. équation 5.8).

Lorsque ω_k ouvre un objet, l'aide n'est pas réalisée si la transaction d'écriture est localisée sur le même coeur que celui sur lequel ω_k s'exécute (cf. partie haute de l'équation 5.7). Si ce n'est pas le cas (cf. partie basse de l'équation 5.7), il convient de prendre en compte le fait que la transaction d'écriture que ω_k va aider, peut elle-même aider au plus $(M-2)$ autres transactions de lecture, l'aide cyclique étant interdite. Ceci est intégré dans les équations 5.4 et 5.5.

Ces résultats nous conduisent à établir une borne supérieure sur l'overhead associé à chaque tâche pouvant exécuter une ou plusieurs transactions de lecture et/ou une ou plusieurs transactions d'écriture.

Lemme 5.3.7 *Toute tâche τ_i qui exécute $|\Omega_{\tau_i}^R|$ transactions de lecture et $|\Omega_{\tau_i}^W|$ transactions d'écriture, subit au pire-cas un overhead égal à :*

$$overhead_i = \sum_{j=1}^{|\Omega_{\tau_i}^R|} T\_overhead_{\omega_j}^{R} + \sum_{j=1}^{|\Omega_{\tau_i}^W|} T\_overhead_{\omega_j}^{W} \tag{5.9}$$

Une tâche exécute $|\Omega_{\tau_i}^R|$ transactions de lecture et $|\Omega_{\tau_i}^W|$ transactions d'écriture. L'overhead total induit par les transactions est ainsi égal à la somme des overheads individuels des transactions, pour lesquelles une borne supérieure a été déterminée dans les lemmes 5.3.5 et 5.3.6.

Ce dernier lemme nous conduit à établir le résultat du Théorème 5.3.4 qui en découle directement.

Théorème 5.3.4 *Une borne supérieure sur la durée d'exécution d'une tâche τ_i dans le contexte de la STM-HRT est telle que :*

$$C_i^{STM-HRT} = C_i + overhead_i \tag{5.10}$$

La durée d'exécution au pire-cas (WCET) d'une tâche dépend de l'overhead induit par le protocole de la STM-HRT. La durée d'exécution initiale C_i considérée lorsque les transactions s'exécutent en totale isolation est augmentée de l'overhead au pire-cas induit par l'ensemble des exécutions concurrentes des transactions de lecture et d'écriture (scénario de conflits au pire-cas vis-à-vis des ressources partagées tel que décrit dans le lemme 5.3.7).

5.4 Bilan et diffusion des résultats

Nous avons présenté une alternative aux protocoles de synchronisation bloquants dans le contexte embarqué automobile. La STM-HRT proposée est un protocole non bloquant qui s'inspire d'une part des travaux sur les mécanismes wait-free, et d'autre part des travaux sur les mémoires transactionnelles. La STM-HRT est un mécanisme wait-free (i.e., elle garantit la progression de *toutes* les transactions). La progression wait-free est notamment assurée grâce au mécanisme d'aide : chaque transaction doit s'assurer de la progression des transactions avec lesquelles elle est en conflit. Au-delà des garanties de progrès offertes, l'aide permet aussi d'atteindre les contraintes de robustesse visées (i.e. résister au crash d'un coeur ou d'une tâche). Notons néanmoins que la progression wait-free est apportée sur la base d'hypothèses simplificatrices, voire restrictives vis-à-vis des possibilités offertes habituellement par les mémoires transactionnelles (transactions homogènes, contexte mono-écrivain). Ces hypothèses sont cependant cohérentes avec les stratégies de développement des systèmes embarqués automobiles. Parmi les solutions basées sur les mêmes hypothèses, nous pouvons nous comparer à l'algorithme de Chen [Chen 1997]. La STM-HRT est plus complète dans le sens où elle bénéficie du contexte transactionnel pour protéger une section critique au sein de laquelle on accède à un groupe de données, là où l'algorithme de Chen ne maintient la cohérence d'une donnée uniquement à la fois.

Afin de valider la conception du protocole de la STM-HRT, ses propriétés ont été évaluées à l'aide du model checker SPIN [Holzmann 1997]. Le paquetage logiciel (code source + scénarii de test) est accessible à l'adresse suivante : `http://stmhrt.rts-software.org`. Nous avons implémenté le protocole STM-HRT en PROMELA et défini un ensemble d'exigences à vérifier telles que :

- Une transaction de lecture peut échouer une seule fois,
- Une transaction d'écriture ne peut jamais échouer,
- Tous les objets ouverts en écriture sont mis à jour,
- Le progrès de toutes les transactions est garanti,
- La mise à jour et le nettoyage des structures de données sont correctement effectués lors du succès d'une transaction,
- Aucune transaction ne peut être en famine.

Le nombre de rejoues des transactions étant borné, la STM-HRT est adaptée aux systèmes temps réel dur. L'analyse temporelle nous a permis de montrer son impact pire-cas sur le temps d'exécution des tâches comportant des transactions. Les accès aux données partagées étant encapsulés dans des transactions et le protocole (vu comme un service du système d'exploitation) gèrant seul les accès et les conflits, le pire temps d'exécution d'une transaction est indépendant de la politique d'ordonnancement choisie. Nous avons mis en évidence que l'overhead temporel lié aux rejoues des transactions peut être intégré dans la durée d'exécution au pire-cas des tâches, afin d'être pris en compte aisément lors de l'analyse d'ordonnançabilité. Barros et al. [Barros 2016] ont traité le problème de l'analyse du temps de réponse de tâches temps réel *hard* partageant des données gérés par une STM sous

une approche d'ordonnancement partitionnée. Bien que leurs travaux poursuivent le même objectif d'analyse temporelle d'une STM, les hypothèses sur lesquelles ils se sont appuyés sont différentes : (i) les transactions ne sont pas forcément homogènes, (ii) les transactions sont sérialisées en fonction de leur date d'arrivée, et (iii) la résolution des conflits repose sur un gestionnaire de contention adapté aux systèmes temps réel (FIFO-CRT) [Barros 2014] mais qui n'offre cependant pas la garantie de progression *wait-free*, contrairement à la STM-HRT.

Nous avons également évalué l'empreinte mémoire de la STM-HRT, en fonction du nombre de coeurs et du nombre de données partagées, afin de vérifier son adéquation avec le secteur industriel automobile visé. Les résultats rapportés dans [Cotard 2013, Cotard 2015] montrent que l'empreinte mémoire du protocole dépend essentiellement du nombre de copies nécessaires à chaque donnée partagée. Dans *AUTOSAR* (*AUTomotive Open System ARchitecture* – l'architecture logicielle standardisée pour les unités de contrôle électronique (ECUs) des véhicules), chaque destinataire d'un message possède également sa propre copie. Notre solution est donc comparable aux solutions existantes.

En résumé, la STM-HRT est une approche robuste, non bloquante, wait-free, représentant une réelle alternative au protocole de synchronisation bloquant spécifié dans *AUTOSAR* multicoeur pour le partage des données inter-coeurs.

Cette contribution est une partie du travail de thèse de Sylvain Cotard [Cotard 2013]. Les résultats de ces travaux ont été présentés dans la revue *ACM Transactions on Embedded Computing Systems* [Cotard 2015].

Troisième partie

Contributions à l'autonomie énergétique des systèmes temps réel embarqués

Chapitre 6

Problématique de l'embarqué temps réel autonome en énergie

Sommaire

6.1 Mise en oeuvre d'un système énergétiquement autonome

De nombreuses sources d'énergie telles que la lumière, les matériaux piézoélectriques, les vibrations ou les mouvements mécaniques, peuvent être utilisées pour alimenter une grande variété de capteurs et d'appareils sans fil présents dans le monde de l'*Internet des Objets* (*IdO* ou *Internet of Things – IoT* en anglais) [Chalasani 2008, Yildiz 2009]. L'alimentation d'un noeud de capteur a toujours été un défi majeur, en particulier lorsque celui-ci est localisé dans un lieu difficile d'accès ou même dangereux. Le remplacement des batteries est également un problème central lors de la conception de ces systèmes. Grâce à des solutions de *récupération d'énergie* (*energy harvesting* en anglais), il est désormais possible de tirer profit de sources d'énergie durables, économiques et autonomes. Les technologies de récupération d'énergie attirent de plus en plus l'attention des ingénieurs car l'énergie disponible dans l'environnement peut désormais être convertie pour permettre de mettre en oeuvre des systèmes nécessitant peu de maintenance. Parmi les exemples d'applications potentielles pouvant bénéficier de la récupération d'énergie environnementale, citons une ligne de production. La maintenance prédictive y devient une activité de toute première importance afin d'anticiper et minimiser l'impact lié à la réparation des machines, suite à la survenue d'une panne. Conserver l'usine de fabrication en état de marche, sans temps d'arrêt imprévu, est souvent nécessaire pour des raisons économiques. Dans ce cas, les systèmes de surveillance des installations à base de capteurs autonomes alimentés par les vibrations ambiantes peuvent permettre de réduire les coûts de maintenance. Considérons un autre exemple, celui de l'industrie aérospatiale où l'on souhaite éliminer la dépendance à la fois aux câbles d'alimentation et aux batteries. Des capteurs énergétiquement autonomes peuvent désormais être déployés pour surveiller les différents composants critiques de la structure de l'avion. Afin d'alimenter les capteurs, les vibrations de rotation créées par l'équipement ou les machines sont continuellement converties en électricité grâce à des dispositifs spécifiques appelés *récupérateurs d'énergie*. Chaque récupérateur doit être monté sur une surface plane afin de capter les vibrations. Cependant, l'intégration d'un récupérateur d'énergie dans

un produit nécessite un dimensionnement préalable de la taille de ce récupérateur afin de rendre le système autonome durant toute la durée de vie de l'application. L'inclusion de techniques spécifiques de gestion de l'énergie est également nécessaire. La source d'énergie ambiante doit produire suffisamment d'énergie pour le système de surveillance à chaque instant. C'est la raison pour laquelle la plupart des capteurs autonomes sont équipés d'un *réservoir d'énergie* (batterie ou supercondensateur) afin de prévenir les cas où la récolte d'énergie ambiante est insuffisante (ou inexistante par intermittence) afin d'alimenter le système.

À titre d'exemple illustratif, considérons l'exemple d'un capteur temps réel de surveillance de mouvement alimenté par une source d'énergie renouvelable (cf. Figure 6.1). L'application de surveillance implémentée dans le capteur a pour mission de détecter et transmettre périodiquement des données à un serveur externe. La caméra récupère des informations via le capteur de détection de mouvement à intervalles réguliers (par exemple toutes les 500 ms) et dès lors qu'un mouvement est détecté, elle envoie des alertes en temps réel tout en enregistrant des clichés ou des vidéos des intrusions.

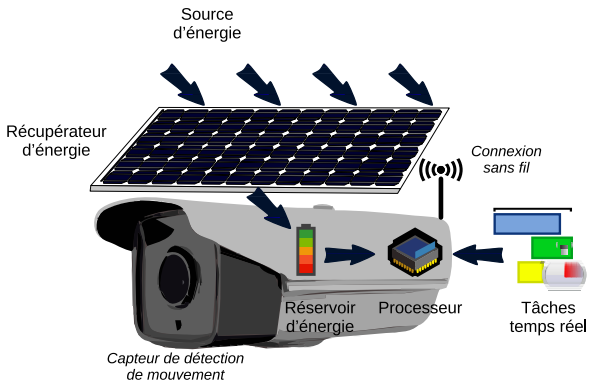


Figure 6.1 – *Exemple de système temps réel de surveillance autonome*

6.2 Comment concilier contraintes de temps et d'énergie ?

6.2.1 Minimisation énergétique vs. neutralité énergétique

De nombreuses techniques de gestion de l'énergie permettent de *réduire* la consommation énergétique d'un système. Leur objectif est d'augmenter la durée d'autonomie de ces systèmes qualifiés de *basse consommation* ou *très basse consommation* (*low-power* ou *ultra low-power* en anglais). Parmi elles, citons les approches *Dynamic Power Management* (*DPM*) [Sinha 2001] et *Dynamic Voltage and Frequency Scaling* (*DVFS*) [Yao 1995, Pillai 2007]. Le mécanisme DPM consiste à mettre hors tension l'ensemble ou une partie des circuits électroniques afin d'économiser l'énergie consommée. La technique du DVFS, très étudiée dans la littérature [Chen 2007] et désormais intégrée à de nombreux processeurs, repose quant à elle sur la diminution de la vitesse de fonctionnement du (ou des) processeur(s) afin de réduire la consommation énergétique. Le DVFS apporte une solution efficace en allongeant effectivement l'intervalle de temps entre deux recharges du système. Il convient tout particulièrement aux systèmes pour lesquels la *minimisation* de l'énergie consommée

est le principal challenge. En revanche, dans le cas de systèmes à récupération d'énergie, le challenge est tout autre : il s'agit de récolter et d'utiliser l'énergie issue uniquement de l'environnement en s'affranchissant de toute recharge externe du système. Le principal défi est d'assurer au système un fonctionnement *neutre* du point de vue énergétique. En d'autres termes, l'énergie consommée par le système doit toujours rester inférieure au égale à l'énergie récupérée dans l'environnement.

En résumé, les défis de conception des systèmes temps réel autonomes en énergie (cf. Figure 6.2) se révèlent être à l'intersection de trois problématiques distinctes. Il est donc impératif de doter les systèmes temps réel à récupération d'énergie de mécanismes intelligents à la fois en termes de gestion de l'énergie et d'ordonnancement temps réel afin de garantir un fonctionnement en *neutralité énergétique*.

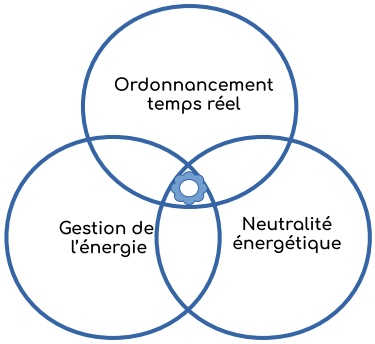


Figure 6.2 – *Les défis de conception des systèmes temps réel autonomes*

6.2.2 Neutralité énergétique et ordonnancement temps réel

L'ordonnanceur d'un système temps réel autonome doit prendre en compte les caractéristiques temporelles des tâches mais il doit aussi intégrer dans ses décisions les propriétés de la source d'énergie (souvent variable par nature) et du réservoir d'énergie. La *neutralité énergétique* doit permettre au système de fonctionner de manière perpétuelle sans *pénurie énergétique*. Afin de garantir cette neutralité energétique, trois paramètres clés doivent être considérés : (i) l'énergie récupérée de l'environnement, (ii) l'énergie consommée par les tâches qui s'exécutent sur le système, et (iii) l'énergie stockée dans le réservoir d'énergie. La mise en relation de ces paramètres au travers de modèles mathématiques doit permettre de comprendre le comportement énergétique du système et de déterminer la quantité d'énergie disponible à court terme. Ces modèles peuvent alors être exploités par l'algorithme d'ordonnancement de manière à adapter dynamiquement l'activité du processeur. Le problème de l'ordonnancement sur un système temps réel énergétiquement autonome peut ainsi être appréhendé au travers de deux modules indissociables : le premier traitant l'ordonnancement des tâches, le second contrôlant l'énergie disponible. Le comportement du système est principalement déterminé par l'algorithme d'ordonnancement, l'énergie en tant que ressource disponible en quantité limitée pouvant être considérée comme une "simple" perturbation.

En résumé, les questions clés auxquelles nous nous intéressons en termes d'ordonnancement temps réel pour les systèmes autonomie énergétiquement sont les suivantes :

— *l'ordonnanceur EDF, optimal en l'absence de toute limitation énergétique et de toute surcharge de traitement [Liu 1973, Dertouzos 1974], reste-t-il un "bon" candidat sous*

les contraintes associées au energy harvesting *?*

— *comment ordonnancer les différents jobs des tâches dans le respect de leurs contraintes temporelles tout en prévenant la survenue de toute pénurie énergétique pour le système ?*

— *un algorithme d'ordonnancement, à la fois optimal et de complexité d'implémentation acceptable, existe-t-il pour les systèmes temps réel à récupération d'énergie ?*

— *comment vérifier et garantir avant le lancement du système que celui-ci fonctionnera de manière perpétuelle en neutralité énergétique, tout en délivrant un niveau de performance toujours acceptable du point de vue temporel ?*

6.3 Contributions

Dans les chapitres suivants, nous présentons nos contributions à l'ordonnancement temps réel pour les systèmes énergétiquement autonomes.

Le chapitre 7 présente nos résultats en considérant des plate-formes temps réel monoprocesseur. Nous présentons tout d'abord un nouveau modèle de tâches intégrant les contraintes d'énergie associées à l'exécution des tâches. Nous nous intéressons ensuite à l'adéquation de l'ordonnanceur optimal *Earliest Deadline First* (*EDF*) au contexte du *energy-harvesting*. Sur la base de ces observations, nous définissons un certain nombre de pré-requis pour un ordonnanceur dans ce contexte. Puis, nous introduisons un nouvel algorithme d'ordonnancement de tâches périodiques adapté aux systèmes autonomes en énergie, *Earliest Deadline with Energy harvesting capabilities* (*ED-H*). Nous présentons son fonctionnement et ses propriétés. Le problème de la faisabilité d'un ensemble de tâches périodiques sous contraintes temporelles et énergétiques est enfin décrit en soulignant l'intérêt de l'hypothèse (plausible dans certains cas) d'une puissance constante.

Le chapitre 8 présente nos contributions à *la gestion des cas de surcharge* dans les systèmes monoprocesseur énergétiquement autonomes. Par surcharge, on entend à la fois la surcharge du point de vue temporel (i.e. demande de traitement excédant les capacités du processeur) et énergétique (i.e. consommation énergétique excédant les capacités de production et/ou stockage de l'énergie). Après avoir décrit le problème de la résolution des cas de surcharge, nous présentons un modèle de tâches étendu permettant de garantir une QoS minimale pour les tâches, même en cas de surcharge. Deux nouveaux ordonnanceurs, *Green-RTO* et *Green-BWP* intégrant conjointement des contraintes de QoS et d'énergie sont introduits. Une analyse de faisabilité sous contraintes de QoS et d'énergie est ensuite présentée. Enfin, nous rapportons les résultats d'évaluation du comportement des algorithmes proposés et discutons du support matériel nécessaire à leur implémentation.

Chapitre 7

Ordonnancement temps réel pour les systèmes monoprocesseur à récupération d'énergie renouvelable

Sommaire

7.1 Ordonnancement de tâches périodiques sous contraintes d'énergie

7.1.1 Modèle du système

Le modèle général du système est représenté sur la Figure 7.1. Comme nous pouvons le voir, le système consiste matériellement en un récupérateur d'énergie alimenté par une source d'énergie renouvelable, un réservoir de stockage de l'énergie, et une unité de traitement type microcontrôleur. On suppose que l'unité de traitement opère à une fréquence donnée et que sa consommation à l'état *idle* (i.e. lorsqu'elle n'exécute aucune tâche) est négligeable. L'ordonnanceur temps réel a accès non seulement aux caractéristiques des tâches mais également aux caractéristiques liées à l'énergie disponible dans le système, et ce, afin de pouvoir ordonnancer les tâches de manière optimale en fonction de la double contrainte : *temps* et *énergie*.

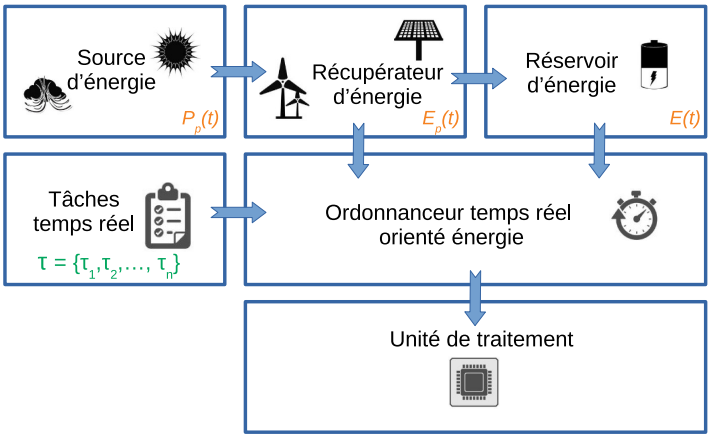


Figure 7.1 – *Modèle d'un système temps réel à récupération d'énergie*

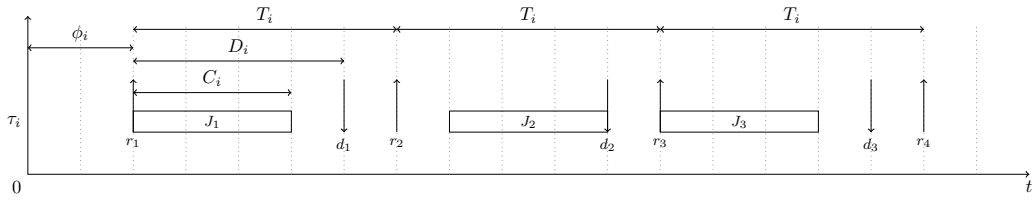


Figure 7.2 – *Modèle de tâches périodiques [Liu 1973]*

7.1.1.1 Modèle de tâches RTEH

Nous étendons le modèle de tâches périodiques initialement introduit par Liu et Layland [Liu 1973] (cf. Figure 7.2) afin d'y intégrer les contraintes d'énergie associées à l'exécution des tâches. Ce nouveau modèle est dénommé *RTEH* (*Real-Time Energy Harvesting*).

Une tâche τ_i est définie par le quintuplet $(\phi_i, C_i, E_i, D_i, T_i)$ avec un instant initial de réveil du premier job de la tâche ϕ_i (également appelé *offset*), une *durée d'exécution au pire-cas* C_i (*worst-case execution time* ou *WCET* en anglais), une *consommation d'énergie au pire-cas* E_i (*worst-case energy consumption* ou *WCEC* en anglais), une *échéance relative* D_i (également appelée *délai critique*), et une *période d'activation* T_i. Le paramètre E_i est supposé exprimé en unités d'énergie (Joules ou Milliwatts). Les tâches τ_i sont considérées *à échéances contraintes* dans le sens où leur délai critique est supposé inférieur ou égal à leur période d'activation (i.e. $D_i \leq T_i$). Dans le cas $D_i = T_i$, les tâches sont dites *à échéances sur requêtes*.

Soit $\mathcal{T}$ représentant *une configuration* de n tâches périodiques : $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$. $\mathcal{T}$ est supposée entièrement caractérisée puisque tous les paramètres de chacun des jobs peuvent être déterminés avant le lancement de l'application. L'ensemble des tâches $\mathcal{T}$ génère une suite infinie de jobs temps réel, dénotée ω. Les jobs arrivent aux instants $(r_i + kT_i)$ pour tout entier k tel que $k \geq 0$. Chaque job J_i de ω est caractérisé par trois paramètres : une date d'arrivée r_i, un temps d'éxécution au pire-cas C_i et une échéance absolue d_i. Chaque job J_i dispose de D_i unités de temps à partir de sa date d'arrivée pour s'exécuter, i.e., il doit recevoir C_i unités d'exécutions sur l'intervalle $[r_i, d_i)$. Soit d_{Max} la plus grande échéance absolue au sein de l'ensemble des jobs (d_{Max} vaut l'infini si ω est un ensemble infini de jobs).

Le facteur d'utilisation du processeur associé à une configuration $\mathcal{T}$ de n tâches pé-

riodiques est défini comme suit : $U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$. Le facteur d'utilisation énergétique $U_e = \sum_{i=1}^{n} \frac{E_i}{T_i}$ caractérise quant à lui la consommation moyenne d'énergie de la configuration $\mathcal{T}$ par unité de temps. Alors que U_p ne peut pas dépasser 1, U_e ne peut pas être plus grand que l'énergie moyenne recoltée par la source d'énergie durant une unité de temps. L'énergie consommée par $\mathcal{T}$ sur un intervalle donné $[t_1, t_2)$ est notée $E_c(t_1, t_2)$.

Soit $H_{\mathcal{T}}$ le plus petit commun multiple des périodes de tâches au sein de $\mathcal{T}$: $H_{\mathcal{T}} = ppcm\{T_1, T_2, ..., T_n\}$. Dans le cas où tous les offsets des tâches valent zéro (i.e. $\forall \tau_i$, $\phi_i = 0$), les tâches sont dites *concrètes* et la configuration $\mathcal{T}$ est qualifiée de *synchrone*. Dans le cas contraire, les tâches sont dites *non-concrètes* et la configuration $\mathcal{T}$ est qualifiée d'*asynchrone*.

On suppose que la consommation d'énergie associée à l'exécution d'un job n'est pas nécessairement proportionnelle à sa durée d'exécution. Les jobs sont supposés indépendants les uns des autres. Ils peuvent être préemptés puis redémarrés plus tard dans le temps, sans surcôut, ni temporel, ni énergétique.

7.1.1.2 Modèle d'énergie

A tout instant t, le récupérateur d'énergie (par exemple un panneau solaire) extrait de l'énergie de l'environnement et la convertit en électricité avec un *taux de recharge instantanée* notée $P_p(t)$, incorporant toutes les pertes. L'énergie récoltée sur un intervalle donné $[t_1, t_2)$ est donnée par $E_p(t_1, t_2) = \int_{t_1}^{t_2} P_p(t)dt$. L'énergie consommée sur ce même intervalle est notée $E_c(t_1, t_2)$. L'énergie produite par la source environnementale est supposée incontrôlable. Par conséquent, celle-ci peut être nulle sur certaines périodes de temps. Nous supposons que la production et la consommation d'énergie peuvent avoir lieu simultanément. Malgré le fait que la puissance délivrée par la source n'est pas toujours constante, nous supposons qu'il est toujours possible de la prédire à court terme avec précision et ce, avec un temps et un coût énergétique négligeables.

Le système considéré utilise par ailleurs un réservoir d'énergie (ex : une batterie rechargeable ou un supercondensateur) qui permet au système de fonctionner de manière continue même s'il n'y a pas suffisamment d'énergie récupérable de l'environnement à un instant t pour exécuter tous les jobs dans le respect de leurs contraintes temporelles et énergétiques. Sa capacité nominale C correspond la quantité d'énergie maximale (exprimée en unités d'énergie) stockable à tout instant. Le réservoir reçoit de l'énergie du récupérateur d'énergie et en délivre à l'unité de traitement. La quantité d'énergie stockée à un instant t est notée $E(t)$. On suppose que le réservoir n'a aucune fuite d'énergie au cours du temps. S'il est entièrement chargé au temps t et que l'on continue de le charger, l'énergie est gaspillée et on parle de *débordement énergétique*. A l'opposé, s'il est entièrement déchargé au temps t, aucun job ne peut être exécuté et l'unité de traitement bascule en mode *idle*. On suppose que l'application démarre avec un réservoir entièrement chargé (c'est-à-dire $E(0) = C$).

7.1.2 Définitions et concepts clés

Nous introduisons ici un certain nombre de définitions et concepts nécessaires à la poursuite de la lecture de ce chapitre.

7.1.2.1 Traitement et laxité temporelle

Commençons par rappeler la notion de *demande processeur* introduite par Baruah et al. [Baruah 1990], notée ici $tdbf_\omega(t_1, t_2)$ (*time demand bound function)* qui se rapporte à la quantité de traitement cumulée générée sur l'intervalle $[t_1, t_2)$ par les jobs de ω possédant une date d'arrivée supérieure ou égale au temps t_1 et une échéance inférieure ou égale à t_2 :

Définition 3 *La demande processeur associée à un ensemble de jobs ω sur l'intervalle $[t_1, t_2)$ est donnée par :*

$$tdbf_\omega(t_1, t_2) = \sum_{t_1 \leq r_k, d_k \leq t_2} C_k \tag{7.1}$$

L'analyse d'ordonnançabilité de EDF repose sur le calcul de la demande processeur sur tout intervalle délimité par le réveil d'un job et l'échéance d'un job en appliquant l'équation 7.1. Il est en effet nécessaire de vérifier que la demande processeur n'excède pas la longueur de l'intervalle considéré. Lorsque les jobs de ω utilisent le processeur à moins de 100%, on observe des temps d'inactivité du processeur, d'où la notion de laxité temporelle (*static slack time* en anglais), notée SST_ω :

Définition 4 *La laxité temporelle associée à un ensemble de jobs ω sur l'intervalle $[t_1, t_2)$ est égale à :*

$$SST_\omega(t_1, t_2) = (t_2 - t_1) - tdbf_\omega(t_1, t_2) \tag{7.2}$$

Définissons ensuite la laxité temporelle statique qui représente la durée maximale durant laquelle le processeur peut rester inactif dans l'intervalle $[t_1, t_2)$, tout en garantissant que les jobs de ω pourront s'exécuter dans le respect de leur échéance.

Définition 5 *La laxité temporelle statique associée à un ensemble de jobs ω est donnée par :*

$$SST_\omega = \min_{0 \leq t_1 < t_2 \leq d_{Max}} SST_\omega(t_1, t_2) \tag{7.3}$$

Par définition, on doit observer que $SST_\omega \geq 0$.

Le calcul dynamique de la laxité temporelle au temps courant t_c associé à un job de tâche τ_i peut quant à lui être spécifié de la manière suivante :

Définition 6 *La laxité temporelle dynamique d'un job de tâche τ_i est donnée par :*

$$ST_{\tau_i}(t_c) = d_i - t_c - tdbf_\omega(\tau_i, t_c, d_i) - \sum_{d_k \leq d_i} C_k(t_c) \tag{7.4}$$

$\sum_{d_k \leq d_i} C_k(t_c)$ correspond ici à la durée d'exécution *restante* des jobs non encore terminés à l'instant t_c et possédant une échance inférieure ou égale à d_i.

Enfin, nous introduisons la notion de *laxité temporelle* d'un ensemble de jobs ω à l'instant courant t_c, notée $ST_\omega(t_c)$. Celle-ci représente la durée maximale durant laquelle le processeur peut rester inactif à l'instant t_c, sans que cela ne remette en cause la faisabilité des jobs susceptibles de le préempter.

Définition 7 *La laxité temporelle associée à un ensemble de jobs ω est donnée par :*

$$ST_\omega(t_c) = \min_{d_i > t_c} ST_{\tau_i}(t_c) \tag{7.5}$$

7.1.2.2 Consommation et laxité énergétique

Nous présentons ici un certain nombre de définitions introduites pour la première fois dans [Chetto 2014a].

A l'image de la *demande processeur*, nous proposons ci-après la notion de *demande énergétique* notée $edbf_\omega(t_1, t_2)$ (*energy demand bound function)* :

Définition 8 *La demande énergétique associée à un ensemble de jobs ω sur l'intervalle $[t_1, t_2)$ est donnée par :*

$$edbf_\omega(t_1, t_2) = \sum_{t_1 \leq r_k, d_k \leq t_2} E_k \tag{7.6}$$

La demande énergétique $edbf_\omega(t_1, t_2)$ de ω sur l'intervalle $[t_1, t_2)$ représente l'énergie cumulée requise par les jobs de ω possédant une date d'arrivée supérieure ou égale au temps t_1 et une échéance inférieure ou égale à t_2. Lorsque les jobs de ω utilisent moins d'énergie que celle exploitable en réalité (l'énergie disponible étant égale à la somme de l'énergie stockée dans le réservoir d'énergie et de l'énergie produite), on observe un surplus d'énergie disponible, d'où la notion de laxité énergétique (*static slack energy* en anglais), notée SSE_ω :

Définition 9 *La laxité énergétique associée à un ensemble de jobs ω sur l'intervalle $[t_1, t_2)$ est égale à :*

$$SSE_\omega(t_1, t_2) = C + E_p(t_1, t_2) - edbf_\omega(t_1, t_2) \tag{7.7}$$

$SSE_\omega(t_1, t_2)$ correspond à la quantité maximale d'énergie disponible sur l'intervalle $[t_1, t_2)$ suite à l'exécution des jobs de ω possédant une date d'arrivée supérieure ou égale au temps t_1 et une échéance inférieure ou égale à t_2.

Définissons à présent la *laxité énergétique statique* qui représente la quantité maximale d'énergie pouvant être consommée ou gaspillée à partir d'un instant donné, tout en garantissant le respect de l'ensemble des contraintes d'énergie de ω.

Définition 10 *La laxité énergétique statique d'un ensemble de jobs ω est donnée par :*

$$SSE_\omega = \min_{0 \leq t_1 < t_2 \leq d_{Max}} SSE_\omega(t_1, t_2) \tag{7.8}$$

Le calcul dynamique de la laxité énergétique au temps courant t_c associé à un job de tâche τ_i peut quant à lui être spécifié de la manière suivante :

Définition 11 *La laxité énergétique dynamique d'un job de tâche τ_i est donnée par :*

$$SE_{\tau_i}(t_c) = E(t_c) + E_p(t_c, d_i) - edbf_\omega(t_c, d_i) \tag{7.9}$$

Enfin, nous introduisons la notion de *laxité énergétique de préemption* d'un ensemble de jobs ω à l'instant courant t_c, notée $PSE_\omega(t_c)$. Celle-ci représente la quantité maximale d'énergie consommable à l'instant t_c par le job actif d'échéance d, sans que cela ne remette en cause la faisabilité des jobs susceptibles de le préempter.

Définition 12 *La laxité énergétique de préemption d'un ensemble de jobs ω est donnée par :*

$$PSE_\omega(t_c) = \min_{t_c < r_i < d_i < d} SE_{\tau_i}(t_c) \tag{7.10}$$

7.1.2.3 Ordonnancement : éléments de terminologie

Nous introduisons ici un certain nombre de définitions associées à l'ordonnancement sur lesquelles nous nous appuyons dans le reste de ce chapitre.

Définition 13 *Une séquence d'ordonnancement Γ pour $\mathcal{T}$ est dite* temporellement valide *si tous les jobs de $\mathcal{T}$ respectent leurs échéances dans Γ avec $E_i = 0, \forall i \in \{1, ..., n\}$.*

Définition 14 *Une séquence d'ordonnancement* Γ *pour* $\mathcal{T}$ *est dite* énergétiquement valide *si tous les jobs de* $\mathcal{T}$ *respectent leurs échéances dans* Γ *avec* $C_i = 0, \forall i \in \{1, ..., n\}$.

Définition 15 *Une configuration de tâches* Γ *est dite* temporellement faisable *s'il existe une séquence temporellement valide pour* $\mathcal{T}$.

Définition 16 *Une configuration de tâches* Γ *est dite* énergétiquement faisable *s'il existe une séquence énergétiquement valide pour* $\mathcal{T}$.

Définition 17 *Un algorithme d'ordonnancement est dit* optimal *s'il trouve une séquence valide (temporellement et énergétiquement) chaque fois qu'il en existe au moins une.*

Définition 18 *Un algorithme d'ordonnancement est dit* en-ligne *s'il prend ses décisions dynamiquement.*

Définition 19 *Un algorithme d'ordonnancement est dit* semi en-ligne *s'il est en-ligne avec une connaissance nécessaire du futur sur un certain intervalle de temps.*

Définition 20 *Un algorithme d'ordonnancement est dit* oisif *s'il est capable de laisser le processeur inactif alors que des jobs attendent d'être exécutés. Dans le cas contraire, il est dit* non-oisif.

Définition 21 *Un algorithme d'ordonnancement est dit* clairvoyant *s'il a une connaissance de la totalité du futur (i.e. caractéristiques des jobs futurs arrivant dans le système et profil de l'énergie produite par la source). Dans le cas contraire, il est dit* non-clairvoyant.

Définition 22 *Un algorithme d'ordonnancement est dit* ld-omnicient *s'il est semi en-ligne et clairvoyant sur les* ld *prochaines unités de temps.*

Définition 23 *La "valeur" d'une tâche définit sa contribution à la performance globale du système. La valeur obtenue est proportionnelle à la durée d'exécution de la tâche. Le système obtient la valeur correspondante d'une tâche seulement si celle-ci est exécutée dans le respect de son échéance. Sinon, le système n'obtient aucune valeur associée à la tâche [Baruah 1991, Buttazzo 2011].*

Définition 24 *La performance d'un algorithme d'ordonnancement est mesurée grâce à son* facteur de compétitivité *: un algorithme en-ligne de facteur de compétitivité* r $(0 < r < 1)$ *garantit une valeur totale cumulée égale au moins* r *fois à la valeur totale cumulée que pourrait fournir le meilleur algorithme clairvoyant [Baruah 1992].*

Définition 25 *Un algorithme d'ordonnancement en-ligne est dit* compétitif *s'il possède une facteur de compétitivité constant strictement supérieur à 0. Sinon, il est dit* non-compétitif.

Définition 26 *Le système souffre d'une* pénurie d'énergie *s'il possède au moins un job de tache* τ_i *qui, lorsqu'il atteint son échéance, est partiellement exécuté car l'énergie requise pour le terminer avant échéance est insuffisante.*

Notre objectif à présent est d'étudier dans quelle mesure l'ordonnancement des tâches peut garantir le fonctionnement perpétuel du système en assurant un mode opératoire conforme aux contraintes de *neutralité énergétique*. Pour ce faire, nous nous intéressons dans un premier temps à l'ordonnanceur EDF pour déterminer s'il peut s'accommoder de façon adaptative aux variations de l'énergie d'alimentation d'un système temps réel autonome.

7.2 Adéquation de EDF aux systèmes temps réel autonomes

7.2.1 Limitations de EDF sous contraintes d'énergie

L'algorithme EDF est l'approche la plus connue pour ordonnancer des tâches temps réel indépendantes en l'absence de surcharge processeur et de limitations d'énergie [Liu 1973, Dertouzos 1974]. EDF est un algorithme en-ligne qui ordonnance la tâche prête (i.e. tâche devant être exécutée et non encore terminée) dont l'échéance absolue est la plus proche du temps courant t. L'ordonnancement en-ligne représente la meilleure option pour un système dont la charge de traitement future n'est pas connue a priori. Il permet en effet de gérer dynamiquement les variations en termes de demandes processeur liées notamment au pattern imprévisible des réveils associés aux jobs sporadiques. L'ordonnanceur EDF existe en version *préemptive* ou *non-préemptive*. Il peut également être implémenté selon un mode *non-oisif* (aussi appelé *EDS* ou *ASAP – As Soon As Possible*) ou *oisif* (aussi appelée *EDL* ou *ALAP – As Late As Possible*). Le mode non-oisif est le plus répondu. Dans ce cas, s'il existe au moins un job à l'état prêt, celui-ci est exécuté, ce qui fait de EDF un ordonnanceur dit *goulu*. La version oisive de EDF a jusqu'à présent trouver son intérêt en présence de tâches apériodiques non critiques pour lesquelles l'objectif est de minimiser leur temps de réponse. Nous étudierons dans la section 7.3 comment tirer bénéfice de cette version afin de prévenir les pénuries d'énergie.

EDF est optimal dans sa version préemptive et sous réserve que les tâches soient indépendantes. Cependant, dans le cas où le système est confronté à une situation de surcharge de traitement (i.e. la demande processeur excède la capacité du système), EDF n'est plus optimal. Dans ce cas, Baruah et al. ont montré qu'aucun ordonnanceur en-ligne, y compris EDF, ne peut garantir un facteur de compétitivité supérieur à 0.25 pour des tâches comportant des densités (i.e. rapports $\frac{C_i}{D_i}$) uniformes [Baruah 1992].

EDF possède de nombreux atouts : son coût d'implémentation (version ASAP) et son overhead d'exécution sont faibles, le nombre de préemptions qu'il génère est réduit par rapport aux autres algorithmes d'ordonnancement. C'est donc tout naturellement que nous nous sommes intéressés à l'adéquation de EDF aux systèmes temps réel autonomes en énergie. Illustrons au travers d'un exemple, les faiblesses de EDF dans le contexte du *energy harvesting*. A noter que pour des raisons de simplifications de la représentation, nous supposerons que les jobs consomment l'énergie de manière constante à raison de $\frac{E_i}{C_i}$ unités d'énergie par unité de temps. On suppose le réservoir d'énergie rempli au départ tel que $E(0) = 3$. Le profil de la source d'énergie est telle que $E_p(0, 2) = 0$ et $E_p(2, 3) = 1$. Soit un premier job issu d'une tâche τ_1 telle que $\phi_1 = 0$, $C_1 = 1$, $E_1 = 1$, $D_1 = 3$.

Supposons dans un premier cas (cf. Figure 7.3(a)) le réveil d'un deuxième job issu d'une deuxième tâche τ_2 avec les paramètres suivants : $\phi_2 = 1$, $C_2 = 1$, $E_2 = 3$, $D_2 = 1$. Au temps $t = 1$, τ_1 termine son exécution dans le respect de son échéance et l'énergie résiduelle dans le réservoir est alors égale à 2 unités. Cependant, étant donné que la consommation d'énergie du job de τ_2 est égale à 3 unités, l'exécution du job de la tâche τ_2 s'interrompt faute d'une quantité d'énergie suffisante pour pouvoir terminer son exécution. Une séquence d'ordonnancement valide existe cependant comme suit : le processeur doit être laissé inactif sur l'intervalle $[0, 1]$ même si le job de τ_1 est prêt ; le job de τ_2 peut ensuite être exécuté car l'énergie disponible au temps $t = 1$ est égale à 3 unités ; au temps $t = 2$, le réservoir d'énergie est vide mais comme l'énergie produite sur l'intervalle $[2, 3]$ est de 1 unité, celle-ci est suffisante pour pouvoir exécuter le job de τ_1 qui peut se terminer dans le respect de son échéance au temps $t = 3$. Ce cas nous montre que EDF ne s'accommode pas des fluctuations de l'énergie disponible et qu'il est incapable d'anticiper les pénuries d'énergie dont pourrait

souffrir le système.

Supposons à présent que l'on décide de laisser le processeur inactif sur l'intervalle $[0, 1]$ (cf. Figure 7.3(b)). L'énergie disponible dans le réservoir à $t = 1$ est alors de 3 unités. Supposons le réveil d'un deuxième job issu d'une deuxième tâche τ_2 avec cette fois-ci les paramètres suivants : $\phi_2 = 1$, $C_2 = 2$, $E_2 = 3$, $D_2 = 2$. Quelle que soit la priorisation des jobs, il est impossible de les exécuter tous les deux avant leur échéance commune située à $t = 3$. Cependant, une séquence d'ordonnancement valide peut être obtenue de la manière suivante : au temps $t = 1$, le job de τ_1 doit avoir terminé son exécution en laissant 2 unités d'énergie disponible dans le réservoir ; le job de τ_2 peut s'exécuter à $t = 2$ en consommant $3/2 = 1.5$ unités d'énergie ; étant donnée que la production d'énergie sur l'intervalle $[2, 3]$ vaut 1, le job de τ_2 peut terminer son exécution dans le respect de son échéance à $t = 3$. Ce cas précis nous montre que EDF conduit à des violations d'échéance, même dans le cas où le temps et l'énergie disponibles pour exécuter les jobs sont en quantité suffisante.

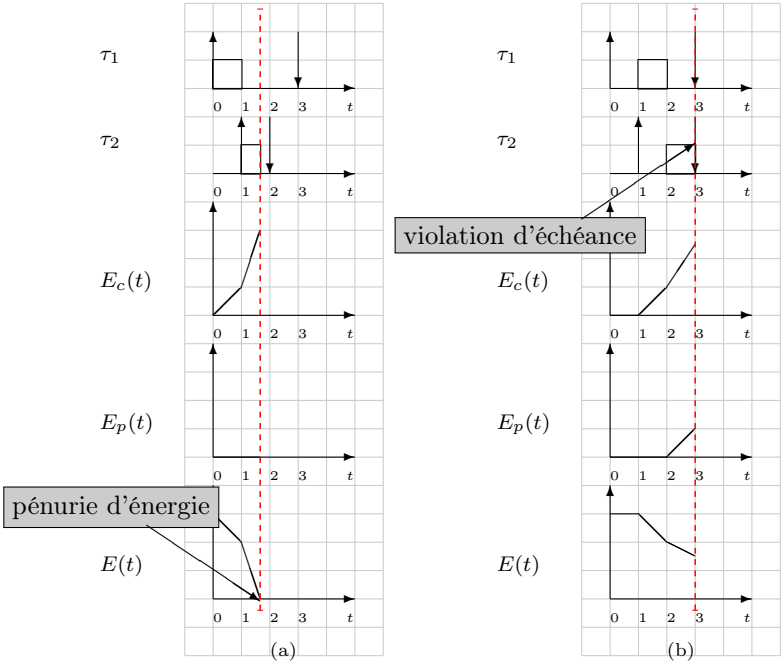


Figure 7.3 – *Illustration de la non-optimalité de EDF du fait (a) des contraintes énergétiques et (b) des contraintes temporelles*

L'exemple illustratif précédent nous montre les limitations de EDF dans le contexte du *energy harvesting* malgré ses nombreuses qualités (faible complexité d'implémentation, faible overhead, aucun dispositif technologique particulier requis). Sa principale faiblesse réside dans sa manière "goulue" de consommer l'énergie disponible, commune à tous les algorithmes d'ordonnancement non oisifs, et conduisant fatalement à des situations de *pénurie d'énergie*. Cependant, compte-tenu du fait que tous les systèmes ne supportent pas la gestion dynamique de puissance (élément support du fonctionnement d'un ordonnanceur oisif), nous avons souhaité montrer que EDF restait le meilleur candidat parmi tous les ordonnanceurs non-oisifs pour un choix d'intégration pour un système temps réel autonome en énergie. Le Théorème 7.2.1 établit ce résultat.

Théorème 7.2.1 *[Chetto 2013] EDF est optimal dans la classe des algorithmes en-ligne non-oisifs pour le modèle RTEH.*

Nous avons également conduit une analyse du facteur de compétitivité de EDF pour le modèle RTEH dans [Chetto 2013] et montré que, même s'il reste le meilleur ordonnanceur non oisif, EDF s'avère non-compétitif (i.e. son facteur de compétitivité est nul), comme l'établit le Théorème 7.2.2.

Théorème 7.2.2 *[Chetto 2013] EDF est un ordonnanceur non-compétitif pour le modèle RTEH.*

Le résultat énoncé dans le Théorème 7.2.2 signifie que la contrainte de devoir disposer d'un ordonnanceur non oisif pour un système temps réel autonome en énergie, implique une performance en-ligne médiocre.

7.2.2 Pré-requis d'un ordonnanceur dans le contexte du *energy harvesting*

Après avoir mis en évidence l'inefficacité des ordonnanceurs en-ligne classiques tels que EDF, nous nous intéressons ici à l'identification de quelques-unes des propriétés clés d'un ordonnanceur dans le contexte des systèmes autonomes en énergie.

Le Théorème 7.2.3 établit que tout algorithme optimal pour le modèle RTEH doit nécessairement être clairvoyant. C'est-à-dire qu'il doit posséder une connaissance complète des éléments futurs (caractéristiques des jobs, profil de l'énergie récupérable, etc.) afin de construire une séquence d'ordonnancement valide.

Théorème 7.2.3 *[Chetto 2014b] Il n'existe aucun algorithme d'ordonnancement en-ligne optimal pour le modèle RTEH qui soit non-clairvoyant.*

Ce résultat prouve que la condition pour construire une séquence d'ordonnancement meilleure que celle produite par un ordonnancement totalement non-clairvoyant comme EDF, implique une omniscience de l'ordonnanceur sur le futur. Le Théorème 7.2.4 complète le théorème précédent en précisant cet intervalle minimum d'anticipation requis par l'ordonnanceur pour prévenir toute pénurie d'énergie et toute violation d'échéance.

Théorème 7.2.4 *[Chetto 2014b] Soit D la plus grande échéance relative de l'application. Aucun algorithme d'ordonnancement en-ligne ld-omniscient ne peut être optimal pour le modèle RTEH si $ld < D$.*

7.3 ED-H : un algorithme d'ordonnancement optimal

7.3.1 Principe

L'algorithme d'ordonnancement ED-H (*Earliest Deadline with energy Harvesting capabilities*) [Chetto 2014a] est une variante de l'ordonnanceur EDF. Les échéances absolues des jobs guident toujours les décisions d'ordonnancement mais, à la différence de EDF, l'ordonnanceur ED-H autorise l'exécution des jobs uniquement si elle ne conduit pas inévitablement à une future situation de pénurie énergétique. Ainsi, afin de prévenir toute surconsommation énergétique, ED-H spécifie dynamiquement des intervalles de mise en veille du processeur. Il adopte donc un comportement oisif de manière à concilier contraintes de temps et d'énergie. La clairvoyance à la fois sur les arrivées des jobs et la production

d'énergie permet à ED-H d'anticiper la possible survenue de pénuries d'énergie, et par conséquent les violations d'échéance. En d'autres termes, ED-H se comporte comme EDF sauf que le processeur est inactif dès que la laxité énergétique de préemption est nulle.

L'ordonnanceur ED-H s'appuie sur plusieurs règles afin de déterminer si un job à l'état prêt peut s'exécuter. En particulier, il comprend une règle permettant de gérer dynamiquement l'activité du processeur en spécifiant les intervalles durant lesquels le processeur doit rester inactif, et ceux durant lesquels un job doit être exécuté. Soit t_c le temps courant. Selon l'algorithme d'ordonnancement ED-H, le processeur ne peut être actif à l'instant t_c si le réservoir d'énergie est vide ou bien si le fait d'éxécuter un job à l'instant t_c entraînera inévitablement une pénurie d'énergie pour l'un des jobs réveillés dans le futur (i.e. le système ne dispose d'aucune laxité énergétique de préemption). Par ailleurs, le processeur ne peut rester inactif si le réservoir d'énergie est plein ou bien si le fait de laisser le processeur inactif empêchera au moins un job de s'exécuter dans le respect de son échéance (i.e. le système ne dispose d'aucune laxité temporelle au temps t_c).

Dans le cas où le réservoir d'énergie n'est ni complètement plein ni complètement vide, et que le système dispose à la fois de laxités temporelle et énergétique, alors l'ordonnanceur peut indifféremment décider d'éxécuter un job ou bien laisser le processeur inactif.

Plus formellement, soit $L_r(t_c)$ la liste des jobs à l'état prêt au temps t_c. L'algorithme d'ordonnancement ED-H répond aux règles de décision suivantes :

- **Règle 1 :** Les priorités affectées selon EDF sont utilisées pour sélectionner le prochain job prêt au sein de $L_r(t_c)$.
- **Règle 2 :** Le processeur est *inactif* sur l'intervalle $[t_c, t_c + 1)$ si $L_r(t_c) = \emptyset$.
- **Règle 3 :** Le processeur est *inactif* sur l'intervalle $[t_c, t_c + 1)$ si $L_r(t_c) \neq \emptyset$ et si *l'une* des conditions suivantes est vérifiée :
  1. $E(t_c) \approx 0$
  2. $PSE_{\mathcal{T}}(t_c) \approx 0$
- **Règle 4 :** Le processeur est *actif* sur l'intervalle $[t_c, t_c + 1)$ si $L_r(t_c) \neq \emptyset$ et si *l'une* des conditions suivantes est vérifiée :
  1. $E(t_c) \approx C$
  2. $ST_{\mathcal{T}}(t_c) = 0$
- **Règle 5 :** Le processeur est indifféremment *actif* ou *inactif* si $L_r(t_c) \neq \emptyset$, $0 < E(t_c) < C$, $ST_{\mathcal{T}}(t_c) > 0$ et $PSE_{\mathcal{T}}(t_c) > 0$.

Par soucis de simplification, $E(t) \approx C$ correspond à $C \leq E(t) < C + e_m$. A l'opposé, le réservoir d'énergie est considéré comme complètement vide au temps t si $0 \leq E(t) < e_m$, noté $E(t) \approx 0$. La règle 3 stipule qu'aucun job ne peut être exécuté si le réservoir d'énergie est vide, ou bien si l'exécution d'un job conduira fatalement à une situation de pénurie énergétique causée par une laxité énergétique de préemption insuffisante. La règle 4 indique que le processeur ne peut rester inactif si le réservoir d'énergie est totalement plein, ou bien si l'inactivité du processeur entraînera nécessairement une violation d'échéance, résultat d'une laxité temporelle nulle. Si le réservoir d'énergie n'est ni plein, ni vide, et que les laxités temporelle et énergétique du système sont non-nulles, alors la règle 5 stipule que le processeur peut rester inactif ou non sans que cela affecte la validité de la séquence d'ordonnancement. A noter que sous ED-H, l'énergie est uniquement gaspillée dans le cas où le réservoir d'énergie est plein et qu'aucun job n'est à l'état prêt.

7.3.1.1 Illustration

Le comportement de ED-H est illustré sur la Figure 7.4 en considérant l'ensemble de jobs ω défini dans la Table 7.1. On suppose que le réservoir d'énergie a une capacité nominale

de 5 unités et qu'il est initialement plein à l'instant $t = 0$, i.e. $E(0) = C = 5$. On suppose également une source d'énergie non-uniforme $P_p(t)$ telle que la quantité d'énergie produite fluctue au cours du temps. Enfin, par soucis de simplification de l'exemple, on supposera que les jobs consomment l'énergie de manière constante (ex : J_3 consomme 2 unités d'énergie durant chacune de ses 4 unités d'exécution, ce qui fait un total de $E_i = 8$ unités d'énergie pour l'ensemble de son exécution).

Table 7.1 – *Ensemble de jobs ω*

Job	r_i	C_i	E_i	d_i
J_1	4	1	5	6
J_2	0	1	1	2
J_3	0	4	8	9

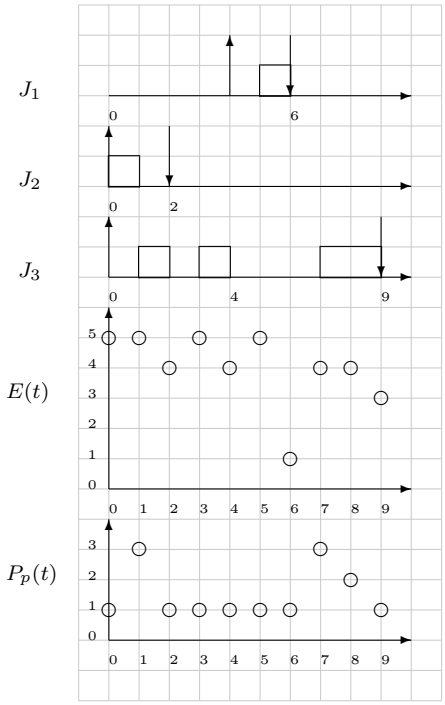


Figure 7.4 – *Séquence d'ordonnancement produite par ED-H*

7.3.2 Propriétés

Le théorème suivant dont la preuve est décrite formellement dans [Chetto 2014a], établit que ED-H est le meilleur candidat pour ordonnancer des systèmes temps réel à récupération d'énergie renouvelable :

Théorème 7.3.1 *[Chetto 2014a] L'algorithme d'ordonnancement ED-H est optimal pour le modèle RTEH.*

Le Théorème 7.3.1 traduit le fait que si une configuration donnée de tâches ne peut pas être fiablement ordonnancée selon ED-H, alors celle-ci n'est pas faisable (i.e. elle n'est ordonnançable selon *aucun* algorithme d'ordonnancement) en considérant la même architecture matérielle et le même contexte énergétique. Cette propriété est générale et reste valable quel que soit le profil de la source d'énergie.

La preuve d'optimalité repose sur le fait que l'ordonnanceur ED-H produit une séquence valide sur un intervalle de temps donnée, à condition que : (i) la demande processeur n'excède pas la durée de l'intervalle, et (ii) la demande énergétique n'excède pas l'énergie disponible dans cet intervalle.

Une solution optimale avait déjà été proposée par ailleurs au travers de l'algorithme LSA (*Lazy Scheduling Algorithm*) introduit par Moser et al. [Moser 2007]. Cependant, celle-ci s'avérait plus restrictive dans le sens où l'énergie consommée par les jobs était supposée proportionnelle à leur durée d'exécution. ED-H n'émet aucune hypothèse à ce niveau, ce qui lui permet d'être plus proche de la réalité. En effet, la puissance instantanée associée à un job en exécution peut varier au cours du temps selon les dispositifs matériels (ex : liaison RF, moteurs, etc.) sollicités par son exécution.

Nous avons vu dans la Section 7.2.2 qu'un des pré-requis en matière d'optimalité pour un ordonnanceur dans le contexte du energy harvesting était d'être omniscient sur une durée au moins égale à D unités de temps, avec D la plus grande échéance relative de l'application. Il en découle donc que l'ordonnanceur ED-H doit disposer d'une clairvoyance sur les D unités de temps suivant l'instant courant pour implémenter ses décisions. Par conséquent, nous introduisons le théorème suivant dont la preuve formelle a été établie dans [Chetto 2014b] :

Théorème 7.3.2 *[Chetto 2014b] L'algorithme d'ordonnancement ED-H est D-omniscient.*

Cette propriété implique pour l'ordonnanceur ED-H de disposer, sur des fenêtres temporelles glissantes de longueur égale à D unités de temps, d'une connaissance non seulement des caractéristique des jobs futurs arrivant dans le système, mais aussi du profil de l'énergie produite par la source. Cela présuppose donc de disposer de méthodes de prédiction de l'énergie [Liu 2011], en fonction du type et de la source d'énergie renouvelable considérés.

7.3.3 Conditions d'ordonnançabilité

L'objectif est de vérifier formellement si l'ordonnanceur ED-H est capable de produire une séquence valide temporellement et énergétiquement pour une configuration de tâches en entrée, en considérant une taille de réservoir d'énergie donnée et un profil de production d'énergie donné. Chetto dans [Chetto 2014a] a établi un test exact d'ordonnançabilité pour ED-H. Un ensemble de jobs est ordonnançable selon ED-H si et seulement si la demande processeur sur tout intervalle n'excède pas la longueur de celui-ci, et si la demande énergétique n'excède pas l'énergie maximale disponible sur ce même intervalle. La complexité de ce test est en $O(n^2)$, avec n le nombre de jobs.

Théorème 7.3.3 *[Chetto 2014a] Un ensemble de jobs ω est ordonnançable selon ED-H si et seulement si pour tout instants t_1, t_2 tels que $t_1 < t_2$,*

$$tdbf_{\omega}(t_1, t_2) \leq t_2 - t_1 \tag{7.11}$$

$$edbf_{\omega}(t_1, t_2) \leq C + E_p(t_1, t_2) \tag{7.12}$$

Le test d'ordonnançabilité peut également s'exprimer sous la forme suivante :

Théorème 7.3.4 *[Chetto 2014a] Un ensemble de jobs ω est ordonnançable selon ED-H si et seulement si $SST_\omega \geq 0$ and $SSE_\omega \geq 0$.*

Le Théorème 7.3.4 permet de ramener la question de l'ordonnançabilité à deux problèmes séparés : (i) un test d'ordonnançabilité du point de vue temporel, et (ii) un test d'ordonnaçabilité du point de vue énergétique.

7.4 Analyse de faisabilité

Toute analyse de faisabilité est menée sur la base des caractéristiques de la plate-forme d'exécution. Rappelons que nous nous intéressons à un système temps réel pour lequel la principale préoccupation est de respecter les contraintes de temps des jobs. Dès lors que la puissance de la source d'énergie est variable, le seul moyen de garantir la faisabilité est d'appliquer un test déterministe, même pessimiste. Par conséquent, la principale problématique est de déterminer la borne la plus proche de manière à limiter l'impact négatif sur le dimensionnement du récupérateur d'énergie et du réservoir d'énergie.

Rappelons également que selon le modèle RTEH, la plate-forme est caractérisée par la vitesse du processeur, la taille du réservoir d'énergie et le profil de la source d'énergie, à confronter aux besoins temporels et énergétiques des jobs.

7.4.1 Test de faisabilité

Considérons le problème de la faisabilité d'un ensemble de tâches périodiques définies selon le modèle RTEH. Etant donné qu'un ordonnanceur optimal tel que ED-H est capable d'ordonnancer tout ensemble faisable de tâches (périodiques ou non), il suffit de considérer cet algorithme pour fournir un test de faisabilité. En effet, en vertu de son optimalité, tester une condition d'ordonnançabilité pour l'algorithme ED-H, revient à tester la faisabilité pour le modèle RTEH. Tout ensemble de tâches périodiques génère une suite de jobs. L'analyse de faisabilité consiste par conséquent à déterminer si cette suite de jobs peut être fiablement ordonnançée par ED-H (i.e. toutes les échéances de toutes les tâches seront respectées).

Découlant directement du Théorème 7.3.3 énoncé précedemment, nous pouvons établir le corollaire suivant :

Corollaire 7.4.1 *[Chetto 2019]* ***(Test de faisabilité)*** *Un ensemble $\mathcal{T}$ de tâches périodiques asynchrones est faisable jusqu'au temps t selon l'ordonnanceur ED-H si et seulement si*

$$tdbf_{\mathcal{T}}(t_1, t_2) \leq t_2 - t_1 \tag{7.13}$$

$$edbf_{\mathcal{T}}(t_1, t_2) \leq C + E_p(t_1, t_2) \tag{7.14}$$

pour tout t_1, t_2, t tels que $0 < t_1 < t_2 < t$,

Dans de nombreuses applications, les dates initiales de réveil des tâches périodiques ne sont pas connues à l'avance car elles dépendent souvent d'événements imprévisibles. Dans ce cas, l'analyse de faisabilité peut être réalisée sur la base du cas synchrone. Le Théorème 7.4.1 exprime le fait que, d'un point de vue ordonnançabilité pour l'ordonnanceur ED-H, le cas synchrone est le scénario pire-cas. Par conséquent, dans le cas de tâches asynchrones pour lesquelles les dates initiales de réveil des tâches peuvent être arbitraires, l'analyse de faisabilité peut être effectuée en considérant le scénario pire-cas synchrone dans lequel toutes les tâches sont initialement réveillées de manière simultanée. Cependant, pour des tâches périodiques asynchrones non-concrètes, les conditions de faisabilité ne sont plus nécessaires, seulement suffisantes.

Théorème 7.4.1 *[Chetto 2019] Soit $\mathcal{T}$ un ensemble de tâches périodiques. Si $\mathcal{T}$ est ordonnançable dans le cas synchrone en utilisant l'ordonnanceur ED-H, alors $\mathcal{T}$ est ordonnançable dans tous les cas asynchrones.*

Le test de faisabilité peut être effectué hors-ligne dès lors que le profil de la source d'énergie est connu sur toute la durée de vie de l'application. Dans le cas contraire, le test doit être réalisé en-ligne sur un horizon temporel fonction des techniques de prédiction de l'énergie considérées, et en limitant au maximum l'overhead d'exécution.

Dans la plupart des cas, il est impossible de prédire le profil de l'énergie récupérable sur un horizon lointain. Par conséquent, il est impossible d'implémenter le test de faisabilité au cours d'une phase pré-opérationnelle. Une solution à ce problème réside dans l'approximation de l'énergie récoltable au pire-cas (i.e. la puissance instantanée minimale récupérable à tout instant depuis l'environnement). Cependant, cette approche nécessite de s'appuyer sur les propriétés de "robustesse" du système afin de pourvoir tester l'ordonnnançabilité hors-ligne. Le système doit en effet rester stable même en cas de changements positifs de ses paramètres caractéristiques (ex : durées d'exécution au pire-cas, énergie récoltée au pire-cas, etc.). Il convient en particulier de prouver au préalable que le test de faisabilité est "robuste" vis-à-vis de la puissance de la source d'énergie.

7.4.2 *"Robustesse"* du test de faisabilité

Nous définissons ici la notion de *robustesse* comme la propriété d'un algorithme d'ordonnancement ou d'un test de faisabilité à maintenir le respect de toutes les échéances des jobs si la puissance $P_p(t)$ de la source est telle que, pour tout instant t, celle-ci est toujours supérieure ou égale à un seuil donné considéré dans l'analyse d'ordonnançabilité. En d'autres termes, un algorithme d'ordonnancement et/ou un test d'ordonnaçabilité associée à une politique d'ordonnancement donnée est *robuste* si tout ensemble de tâches jugé ordonnançable, reste ordonnançable si la puissance de la source d'énergie s'avère plus élévée.

L'avantage de prouver que ED-H est robuste réside dans le fait de pouvoir appliquer l'analyse de faisabilité sur la base d'un minorant sur la puissance instantanée récupérable depuis l'environnement à tout instant. Cette question est fondamentale dans le sens où :

- beaucoup de sources d'énergie renouvelables (ex : vibrations) sont instables par nature. Etant donné qu'elles présentent des fluctuations imprévisibles, aucun test exact ne peut être effectué hors-ligne afin de garantir l'ordonnançabilité lors de l'excution de l'application.
- il peut être impossible d'instrumenter et de mesurer avec précision la puissance de la source, même si celle-ci est stable.

Ainsi, nous avons énoncé le théorème suivant :

Théorème 7.4.2 *[Chetto 2019] Le test de faisabilité est robuste vis-à-vis de la puissance de la source d'énergie.*

Le théorème 7.4.2 signifie que tout ensemble de tâches, periodiques ou non, ordonnançable selon ED-H avec un récupérateur d'énergie donné, reste ordonnançable avec un récupérateur d'énergie plus puissant. Par conséquent, l'idée est de mener l'analyse de faisabilité sur la base d'un scénario pire-cas pour lequel le récupérateur d'énergie délivre une puissance plus faible mais constante.

7.4.3 Intérêt de l'hypothèse d'une puissance constante

L'énergie solaire peut être récupérée efficacement au travers de modules photovoltaïques. Le niveau de radiation solaire fluctue au cours du temps puisqu'il suit un cycle diurne. Néanmoins, on peut considérer que la puissance récupérée est constante sur de "grands" intervalles, au regard notamment des périodes des tâches. C'est le cas également de l'énergie récoltée sur le corps humain via des générateurs thermoélectriques qui, exploitant les différences de température existant entre l'environnement et le corps humain, fournissent une source d'énergie continue et quasi-constante. Les variations d'énergie peuvent là-encore être considérées comme faibles au regard des périodes d'activation des tâches.

Soit P_p la puissance constante récupérée depuis la source d'énergie environnementale supposée entièrement caractérisée sur un horizon temporel donné. Supposons que l'ensemble $\mathcal{T}$ des tâches périodiques satisfasse $U_p \leq 1$ et $U_e \leq P_p$. En d'autres termes, nous supposons que la puissance récupérée est suffisante pour satisfaire les demandes énergétiques de l'ensemble de tâches sur ce même horizon temporel. Dans le cas contraire, les violations d'échéances sont inévitables.

Nous nous intéressons à la question suivante : *sous l'hypothèse d'une source d'énergie de puissance constante, comment décider si un ensemble $\mathcal{T}$ de tâches périodiques synchrones est faisable ou non ?* En l'absence de contraintes énergétiques, $H_{\mathcal{T}} = lcm\{T_1, T_2, ..., T_n\}$ représente un horizon temporel naturel au sein duquel l'ordonnançabilité associée à la séquence d'ordonnancement générée par un ordonnanceur optimal tel que EDF, est vérifiée. Néanmoins, il n'est pas évident que cet horizon temporel existe toujours avec l'ordonnanceur ED-H en considérant les contraintes énergétiques.

Considérons un ensemble de tâches faisable. Nous prouvons à présent que, si le réservoir d'énergie est plein (i.e. $E(0) = C$) au moment de l'initialisation du système, alors il sera à nouveau plein à la fin de la première hyperpériode, et également à la fin de toutes les hyperpériodes suivantes.

Lemme 7.4.1 *[Chetto 2019] Soit $\mathcal{T}$ un ensemble de tâches périodiques tel que $U_p \leq 1$ et $U_e \leq P_p$. Si la séquence d'ordonnancement produite par ED-H est valide sur l'intervalle $[O, H_{\mathcal{T}})$, alors $E(H_{\mathcal{T}}) = C$.*

Illustrons ce résultat au travers d'un exemple. Considérons l'ensemble de tâches périodiques dont les caractéristiques sont données dans la Table 7.2. La taille du réservoir d'énergie est supposée telle que $C = 4$. On suppose que celui-ci est initialement plein (i.e. $E(0) = C$). La puissance de production d'énergie est constante avec $P_p = 1$. La séquence d'ordonnancement générée par ED-H sur la première hyperpériode est décrite sur la Figure 7.5.

Le test de faisabilité revient par conséquent à simplement vérifier qu'il n'y a aucune violation d'échéance ni aucune pénurie d'énergie sur une hyperpériode, comme énoncé dans le Théorème 7.4.3. Il se révèle être d'une grande pertinence d'un point de vue pratique en offrant une faible complexité d'implémentation.

Table 7.2 – *Ensemble $\mathcal{T}$ de tâches périodiques*

Task	r_i	C_i	E_i	D_i	T_i
τ_1	0	3	6	7	20
τ_2	0	2	2	4	5
τ_3	0	1	2	8	10

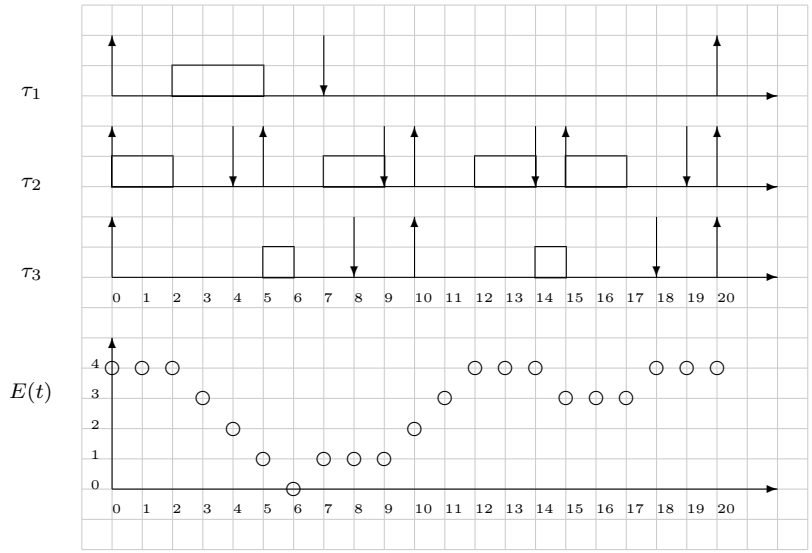


Figure 7.5 – *Séquence d'ordonnancement générée par ED-H sur la première hyperpériode*

Théorème 7.4.3 *[Chetto 2019] Un ensemble $\mathcal{T}$ de tâches synchrones est faisable si et seulement si toutes les échéances dans l'intervalle $[O, H_{\mathcal{T}})$ sont respectées selon l'ordonnanceur ED-H.*

A noter que dans le cas où la puissance récupérée depuis l'environnement peut présentée des fluctuations stochastiques lors de l'exécution et qu'aucun minorant sur la puissance récupérée ne peut alors être calculé en-ligne, seul un test d'admission en-ligne peut être implémenté de manière à recalculer l'ordonnançabilité sur la base de la prédiction d'une puissance constante. Dans ce cas, le fait de ramener l'analyse d'ordonnaçabilité à une seule hyperpériode permet de limiter à la fois l'overhead à l'exécution et l'empreinte mémoire associés.

7.5 Bilan et diffusion des résultats

La technologie du *energy harvesting* destinée à capter l'énergie issue de l'environnement ou de l'activité humaine apporte une solution à l'autonomie et à la gestion des piles de nombreux systèmes embarqués. Elle se révèle en particulier parfaitement adaptée aux objets connectés, disposant par définition de peu de ressources. Ces objets interagissant le plus souvent avec le monde extérieur avec des contraintes temps réel, il est donc naturel de se poser la question de la faisabilité du *energy harvesting* dans le contexte temps réel, et plus précisément au niveau des services offerts par l'OS : c'est *l'ordonnanceur* qui nous intéresse dans le cas présent. Les premiers travaux de recherche concernant l'ordonnancement temps réel dans le contexte du *energy harvesting* ont été proposés au début des années 2000s, en supposant que l'énergie récupérée depuis l'environnement était constante et en considérant des tâches périodiques de périodes identiques [Allavena 2001]. En 2007, Moser et al. ont considéré un modèle d'énergie plus réaliste dans lequel l'énergie récupérée pouvait être non uniforme mais prédictible dans un futur proche. Sous l'hypothèse d'une prédiction précise de l'énergie récoltable dans le futur, l'algorithme d'ordonnancement *Lazy Scheduling algorithm (LSA)*

fournit une solution optimale [Moser 2007]. Des extensions DVFS de LSA ont été proposées dans [Liu 2008] et [Liu 2009]. Les auteurs émettent cependant une hypothèse restrictive : les durées d'exécution au pire cas C_i des tâches sont proportionnelles à leur consommation d'énergie au pire-cas E_i. Pour de nombreux systèmes réels, la corrélation entre ces paramètres n'existe pas [Jayaseelan 2006]. Par exemple, un job transmettant des données via une interface RF consommera une quantité d'énergie importante sur un laps de temps assez court.

Forts de ce constat, nous avons donc tout d'abord présenté un nouveau modèle de tâches temps réel intégrant les contraintes d'énergie associées à l'exécution des tâches, le modèle RTEH, dans lequel les paramètres C_i et E_i ne sont pas nécessairement proportionnels. Puis, nous avons étudié le comportement d'un des ordonnanceurs temps réel "phares", *EDF*, sous contraintes énergétiques. Nous avons montré que EDF n'est pas compétitif et que seul un ordonnanceur clairvoyant peut être optimal pour le modèle de tâches RTEH. Nous avons également établi que l'horizon de clairvoyance requis par tout ordonnanceur optimal pour construire une séquence énergétiquement valide est au minimum la plus grande échéance relative D de l'application. Il est donc nécessaire de disposer d'une prédiction de la quantitié d'énergie pouvant être récupérée sur cet intervalle de temps. En pratique, il s'agit d'estimer cycliquement l'énergie récoltable sur des fenêtres temporelles glissantes. Nous avons cependant montré que EDF reste le meilleur ordonnanceur non-oisif dans le contexte RTEH, et qu'il offre, de par sa simplicité d'implémentation, une solution attractive pour tout système pour lequel une estimation prédictive de l'énergie récoltable est impossible.

Nous avons ensuite présenté un ordonnanceur optimal, *ED-H* [Chetto 2014a], variante de EDF. Contrairement à ce dernier, ED-H n'est pas un ordonnanceur "goulu". Il prévient la surconsommation énergétique, et par conséquent les situations de pénurie d'énergie, en spécifiant des intervalles de mise en veille du processeur. Afin d'atteindre l'optimalité, ED-H doit disposer d'une connaissance non seulement des caractéristiques temporelles et énergétiques des jobs mais aussi de la quantité d'énergie récoltable et ce, sur les D unités de temps consécutives au temps courant. Ce pré-requis d'anticipation constitue un verrou technologique majeur à la mise en oeuvre de l'ordonnanceur ED-H. Cela suppose de disposer en particulier de méthodes de prédiction de l'énergie telles que les techniques *Exponentially Weighted Moving Average (EWMA)* [Hsu 2006] et *Weather Conditioned Moving Average (WCMA)* [Piorno 2009] proposées dans le cas de l'énergie solaire par exemple. La prédiction en-ligne de l'énergie future disponible a également été traitée par le passé dans [Lu 2010].

Nous avons enfin exploré le problème de la faisabilité d'un ensemble de tâches temps réel à contraintes strictes dans le contexte du *energy harvesting*. Nous avons mené une analyse de la "robustesse" du test de faisabilité proposé en tant que capacité du système à rester *stable* même en cas de changements positifs dans son environnement d'exécution. Dans un premier temps, nous avons prouvé que notre test de faisabilité était robuste vis-à-vis de la puissance de la source d'énergie. Dans un second temps, sous l'hypothèse d'une puissance constante pour la source d'énergie (ou d'un minorant sur la puissance lorsqu'une prédiction précise de l'énergie environnementale récoltable est impossible), nous avons prouvé que la faisabilité de tâches asynchrones pour lesquelles les dates de réveil initiales ne sont pas connues, peut être vérifiée en considérant un pattern d'activation synchrone qui correspond au scénario pire-cas. Dans ce cas, nous avons déterminé qu'il était suffisant de vérifier l'ordonnançabilité des tâches sur une seule hyperpériode, ce qui permet de disposer d'un test pour lequel le gain d'efficience à l'implémentation est important.

Ce travail a été réalisé en collaboration avec Maryline Chetto (Professeure des Universités à l'IUT de Nantes). Notre collaboration a notamment donné lieu à la rédaction des livres [Chetto 2016, Chetto 2017, Chetto 2020b] publiés aux éditions ISTE et Elsevier. Les contributions sur l'analyse de EDF dans le contexte du *energy harvesting* ont été publiées dans les revues *IEEE Transactions on Computers* [Chetto 2013] et *Real-Time Systems Journal* [Chetto 2014b]. Les résultats sur la faisabilité d'un ensemble de tâches sous contraintes de temps et d'énergie ont été publiés dans la revue *Sustainable Computing : Informatics and Systems* [Chetto 2019]. Plus récemment, une synthèse des principaux résultats théoriques énoncés dans ce chapitre a été présentée à la conférence *IEEE International Conference on Intelligent Systems and Computer Vision (ISCV'20)* [Chetto 2020a].

Chapitre 8

Gestion de la surcharge dans les systèmes monoprocesseur à récupération d'énergie renouvelable

Sommaire

8.1 Résolution des cas de surcharge

Lorsque les ressources du système (processeur, mémoire, énergie, etc.) nécessaires pour exécuter les tâches sont plus importantes que les ressources disponibles, le système entre en phase de *surcharge*. Idéalement, un système temps réel ne devrait jamais être surchargé et son comportement devrait être conforme à la prévision faite au moment de sa conception. En réalité, les systèmes peuvent être soumis en pratique à des surcharges temporaires de traitement et/ou de consommation énergétique. Il est alors impossible de respecter toutes les

échéances des tâches. En particulier, des tâches critiques peuvent manquer leur échéance, ce qui peut avoir des conséquences catastrophiques sur le système ou l'environnement contrôlé. Par conséquent, pour faire face aux surcharges, la performance du système doit pouvoir être dégradée localement ou globalement de manière à conserver le système dans un état sécuritaire. En effet, cette maîtrise est nécessaire car un système qui s'effrondre et subit une importante perte de performance lors d'une situation d'urgence tendra davantage à empirer la situation qu'à l'améliorer.

8.1.1 Les causes de surcharge

On distingue plusieurs causes de surcharge relevant de paramètres divers et variés associés au système et à son environnement. En premier lieu, la surcharge peut être conséquente à une mauvaise conception du système lui-même. Les durées d'exécution effectives des tâches peuvent s'avérer supérieures aux durées d'exécution pire-cas estimées en amont par une analyse WCET (*Worst-Case Execution Time*). Les surcoûts associés au système d'exploitation ou au matériel peuvent également avoir été sous-estimés ou mésestimés. Le mauvais fonctionnement d'un périphérique d'entrée peut également mener à un cas de surcharge du système. Les exceptions système soulevées par le noyau peuvent aussi faire partie des facteurs influant sur la charge potentielle infligée au système. De même, des variations au niveau de l'environnement, ou bien l'arrivée simultanée d'événements asynchrones, vont contribuer à moduler l'évolution de la charge. Une ligne de communication encombrée, une transmission défectueuse ou la contention sur une ressource critique sont autant de facteurs pouvant influer sur la fin d'exécution tardive d'une tâche, donnant lieu à un cas de surcharge. Enfin, on peut citer d'autres causes de surcharge : un flux élevé de tâches apériodiques, un test d'ordonnançabilité erroné, etc.

8.1.2 "Robustesse" de RM et EDF vis-à-vis des cas de surcharge

On considère ici qu'un algorithme d'ordonnancement est *robuste* si la séquence d'ordonnancement générée reste valide, même en cas de surcharge du système.

Le comportement de Rate-Monotonic en surcharge est tel que ce sont les tâches de plus longues périodes qui violent leur échéance [Delacroix 1994]. Une solution pratique consiste alors à réduire la période d'exécution d'une tâche pour la rendre prioritaire et non impactée par une surcharge. Cependant, cette action délicate entraîne une augmentation de l'utilisation processeur de la tâche et une confusion des notions d'urgence et d'importance.

Quant à l'algorithme EDF, son comportement est *instable* (i.e. non déterministe) en cas de surcharge. Des expériences menées par Locke [Locke 1987] ont montré que EDF est sujet à *l'effet domino* (cas d'avalanche de fautes temporelles, les tâches qui manquent leur échéances retardent les autres qui à leur tour manquent leur échéances) et que sa performance est rapidement dégradée sur les intervalles de surcharge. Ceci est dû au fait que l'algorithme EDF octroît toujours la plus grande priorité à la tâche dont l'échéance est la plus proche. En cas de surcharge, EDF ne fournit aucun type de garantie sur l'identité des tâches qui respecteront leurs contraintes temporelles : ce comportement qualifié d'indéterministe est particulièrement indésirable lorsque des cas de surcharges, dus à des modifications de l'environnement par exemple, surviennent inopinément et de manière intermittente.

Buttazzo et Stankovic ont proposé dans [Buttazzo 1993] plusieurs ordonnanceurs dérivés de EDF, robustes aux cas de surcharge. Les algorithmes d'ordonnancement GED (*Guaranteed Earliest Deadline*), RED (*Robust Earliest Deadline with single task rejection*), et MED (*Robust Earliest Deadline with multiple task rejection*), incorporent des politiques

d'acceptation et/ou d'abandon de tâches, basées sur l'évaluation de l'importance des tâches et de leur tolérance à respecter ou non leur échéance.

8.1.3 Méthodes de résolution des cas de surcharge de traitement

L'objectif de la résolution des cas de surcharge est de concevoir des stratégies d'ordonnancement assurant une dégradation contrôlée de la *Qualité de Service (QoS)*, celle-ci étant évaluée en termes de *ratio du nombre de jobs exécutés dans le respect de leur échéance sur le nombre total de jobs*. Buttazzo regroupe les méthodes existantes pour faire face aux cas de surcharge de traitement en trois catégories distinctes [Buttazzo 2011] :

— *les approches à pertes contraintes* [Koren 1995, Hamdaoui 1995, Bernat 2001] : la charge processeur est réduite en abandonnant l'exécution de certains jobs de tâches tout en garantissant l'exécution d'un nombre minimal de jobs dans le respect de leur échéance.

— *les approches à adaptation de période* [Buttazzo 1998] : les périodes des tâches sont dites "élastiques" et peuvent être allongées de manière à ramener la charge processeur en-dessous d'un certain seuil.

— *les approches à tâches bi-partites* [Campbell 1979, Silly 1986, Liu 1987] : les tâches existent en deux versions dont les durées d'exécution sont distinctes. En cas de surcharge, une commutation sur la version la plus "courte" permet d'abaisser la charge processeur.

Dans ce chapitre, nous nous focalisons uniquement sur les approches à pertes contraintes pour maîtriser le niveau de dégradation de la QoS en cas de surcharge. Plus particulièrement, nous appuyons nos travaux sur le modèle de tâches *firm* Skip-Over [Koren 1995] introduit par Koren et Shasha. Ce modèle et les algorithmes d'ordonnancement associés sont présentés dans la section suivante.

8.1.4 L'approche Skip-over

8.1.4.1 Modèle de tâches *firm*

Au sein de ce modèle [Koren 1995], chaque tâche périodique τ_i autorisant des pertes au sens Skip-Over, est caractérisée par les paramètres suivants :

$$\tau_i(C_i, T_i, D_i, s_i)$$

où l'on voit apparaître le paramètre additionnel s_i, tel que $2 \leq s_i \leq \infty$, qui représente la distance maximale entre deux abandons successifs de jobs pour la tâche. Par exemple, si $s_i = 3$, la tâche peut abandonner un job toutes ses trois activations. Ce paramètre traduit le seuil de tolérance de la tâche à subir des violations d'échéance pour ses jobs. Il peut être vu comme une métrique de QoS : plus la valeur du paramètre s_i est élevée, plus la QoS observée pour la tâche en termes de nombre de jobs exécutés dans le respect de leur échéance, sera élevée. Lorsque $s_i = \infty$, aucun abandon de job n'est toléré. Dans ce cas, la tâche est équivalente à une tâche temps réel *hard*.

A titre d'illustration, nous pouvons considérer la Figure 8.1 :

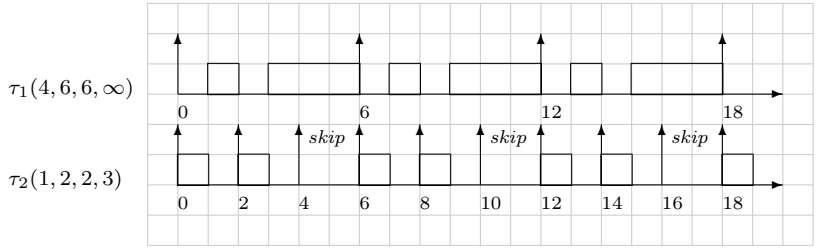


Figure 8.1 – *Exemple de pertes au sens Skip-Over*

Le système est surchargé ($U_p = \sum_{i=1}^{n} \frac{C_i}{P_i} = \frac{4}{6} + \frac{1}{2} = 1.17$), mais les tâches peuvent être ordonnancées si τ_2 n'exécute pas un job sur trois.

Selon la terminologie utilisée par Koren et Shasha [Koren 1995], chaque job peut être *rouge* ou *bleu*. L'exécution d'un job *rouge* dans le respect de son échéance est obligatoire. Les jobs *bleus* peuvent, quant à eux, être abandonnés à n'importe quel instant. Si un job *bleu* a été abandonné, les $(s_i - 1)$ jobs suivants doivent être *rouges* afin de respecter une distance minimale de s_i périodes entre deux abandons successifs de jobs pour la tâche. Si un job *bleu* termine son exécution dans le respect de son échéance, le job suivant sera à nouveau *bleu*, étant donné qu'il n'a pas été abandonné.

8.1.4.2 RTO et BWP : ordonnanceurs avec QoS garantie

Sur la base de ce modèle, Koren et Shasha ont proposé deux algorithmes d'ordonnancement en-ligne [Koren 1995] basés sur EDF :

— RTO (*Red Tasks Only*) qui ordonnance les jobs rouges uniquement et abandonne systématiquement les jobs bleus. Il garantit ainsi simplement une QoS minimale pour le système,

— BWP (*Blue When Possible*) qui ordonnance les jobs rouges et autorise également les jobs bleus à s'exécuter dès lors qu'il n'y a plus aucun job rouge à l'état prêt. La QoS offerte par BWP est par conséquent meilleure dans le sens où un certain nombre de jobs bleus peuvent effectivement être exécutés dans le respect de leur échéance.

Les auteurs ont montré que, dans le cas général, ces algorithmes ne sont pas optimaux, exception faite du cas où les tâches sont activées de manière synchone et que les $(s_i - 1)$ premiers jobs de chaque tâche sont rouges. La configuration de tâches est alors qualifiée de *profondément rouge* (*deeply-red* en anglais).

8.1.4.3 Eléments de terminologie

Sous l'hypothèse *profondément rouge* et en considérant la demande processeur associées aux jobs *rouges* uniquement, la notion de *demande processeur équivalente* sur tout intervalle $[0, L)$, est alors définie comme suit :

Définition 27 *[Koren 1995] La demande processeur* équivalente *sur tout intervalle* $[0, L)$*, associée à une configuration* $\mathcal{T} = \{\tau_i(C_i, T_i, s_i)\}$ profondément rouge *de n tâches périodiques autorisant des pertes au sens Skip-Over, est donnée par :*

$$tdbf^*_{\mathcal{T}}(0, L) = \sum_{i=1}^{n} (\lfloor \frac{L}{T_i} \rfloor - \lfloor \frac{L}{T_i s_i} \rfloor) C_i \tag{8.1}$$

Caccamo et Buttazzo ont par ailleurs introduit la notion de *facteur équivalent d'utilisation du processeur* [Caccamo 1997] qui représente la charge imposée au processeur dans le mode de fonctionnement le plus dégradé (seuls les jobs rouges sont exécutés) :

Définition 28 *[Caccamo 1997] Etant donné une configuration* $\mathcal{T} = \{\tau_i(C_i, T_i, s_i)\}$ *de n tâches périodiques autorisant des pertes au sens Skip-Over, le facteur d'utilisation équivalent du processeur est défini par :*

$$U^*_p = \max_{L \geq 0} \left\{ \frac{tdbf^*_{\mathcal{T}}(0, L)}{L} \right\} \tag{8.2}$$

Les auteurs ont montré qu'il est suffisant d'évaluer $tdbf^*_{\mathcal{T}}(0, L)$ pour tous les instants t correspondant aux échéances des jobs rouges sur *l'hyperpériode équivalente* $H^* = PPCM(T_1.s_1, ..., T_i.s_i, ..., T_n.s_n)$.

8.1.4.4 Analyse de faisabilité

La configuration *profondément rouge* représentant le pire-cas [Koren 1995], il en découle que si une configuration de tâches *profondément rouge* est ordonnançable selon RTO, alors la même configuration de tâches le sera également selon RTO pour tout pattern d'activation différent. Etant donnée l'optimalité de RTO sous la condition *profondément rouge*, le test de faisabilité est établi comme suit :

Théorème 8.1.1 *[Caccamo 1997] Une configuration* $\mathcal{T} = \{\tau_i(C_i, T_i, s_i)\}$ *profondément rouge de n tâches périodiques autorisant des pertes au sens Skip-Over, est faisable* temporellement *si et seulement si :*

$$U^*_p \leq 1 \tag{8.3}$$

8.2 Ordonnancement de tâches périodiques sous contraintes de QoS et d'énergie

8.2.1 Modèle de tâches RTEH-QoS

Nous étendons le modèle RTEH de tâches périodiques sous contraintes d'énergie (cf. Section 7.1.1.1) afin d'y intégrer des contraintes de QoS associées à l'exécution des tâches. Ce nouveau modèle est dénommé *RTEH-QoS* (*Real-Time Energy Harvesting with QoS guarantees*). Chaque tâche périodique τ_i à contraintes d'énergie et autorisant des pertes au sens Skip-Over, est caractérisée par les paramètres suivants :

$$\tau_i(\phi_i, C_i, E_i, D_i, T_i, s_i)$$

Le modèle intègre à présent à la fois les contraintes temporelles (paramètres C_i, T_i et D_i), les contraintes énergétiques (paramètre E_i), et les contraintes de QoS (paramètre s_i).

8.2.2 Green-RTO et Green-BWP

Cette contribution réalisée dans le cadre de la thèse de Maïssa Abdallah [Abdallah 2014a] (2010-2014) est antérieure à la proposition de l'ordonnanceur optimal ED-H [Chetto 2014a] décrit dans le chapitre 7. Les travaux présentés dans ce chapitre s'appuient donc sur une heuristique d'ordonnancement appelée *EDeg* (*Earliest Deadline with energy guarantee*), proposée par El Ghor et. al en 2009 [Chetto 2009]. Son fonctionnement est rappelé ci-après.

8.2.2.1 EDeg

L'heuristique EDeg [Chetto 2009] (cf. Algorithme 3) est un algorithme d'ordonnancement pour des configurations de tâches périodiques. EDeg est une variante de l'algorithme EDF contrôlant les intervalles d'oisiveté du processeur afin de fournir l'énergie nécessaire à l'exécution des tâches temps réel dans le respect de leur échéance. Ainsi, un job à l'état prêt est exécuté uniquement dans le cas où le réservoir d'énergie n'est pas vide et qu'il y a assez d'énergie dans le système pour ne pas compromettre l'exécution des jobs futurs. Dans le cas contraire, si l'énergie disponible n'est pas suffisante ou si le réservoir est vide, le processeur doit être inactif pendant une durée donnée permettant de toujours garantir le respect des contraintes temporelles des jobs. Le niveau d'énergie dans le réservoir est supposé évoluer entre deux bornes E_{min} et E_{max} telles $C = E_{max} - E_{min}$. *PENDING* est une variable booléenne qui prend la valeur *'true'* dès lors qu'il y a au moins une tâche prête en attente, *'false'* dans le cas contraire.

Algorithm 3 EDeg [Chetto 2009]

```
t : temps courant
while true do
  while PENDING=true do
    while (E(t) > E_min and Slack.energy(t) > 0) do
      execute()
    end while
    while (E(t) < E_max and Slack.time(t) > 0) do
      wait()
    end while
  end while
  while PENDING=false do
    wait()
  end while
end while
```

Plus formellement, le fonctionnement de l'heuristique EDeg repose sur deux concepts clés :

- *la laxité temporelle du système au temps courant t*, notée $Slack.time(t)$, égale à la longueur du plus grand intervalle de temps débutant au temps t durant lequel le processeur peut rester inactif sans remettre en cause le respect des échéances des tâches,
- *la laxité énergétique du système au temps courant t*, notée $Slack.energy(t)$, représentant la quantité maximale d'énergie pouvant être consommée au temps courant t tout en satisfaisant les contraintes temporelles des tâches.

Tel que décrit dans [Silly 1999], la *laxité temporelle du système au temps courant* t associée à un ensemble de tâches périodiques peut être calculé en-ligne en déterminant la localisation et les durées des temps creux d'une séquence d'ordonnancement *EDL* (*Earliest Deadline as Late as possible*). La complexité de ce calcul est en $O(K.n)$ où n représente le nombre de tâches périodiques et K est égal à $\lfloor \frac{R}{p} \rfloor$ avec R et p correspondant respectivement à la plus grande échéance et la plus petite période parmi les tâches prêtes.

La *laxité énergétique du système au temps courant* t est, quant à elle, égale à la valeur minimale parmi les laxités énergétiques calculées sur l'ensemble des jobs actifs à t ayant une date de réveil supérieure ou égale à t et une échéance absolue inférieure à d (d correspondant à l'échéance absolue du job le plus prioritaire à l'instant t selon EDF). La complexité de ce calcul est également en $O(K.n)$. La laxité énergétique du système au temps courant t est calculée uniquement lorsqu'il y a au moins un job J_i dont la date de réveil est postérieure à t et dont l'échéance absolue d_j est inférieure ou égale à celle du job actif le plus prioritaire au temps t. La laxité énergétique de J_i est calculée comme suit :

$$Slack.energy(J_i, t) = E(t) + E_p(t, d_i) - \sum_{\substack{Jk \\ t \leq r_k, d_k \leq d_i}} E_k \tag{8.4}$$

L'ordonnanceur EDeg est illustré sur la Figure 8.2, en considérant l'ensemble de tâches périodiques $\mathcal{T}$=$\{\tau_i(C_i,D_i, T_i, E_i)\} = \{\tau_1(2, 7, 20, 16), \tau_2(1, 4, 5, 10), \tau_3(1, 9, 10, 6)\}$. On suppose que le réservoir d'énergie est initialement plein (i.e. $E(0) = C = 10$) et que l'énergie récupérée depuis l'environnement est constante avec $P_p = 4$. Par soucis de simplification de la représentation, on suppose enfin que les tâches consomment l'énergie à taux constant.

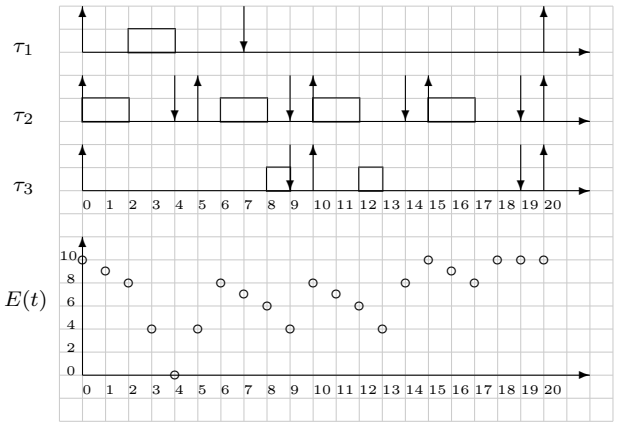


Figure 8.2 – *Séquence d'ordonnancement générée par EDeg [Chetto 2009]*

L'ordonnanceur EDeg a fait l'objet d'une étude de performance exhaustive rapportée dans la thèse de El Ghor [El Ghor 2012] et publiée dans [Ghor 2011, El Ghor 2013]. L'intérêt de EDeg par rapport à EDF a notamment été montré au travers du nombre de configurations de tâches ordonnançables en fonction de la taille du réservoir d'énergie, du profil de consommation d'énergie et du temps moyen d'oisiveté du processeur.

8.2.2.2 Green-RTO

L'algorithme d'ordonnancement Green-RTO est basé sur les algorithmes RTO et EDeg. A l'image de RTO, Green-RTO ordonnance les jobs rouges uniquement, garantissant le seuil minimal de QoS pour le système.

Green-RTO répond aux règles suivantes (cf. pseudo-code décrit par l'Algorithme 4) :

1. lorsque le réservoir n'est pas vide ($E(t) > E_{min}$), le processeur est *actif* s'il y au moins un job rouge à l'état prêt et si la laxité énergétique du système est positive. Les jobs rouges sont ordonnancés selon l'algorithme EDF,
2. le processeur est *inactif* si la laxité temporelle n'est pas nulle ou s'il n'y a aucun job rouge à l'état prêt.

Algorithm 4 Green-RTO

```
t : temps courant
J_r(t) : la liste des jobs rouges prêts à l'instant t
while VRAI do
  idle ← VRAI
  if J_r(t) = not(φ) then
    Soit J_i le job rouge ayant la plus proche échéance
    if E(t) > E_min then
      Calculer SlackEnergy*(t)
      if SlackEnergy*(t) ≥ 0 then
        idle← FAUX
        Exécuter J_i
      else
        /* SlackEnergy*(t) < 0 */
        if E(t) < E_max then
          /* E_min < E(t) < E_max */
          Calculer SlackTime*(t)
          if SlackTime*(t) = 0 then
            idle ← FAUX
            Exécuter J_i
          end if
        end if
      end if
    end if
  end if
end while
```

Afin de déterminer les intervalles d'activité/d'inactivité du processeur, Green-RTO s'appuie sur le calcul en-ligne des laxités temporelle et énergétique du système intégrant les contraintes de QoS des tâches, ce que nous dénommons ci-après *laxités équivalentes*.

Calcul de la laxité temporelle *équivalente* du système. Afin de déterminer dynamiquement les laxités temporelles pour des jobs avec pertes au sens Skip-over, nous nous sommes appuyés sur le calcul des temps creux selon EDL adapté à RTO, tel que décrit dans [Marchand 2006]. La laxité temporelle *équivalente* associée à un job au temps courant peut alors s'exprimer comme suit :

Définition 29 *La laxité temporelle* équivalente *d'un job J_i à l'instant t est donnée par :*

$$SlackTime^*(J_i, t) = \Omega^{EDL-RTO}(t, d_i) \tag{8.5}$$

$\Omega^{EDL-RTO}(t, d_i)$ représente la somme totale des temps creux calculée sur l'intervalle $[t, d_i)$ pour un modèle de tâches RTO. La complexité algorithmique du calcul du vecteur dynamique des temps creux avec un modèle de tâches RTO est la même que celle observée avec un modèle de tâches classiques, soit en $O(\lceil \frac{R}{p} \rceil n)$ où n désigne le nombre de tâches périodiques, R la plus grande échéance parmi les tâches actives et p la plus petite période [Marchand 2006].

La *laxité temporelle équivalente du système* à l'instant courant t, notée $SlackTime^*(t)$ est égale à la valeur minimale parmi les laxités temporelles équivalentes de l'ensemble des jobs actifs à l'instant t. Elle représente la durée maximale depuis t durant laquelle le processeur peut rester inactif sans pour autant compromettre l'exécution des jobs *rouges* dans le respect de leur échéance.

Calcul de la laxité énergétique *équivalente* du système. Le calcul dynamique de la laxité énergétique *équivalente* au temps courant associé à un job peut, quant à lui, être spécifié de la manière suivante :

Définition 30 *La laxité énergétique* équivalente *d'un job J_i à l'instant t est donnée par :*

$$SlackEnergy^*(J_i, t) = E(t) + E_p(t, d_i) - \sum_{\substack{Jk \in jobs\,rouges \\ t \leq r_k, d_k \leq d_i}} E_k \tag{8.6}$$

$SlackEnergy^*(J_i, t)$ représente la quantité maximale d'énergie pouvant être utilisée à partir de t par des jobs *rouges* de plus haute priorité (i.e. jobs possédant une échéance inférieure ou égale à celle du job J_i).

Sur la base de la définition précédente, nous introduisons la notion de *laxité énergétique équivalente du système* à l'instant courant t, notée $SlackEnergy^*(t)$. Celle-ci représente le surplus d'énergie consommable à l'instant t dans le système tout en satisfaisant les contraintes temporelles et énergétiques des jobs *rouges* actifs ayant leur date de réveil postérieure ou égale à t et leur échéance inférieure ou égale à d (avec d l'échéance du job *rouge* le plus prioritaire à l'instant t) :

Définition 31 *La laxité énergétique* équivalente *du système à l'instant t est donnée par :*

$$SlackEnergy^*(t) = \min_{t \leq r_i < d_i \leq d} SlackEnergy^*(J_i, t) \tag{8.7}$$

8.2.2.3 Green-BWP

L'algorithme d'ordonnancement Green-BWP (cf. pseudo-code décrit par l'Algorithme 5) est basé sur les algorithmes BWP et EDeg. Green-BWP vise à améliorer la QoS observée pour le système, en ordonnançant les jobs bleus dès lors qu'aucun job rouge n'est à l'état prêt. Il diffère cependant de BWP dans le sens où un job est exécuté uniquement après avoir vérifié que son exécution ne génèrera pas de pénurie énergétique ni pour l'ensemble des jobs *rouges* actifs, ni pour aucun job *rouge* réveillé dans le futur. Le processeur est actif à l'instant t si l'énergie disponible dans le réservoir est non nulle ET si la laxité énergétique équivalente du système calculée à l'instant t est positive. Dans le cas contraire, le calcul de la laxité temporelle équivalente permet de déterminer la durée durant laquelle le processeur peut être laissé inactif tout en garantissant les contraintes temporelles des jobs *rouges*.

Algorithm 5 Green-BWP

```
t : temps courant
J_r(t) : la liste des jobs rouges prêts à l'instant t
J_b(t) : la liste des jobs bleus prêts à l'instant t
while VRAI do
  idle ← VRAI
  if J_r(t) = not(φ) then
    soit J_i le job rouge ayant la plus petite échéance
  else if J_b(t) = not(φ) then
    soit J_i le job bleu ayant la plus petite échéance
  end if
  if E(t) > E_min then
    Calculer SlackEnergy*(t)
    if SlackEnergy*(t) ≥ 0 then
      idle← FAUX
      Exécuter J_i
    else
      /* SlackEnergy*(t) < 0 */
      if E(t) < E_max then
        /* E_min < E(t) < E_max */
        Calculer SlackTime*(t)
        if SlackTime*(t) = 0 then
          idle ← FAUX
          Exécuter J_i
        end if
      end if
    end if
  end if
end while
```

Calcul de la laxité temporelle *équivalente* du système. Calculer dynamiquement la laxité temporelle équivalente $SlackTime^*(J_i, t)$ d'un job J_i sous Green-BWP, est plus complexe que dans le cas de Green-RTO. Il est nécessaire d'identifier où sont situés les réveils de jobs rouges sur l'intervalle $[t, d_i)$. Sous RTO, le pattern d'activation des jobs rouges est fixe. Sous BWP, ce n'est plus le cas. En effet, lorsqu'un job bleu est exécuté dans le respect de son échéance, le job suivant est également bleu, introduisant par conséquent un décalage dans le positionnenement des activations des jobs rouges. Aussi, afin de déterminer dynamiquement les laxités temporelles sous Green-BWP, nous avons utilisé le calcul des temps creux selon EDL adapté à BWP, tel que décrit dans [Marchand 2006]. La laxité temporelle *équivalente* d'un job peut alors s'exprimer comme suit :

Définition 32 *La laxité temporelle* équivalente *d'un job J_i au temps t est donnée par :*

$$SlackTime^*(J_i, t) = \Omega^{EDL-BWP}(t, d_i) \tag{8.8}$$

$\Omega^{EDL-BWP}(t, d_i)$ représente la somme totale des temps creux calculée sur l'intervalle $[t, d_i)$ pour un modèle de tâches BWP. La complexité algorithmique du calcul du vecteur dynamique des temps creux avec un modèle de tâches BWP est en $O(N)$ où N représente le nombre de jobs périodiques *rouges* sur l'intervalle $[t, kH[$ avec $k = (\lfloor \frac{t}{H} \rfloor + 1)$ figurant la date de fin de l'hyperpériode courante contenant t [Marchand 2006].

La *laxité temporelle équivalente du système* à l'instant courant t, notée $SlackTime^*(t)$ est à nouveau égale à la valeur minimale parmi les laxités temporelles *équivalentes* de l'ensemble des jobs actifs à l'instant t.

Calcul de la laxité énergétique *équivalente* du système. Le calcul est identique à celui réalisé par Green-RTO, avec cependant la problématique supplémentaire de détermination des localisations des jobs *rouges* sur l'intervalle $[t, d)$. En effet, les exécutions de jobs *bleus* dans le respect de leur échéance introduisent pour les tâches, un décalage dans le pattern d'activation des jobs *rouges*.

8.2.3 Illustration

Les Figures 8.3 et 8.4 illustrent les séquences d'ordonnancement générées respectivement par Green-RTO et Green-BWP, ainsi que la courbe de variation du niveau d'énergie dans le réservoir, sur l'ensemble de tâches périodiques $\mathcal{T}$=$\{\tau_i(C_i,D_i, T_i, s_i, E_i)\ \} = \{\tau_1(3, 6, 6, 2, 7)$, $\tau_2(5, 9, 9, 2, 12)\}$. On suppose que le réservoir d'énergie est initialement plein (i.e. $E(0) = C = 7$) et que l'énergie récupérée depuis l'environnement est constante avec $P_p = 2$. Dans le cas présent, on suppose également que l'énergie consommée par les tâches est équitablement répartie sur l'ensemble de leur durée d'exécution de sorte qu'elles consomment exactement $\frac{E_i}{C_i}$ unités d'énergie par unité de temps d'exécution.

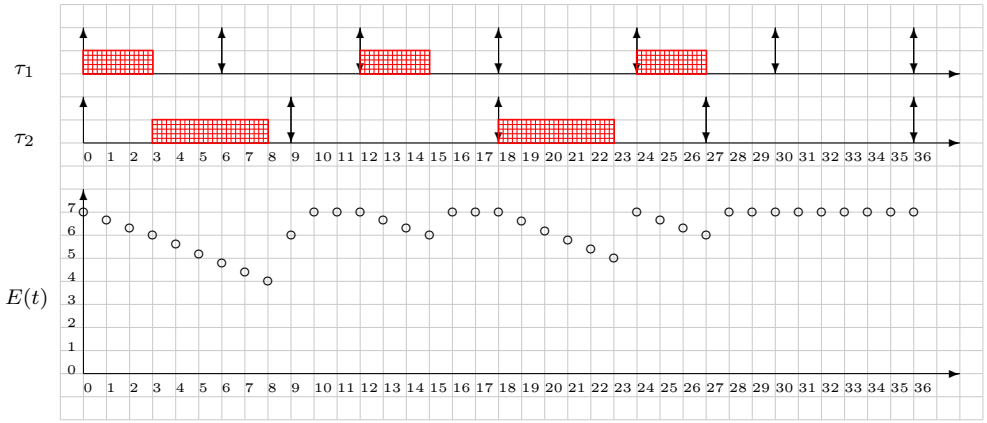


Figure 8.3 – *Séquence d'ordonnancement générée par Green-RTO*

On observe que l'ordonnanceur Green-BWP exécute davantage de jobs dans le respect de leurs contraintes temporelles et énergétiques que l'ordonnanceur Green-RTO. Dans l'exemple considéré, 70% des jobs sont effectivement exécutés avec Green-BWP alors que 50% le sont avec Green-RTO.

On constate également que sous Green-RTO, le niveau d'énergie dans le réservoir n'est jamais proche de zéro. Un réservoir de taille inférieure pourrait donc être envisagé. Sous Green-BWP, du fait de l'exécution des jobs bleus, la quantité d'énergie consommée est plus importante. A $t = 31$, le job bleu de τ_2 réveillé à $t = 27$ ne peut plus s'exécuter faute d'énergie. A sa date d'échéance, le job n'est pas terminé. Il viole donc son échéance. Notons que Green-BWP génère dans ce cas un gaspillage de temps processeur et d'énergie.

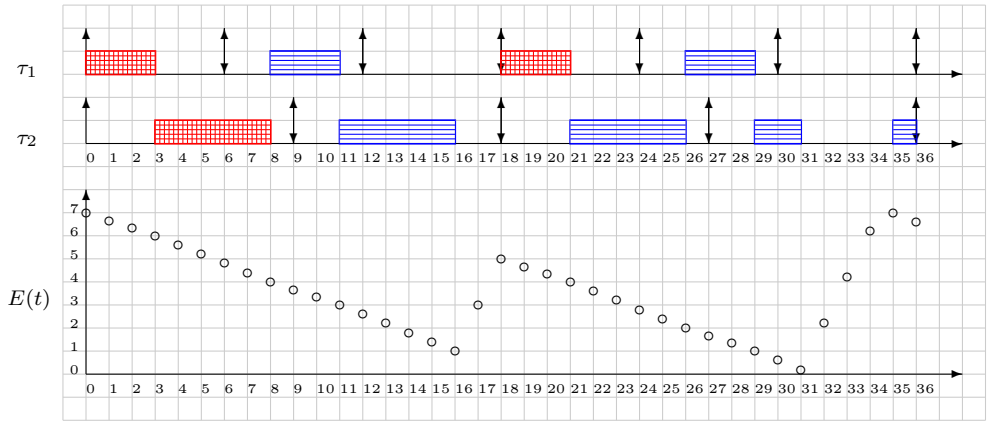


Figure 8.4 – *Séquence d'ordonnancement générée par Green-BWP*

8.3 Analyse de faisabilité sous contraintes de QoS et d'énergie

8.3.1 Nouveaux éléments de terminologie

A l'image de la *demande processeur équivalente*, nous proposons ci-après la notion de *demande énergétique équivalente* sur tout intervalle $[0, L)$, notée $edbf^*(0, L)$:

Définition 33 *La demande énergétique* équivalente *sur tout intervalle* $[0, L)$*, associée à une configuration* $\mathcal{T}$ profondément rouge *de n tâches périodiques définies selon le modèle RTEH-QoS, est donnée par :*

$$edbf^*(0, L) = \sum_{i=1}^{n} (\lfloor \frac{L}{T_i} \rfloor - \lfloor \frac{L}{T_i s_i} \rfloor) E_i \tag{8.9}$$

La demande énergétique *équivalente* $edbf^*(0, L)$ sur l'intervalle $[0, L)$ représente l'énergie cumulée requise par les jobs *rouges* de $\mathcal{T}$ possédant une échéance inférieure ou égale à L.

Il en découle la notion de *facteur d'utilisation énergétique équivalent*, noté U_e^* :

Définition 34 *Etant donné une configuration* $\mathcal{T}$ profondément rouge *de n tâches périodiques définies selon le modèle RTEH-QoS, le facteur d'utilisation énergétique équivalent est défini par :*

$$U_e^* = \max_{L \geq 0} \left\{ \frac{edbf^*(0, L)}{E(0) + E_p(0, t)} \right\} \tag{8.10}$$

où $E(0)$ est le niveau d'énergie initial dans le réservoir d'énergie et $E_p(0, t)$ est l'énergie produite sur l'intervalle $[0, t)$.

8.3.2 Analyse du point de vue temporel

Nous avons vu précédemment (cf. Section 8.1.4.4, Théorème 8.1.1) qu'une condition de faisabilité ($U_p^* \leq 1$) avait été établie par Koren et Shasha dans [Koren 1995] pour tester si une configuration de tâches était faisable *temporellement*. Sous l'hypothèse d'une configuration profondément rouge, cette condition est nécessaire et suffisante.

8.3.3 Analyse du point de vue énergétique

Sur la base de la connaissance du niveau d'énergie initial dans le réservoir d'énergie, et de l'évaluation à la fois de la demande énergétique équivalente $edbf^*(0, L)$ et de l'énergie reçue sur tout intervalle $[0, L)$, il est possible de déterminer une condition nécessaire à la faisabilité *énergétique* d'une configuration de tâches :

Théorème 8.3.1 *[Abdallah 2012a] Une configuration $\mathcal{T} = \{\tau_i(C_i, E_i, D_i, T_i, s_i)\}$ profondément rouge de n tâches périodiques autorisant des pertes au sens Skip-Over, est faisable* énergétiquement *seulement si :*

$$\forall L | \sum_{i=1}^{n} edbf_i^*(0, L) \leq E(0) + E_p(0, L) \tag{8.11}$$

Ceci nous amène à proposer un deuxième théorème :

Théorème 8.3.2 *[Abdallah 2012a] Une configuration $\mathcal{T} = \{\tau_i(C_i, E_i, D_i, T_i, s_i\}$ profondément rouge de n tâches périodiques autorisant des pertes au sens Skip-Over, est faisable* énergétiquement *seulement si :*

$$U_e^* \leq 1 \tag{8.12}$$

Si $U_e^* > 1$, ceci implique que sur un intervalle $[0, t_1)$, $\sum_{i=1}^{n} edbf_i^*(0, t_1) > E(0) + E_p(0, t_1)$ et le Théorème 8.3.1 montre que $\mathcal{T}$ est alors non faisable.

8.3.4 Analyse du point de vue temporel et énergétique

A partir des théorèmes 8.1.1 et 8.3.2, nous établissons la condition nécessaire suivante relative à la faisabilité temporelle et énergétique d'une configuration de tâches sous contraintes de QoS et d'énergie :

Théorème 8.3.3 *[Abdallah 2012a] Une configuration $\mathcal{T} = \{\tau_i(C_i, E_i, D_i, T_i, s_i\}$ profondément rouge de n tâches périodiques autorisant des pertes au sens Skip-Over, est faisable* temporellement *ET* énergétiquement *seulement si :*

$$U_p^* \leq 1 \tag{8.13}$$

et

$$U_e^* \leq 1 \tag{8.14}$$

La preuve triviale, découle directement des Théorèmes 8.1.1 et 8.3.2.

8.4 Evaluation de performance

Dans cette section, nous décrivons les résultats des expérimentations menées afin d'évaluer la performance des algorithmes d'ordonnancement Green-RTO et Green-BWP par rapport à l'algorithme d'ordonnancement EDeg qui n'intègre aucune garantie de QoS pour les tâches en cas de surcharge de traitement et/ou de pénurie énergétique.

8.4.1 Contexte et métriques d'évaluation

Un simulateur en langage C a été développé afin de étudier le comportement des différents algorithmes d'ordonnancement. Il est composé d'un générateur de configurations de tâches périodiques temps réel définies selon le modèle RTEH-QoS ($\mathcal{T}^* = \{\tau_i(\phi_i, C_i, E_i, D_i, T_i, s_i)$, i=1 à $n\}$) acceptant en entrée les paramètres suivants :

— n : le nombre de tâches périodiques au sein de la configuration,
— $PPCM_{max}$: la valeur maximale du $PPCM(T_1, ..., T_i, ..., T_n)$ des périodes des tâches de la configuration,
— $U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$: le facteur d'utilisation du processeur,
— $U_e = \sum_{i=1}^{n} \frac{E_i}{T_i}$: le facteur d'utilisation énergétique,
— s_i : les pertes autorisées (supposées identiques pour toutes les tâches).

Au sein de chaque configuration, les tâches sont supposées synchrones ($\forall i, \phi_i = 0$), indépendantes, préemptables et à échéances sur requêtes ($D_i = T_i$). Les périodes des tâches T_i sont générées aléatoirement en fonction de la valeur $PPCM_{max}$ fournie en entrée. Les durées d'exécution au-pire cas C_i (resp. les demandes énergétiques E_i) sont également générées aléatoirement en respectant le facteur U_p (resp. U_e) spécifié en entrée. Aucune corrélation n'existe entre les valeurs E_i et C_i.

Les métriques considérées pour les évaluations sont les suivantes :
— *la QoS* : le nombre de jobs exécutés dans le respect de leur échéance sur le nombre total de jobs,
— *le taux de réservoir plein* : le pourcentage de temps durant lequel le réservoir d'énergie est plein,
— *le taux d'oisiveté* : le pourcentage de temps durant lequel le processeur est inactif,
— *le taux de gaspillage de temps processeur* : le pourcentage de temps processeur gaspillé à exécuter des jobs ayant violé leur échéance par rapport à la durée totale des périodes d'activité du processeur,
— *le taux de gaspillage d'énergie* : le pourcentage d'énergie gaspillée à exécuter des jobs ayant violé leur échéance par rapport à la la quantité totale d'énergie consommée.

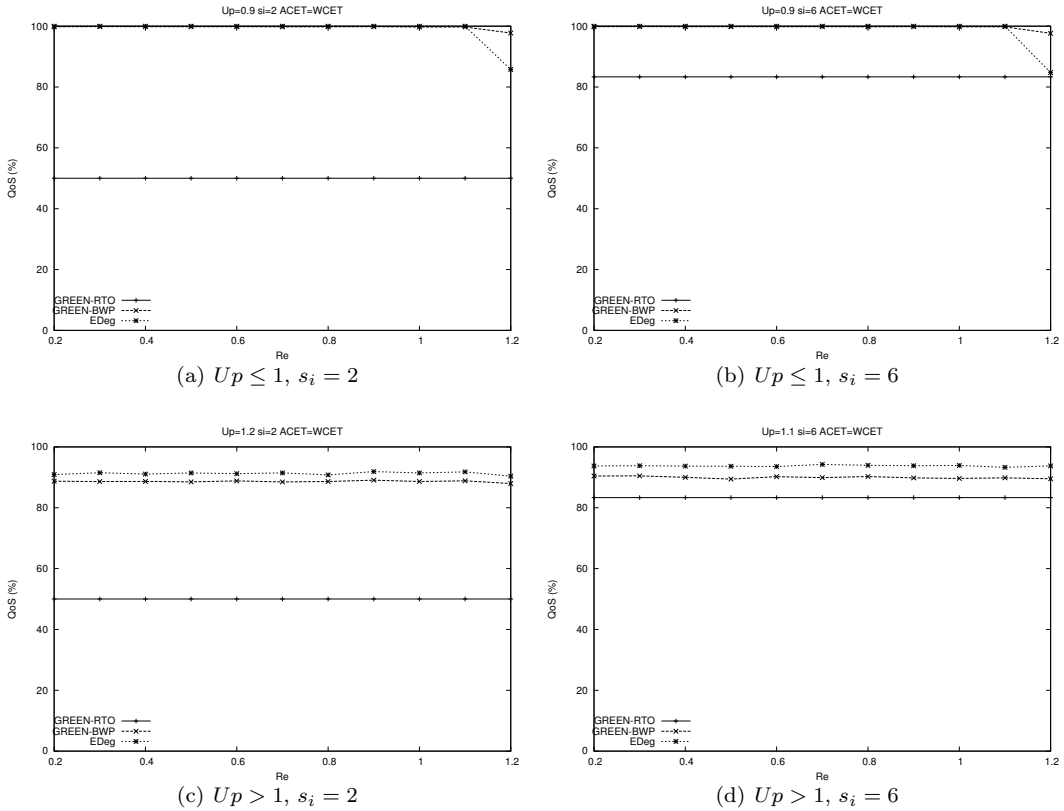
Les paramètres d'entrée que nous avons fait varier sont listés ci-dessous :
— *le facteur d'utilisation du processeur* U_p. Si $U_p > 1$, le système est en surcharge du point de vue temporel,
— *le paramètre de pertes* s_i pour les tâches,
— *le ratio de criticité énergétique* $R_e = \frac{U_e}{P_r}$ qui est égal à la consommation énergétique moyennes des tâches ($U_e = \sum_{i=1}^{n} \frac{E_i}{T_i}$) divisée par la puissance reçue (P_r). Si $R_e \leq 1$, ceci signifie que la consommation moyenne énergétique est inférieure à l'énergie délivrée au système. Dans le cas contraire où $R_e > 1$, cela implique que l'énergie délivrée au système est inférieure à la demande énergétique de la configuration de tâches et que le système est par conséquent fortement contraint énergétiquement.

Les simulations ont été réalisées sur 100 configurations de tâches différentes, chacune constituée de 10 tâches. Les algorithmes d'ordonnancement Green-RTO, Green-BWP, et EDeg ont été simulés sur 6 hyperpériodes $H^* = PPCM(T_1 s_1, ..., T_n s_n)$. On a supposé le réservoir d'énergie initialement plein (i.e., $E(0) = C$) et ayant une capacité égale à $C = H^*.P_r$.

8.4.2 Résultats de performance

Nous avons choisi d'observer les différentes métriques selon divers profils temporels : dans le cas d'un système fortement chargé temporellement (i.e. $U_p \leq 1$) et d'un système surchargé temporellement (i.e. $U_p > 1$). Deux cas de contraintes de QoS ($s_i = 2$ et $s_i = 6$) ont été considérés afin d'étudier l'impact de la tolérance aux pertes autorisées. Les résultats de performance sont tous affichés en fonction du ratio de criticité énergétique R_e.

La QoS. Nous cherchons ici à quantifier les performances des algorithmes d'ordonnancement du point de vue de la QoS observée au niveau du système. Les résultats sont rapportés sur la Figure 8.5.



(a) $Up \leq 1$, $s_i = 2$ (b) $Up \leq 1$, $s_i = 6$

(c) $Up > 1$, $s_i = 2$ (d) $Up > 1$, $s_i = 6$

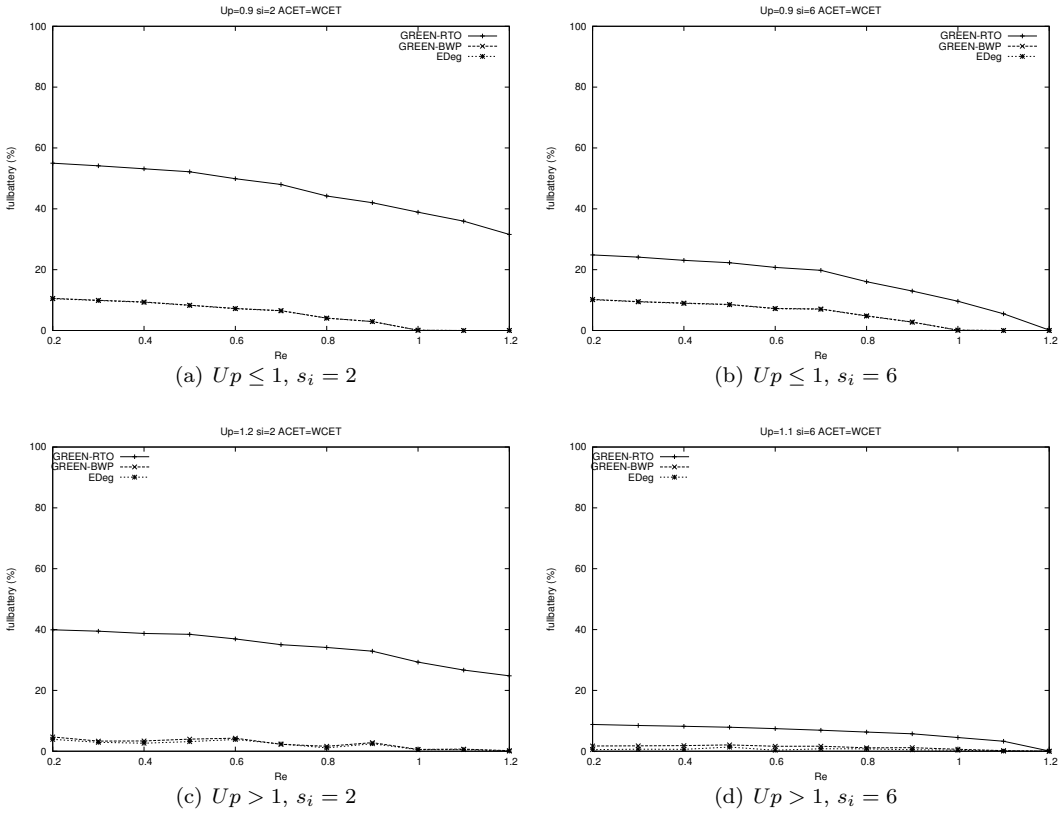
Figure 8.5 – *QoS*

Au vu de ces résultats, nous relevons l'ensemble des conclusions suivantes :

1. Green-RTO offre une QoS, constante, minimale, indépendante du profil temporel et énergétique, qui ne dépend que du paramètre de pertes s_i. Son avantage réside cependant le fait que les abandons de jobs sont répartis régulièrement dans le temps, dans le respect du seuil de tolérance fixé par l'utilisateur au travers du paramètre s_i pour les tâches,
2. Dans le cas où le système n'est pas *surchargé temporellement ou énergétiquement*, Green-BWP et EDeg offrent une QoS de 100%,
3. Dans le cas où le système est *surchargé temporellement*, Green-BWP et EDeg affichent une QoS inférieure à 100% parce que des jobs violent leur échéance par manque de temps. On observe une QoS légèrement supérieure sous EDeg par rapport à Green-BWP. L'avantage de Green-BWP réside cependant dans le fait que les jobs non exécutés sont "maîtrisés" (i.e., la distance maximale entre deux abandons de jobs est égale à s_i) alors qu'avec EDeg, l'identité des jobs en échec n'est pas définie,
4. Dans le cas où le système est *surchargé énergétiquement*, Green-BWP et EDeg affichent une QoS inférieure à 100% parce que des jobs violent leur échéance par

manque d'énergie. Dans le cas où $Up \leq 1$, Green-BWP s'avère être l'ordonnanceur le plus performant en termes de QoS. En cas de surcharge temporelle (i.e., $U_p > 1$), c'est EDeg qui offre la meilleure QoS mais toujours sans aucun contrôle sur l'identité des jobs abandonnés.

Taux de réservoir plein. Nous nous intéressons à présent au taux de réservoir plein afin de vérifier si le dimensionnement du réservoir d'énergie est adapté aux contraintes énergétiques du système. La Figure 8.6 montre les performances respectives des trois algorithmes d'ordonnancement pour les deux configurations $Up \leq 1$ et $U_p > 1$.



(a) $Up \leq 1$, $s_i = 2$ (b) $Up \leq 1$, $s_i = 6$

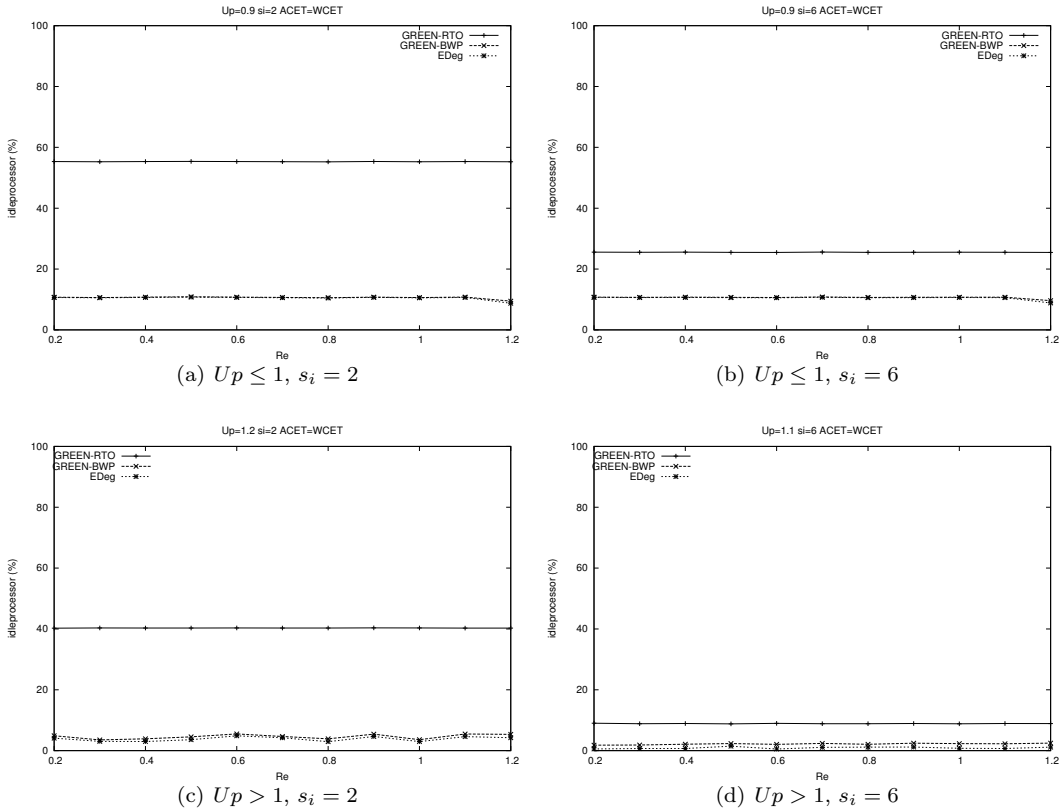
(c) $Up > 1$, $s_i = 2$ (d) $Up > 1$, $s_i = 6$

Figure 8.6 – *Taux de réservoir plein*

Nous pouvons relevé les points de conclusion suivants :

1. Dans tous les cas, on observe que le nombre d'instants durant lequel le réservoir d'énergie est plein décroît en fonction du facteur de criticité énergétique R_e. En effet, si la consommation moyenne des tâches est plus élevée, l'utilisation de l'énergie disponible dans le réservoir est plus importante,
2. Le réservoir d'énergie est plus souvent plein avec Green-RTO qu'avec Green-BWP et EDeg car Green-RTO n'exécute que les jobs rouges. Cela signifie qu'on peut se contenter d'un réservoir d'énergie de capacité plus petite que celle requise par EDeg ou Green-BWP,
3. Le nombre d'instants durant lequel le réservoir d'énergie est plein est similaire avec EDeg et Green-BWP mais ce ne sont pas forcément les mêmes jobs qui sont exécutés.

Taux d'oisiveté. La Figure 8.7 affiche les taux d'oisiveté en fonction des variations des mêmes paramètres temporels (U_p, s_i) et énergétiques (R_e) que précédemment.



(a) $Up \leq 1$, $s_i = 2$ (b) $Up \leq 1$, $s_i = 6$

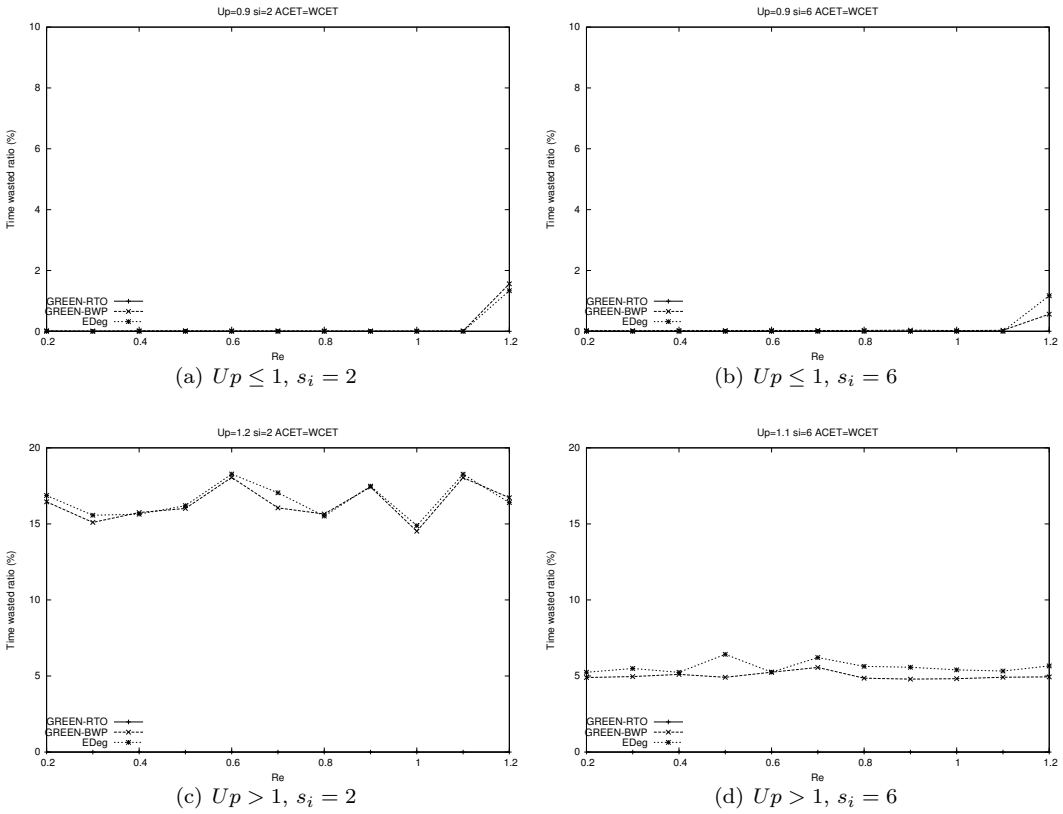
(c) $Up > 1$, $s_i = 2$ (d) $Up > 1$, $s_i = 6$

Figure 8.7 – *Taux d'oisiveté*

Les points remarquables sont listés ci-dessous :

1. Les taux d'oisiveté obtenus avec Green-RTO, Green-BWP et EDeg sont d'autant plus faibles que l'utilisation du processeur est élevé et/ou que le nombre pertes autorisées est faible,
2. Green-RTO affiche le taux d'oisiveté le plus élevé, et ce, quels que soient le facteur d'utilisation du processeur et le facteur de criticité énergétique. Green-RTO exécutant uniquement les jobs rouges, le processeur est naturellement moins sollicité,
3. Green-BWP et EDeg affichent des taux d'oisiveté très proches qui tendent vers zéro quand le système souffre d'une surcharge temporelle.

Taux de gaspillage de temps processeur. L'objectif ici est de quantifier les performances des algorithmes d'ordonnancement du point de vue du gaspillage de temps processeur, généré par l'ordonnancement de jobs non exécutés complètement au terme de leur échéance. Les résultats sont rapportés sur la Figure 8.8.



(a) $Up \leq 1$, $s_i = 2$ (b) $Up \leq 1$, $s_i = 6$

(c) $Up > 1$, $s_i = 2$ (d) $Up > 1$, $s_i = 6$

Figure 8.8 – *Taux de gaspillage de temps processeur*

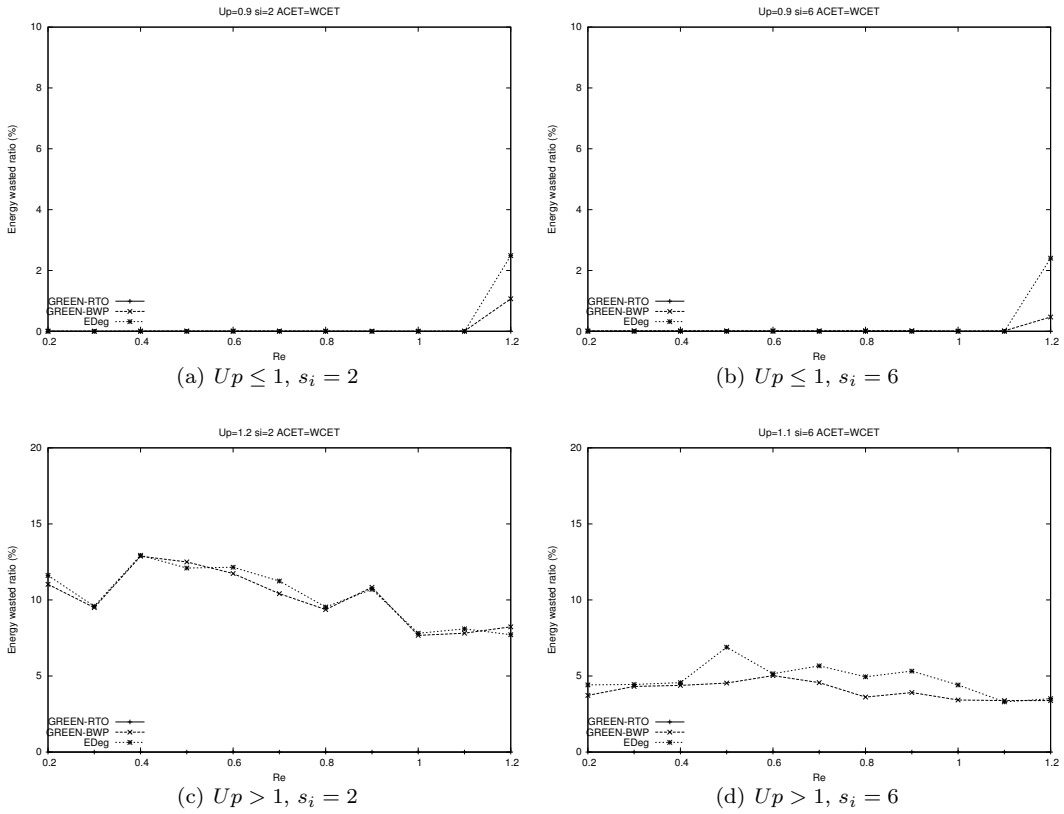
Nous remarquons que :

1. Green-RTO n'engendre pas de gaspillage de temps processeur quels que soient les profils énergétiques et temporels,
2. Dans le cas où le système n'est pas *surchargé temporellement ou énergétiquement*, les ratios de temps gaspillé sont nuls pour tous les algorithmes,
3. Dans le cas où le système est *surchargé temporellement et/ou énergétiquement*, Green-BWP et EDeg présente des gaspillages de temps processeur dû à des violations d'échéance de jobs partiellement exécutés, et ceux-ci sont d'autant moindres que le paramètre s_i est élevé.

Taux de gaspillage d'énergie. Nous nous intéressons enfin au taux de gaspillage d'énergie. La Figure 8.9 montre les performances respectives des trois algorithmes d'ordonnancement pour les deux configurations $Up \leq 1$ et $U_p > 1$.

Nous observons que :

1. Green-RTO n'engendre pas de gaspillage d'énergie quel que soit le profil énergétique et temporel étudiés,
2. Pour l'ensemble des observations, Green-BWP et EDeg génèrent des taux de gaspillage d'énergie très proches les uns des autres,



(a) $Up \leq 1$, $s_i = 2$ (b) $Up \leq 1$, $s_i = 6$

(c) $Up > 1$, $s_i = 2$ (d) $Up > 1$, $s_i = 6$

Figure 8.9 – *Taux de gaspillage d'énergie*

3. Dans le cas où le système est *non surchargé temporellement*, Green-BWP et EDeg génèrent des gaspillages d'énergie uniquement lorsque le système est surchargé énergétiquement. Dans ce cas, des violations d'échéances de jobs partiellement exécutés par manque d'énergie apparaissent.
4. Dans le cas où le système est *surchargé temporellement*, Green-BWP et EDeg induisent des gaspillages d'énergie quel que soit le profil énergétique. Ce gaspillage d'énergie est dû à des violations d'échéances de jobs partiellement exécutés par manque de temps et s'avère d'autant plus important que le paramètre de pertes est faible.

8.4.3 Synthèse des résultats.

Le but de ces simulations a été de nous permettre d'évaluer les performances des deux algorithmes proposés dans ce chapitre, à savoir Green-RTO et Green-BWP, par rapport à celles de l'algorithme EDeg. Au vu d'une part des descriptions algorithmiques, et d'autre part des résultats de simulation, nous pouvons dresser le tableau récapitulatif suivant (cf. Table 8.1) résumant les performances des différents ordonnanceurs en termes de performances, de complexité algorithmique, et de support matériel sous-jacent nécessaire (i.e., capacité du réservoir d'énergie et puissance de traitement requis pour garantir le bon fonctionnement du système). Dans ce tableau, les '★' représentent les bonnes performances de l'algorithme d'ordonnancement au niveau d'un critère donné.

Algorithmes d'ordonnancement	Performance globale	Garanties QoS	Complexité algorithmique	Réservoir requis	Processeur requis
Green-RTO	★	★★	★★★	★★★	★★★
Green-BWP	★★★	★★★	★★	★★	★★
EDeg	★★★	★	★★★	★	★

Table 8.1 – *Synthèse des performances comparatives de Green-RTO, Green-BWP, et EDeg*

Cette classification nous montre la performance assez médiocre de l'algorithme Green-RTO. Celle-ci s'accompagne cependant d'une facilité de mise en œuvre et d'implémentation. Ce tableau souligne la prédominance de performance de l'algorithme Green-BWP en termes de garanties de QoS. Les jobs abandonnés sont quantifiés et répartis dans le temps, selon la tolérance aux pertes spécifiée pour chacune des tâches par l'utilisateur. Ces garanties de seuil de QoS par tâche sont cependant acquises au détriment d'une augmentation de la complexité de calcul et d'implémentation. Ces résultats mettent enfin en avant l'existence d'un compromis performance-matériel requis. Green-RTO ou Green-BWP garantissent que le résultat est toujours acceptable même si la QoS est dégradée. De ce point de vue, avec Green-RTO ou Green-BWP, il est possible de réduire la taille du réservoir d'énergie et continuer à avoir un résultat acceptable contrairement à EDeg qui a besoin d'une taille prédéfinie à l'avance de sorte à ne pas manquer d'énergie en cas de surcharge.

8.5 Autres travaux

Nous rapportons ici les autres travaux que nous avons menés en lien avec la problématique de l'ordonnancement sous contraintes de QoS et d'énergie.

Dans un axe d'étude complémentaire, nous avons apprécié la performance de Green-BWP non plus uniquement sur la base de la QoS globale du système mais sur la base de la QoS observée individuellement pour chaque tâche. L'analyse des taux de respect individuels sous Green-BWP nous a montré le défaut d'*équité de service* (*fairness* en anglais) dont souffrait cet algorithme. Nous avons donc proposé deux variantes appelées *Green-BWP-MS (Minimum success)* et *Green-BWP-LF (Last Failure)*. Selon *Green-BWP-MS*, le job bleu prêt avec le plus faible taux de respect observé depuis l'instant initial obtient la plus haute priorité. Quant à *Green-BWP-LF*, il consiste à exécuter à tout instant t, le job bleu prêt dont le nombre de succès successifs mesuré depuis le dernier abandon, est le plus petit. L'évaluation menée par le biais de diverses simulations, nous a permis d'observer le gain d'équité de service obtenu grâce à ces nouveaux ordonnanceurs. Cependant, nous avons pu remarquer que ce gain engendrait une légère dégradation de la QoS globale du système. Pour certaines applications, le critère d'équité de service peut prévaloir sur la performance globale, et seules les spécifications du système peuvent guider le choix de l'algorithme d'ordonnancement le plus adapté.

Nous nous sommes par ailleurs intéressés à l'ordonnancement d'un ensemble hybride de tâches constitué de tâches périodiques et de tâches apériodiques définies sous contraintes de temps et d'énergie. Nous avons présenté l'adaptation des serveurs de tâches apériodiques *Background Server (BG)*, et *Total Bandwidth Server (TBS)* à un fonctionnement avec les

ordonnanceurs EDeg, Green-RTO, et Green-BWP. L'objectif de ce travail était de fournir des solutions afin de gérer les cas de surcharge de traitement et/ou de pénurie d'énergie tout en minimisant le temps de réponse des tâches apériodiques.

Nous avons enfin étendu ces travaux au cas multiprocesseur. Nous avons proposé une nouvelle approche de partitionnement, appelée *KTS* (*K-level Task Splitting*) basée sur le fractionnement de tâches (*task splitting* en anglais) pour des systèmes temps réel multiprocesseur homogènes énergétiquement autonomes. L'objectif était le même que pour le cas monoprocesseur : assurer une dégradation contrôlée de la QoS, en particulier dans des phases de famine énergétique et/ou de surcharge de traitement. La problématique consistait alors à répartir les tâches en respectant à la fois leurs contraintes temporelles et leurs contraintes énergétiques, tout en assurant l'exécution des jobs obligatoires permettant de garantir la QoS minimale requise pour le système. L'approche proposée présente l'avantage de n'induire aucun coût de migrations et ce, quels que soient le type de la tâche ($D = T$ ou $D \leq T$, lourde/légère) et le nombre de processeurs du système. Elle requiert uniquement que les horloges des différents coeurs soient synchronisées. Au travers des simulations, nous avons exploré la façon dont à la fois les critères de tri des tâches temps réel et le niveau de criticité énergétique peuvent influer sur la performance des heuristiques. De plus, nous avons estimé une taille de réservoir d'énergie minimale permettant d'obtenir les meilleures performances quels que soient le niveau de contrainte énergétique du système et/ou l'heuristique considérée. L'approche proposée offre des performances intermédiaires entre l'approche semi-partitionnée existante *EDF split (DD)* [Burns 2012] et l'heuristique de partitionnement classique *FFDD (First-Fit Decreasing Density)*.

8.6 Bilan et diffusion des résultats

En cas de surcharge temporelle et/ou énergétique, EDeg n'est pas fiable dans le sens où un nombre non contrôlé de jobs peuvent être abandonnés faute de temps ou d'énergie. Les jobs abandonnés n'étant pas clairement identifiés en termes d'appartenance de tâches, cela peut avoir de graves conséquences sur le système et l'environnement contrôlé. Par conséquent, afin de garantir un seuil de QoS minimale pour les tâches applicatives même en cas de surcharge, nous avons proposé deux nouveaux ordonnanceurs monoprocesseur, Green-RTO et Green-BWP, capables de prendre en compte des ensembles composés de tâches périodiques munies de contraintes de QoS et d'énergie. Nous avons également proposé un test de faisabilité nécessaire intégrant à la fois les contraintes de QoS et d'énergie.

Au travers d'un certain nombre de simulations, nous avons observé que Green-RTO garantit toujours une QoS minimale quel que soit le profil temporel ou énergétique du système. Cet algorithme a néanmoins l'avantage de pouvoir se contenter d'un réservoir d'énergie de faible capacité et d'un processeur à faible puissance de traitement. Green-BWP permet quant à lui d'améliorer la QoS globale en essayant de maximiser le nombre de jobs exécutés dans le respect de leur échéance en fonction du profil énergétique et temporel. Notons cependant que Green-BWP et EDeg présentent des overheads d'exécution supérieurs à Green-RTO et engendrent tous les deux des gaspillages de temps processeur et d'énergie, causés par des jobs abandonnés dans les situations de surcharge. En résumé, le choix de l'ordonnanceur dans le contexte du *energy harvesting sous contraintes de QoS* sera dicté par les spécifications de l'application et du système afin d'identifier le meilleur compromis entre le niveau de QoS souhaité et les contraintes de la plate-forme d'exécution (coût du processeur, coût et dimensionnement du réservoir d'énergie, etc.).

Cette contribution, extension des travaux sur l'heuristique EDeg menés dans la thèse de El Ghor [El Ghor 2012], est une partie du travail de thèse de Maïssa Abdallah [Abdallah 2014a]. Les ordonnanceurs Green-RTO et Green-BWP ont été présentés dans les conférences suivantes :

- *IEEE Int. Conference on Electronics, Circuits and Systems* [Abdallah 2011a],
- *Int. Conference on Advances in Computational Tools for Engineering Applications* [Abdallah 2012a],
- *IEEE Int. Conference on Green Computing and Communications* [Abdallah 2012b].

Les contributions sur les variantes de Green-BWP à visée d'équité de service ont été publiées à la conférence *IEEE International Green Computing and Communications conference (GreenCom'2013)*[Abdallah 2013].

Les travaux concernant l'adaptation de BG et TBS à des systèmes temps réel énergétiquement autonomes ont été présentés récemment à la conférence *IEEE International Conference on Green Computing and Communications (GreenCom'2020)* [Queudet 2020].

L'extension au cas multiprocesseur, réalisée dans le cadre du travail de thèse de Nadine Abdallah [Abdallah 2014b], a été publiée dans les conférences *IEEE International Conference on Electronics, Circuits and Systems (ICECS11)* [Abdallah 2011b] et *IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing* [Abdallah 2014c], ainsi que dans la revue *The Journal of Supercomputing* [Queudet 2017].

Quatrième partie

Bilan et Perspectives de recherche

Chapitre 9

Bilan et Perspectives de recherche

Sommaire

9.1 Bilan des travaux

Sur la période 2007-2020, j'ai co-encadré 6 thèses de doctorat (dont l'une est en cours) et supervisé de nombreux projets de Master (niveau M1 et M2). J'ai également participé à des projets partenariaux (nationaux et internationaux) et publié mes travaux dans une dizaine de revues internationales, trois livres, et plus d'une vingtaine de conférences internationales.

Mes travaux se sont articulés selon deux axes de recherche principaux : (i) la synchronisation d'applications temps réel multicoeur, et (ii) l'autonomie énergétique des systèmes temps réel embarqués. Ces problématiques ont été abordées du point de vue de *l'ordonnancement* en prenant en compte plusieurs types de contraintes : des contraintes *temporelles*, des contraintes *de partage de ressources*, des contraintes de *QoS*, et des contraintes *d'énergie*.

9.1.1 Synchronisation d'applications temps réel multicoeur

Gérer efficacement les accès concurrents à la mémoire dans un contexte temps réel multicoeur se révèle complexe. Dans la littérature, il existe relativement peu de modèles et d'outils visant à réduire cette complexité. Conceptualiser le parallélisme pour le programmeur n'est pas une tâche aisée. Par conséquent, des solutions à base de verrouillage sont communément utilisées afin d'assurer l'intégrité du système. En multicoeur, le verrouillage systématique peut cependant réduire le parallélisme de la plate-forme d'exécution et engendrer par là-même une baisse significative des performances. Par ailleurs, une mauvaise utilisation de ces mécanismes de synchronisation peut mener à des interblocages ou des inversions de priorités. Ces différentes situations s'avèrent le plus souvent difficiles à détecter et/ou reproduire. Les *mémoires transactionnelles*, en revanche, offrent une interface de programmation simplifiée : le programmeur encapsule les accès aux données partagées au sein de *transactions* et c'est le mécanisme de synchronisation sous-jacent qui se charge de gérer l'exécution atomique du bloc transactionnel.

Jusqu'alors, de tels mécanismes de synchronisation n'avaient pas été considérés dans un contexte temps réel. Forts de ce constat, nous avons donc étudié l'adaptation des mémoires transactionnelles aux systèmes temps réel multicoeur. Au travers d'une étude expérimentale, nous avons comparé les performances de plusieurs mémoires transactionnelles afin de déterminer les facteurs clés (système d'exploitation, politique d'ordonnancement, allocateur mémoire) influant sur la gigue d'exécution des transactions, gigue pouvant impacter directement le respect des contraintes temporelles des tâches dans un système temps réel. Nous avons en particulier montrer l'intérêt de considérer un RTOS, une politique d'ordonnancement partitionnée telle que *P-EDF*, et un gestionnaire mémoire à temps d'allocation constant tel que TLSF. Notre contribution suivante a reposé sur la proposition d'un modèle transactionnel temps réel et d'une STM temps réel *soft*, la RT-STM, intégrant deux protocoles de concurrence (un mono-écrivain, un multi-écrivain). Les résultats expérimentaux ont montré d'une part l'intérêt de cette approche en termes de nombre de transactions exécutées dans le respect de leur échéance, et d'autre part la faisabilité de la mise en oeuvre d'une telle STM vis-à-vis des performances observées en termes de bande passante.

Nous avons ensuite présenté les travaux menés autour de la problématique des accès concurrents en mémoire dans le cadre des *garanties strictes* requises par les systèmes temps réel *hard*. Dans un cadre aux hypothèses plus restreintes (i.e. transactions homogènes, mono-écrivain) mais en adéquation cependant avec le domaine d'application visée (i.e. le domaine automobile), nous avons proposé une STM temps réel *hard*, la STM-HRT, permettant de garantir la progression de *toutes* les transactions du système. Une analyse fonctionnelle et temporelle de la STM-HRT nous a permis de valider le mécanisme de synchronisation non bloquant proposé.

9.1.2 Autonomie énergétique des systèmes temps réel embarqués

Les systèmes embarqués de nouvelle génération tels que les noeuds de capteurs sans fil se sont multipliés ces dernières années. Pour nombre de ces systèmes, l'autonomie énergétique est une problématique capitale. C'est le cas par exemple des noeuds de capteurs sans fil dispersés aléatoirement en cas de marée noire, dans le but de récolter un certain nombre d'informations sur l'évolution des nappes de pétrole. Typiquement déployés en zone "hostile", ils doivent fonctionner de manière autonome, sans nécessiter une quelconque intervention humaine. La technologie du *energy harvesting* consistant à capter l'énergie dans l'environnement pour alimenter un système, permet en particulier de doter ces systèmes embarqués aux ressources contraintes, d'une capacité d'autosuffisance énergétique. L'informatique "intelligente" embarquée au sein de ces systèmes possèdent très souvent des exigences temps réel au niveau des traitements logiciels. Il convient alors de garantir le fonctionnement perpétuel du système en gérant conjointement deux types de contraintes : *le temps* et *l'énergie*. C'est précisément l'objet de nos contributions sur cette thématique, en considérant le problème du point de vue de l'ordonnancement des tâches applicatives sur le processeur.

Nous avons tout d'abord mis en évidence l'inefficacité des ordonnanceurs temps réel "classiques" tels que EDF. Bien que démontré optimal en l'absence de contraintes énergétiques, nous avons montré que l'algorithme EDF révèle un certain nombre de faiblesses en présence de limitations énergétiques : consommation "goulue" de l'énergie, incapacité à s'accomoder des fluctuations de l'énergie d'alimentation. Nous avons cependant montré qu'il reste le meilleur ordonnanceur non-oisif dans le contexte du *energy harvesting* et qu'il constitue le meilleur choix d'intégration pour un système ne pouvant disposer d'une estimation prédictive de l'énergie exploitable. Nos travaux ont ensuite consisté en l'identification de quelques-unes

de propriétés clés d'un ordonnanceur dans le contexte du *energy harvesting*. En particulier, nous avons établi le fait que tout algorithme d'ordonnancement optimal doit nécessairement être clairvoyant et que l'omniscience sur le futur doit être égale au minimum à la plus grande échéance relative D de l'application. Nous nous sommes ensuite intéressés à un ordonnanceur optimal pour le modèle RTEH, l'ordonnanceur ED-H. Nous avons montré que, conformément au résultat précédent, l'algorithme ED-H est D-omniscient. Puis, nous avons contribué au problème de la faisabilité d'un ensemble de tâches temps réel à contraintes strictes dans le contexte du *energy harvesting*, par la proposition d'un nouveau test. Sur la base de la preuve de sa robustesse vis-à-vis de la puissance de la source d'énergie, nous avons souligné l'intérêt de l'hypothèse d'une puissance constante afin de ramener l'analyse d'ordonnançabilité à une seule hyperpériode.

Les contributions présentées par la suite visent à apporter des solutions adaptées aux situations de surcharge temporaire de traitement et/ou de consommation énergétique dont un système peut souffrir. Sur la base d'un nouveau modèle de tâches intégrant des contraintes de QoS, nous avons proposé deux nouveaux ordonnanceurs : Green-RTO et Green-BWP. Ceux-ci reposent sur deux calculs clés au sein desquels nous avons intégré les contraintes de QoS des tâches : les calculs de *la laxité temporelle du système* et de *la laxité énergétique du système* à un instant t. Au travers d'une étude en simulation intégrant divers profils temporels et énergétiques, nous avons montré dans quelle mesure Green-RTO et Green-BWP apportent des garanties de QoS, en contrôlant notamment le nombre et l'identité des jobs de tâches abandonnés en cas de surcharge. Nous avons enfin proposé un test de faisabilité nécessaire intégrant conjointement les contraintes de QoS et d'énergie.

9.2 Perspectives de recherche

9.2.1 Prise en compte de contraintes de précédence

La théorie de l'ordonnancement présentée dans ce document concernant les systèmes temps réel autonomes énergétiquement, repose sur l'hypothèse de tâches *indépendantes*. En réalité, dans de nombreuses applications, les tâches peuvent être reliées entre elles par des contraintes de précédences, de par la conception de l'application et/ou de par le fait qu'elles s'échangent des données. Prenons le cas d'un noeud de capteur au sein duquel une grandeur issue de l'environnement est récupérée (ex : valeur de température, taux de CO_2, etc.). Imaginons que l'application embarquée sur le capteur comporte d'une part une tâche τ_1 chargée de lire périodiquement la grandeur mesurée par le capteur pour la stocker dans une variable *var* et d'autre part, une tâche τ_2 qui, une fois la variable *var* mise à jour, lit sa valeur pour déclencher une alarme si *var* dépasse un seuil donné. On se rend compte aisément de la dépendance d'exécution entre ces deux tâches : afin que la tâche τ_2 puisse tester la valeur de *var*, il est nécessaire que la tâche τ_1 ait été exécutée. Une contrainte de précédence entre deux tâches impose donc un ordre d'exécution pour ces tâches [Orozco 1997]. Cela implique que le début d'exécution d'une tâche soit contrôlée et qu'il ne soit possible qu'une fois que toutes ses tâches prédécesseurs ont été exécutées.

Nous souhaitons donc étendre les travaux présentés dans les chapitres 7 et 8 à la prise en comptes de contraintes de précédence. L'idée est de transposer à ED-H les travaux réalisés sur EDF dans [Chetto 1990], permettant de transformer un ensemble de tâches dépendantes en un ensemble de tâches indépendantes par des modifications adéquates de leurs paramètres temporels. En particulier, les dates de réveils et les dates d'échéance doivent être modifiées de manière à ce que chaque tâche ne démarre pas son exécution avant ses prédécesseurs et ne puissent pas préempter ses sucesseurs.

9.2.2 Transfert technologique pour l'embarqué autonome en énergie

L'embarqué autonome en énergie apparaît comme une technologie stratégique des systèmes manufacturiers du futur. Nous souhaitons démontrer la faisabilité de la technologie du *energy harvesting* appliquée à une plateforme réelle. L'idée est d'évaluer l'intérêt pratique des mécanismes d'ordonnancement et de gestion de l'énergie proposées dans les chapitres 7 et 8. Au travers notamment de la collaboration initiée avec la société E-Cobot (thèse CIFRE + projet NExT), nous envisageons d'intégrer et tester nos mécanismes logiciels au sein d'un RTOS libre qui sera implanté dans un premier temps sur le cobot Husky conçu par la société E-Cobot. Plusieurs RTOS temps réel "sensibles à l'énergie" ont été conçus ces dernières années (ex : Contiki [Dunkels 2004], TinyOS [Levis 2005], etc.), mais aucun d'eux ne fournit les fonctionnalités d'un "Energy Harvesting aware RTOS".

Nous voulons ainsi démontrer que toute plate-forme cobotique peut satisfaire ses exigences opérationnelles en temps très contraint et ce, de manière autonome au regard de son alimentation électrique, grâce à l'énergie ambiante (énergie lumineuse, cinétique, etc.) et à un réservoir d'énergie de taille adéquatement dimensionnée. Le risque est principalement lié aux verrous technologiques majeurs concernant la prédiction de l'énergie environnementale, la fuite d'énergie, un décalage entre les dynamiques de la batterie liées au temps de recharge de la batterie et celles de l'ordonnanceur de tâches implémenté dans le système d'exploitation temps réel, etc. En contrepartie, le bénéfice sera de mieux comprendre et décrire ces verrous pour corriger les modèles sur lesquels notre théorie reposait jusqu'alors.

9.2.3 Ordonnancement temps réel au service de l'IoT

Parmi les différents réseaux de l'Internet des Objets (IoT), on peut citer LoRa, Sigfox, LTE-M, NB-IoT et 5G. Alors que LoRa et SigFox permettent l'envoi de messages légers tout en étant économes en énergie, LTE-M, NB-IoT et 5G permettent d'envoyer des messages plus lourds mais au détriment d'un fonctionnement plus énergivore.

La technologie radio LoRa (*Long Range* – "longue portée" environ 15 km) est actuellemnt au coeur de la révolution IoT dans le secteur de l'énergie. Les dispositifs qui l'utilisent se multiplient aujourd'hui depuis les capteurs de données impulsionnelles (ex : température, humidité, présence, luminosité...) aux concentrateurs de données (GPS, accéléromètre, gyroscope, microcontrôleur...) en passant par les applications de monitoring (véhicules et matériels roulants, structures immobiles et zones de stockage, chaîne du froid de marchandises...).

Parmi les caractéristiques contribuant à faire de LoRa un réseau économe en énergie (et maximisant par là-même la durée de vie de la batterie des objets connectés), l'adaptabilité de la puissance de transmission en fonction de la taille des données, du débit d'envoi, et de la portée d'émission est un élément clé. Le protocole de communication s'appuie en effet sur un algorithme chargé de déterminer le niveau de puissance requis pour l'envoi d'un message.

L'idée des travaux futurs envisagés dans cet axe (au travers d'une thèse de doctorat et/ou d'un dépôt de projet ANR incluant des partenaires dans le domaine des télécommunications) est d'adapter le protocole de communication LoRa à des objets connectés supposés autonomes du point de vue énergétique (c'est-à-dire connectés par exemple à une source d'énergie renouvelable). L'objectif n'est plus alors de maximiser la durée de vie de la batterie mais bien d'assurer un fonctionnement autonome énergétiquement en fonction de l'énergie collectée. Il s'agirait en particulier de transposer les travaux précédemment menés sur l'ordonnancement de tâches temps réel avec autonomie d'énergie à de l'ordonnancement de données LoRa avec autonomie d'énergie.

Bibliographie

[Abbott 1992] Robert K Abbott et Hector Garcia-Molina. *Scheduling real-time transactions : A performance evaluation.* ACM Transactions on Database Systems (TODS), vol. 17, no. 3, pages 513–560, 1992. (Cité en page 50.)

[Abdallah 2011a] Maissa Abdallah, Maryline Chetto, Audrey Queudet et R Hage Chehade. *Quality of service facilities for firm real-time energy harvesting systems.* In Proceedings of the 18th IEEE International Conference on Electronics, Circuits, and Systems, pages 768–771, 2011. (Cité en pages 7 et 114.)

[Abdallah 2011b] Nadine Abdallah, Audrey Queudet, Maryline Chetto et Rafic Hage Chehade. *Partitioned EDF Scheduling in Multicore systems with Quality of Service constraints.* In Proceedings of the 18th IEEE International Conference on Electronics, Circuits, and Systems, pages 764–767, 2011. (Cité en pages 7 et 114.)

[Abdallah 2012a] Maissa Abdallah, Maryline Chetto et Audrey Queudet. *Energy-aware schedulers for Real-Time Energy Harvesting systems with Quality of Service requirements.* In Proceedings of the 2nd IEEE International Conference on Advances in Computational Tools for Engineering Applications (ACTEA), pages 342–347, 2012. (Cité en pages 7, 105 et 114.)

[Abdallah 2012b] Maissa Abdallah, Maryline Chetto et Audrey Queudet. *Scheduling with quality of service requirements in real-time energy harvesting sensors.* In Proceedings of the IEEE International Conference on Green Computing and Communications, pages 644–646, 2012. (Cité en pages 7 et 114.)

[Abdallah 2013] Maissa Abdallah, Maryline Chetto, Audrey Queudet et Rafic Hage Chehade. *Stability and Robustness issues in Real-time Sustainable Wireless Sensors.* In Proceedings of the IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, pages 86–93, 2013. (Cité en pages 7 et 114.)

[Abdallah 2014a] Maissa Abdallah. *Ordonnancement temps réel pour l'optimisation de la Qualité de Service dans les systèmes autonomes en énergie.* PhD thesis, University of Nantes, France, 2014. (Cité en pages 98 et 114.)

[Abdallah 2014b] Nadine Abdallah. *Partitionnement temps réel multiprocesseur sous contraintes de qualité de service et d'énergie.* PhD thesis, University of Nantes, France, 2014. (Cité en page 114.)

[Abdallah 2014c] Nadine Abdallah, Audrey Queudet et Maryline Chetto. *Task partitioning strategies for multicore real-time energy harvesting systems.* In Proceedings of the 17th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, pages 125–132, 2014. (Cité en pages 7 et 114.)

[Abu Taha 2015] Mohammed Abu Taha, Safwan El Assad, Mousa Farajallah, Audrey Queudet et Olivier Deforge. *Chaos-based cryptosystems using dependent diffusion : An overview.* In Proceedings of the 10th IEEE International Conference for Internet Technology and Secured Transactions (ICITST), pages 44–49, 2015. (Cité en page 7.)

[Abu Taha 2017] Mohammed Abu Taha, Safwan El Assad, Audrey Queudet et Olivier Deforges. *Design and efficient implementation of a chaos-based stream cipher.* International Journal of Internet Technology and Secured Transactions, vol. 7, no. 2, pages 89–114, 2017. (Cité en page 7.)

[Abu Taha 2018] Mohammed Abu Taha, Safwan El Assad et Audrey Queudet. *Parallel Generator of Discrete Chaotic Sequences Using Multi-threading Approach*. In Advanced Multimedia and Ubiquitous Engineering, pages 161–167. Springer, 2018. (Cité en page 7.)

[Allavena 2001] Andre Allavena et Daniel Mosse. *Scheduling of frame-based embedded systems with rechargeable batteries*. In Workshop on Power Management for Real-time and Embedded systems (in conjunction with RTAS 2001), 2001. (Cité en page 90.)

[Anderson 1995] James H Anderson et Mark Moir. *Universal constructions for multi-object operations*. In Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing, pages 184–193, 1995. (Cité en page 35.)

[Anderson 1997] James H Anderson, Srikanth Ramamurthy et Rohit Jain. *Implementing wait-free objects on priority-based systems*. In Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing, pages 229–238, 1997. (Cité en page 35.)

[Anderson 2000] James H Anderson et Anand Srinivasan. *Pfair scheduling : Beyond periodic task systems*. In Proceedings 7th IEEE International Conference on Real-Time Computing Systems and Applications, pages 297–306, 2000. (Cité en page 40.)

[Axboe 2004] Jens Axboe. *Linux block IO—present and future*. In Ottawa Linux Symp, pages 51–61, 2004. (Cité en page 40.)

[Barros 2014] António Barros et Luis Miguel Pinho. *Non-preemptive scheduling of real-time software transactional memory*. In Proceedings of the International Conference on Architecture of Computing Systems, pages 25–36. Springer, 2014. (Cité en page 68.)

[Barros 2016] António Barros, Patrick Meumeu Yomsi et Luis Miguel Pinho. *Response time analysis of hard real-time tasks sharing software transactional memory data under fully partitioned scheduling*. In Proceedings of the 11th IEEE Symposium on Industrial Embedded Systems (SIES), pages 1–10, 2016. (Cité en page 67.)

[Baruah 1990] Sanjoy K Baruah, Louis E Rosier et Rodney R Howell. *Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor*. Real-time systems, vol. 2, no. 4, pages 301–324, 1990. (Cité en page 77.)

[Baruah 1991] Sanjoy Baruah, Gilad Koren, Bhubaneswar Mishra, Arvind Raghunathan, Louis Rosier et Dennis Shasha. *On-line scheduling in the presence of overload*. In Proceedings 32nd IEEE Annual Symposium of Foundations of Computer Science, pages 100–110, 1991. (Cité en page 80.)

[Baruah 1992] Sanjoy Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis Rosier, Dennis Shasha et Fuxing Wang. *On the competitiveness of on-line real-time task scheduling*. Real-Time Systems, vol. 4, no. 2, pages 125–144, 1992. (Cité en pages 80 et 81.)

[Bernat 2001] Guillem Bernat, Alan Burns et Albert Liamosi. *Weakly hard real-time systems*. IEEE transactions on Computers, vol. 50, no. 4, pages 308–321, 2001. (Cité en page 95.)

[Block 2007] Aaron Block, Hennadiy Leontyev, Bjorn B Brandenburg et James H Anderson. *A flexible real-time locking protocol for multiprocessors*. In Proceedings of the 13th IEEE international conference on embedded and real-time computing systems and applications (RTCSA 2007), pages 47–56, 2007. (Cité en page 45.)

[Bobba 2007] Jayaram Bobba, Kevin E Moore, Haris Volos, Luke Yen, Mark D Hill, Michael M Swift et David A Wood. *Performance pathologies in hardware transactional*

memory. ACM SIGARCH Computer Architecture News, vol. 35, no. 2, pages 81–91, 2007. (Cité en page 34.)

[Brandenburg 2008] Björn B Brandenburg, John M Calandrino, Aaron Block, Hennadiy Leontyev et James H Anderson. *Real-time synchronization on multiprocessors : To block or not to block, to suspend or spin ?* In Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, pages 342–353, 2008. (Cité en page 40.)

[Burns 2012] Alan Burns, Robert I Davis, Pengyu Wang et Fengxiang Zhang. *Partitioned EDF scheduling for multiprocessors using a C=D task splitting scheme*. Real-Time Systems, vol. 48, no. 1, pages 3–33, 2012. (Cité en page 113.)

[Buttazzo 1993] Giorgio C Buttazzo et John A Stankovic. Red : Robust earliest deadline scheduling. University of Massachusetts, Department of Computer Science, 1993. (Cité en page 94.)

[Buttazzo 1998] Giorgio C Buttazzo, Giuseppe Lipari et Luca Abeni. *Elastic task model for adaptive rate control*. In Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279), pages 286–295, 1998. (Cité en page 95.)

[Buttazzo 2011] Giorgio C Buttazzo. Hard real-time computing systems : predictable scheduling algorithms and applications, volume 24. Springer Science & Business Media, 2011. (Cité en pages 80 et 95.)

[Caccamo 1997] Marco Caccamo et Giorgio Buttazzo. *Exploiting skips in periodic tasks for enhancing aperiodic responsiveness*. In Proceedings of the IEEE Real-Time Systems Symposium, pages 330–339, 1997. (Cité en page 97.)

[Campbell 1979] Roy H Campbell, Kurt H Horton et Geneva G Belford. *Simulations of a Fault-tolerant Deadline Mechanism*. In Proceedings-Annual International Conference on Fault-Tolerant Computing, pages 95–101, 1979. (Cité en page 95.)

[Chalasani 2008] Sravanthi Chalasani et James M Conrad. *A survey of energy harvesting sources for embedded systems*. In Proceedings of the IEEE SoutheastCon 2008, pages 442–447, 2008. (Cité en page 71.)

[Chen 1997] Jing Chen et Alan Burns. *A fully asynchronous reader/writer mechanism for multiprocessor real-time systems*. Technical report, University of York, Department of Computer Science, 1997. (Cité en page 67.)

[Chen 2007] Jian-Jia Chen et Chin-Fu Kuo. *Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms*. In Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), pages 28–38, 2007. (Cité en page 72.)

[Chetto 1990] Houssine Chetto, Maryline Silly et T Bouchentouf. *Dynamic scheduling of real-time tasks under precedence constraints*. Real-Time Systems, vol. 2, no. 3, pages 181–194, 1990. (Cité en page 119.)

[Chetto 2009] Maryline Chetto et Hussein El Ghor. *Real-time Scheduling of periodic tasks in a monoprocessor system with rechargeable energy storage*. Work-In-Progress Proceedings of the International Real-Time Systems Symposium, page 45, 2009. (Cité en pages 98 et 99.)

[Chetto 2013] Maryline Chetto et Audrey Queudet. *A note on edf scheduling for real-time energy harvesting systems*. IEEE Transactions on Computers, vol. 63, no. 4, pages 1037–1040, 2013. (Cité en pages 7, 83 et 92.)

[Chetto 2014a] Maryline Chetto. *Optimal scheduling for real-time jobs in energy harvesting computing systems*. IEEE Transactions on Emerging Topics in Computing, vol. 2, no. 2, pages 122–133, 2014. (Cité en pages 78, 83, 85, 86, 87, 91 et 98.)

[Chetto 2014b] Maryline Chetto et Audrey Queudet. *Clairvoyance and online scheduling in real-time energy harvesting systems*. Real-Time Systems, vol. 50, no. 2, pages 179–184, 2014. (Cité en pages 7, 83, 86 et 92.)

[Chetto 2016] Maryline Chetto et Audrey Queudet. Energy autonomy of real-time systems. Elsevier, 2016. (Cité en pages 7 et 92.)

[Chetto 2017] Maryline Chetto et Audrey Queudet. Systèmes temps réel autonomes en énergie, volume 2. ISTE Group, 2017. (Cité en page 92.)

[Chetto 2019] Maryline Chetto et Audrey Queudet. *Feasibility analysis of periodic real-time systems with energy harvesting capabilities*. Sustainable Computing : Informatics and Systems, vol. 22, pages 84–95, 2019. (Cité en pages 7, 87, 88, 89, 90 et 92.)

[Chetto 2020a] Maryline Chetto et Audrey Queudet. *EDF-based real-time scheduling for self-powered sensors : a survey of main theoretical results*. In Proceedings of the IEEE International Conference on Intelligent Systems and Computer Vision (ISCV), pages 1–7, 2020. (Cité en page 92.)

[Chetto 2020b] Maryline Chetto et Audrey Queudet. Sistemas de tiempo real autónomos en energía. ISTE Group, 2020. (Cité en page 92.)

[Cotard 2013] Sylvain Cotard. *Contribution à la robustesse des systèmes temps réel embarqués multicœur autodmobile*. PhD thesis, University of Nantes, France, 2013. (Cité en page 68.)

[Cotard 2015] Sylvain Cotard, Audrey Queudet, Jean-Luc Béchennec, Sébastien Faucou et Yvon Trinquet. *Stm-hrt : A robust and wait-free stm for hard real-time multicore embedded systems*. ACM Transactions on Embedded Computing Systems (TECS), vol. 14, no. 4, pages 1–25, 2015. (Cité en pages 7 et 68.)

[Damron 2006] Peter Damron, Alexandra Fedorova, Yossi Lev, Victor Luchangco, Mark Moir et Daniel Nussbaum. *Hybrid transactional memory*. In Proceedings of the 12th international conference on Architectural support for programming languages and operating systems, pages 336–346, 2006. (Cité en page 36.)

[Delacroix 1994] Joëlle Delacroix. *Un contrôleur d'ordonnancement temps réel pour la stabilité de Earliest Deadline en surcharge : le Régisseur*. PhD thesis, Paris 6, 1994. (Cité en page 94.)

[Dertouzos 1974] Michael Dertouzos. *Control robotics : the procedural control of physical processeds*. In Proceedings of the IFIP congress, pages 807–813, 1974. (Cité en pages 73 et 81.)

[Dice 2006] Dave Dice, Ori Shalev et Nir Shavit. *Transactional locking II*. In Proceedings of the International Symposium on Distributed Computing, pages 194–208. Springer, 2006. (Cité en page 39.)

[Dunkels 2004] Adam Dunkels, Bjorn Gronvall et Thiemo Voigt. *Contiki-a lightweight and flexible operating system for tiny networked sensors*. In Proceedings of the 29th annual IEEE international conference on local computer networks, pages 455–462, 2004. (Cité en page 120.)

[El Ghor 2012] Hussein El Ghor. *Ordonnancement monoprocesseur pour les applications temps réel récupérant l'énergie ambiante*. PhD thesis, University of Nantes, France, 2012. (Cité en pages 99 et 114.)

[El Ghor 2013] Hussein El Ghor, Maryline Chetto et Rafic Hage Chehade. *A nonclairvoyant real-time scheduler for ambient energy harvesting sensors*. International Journal of Distributed Sensor Networks, vol. 9, no. 5, page 732652, 2013. (Cité en page 99.)

[Ennals 2006] Robert Ennals. *Software transactional memory should not be obstruction-free*. Rapport technique, Technical Report IRC-TR-06-052, Intel Research Cambridge Tech Report, 2006. (Cité en page 37.)

[Eswaran 1976] Kapali P. Eswaran, Jim N Gray, Raymond A. Lorie et Irving L. Traiger. *The notions of consistency and predicate locks in a database system*. Communications of the ACM, vol. 19, no. 11, pages 624–633, 1976. (Cité en pages 32 et 33.)

[Fraser 2004] Keir Fraser. *Practical lock-freedom*. Rapport technique UCAM-CL-TR-579, University of Cambridge, Computer Laboratory, F ?rier 2004. (Cité en pages 31, 32, 38 et 47.)

[Fraser 2007] Keir Fraser et Tim Harris. *Concurrent programming without locks*. ACM Transactions on Computer Systems (TOCS), vol. 25, no. 2, pages 5–es, 2007. (Cité en page 37.)

[Ghor 2011] Hussein EL Ghor, Maryline Chetto et Rafic Hage Chehade. *A real-time scheduling framework for embedded systems with environmental energy harvesting*. Computers & Electrical Engineering, vol. 37, no. 4, pages 498–510, 2011. (Cité en page 99.)

[Haerder 1983] Theo Haerder et Andreas Reuter. *Principles of transaction-oriented database recovery*. ACM computing surveys (CSUR), vol. 15, no. 4, pages 287–317, 1983. (Cité en page 33.)

[Hamdaoui 1995] Moncef Hamdaoui et Parameswaran Ramanathan. *A dynamic priority assignment technique for streams with (m, k)-firm deadlines*. IEEE transactions on Computers, vol. 44, no. 12, pages 1443–1451, 1995. (Cité en page 95.)

[Hammoudi 2020] Karim Hammoudi, Mohammed Abu Taha, Halim Benhabiles, Mahmoud Melkemi, Feryal Windal, Safwan El Assad et Audrey Queudet. *Image-based Ciphering of Video Streams and Object Recognition for Urban and Vehicular Surveillance Services*. In Proceedings of the 4th International Congress on Information and Communication Technology, pages 519–527. Springer, 2020. (Cité en page 7.)

[Harris 2014] Tim Harris et Keir Fraser. *Language support for lightweight transactions*. ACM Sigplan Notices, vol. 49, no. 4S, pages 64–78, 2014. (Cité en page 35.)

[Herlihy 1993] Maurice Herlihy et J Eliot B Moss. *Transactional memory : Architectural support for lock-free data structures*. In Proceedings of the 20th annual international symposium on computer architecture, pages 289–300, 1993. (Cité en pages 32 et 35.)

[Herlihy 2003] Maurice Herlihy, Victor Luchangco, Mark Moir et William N Scherer III. *Software transactional memory for dynamic-sized data structures*. In Proceedings of the 22nd annual symposium on Principles of distributed computing, pages 92–101, 2003. (Cité en pages 35 et 37.)

[Holzmann 1997] Gerard J. Holzmann. *The model checker SPIN*. IEEE Transactions on software engineering, vol. 23, no. 5, pages 279–295, 1997. (Cité en page 67.)

[Hsu 2006] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani Srivastava et Vijay Raghunathan. *Adaptive duty cycling for energy harvesting systems*. In Proceedings of the International Symposium on Low power electronics and design, pages 180–185, 2006. (Cité en page 91.)

[Jallouli 2016] Ons Jallouli, Mohammed Abutaha, Safwan El Assad, Maryline Chetto, Audrey Queudet et Olivier Déforges. *Comparative study of two pseudo chaotic number generators for securing the iot*. In Proceedings of the IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), pages 1340–1344, 2016. (Cité en page 7.)

[Jayaseelan 2006] Ramkumar Jayaseelan, Tulika Mitra et Xianfeng Li. *Estimating the worst-case energy consumption of embedded software*. In Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06), pages 81–90, 2006. (Cité en page 91.)

[Koren 1995] Gilad Koren et Dennis Shasha. *Skip-over : Algorithms and complexity for overloaded systems that allow skips*. In Proceedings of the 16th IEEE Real-Time Systems Symposium, pages 110–117, 1995. (Cité en pages 95, 96, 97 et 104.)

[Kumar 2006] Sanjeev Kumar, Michael Chu, Christopher J Hughes, Partha Kundu et Anthony Nguyen. *Hybrid transactional memory*. In Proceedings of the 11th ACM SIGPLAN symposium on Principles and practice of parallel programming, pages 209–220, 2006. (Cité en page 36.)

[Levis 2005] Philip Levis, Samuel Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer*et al. TinyOS : An operating system for sensor networks*. In Ambient intelligence, pages 115–148. Springer, 2005. (Cité en page 120.)

[Liu 1973] Chung Laung Liu et James W Layland. *Scheduling algorithms for multiprogramming in a hard-real-time environment*. Journal of the ACM (JACM), vol. 20, no. 1, pages 46–61, 1973. (Cité en pages 73, 76 et 81.)

[Liu 1987] Jane W-S Liu, Kwei-Jay Lin et Swaminathan Natarajan. *Scheduling algorithms for multiprogramming in a hard real-time environment*. pages 252–260, 1987. (Cité en page 95.)

[Liu 2008] Shaobo Liu, Qinru Qiu et Qing Wu. *Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting*. In IEEE Design, Automation and Test in Europe, pages 236–241, 2008. (Cité en page 91.)

[Liu 2009] Shaobo Liu, Qing Wu et Qinru Qiu. *An adaptive scheduling and voltage/frequency selection algorithm for real-time energy harvesting systems*. In Proceedings of the 46th Annual Design Automation Conference, pages 782–787, 2009. (Cité en page 91.)

[Liu 2011] Shaobo Liu, Jun Lu, Qing Wu et Qinru Qiu. *Harvesting-aware power management for real-time systems with renewable energy*. IEEE Transactions on Very large Scale Integration (VLSI) systems, vol. 20, no. 8, pages 1473–1486, 2011. (Cité en page 86.)

[Locke 1987] C Douglass Locke. *Best-Effort Decision-making for Real-Time Scheduling*. PhD thesis, Carnegie Mellon University, United States, 1987. (Cité en page 94.)

[Lu 2010] Jun Lu, Shaobo Liu, Qing Wu et Qinru Qiu. *Accurate modeling and prediction of energy availability in energy harvesting real-time embedded systems*. In Proceedings of the IEEE International Conference on Green Computing, pages 469–476, 2010. (Cité en page 91.)

[Marchand 2006] Audrey Marchand. *Ordonnancement temps réel avec contraintes de qualité de service : de la théorie à l'intégration*. PhD thesis, University of Nantes, France, 2006. (Cité en pages 100, 101 et 102.)

[Masmano 2004] Miguel Masmano, Ismael Ripoll, Alfons Crespo et Jorge Real. *TLSF : A new dynamic memory allocator for real-time systems*. In Proceedings of the 16th IEEE Euromicro Conference on Real-Time Systems, pages 79–88, 2004. (Cité en page 41.)

[Moser 2007] Clemens Moser, Davide Brunelli, Lothar Thiele et Luca Benini. *Real-time scheduling for energy harvesting sensor nodes*. Real-Time Systems, vol. 37, no. 3, pages 233–260, 2007. (Cité en pages 86 et 91.)

[Orozco 1997] Javier Orozco, Ricardo Cayssials, Jorge Santos et Edgardo Ferro. *Precedence constraints in hard real-time distributed systems.* In Proceedings of the 3rd IEEE International Conference on Engineering of Complex Computer Systems, pages 33–38, 1997. (Cité en page 119.)

[Pillai 2007] Padmanabhan S Pillai et Kang G Shin. *Dynamic DVFS scheduling.* In Designing Embedded Processors, pages 243–258. Springer, 2007. (Cité en page 72.)

[Piorno 2009] Joaquin Recas Piorno, Carlo Bergonzini, David Atienza et Tajana Simunic Rosing. *Prediction and management in energy harvested wireless sensor nodes.* In Proceedings of the 1st IEEE International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, pages 6–10, 2009. (Cité en page 91.)

[Queudet 2017] Audrey Queudet, Nadine Abdallah et Maryline Chetto. *KTS : a real-time mapping algorithm for NoC-based many-cores.* The Journal of Supercomputing, vol. 73, no. 8, pages 3635–3651, 2017. (Cité en pages 7 et 114.)

[Queudet 2020] Audrey Queudet et Maryline Chetto. *Energy-aware Aperiodic Task Servers for Firm Real-time Energy harvesting Systems.* In Proceedings of the IEEE International Conference on Green Computing and Communications, 2020. (Cité en page 114.)

[Sarni 2009a] Toufik Sarni, Audrey Queudet et Patrick Valduriez. *Real-Time Scheduling of Transactions in Multicore Systems.* In Workshop on Massively Multiprocessor and Multicore Computers, 2009. (Cité en pages 7 et 51.)

[Sarni 2009b] Toufik Sarni, Audrey Queudet et Patrick Valduriez. *Real-time support for software transactional memory.* In Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pages 477–485, 2009. (Cité en pages 7 et 51.)

[Sarni 2009c] Toufik Sarni, Audrey Queudet et Patrick Valduriez. *Software transactional memory : Worst case execution time analysis.* In Proceedings of the 17th IEEE International Conference on Real-Time and Network Systems, 2009. (Cité en pages 7 et 51.)

[Sarni 2012] Toufik Sarni. *Vers une mémoire transactionnelle temps réel.* PhD thesis, University of Nantes, France, 2012. (Cité en pages 48 et 51.)

[Scherer III 2004] William N Scherer III et Michael L Scott. *Contention management in dynamic software transactional memory.* In PODC Workshop on Concurrency and Synchronization in Java programs, volume 10, 2004. (Cité en page 35.)

[Shavit 1997] Nir Shavit et Dan Touitou. *Software transactional memory.* Distributed Computing, vol. 10, no. 2, pages 99–116, 1997. (Cité en page 35.)

[Shriraman 2009] Arrvindh Shriraman et Sandhya Dwarkadas. *Refereeing conflicts in hardware transactional memory.* In Proceedings of the 23rd international conference on Supercomputing, pages 136–146, 2009. (Cité en page 34.)

[Silly 1986] Maryline Silly. *La tolérance aux fautes dans un système temps-réel à contraintes strictes.* Rapport technique, INRIA, Rapport de recherche N°512, 1986. (Cité en page 95.)

[Silly 1999] Maryline Silly. *The EDL server for scheduling periodic and soft aperiodic tasks with resource constraints.* Real-Time Systems, vol. 17, no. 1, pages 87–111, 1999. (Cité en page 99.)

[Sinha 2001] Amit Sinha et Anantha Chandrakasan. *Dynamic power management in wireless sensor networks.* IEEE Design & Test of Computers, vol. 18, no. 2, pages 62–74, 2001. (Cité en page 72.)

[Yao 1995] Frances Yao, Alan Demers et Scott Shenker. *A scheduling model for reduced CPU energy.* In Proceedings of IEEE 36th annual foundations of computer science, pages 374–382, 1995. (Cité en page 72.)

[Yildiz 2009] Faruk Yildiz. *Potential Ambient Energy-Harvesting Sources and Techniques.* Journal of technology Studies, vol. 35, no. 1, pages 40–48, 2009. (Cité en page 71.)

[Zhang 2019] Jiange Zhang, Qing Yi et Damian Dechev. *Automating non-blocking synchronization in concurrent data abstractions.* In Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 735–747, 2019. (Cité en page 32.)

UNIVERSITÉ DE NANTES

LS2N LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

Titre : Synchronisation et Autonomie énergétique des Systèmes temps réel embarqués

Mots clés : Systèmes temps réel, ordonnancement temps réel dynamique, autonomie énergétique, mémoires transactionnelles logicielles, synchronisation multicœur.

Résumé : Les travaux présentés dans ce document s'articulent selon deux axes de recherche principaux : (i) *la synchronisation d'applications temps réel multicœur*, et (ii) *l'autonomie énergétique des systèmes temps réel embarqués*. Ces problématiques sont abordées du point de vue de *l'ordonnancement* en prenant en compte plusieurs types de contraintes : des contraintes temporelles, des contraintes de partage de ressources, des contraintes de qualité de service (QoS), et des contraintes d'énergie.

Gérer efficacement les accès concurrents à la mémoire dans un contexte temps réel multicœur se révèle complexe. Le verrouillage systématique peut réduire le parallélisme de la plate-forme d'exécution et engendrer par là-même une baisse significative des performances. Forts de ce constat, nous avons donc étudié l'adaptation des *mémoires transactionnelles* (mécanismes de synchronisation non-bloquant) aux systèmes temps réel multicœur. Au travers d'une étude expérimentale, nous avons comparé les performances de plusieurs mémoires transactionnelles afin de déterminer les facteurs clés (ex : système d'exploitation, politique d'ordonnancement, allocateur mémoire) influant sur la gigue d'exécution des transactions, gigue pouvant impacter directement le respect des contraintes temporelles des tâches dans un système temps réel. Nous avons ensuite proposé une mémoire transactionnelle logicielle temps réel *hard*, la STM-HRT, permettant de garantir la progression de toutes les transactions du système. Une analyse fonctionnelle et temporelle de la STM-HRT nous a permis de valider le mécanisme de synchronisation non bloquant proposé.

Les systèmes embarqués de nouvelle génération tels que les nœuds de capteurs sans fil se sont multipliés ces dernières années. Pour nombre de ces systèmes, l'autonomie énergétique est une problématique capitale. La technologie du *energy harvesting* consistant à capter l'énergie dans l'environnement pour alimenter un système, permet en particulier de doter ces systèmes embarqués aux ressources contraintes, d'une capacité d'autosuffisance énergétique.

L'informatique "intelligente" embarquée au sein de ces systèmes possède très souvent des exigences temps réel au niveau des traitements logiciels. Il convient alors de garantir le fonctionnement perpétuel du système en gérant conjointement deux types de contraintes : *le temps* et *l'énergie*. C'est précisément l'objet de nos contributions sur cette thématique, en considérant le problème du point de vue de l'ordonnancement des tâches applicatives sur le processeur. Nous avons tout d'abord mis en évidence l'inefficacité des ordonnanceurs temps réel "classiques" tels que EDF, incapable à s'accommoder des fluctuations de l'énergie d'alimentation. Nous avons cependant montré qu'il reste le meilleur ordonnanceur non-oisif dans le contexte du energy harvesting et qu'il constitue le meilleur choix d'intégration pour un système ne pouvant disposer d'une estimation prédictive de l'énergie exploitable. Nos travaux ont ensuite consisté en l'identification de quelques-unes propriétés clés d'un ordonnanceur dans le contexte du energy harvesting. Puis, nous avons contribué au problème de la faisabilité d'un ensemble de tâches temps réel à contraintes strictes dans ce même contexte, par la proposition d'un nouveau test, robuste vis-à-vis de la puissance de la source d'énergie. Les contributions présentées par la suite visent à apporter des solutions adaptées aux situations de *surcharge temporaire de traitement et/ou de consommation énergétique* dont un système peut souffrir. Sur la base d'un nouveau modèle de tâches intégrant des contraintes de QoS, nous avons proposé de nouveaux ordonnanceurs contrôlant notamment le nombre et l'identité des jobs de tâches abandonnés en cas de surcharge. Nous avons enfin proposé un test de faisabilité nécessaire intégrant conjointement les contraintes de QoS et d'énergie.