UNIVERSITÉ DE NANTES
FACULTÉ DES SCIENCES ET DES TECHNIQUES

ÉCOLE DOCTORALE STIM
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET MATHÉMATIQUES

Année 2012

N° attribué par la bibliothèque

Gestion de l'énergie renouvelable et ordonnancement temps réel dans les systèmes embarqués

THÈSE DE DOCTORAT

Discipline : Automatique et Informatique Appliquée
Spécialité : Systèmes temps réel

Présentée
et soutenue publiquement par

Hui ZHANG

Le 4 juin 2012, devant le jury ci-dessous

Président :		
Rapporteurs :	Emmanuel GROLLEAU	Professeur, ENSMA Poitiers
	Laurent GEORGE	Maître de Conférences & HDR, Université Paris-Est Créteil
Examinateurs :	Maryline CHETTO	Professeur, Université de Nantes
	Serge MIDONNET	Maître de Conférences, Université Paris-Est Marne-la-Vallée
Membre invité :	Audrey QUEUDET	Maître de Conférences, Université de Nantes
Directrice de thèse :	Maryline CHETTO	Professeur, Université de Nantes

N° ED : 503 -

Remerciements

Cette thèse est le fruit d'un travail mené au sein du Laboratoire IRCCyN (Institut de Recherche en Communications et Cybernétique de Nantes). Je remercie chaleureusement Monsieur Michel MALABRE, directeur de l'IRCCyN, pour m'avoir accueilli dans ce laboratoire prestigieux afin de préparer ma thèse de Doctorat. Je souhaite également remercier le Professeur Yvon TRINQUET, responsable de l'équipe STR (Systèmes Temps Réel) dans laquelle j'ai travaillé et tous les membres de l'équipe STR, pour les bons moments passés en leur compagnie.

Je tiens particulièrement à exprimer mes plus sincères remerciements et ma grande reconnaissance à ma directrice de thèse Maryline CHETTO pour l'encadrement dont elle m'a fait bénéficier, sa disponibilité et son suivi tout au long de ma thèse, pour sa persévérance et sa confiance en moi au fil de ces années, malgré les échéances que je n'ai pas toujours pu respecter. Sa bonne humeur et son amitié m'ont permis de dépasser certains moments difficiles et d'accomplir cette thèse. Ses très nombreux conseils précieux, ainsi que les discussions que nous avons eues m'ont été d'une aide inestimable et ont été déterminants pour ma formation de chercheur.

J'adresse un grand remerciement aux rapporteurs Laurent GEORGE et Emmanuel GROLLEAU qui ont eu la lourde tâche de rapporter sur ma thèse, ainsi qu'aux membres du jury, Audrey QUEUDET et Serge MIDONNET qui m'ont fait l'honneur d'accepter d'être examinateurs.

Enfin, je souhaite remercier ma famille pour son soutien ainsi que mes proches et mes amis pour leur présence et leurs encouragements.

Gestion de l'énergie renouvelable et Ordonnancement temps réel dans les systèmes embarqués

Résumé : Les dispositifs embarqués de nouvelle génération, comme les réseaux de capteurs sans-fil, limitent les interventions humaines. Ils fonctionnent avec un réservoir d'énergie qui se recharge grâce à une source d'énergie renouvelable, par exemple l'énergie solaire. Concevoir de tels systèmes embarqués, entièrement autonomes, nécessite la résolution d'un certain nombre de problèmes liés à la récolte de l'énergie ambiante, son stockage et son utilisation, de façon à assurer une autonomie durable.

L'étude présentée dans ce manuscrit se restreint à une architecture monoprocesseur, pourvue d'un seul niveau de fréquence et destinée à supporter une application temps réel à contraintes fermes. Le travail de thèse traite de la recherche et la validation de mécanismes qui permettent d'adapter au mieux l'activité d'un système embarqué au profil de la source d'énergie ambiante. Dans cette thèse, nous avons d'abord mis en évidence les points faibles de l'ordonnanceur LSA (Lazy Scheduling Algorithm). Puis nous avons entrepris une étude de simulation étendue de LSA dans deux cas. Le premier concerne les applications intégrées dans un environnement qui produit son énergie régulièrement au cours du temps et le second lorsque cette énergie est produite selon un profil variable.

Cette étude montre que l'optimalité de LSA est contrebalancée par une complexité d'implémentation importante. Ainsi les gains de performance ne sont pas si notoires notamment si nous comparons LSA à certaines heuristiques non clairvoyantes que nous avons proposées.

Mots-clés : Systèmes embarqués, ordonnancement temps réel, récupération de l'énergie ambiante, surcoût d'exécution.

Renewable energy harvesting and real-time scheduling in embedded systems

Abstract : The new generation of embedded systems, working in some locations with limited human intervention, will have the capability to harvest energy from the environment, such as wireless sensor networks. Usually, it work with an energy storage that can be recharged from a renewable energy source, e.g., solar energy. The systematic design of such fully autonomous systems, requires to solve a number of problems related to harvesting, storage and use of ambient energy, in order to guarantee a durable running of the device.

The work presented in this manuscript focusses on a single-processor architecture using mono-frequency which supports firm real-time applications. In a first part, we realized a background on real-time computing and we described advanced technical developments in energy harvesting. In a following part, we pointed out the benefits and limitations of the LSA scheduler (Lazy Scheduling Algorithm) designed for real-time energy harvesting systems, and we proposed heuristics based on EDF (Earliest Deadline First) scheduler so as to reduce scheduling overhead. In order to compare those scheduling strategies, we carried out extensive simulation studies in two cases. In the first one, we were only concerned about the real-time embedded systems working in an environment where renewable energy can be maintained in a constant supply over time. In contrast, the second case focussed on the applications, supplied with a variable renewable energy source.

This study shows that the benefits of LSA algorithm are countervailed by the implementation complexity. Thus, the theoretical performances of LSA are not so significant if we compare this optimal scheduler with certain non clairvoyant and non idling scheduling heuristics, very practical to implement in any operating system.

Keywords : Embedded systems, real-time scheduling, energy harvesting from renewable sources, scheduling overhead.

Table des matières

Table des figures

Liste des tableaux

Lexique terminologique

Définitions des termes employés dans la thèse :

ALAP : As Late As Possible

ASAP : As Soon As Possible

C4ISRT : Command, Control, Communications, Computing, Intelligence, Surveillance, Reconnaissance, and Targeting

CNS : Condition Nécessaire et Suffisante

DM : Deadline Monotonic

DPM : Dynamic Power Management

DVS : Dynamic Voltage Scaling

EA-DVFS : Energy Aware Dynamic Voltage and Frequency Selection

EDc : Earliest Deadline with discard the current task

EDd : Earliest Deadline with discard

EDF : Earliest Deadline First

EDi : Earliest Deadline with inserted idle times

EDt : Earliest Deadline with test

EDu : Earliest Deadline with inserted unit time

EH : Energy Harvesting

ER : Echéance sur Requête

EVCC : Energy Variability Characterization Curves

LLF : Least Laxity First

LSA : Lazy Scheduling Algorithms

MEMS : Micro Electro Mechanical Systems

PPCM : Plus Petit Commun Multiple

QdS : Qualité de Service

RM : Rate Monotonic

RTOS : Real-Time Operating System

SREC : Système de Récupération de l'Energie Cinétique

WCET : Worst Case Execution Time

WSN : Wireless Sensor Network

Introduction générale

Contexte des travaux de thèse

Les matériels numériques portables tels qu'implants biomédicaux et réseaux de capteurs sans-fil prennent une place de plus en plus prépondérante dans notre vie. Dans ces dispositifs embarqués, la gestion de l'énergie électrique est une problématique cruciale. Pour un téléphone cellulaire dont la batterie stocke une énergie en quantité limitée et peut être rechargée régulièrement par son utilisateur, cette problématique revient à minimiser l'énergie consommée pour maximiser sa durée d'autonomie. Elle est en général traitée par des méthodes de type DVS (Dynamic Voltage Scaling) visant à diminuer la vitesse du processeur tant que possible.

Mais bon nombre de systèmes embarqués de nouvelle génération limitent voire interdisent les interventions humaines notamment parce qu'ils sont inaccessibles ou déployés en trop grand nombre. Ils fonctionnent grâce à des batteries ou/et des supercondensateurs qui se rechargent grâce à une source d'énergie renouvelable, par exemple l'énergie solaire. Cette énergie peut être présente en quantité non limitée et se caractériser par une puissance qui fluctue au cours du temps.

Concevoir de tels systèmes embarqués, entièrement autonomes, nécessite la résolution d'un certain nombre de problèmes liés à la récolte de l'énergie ambiante, son stockage et son utilisation, de façon à assurer une autonomie durable (d'une à une dizaine d'années) et ce, tout en maintenant un niveau de performance acceptable au système de traitement.

Problématique traitée

Le travail de thèse traite de la recherche et la validation de mécanismes qui permettent d'adapter au mieux l'activité d'un système embarqué au profil de la source d'énergie ambiante qui sert à l'alimenter. Pour les applications temps réel dites à contraintes strictes (en anglais, hard real-time), il s'agit de déterminer les conditions portant sur la taille minimale du réservoir d'énergie qui permettra de garantir à la fois une autonomie durable et le respect des contraintes temporelles exprimées en termes d'échéances. Cela suppose de déterminer des mécanismes d'ordonnancement pour planifier l'exécution des tâches en fonction de l'énergie qu'elles requièrent et de leur échéance, ainsi que du profil de la variation de puissance d'énergie environnementale.

En ce qui concerne les applications dites à contraintes fermes (en anglais, firm real-time), l'ordonnancement des tâches devra permettre d'optimiser un critère de qualité de service, ici le ratio d'échéances satisfaites, lorsque l'énergie disponible s'avère insuffisante pour satisfaire la

totalité des contraintes temporelles sur toute la durée de vie de l'application. L'étude présentée dans ce manuscrit se restreint ici à une architecture monoprocesseur, pourvue d'un seul niveau de fréquence et destinée à supporter une application temps réel ferme.

La thématique de recherche portant sur la minimisation de l'énergie consommée dans les systèmes temps réel a été étudiée largement depuis une quinzaine d'années. Par contre, celle relative à la gestion de l'énergie renouvelable et au problème sous-jacent de l'ordonnancement temps réel s'avère largement ouverte encore aujourd'hui. Elle n'a été étudiée que depuis le début des années 2000 et de façon relativement marginale. Comme moins d'une dizaine d'équipes de recherche se sont intéressés à cette thématique connue sous le nom anglais de *real-time energy harvesting*, un nombre très restreint de résultats sont disponibles.

Contributions

Les travaux menés au cours de cette thèse ont d'abord consisté, après une étude bibliographique, à introduire une nouvelle terminologie inexistante jusqu'alors, relative à l'ordonnancement des systèmes temps réel sous contraintes énergétiques. A ce jour, seul l'ordonnanceur LSA (Lazy Scheduling Algorithm), proposé en 2006 a fait l'objet d'une étude qui démontre de façon théorique son optimalité. Dans cette thèse, nous avons d'abord mis en évidence les points faibles de cet ordonnanceur compte tenu du modèle sur lequel il s'appuie. Puis nous avons entrepris une étude de simulation étendue de LSA dans deux cas. Le premier concerne les applications intégrées dans un environnement qui produit son énergie régulièrement au cours du temps c'est-à-dire à puissance constante et le second lorsque cette énergie est produite selon un profil variable. Cette étude montre que l'optimalité de LSA est contrebalancée par une complexité d'implémentation importante. Ainsi les gains de performance ne sont pas si notoires notamment si nous comparons LSA à certaines heuristiques non clairvoyantes et non oisives que nous avons proposées. De plus, nous avons suggéré une version approximée de LSA offrant ainsi un bon compromis performance/complexité.

Plan du mémoire

En conséquence, le manuscrit se compose des chapitres suivants :

Le chapitre 1 effectue un rappel de généralités sur les systèmes temps réel et plus précisément sur l'ordonnancement de tâches temps réel. Nous évoquons aussi les réseaux de capteurs qui constituent sans aucun doute le secteur privilégié d'utilisation de l'énergie ambiante pour l'alimentation de systèmes informatiques soumis à des contraintes temps réel.

Dans le chapitre 2, nous décrivons un état de l'art des technologies dites de "energy harvesting" qui recouvrent à la fois la récupération de l'énergie ambiante, la conversion de celle-ci en énergie électrique et le stockage temporaire de l'énergie électrique. C'est en particulier parce que l'avancée de ces technologies a été conséquente au cours de ces dix dernières années que les systèmes autonomes se répandent dans des secteurs d'applications trés divers que nous décrivons aussi.

Le chapitre 3 décrit le modèle de système étudié ainsi que le problème d'ordonnancement

sous-jacent. Nous y donnons des conditions d'ordonnançabilité. Ce chapitre se termine par la mise en évidence de la faiblesse de l'ordonnanceur conventionnel EDF.

Nous consacrons le chapitre 4 à une description détaillée de l'ordonnanceur optimal appelé LSA tant du point de vue de son principe que de sa complexité d'implémentation. Nous mettons ici en exergue les hypothèses irréalistes sans lesquelles l'optimalité de cet ordonnanceur ne tient plus.

Au vu des constatations précèdentes, nous consacrons les chapitres 5 et 6 à une étude de simulation dans le but de comparer l'ordonnanceur LSA à plusieurs heuristiques que nous proposons. Cette étude a lieu sous l'hypothèse où l'énergie est produite par l'environnement avec une puissance constante (chapitre 5) puis variable au cours du temps (chapitre 6).

Dans le chapitre 7, nous décrivons les résultats d'une étude de simulation qui compare une version approximée de LSA à la version optimale de LSA.

La réalisation d'un outil de test dédié à l'analyse de faisabilité d'une application temps réel récupérant l'énergie renouvelable est reportée dans le chapitre 8.

Le mémoire se conclut par un résumé de nos contributions et par la description de plusieurs perspectives de travaux futurs.

Chapitre 1

Généralités sur les systèmes temps réel

1.1 Introduction aux systèmes temps réel

Pendant les vingt dernières années, l'informatique temps réel a profondément évolué en se complexifiant. Nous la trouvons dans des applications extrèmement variées allant du grand public à celles d'une industrie très pointue. Le système de freinage ABS, les radars, les dispositifs multimédias portables tels que iPad en sont des exemples caractéristiques. Par rapport à un système informatique conventionnel, un système temps réel n'est pas nécessairement un système "qui va vite" mais celui qui satisfait à des contraintes temporelles imposées par l'application dans laquelle il est mis en oeuvre. En effet, cette dynamique peut être de l'ordre de la milliseconde pour les systèmes radar, de la fraction de seconde pour les systèmes multimédias, de la minute pour le contrôle de production, de l'heure pour la prévision météo, etc. Mais dans la grande majorité des cas, ces contraintes portent sur les délais exigés pour le calcul de données. Lorsque ces contraintes temporelles ne sont pas respectées, le système peut alors être considéré comme défaillant. Il s'avère donc essentiel de pouvoir garantir le respect des contraintes temporelles, en totalité ou en partie, avant même la mise en opération d'un système temps réel.

1.1.1 Définitions

Dans la littérature scientifique, de nombreuses définitions ont été proposées pour préciser la notion de système temps réel. Ci-après, sont énumérées deux définitions dont la première semble la plus couramment utilisée. Alors qu'il y a une trentaine d'années, l'informatique temps réel ne concernait que les applications de contrôle de processus industriels, elle est présente maintenant dans de multiples secteurs qui dépassent largement celui de l'automatique.

Définition 1.1. *Les systèmes temps réel sont des systèmes informatiques qui doivent obligatoirement réagir à l'environnement dans les contraintes précises du temps. En conséquence, le comportement correct de ces systèmes dépend non seulement de la valeur du résultat logique, mais aussi du temps de réponse auquel le résultat est produit [49].*

Autrement dit, les sorties du système temps réel doivent satisfaire non seulement aux contraintes logiques, mais aussi aux contraintes temporelles (dans un temps borné) qui lui sont imposées.

Définition 1.2. *Nous qualifions de "temps réel" la vitesse de réaction d'un système de pilotage lorsqu'il est assujetti à l'évolution dynamique d'un procédé qui lui est connecté et qu'il doit piloter (ou superviser) en réagissant à tous ses changements d'état [14].*

1.1.2 Système de contrôle/commande temps réel

Un système informatique temps réel se compose d'une architecture matérielle sur laquelle s'exécutent les composants d'une architecture logicielle. Lorsque celui-ci est destiné à une application de contrôle/commande, il peut intéragir avec le système contrôlé via une instrumentation composée de capteurs et d'actionneurs.

La figure 1.1 nous présente un système temps réel qui consiste essentiellement en deux parties.

1. Le système contrôlé (appelé aussi *procédé*) désigne un environnement (ou les entités physiques) dont l'état évolue dynamiquement au cours du temps.
2. Le système de contrôle (appelé aussi *contrôleur*), qui prend régulièrement connaissance de l'état du procédé par le biais de capteurs, et calcule une consigne sur les valeurs mesurées décrivant l'état actuel du procédé et les algorithmes appliqués, émise vers des actionneurs pour commander les changements de procédé.

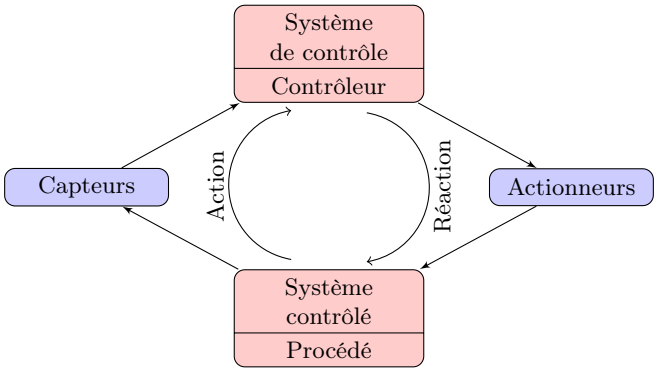


Figure 1.1: – Architecture générale d'un système de contrôle/commande temps réel

L'architecture matérielle présentée précèdemment s'applique également à un système de contrôle /commande non temps réel. La différence entre ces deux types de systèmes se reflète principalement au niveau de l'architecture logicielle et en particulier de l'ordonnanceur.

Depuis quelques années, l'apport de nouvelles technologies, en particulier celles de la communication sans fil a permis d'élargir considérablement le champ des applications temps réel. La figure précèdente ne saurait donc décrire de façon générique un système informatique temps réel. Elle montre toutefois ce qui le distingue d'un système informatique conventionnel : sa dépendance vis à vis d'un environnement (processus, être humain, environnement, etc.) qui évolue avec sa propre dynamique et qui lui impose d'avoir des temps de réponse contraints sur ses calculs et ses actions.

1.1.3 Secteurs de prédilection de l'informatique temps réel

- Automobile (ABS, Airbag, autocroisière, etc.),
- Aéronautique et aérospatial (télécommande d'engin spatial, contrôle de trajectoire de fusée, contrôle de gouvernail, etc.),
- Systèmes militaires et médicaux,
- Processus industriels (centrales nucléaires, robotique, production chimique, etc.),
- Systèmes de surveillance et d'alarme,
- Systèmes multimédias (MP3, PSP, iPhone, etc.),
- Réalité virtuelle (jeux immergés, etc.),
- Systèmes de télécommunication.

1.1.4 Systèmes embarqués

Une grande partie des systèmes temps réel sont aussi des systèmes dits *embarqués*. Nous parlons de système embarqué lorsque le dispositif informatique est intégré au dispositif physique dont il assure le contrôle et/ou la commande et dont il partage inévitablement les contraintes d'environnement.Un tel système embarqué peut évoluer dans des conditions environnementales non prédictibles et variables en termes de température, humidité, vibrations, chocs, variations d'alimentation, interférences RF, corrosion, etc. Sa caractérisation essentielle repose donc sur la forte interaction existant entre le matériel et le milieu où il opère.

Lorsqu'un système embarqué est intégré dans un produit de consommation courante, sa fabrication en grande série imposera une exigence de coût de production le plus bas possible. Cela impliquera de faire appel à des composants matériels peu onéreux par exemple des mémoires de faible capacité, des processeurs basiques (8 bits) faiblement cadencés, des batteries de faible capacité, etc. La problématique centrale liée à la conception de ce type de système temps réel sera d'offrir la meilleure performance tout en respectant ces contraintes d'ordre économique. Cette conception sera d'autant plus difficile que les objets tendent à se miniaturiser, d'où en plus des contraintes physiques à prendre en compte tel que le volume et le poids qui vont donc influer sur le choix des composants matériels et en particulier du matériel destiné au stockage de son énergie d'alimentation.

Les systèmes embarqués sont utilisés dans des applications de plus en plus critiques dont le dysfonctionnement peut générer des nuisances et des pertes économiques voire aussi avoir des conséquences inacceptables telles que la perte de vies humaines (applications médicales, de transport).

1.1.5 Classification des applications

Dans le contexte *temps réel*, les résultats valides se caractérisent à la fois par leur exactitude logique et leur exactitude temporelle. Lorsqu'une contrainte temporelle n'est pas respectée (échéance dépassée sur la fin d'un programme), les conséquences pourront être, soit catastrophiques, soit juste dommageables à l'application. Selon la gravité qu'entraîne un non-respect de contrainte temporelle (exprimée en terme d'échéance généralement), nous distinguons habituellement trois catégories de système temps réel.

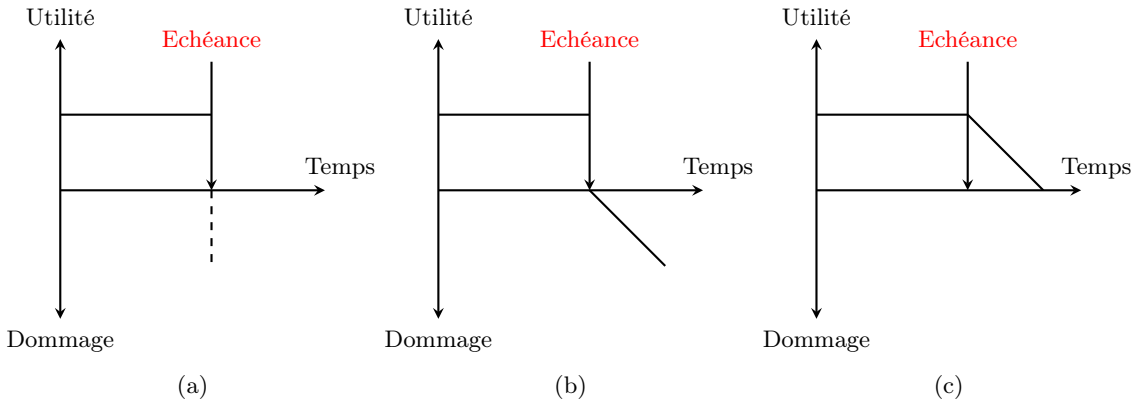


Figure 1.2: – Classification des systèmes temps réel [6]

- Systèmes temps réel à contraintes dures : Le respect d'une échéance est vital pour l'application. En d'autres termes, aucun dépassement d'échéance ne peut être toléré car sinon cela engendrerait l'arrêt de l'application avec des conséquences désastreuses. Voir la figure 1.2 (a).

- Systèmes temps réel à contraintes fermes : Nous tolérons que dans une certaine proportion, certaines échéances ne soient pas respectées. Voir la figure 1.2 (b).

- Systèmes temps réel à contraintes souples : Le dépassement d'une échéance est accepté avec comme unique conséquence une dégradation de performance mais pas l'arrêt de l'application. Voir la figure 1.2 (c).

1.2 Tâches temps réel

1.2.1 Définitions

Nous appelons *tâche* une entité logicielle réalisant une fonction particulière au sein d'un logiciel d'application. Elle correspond à l'exécution d'une séquence d'opérations donnée. Cette suite d'opérations relative au service fourni par la tâche peut être réexécutée plusieurs fois de façon cyclique ou non. Chacune de ces exécutions est alors considérée individuellement sous forme d'*instance* [29] parfois appelée *job* dans la littérature. L'ensemble des tâches à exécuter sur une machine donnée est appelée *configuration de tâches* et sera notée $\mathscr{T}$ dans la suite de ce document.

1.2.2 Modèle canonique de tâches temps réel

Le modèle canonique d'une tâche temps réel τ_i est illustré par la figure 1.3. Les paramètres temporels usuels concernant la modélisation de la tâche τ_i et la configuration de tâches $\mathscr{T}$ sont cités ci-dessous :

- **Date de démarrage du système** t_0 : l'instant auquel le système démarre et que nous considérons comme temps initial ;
- **Date de réveil** (ou d'arrivée) a_i : l'instant à partir duquel la tâche τ_i est autorisée à démarrer son exécution. Nous disons alors qu'elle est *prête* ou en attente d'exécution. Lorsque

plusieurs tâches sont réveillées en même temps ($a_0 = a_1 = ... = a_i$), nous disons qu'elles sont *synchrones* ;

- **Date de début d'exécution** s_i : l'instant auquel la tâche τ_i commence son exécution. Nous disons aussi qu'elle devient *active* à s_i ;
- **Date de fin d'exécution** f_i : l'instant auquel l'exécution de τ_i est terminée ;
- **Echéance** d_i : la date avant laquelle l'exécution entière de τ_i doit être achevée ;
- **Offset de démarrage** O_i : $O_i = a_i - t_0$. Si O_i (ou a_i) est connu a priori alors la tâche est *concrète*. Sinon elle est dite *non concrète* ;
- **Durée d'exécution pire cas** C_i : le temps processeur requis dans le pire des cas pour exécuter totalement la tâche τ_i sans interruption. Cette quantité est appelé généralement WCET (Worst Case Execution Time) ;
- **Temps de réponse** R_i : le délai séparant le réveil et la fin d'exécution de τ_i : $R_i = f_i - a_i$;
- **Délai critique** D_i : le délai maximum acceptable pour l'exécution de τ_i : $D_i = d_i - a_i$;
- **Laxité** L_i : le délai maximum pendant lequel le début d'exécution de τ_i peut être retardé tout en assurant le respect de l'échéance : $L_i = D_i$ - C_i ;
- **Période** T_i : l'intervalle de temps séparant deux réveils successifs de la même tâche τ_i lorsque τ_i doit être exécutée cycliquement ;
- **Facteur d'utilisation** U_i : le taux d'utilisation du processeur par τ_i rapporté à sa période : $U_i = C_i/T_i$;
- **Facteur de charge** Ch_i : le taux d'utilisation du processeur par τ_i rapporté à son délai critique : $Ch_i = C_i/D_i$.

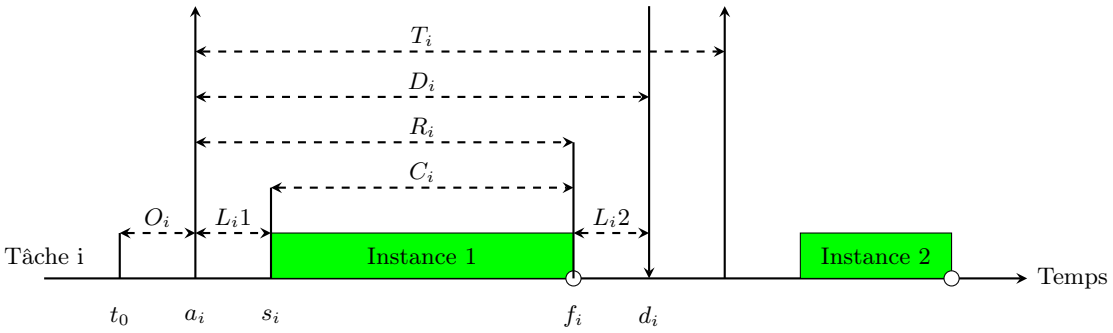


Figure 1.3: – Paramètres temporels d'une tâche temps réel

1.2.3 Types de tâches temps réel

- Tâches périodiques : Elles sont réveillées à intervalles réguliers. L'intervalle de temps entre deux réveils consécutifs est donc constant (T_i), comme l'illustre la figure 1.4. Une configuration de n tâches périodiques est ainsi notée : $\mathscr{T} = \{\ \tau_i | 1 \leq i \leq n, avec\ \tau_i = (a_i, C_i, D_i, T_i)\ \}$.

Nous rencontrons ce type de tâches dans la quasi-totalité des applications de contrôle/commande et dans une majorité des applications temps réel. Pour une tâche T_i, la date de réveil de la m-ième

instance est alors donné par :

$$a_i^m = a_i^1 + (m-1)T_i \quad ; \quad m \geq 1$$

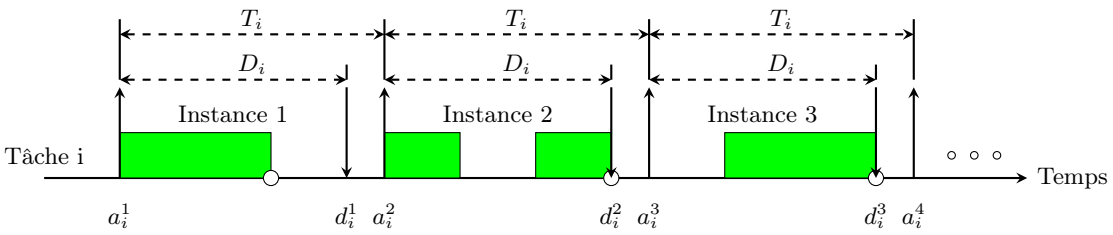


Figure 1.4: – Modèle de tâche périodique

Si $0 < C_i \leq D_i \leq T_i$, nous disons que la tâche T_i est *bien formée*. Si $D_i = T_i$, la tâche est dite à *Echéance sur Requête* (noté ER).

- Tâches sporadiques : Elles sont réveillées de façon cyclique à des instants irréguliers, contrairement aux tâches périodiques. Il existe un intervalle de temps minimal entre deux réveils d'instances successives noté aussi T_i, d'où la condition temporelle suivante : $a_i^{m+1} - a_i^m \geq T_i$. Ce modèle est illustré sur la figure 1.5. Nous pouvons considérer une tâche périodique comme un cas spécial de tâche sporadique. Ainsi tous les résultats de recherche portant sur les tâches sporadiques s'étendent aux tâches périodiques.

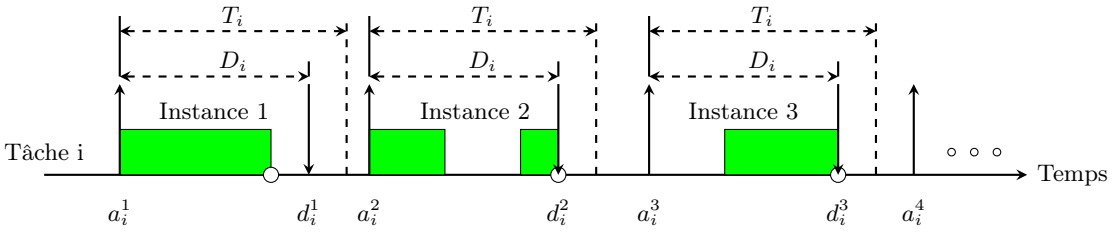


Figure 1.5: – Modèle de tâche sporadique

- Tâches apériodiques : Elles sont réveillées à des instants imprévisibles et sans cyclicité. Une tâche apériodique T_i se caractérise par au moins deux paramètres : sa durée d'exécution pire cas C_i et sa date de réveil a_i. Le modèle est décrit sur la figure 1.6. Comme exemple typique, nous pouvons citer une tâche de traitement d'alarme réveillée sur interruption liée à la réception d'une mesure en provenance d'un capteur ou une tâche d'affichage à l'écran liée à une interruption clavier.

Le réveil d'une tâche apériodique peut être lié à l'occurence d'un événement extérieur au système informatique, en provenance de l'environnement, ou lié à l'occurence d'un signal logiciel émis une autre tâche au cours de son exécution.

Nous distinguons deux catégories de tâches apériodiques selon que leur exécution doit se faire dans des temps contraints ou non. Lorsqu'une tâche apériodique possède une échéance stricte à

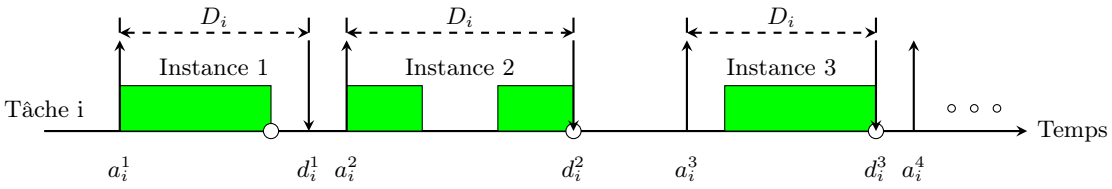


Figure 1.6: – Modèle de tâche apériodique

respecter une fois réveillée, elle est dite *apériodique critique* (cf. figure 1.6). Sinon, elle est dite *apériodique non critique*. Dans ce dernier cas, l'ordonnanceur visera trés souvent à optimiser un temps de réponse moyen des tâches apériodiques non critiques. Nous pourrons aussi par exemple associer à toute tâche, une valeur indiquant l'importance de la tâche d'un point de vue applicatif et utiliser ce paramètre pour optimiser un temps de réponse moyen pondéré.

1.2.4 Caractérisation des tâches

Au vu des descriptions des paramètres temporels, nous faisons ici un petit résumé relatif aux différentes tâches temps réel. Dans le tableau 1.1, nous citons les caractérisatiques les plus usuelles.

Périodique ER	Périodique	Sporadique	Apériodique critique	Apériodique non critique
(a_i, C_i, T_i)	(a_i, C_i, D_i, T_i)	(a_i, C_i, D_i, T_i)	(a_i, C_i, D_i)	(a_i, C_i)

Tableau 1.1: – Caractérisation des tâches

1.3 L'ordonnancement temps réel

Du point de vue logiciel, une application temps réel se structure donc en tâches qui effectuent chacune des traitements avec des besoins propres en termes de ressources logicielles et matérielles. Un *système d'exploitation temps réel* (en anglais, RTOS) (parfois appelé exécutif temps réel) et plus précisément son *noyau* fournit alors tous les services requis pour octroyer ces ressources aux tâches, et ce, en visant le respect de toutes les contraintes spécifiées.

Nous appelons donc *ordonnanceur*, ce composant logiciel du noyau temps réel en charge de planifier l'exécution des tâches au cours du temps c'est-à-dire d'octroyer au bon moment et dans le bon ordre les ressources nécéssaires à l'exécution des tâches. Nous appelons *séquence* la description de cette planification qui se fait généralement par chronogramme ou diagramme de Gantt.

Si nous nous concentrerons sur la principale ressource matérielle qu'est le processeur, concevoir un ordonnanceur consiste à définir une politique (ou une stratégie) d'affectation du processeur (ou des processeurs) aux tâches afin de respecter leurs contraintes temporelles voire en plus leurs contraintes énergétiques comme nous le verrons par la suite.

L'*algorithme d'ordonnancement* représente donc la suite des opérations mises en oeuvre pour sélectionner, dans la liste des tâches prêtes (ayant reçu leur signal de réveil), celle qui se verra affectée le processeur pour son exécution. De la complexité calculatoire de cet algorithme dépendra donc le surcoût temporel d'exécution de l'ordonnanceur à chaque invocation de ce dernier.

1.3.1 Classification des ordonnancements

- Ordonnancement en-ligne vs hors-ligne : L'ordonnancement *hors-ligne* signifie que la séquence d'exécution des tâches est déterminée avant le lancement de l'application. Cette séquence est établie à partir de tous les paramètres connus a priori. Et une fois la séquence effectuée, elle ne change plus pendant toute la vie de d'application. Avec un ordonnancement hors-ligne, la séquence d'ordonnancement caractérisée par les dates de début d'exécution des tâches, de préemption et reprise éventuelles, est déterminée à l'avance. Cette séquence est en général mémorisée dans une table et utilisée par un séquenceur. Nous parlons aussi d'ordonnancement *statique*.

L'ordonnancement *en-ligne* signifie que la séquence d'exécution se construit dynamiquement en fonction des événements qui surviennent et peuvent modifier à tout moment la liste des tâches prêtes. Parmi ces événements, citons le signal de réveil périodique d'une tâche, la libération d'une ressource critique, etc. Par rapport à un ordonnancement hors-ligne, un ordonnancement en-ligne présente plus de flexibilité. Notamment, il sera plus efficace dans la prise en compte des tâches apériodiques dont l'arrivée n'est pas prévisible. Son inconvénient majeur pourra provenir de sa complexité d'implémentation (overhead) et donc du temps perdu chaque fois qu'il est appelé. La problématique de l'ordonnancement temps réel en-ligne est de fournir le meilleur compromis entre performance et surcôuts d'exécution.

- Ordonnancement monoprocesseur vs multiprocesseur : Si toutes les tâches ne peuvent s'exécuter que sur un seul et unique processeur, l'ordonnancement est qualifié de *monoprocesseur*. Sinon, nous parlons d'ordonnancement multiprocesseur. Cette thèse n'ayant trait qu'aux architectures monoprocesseur, nous ne détaillons pas ici les différents types d'ordonnancements applicables en environnement multiprocesseur.

- Ordonnancement préemptif vs non préemptif : Lorsqu'une tâche peut être interrompue par une ou plusieurs tâches au cours de son exécution, nous parlons d'*ordonnancement préemptif*, sinon d'*ordonnancement non préemptif*. Les deux avantages principaux d'un ordonnancement non préemptif sont d'une part un surcôut d'exécution minimisé vu l'absence de préemptions entre tâches et d'autre part, la résolution implicite d'un certain nombre de problèmes de synchonisation tel que l'accès aux ressources partagées entre les tâches.

- Ordonnancement à priorité fixe vs dynamique : Dans la trés grande majorité des ordonnanceurs en-ligne, la politique d'attribution du processeur se base sur une grandeur attribuée à chaque tâche appelée *priorité*. A chaque décision d'ordonnancement, le processeur est attribué à la tâche prête la plus prioritaire. Lorsque les priorités sont attribuées au démarrage de l'application et pour toute la durée de vie de l'application, nous parlons alors d'*ordonnancement à priorité fixe*. Ainsi pour une tâche périodique, toutes ses instances auront la même priorité. Si les priorités peuvent changer au cours du temps, nous parlons d'*ordonnancement à priorité*

dynamique. Ce type d'ordonnancement a l'avantage de s'adapter à un contexte évolutif tel que celui de tâches apériodiques dont les occurences sont imprévisibles et nécessitent d'être traitées conjointement avec des tâches périodiques. L'urgence d'une tâche pourra être directement caractérisée par une priorité qui grandit plus nous nous rapprocherons de son échéance.

- Ordonnancement oisif vs non oisif : Si le processeur peut être laissé inactif alors même que des tâches sont prêtes à l'exécution, nous parlons d'*ordonnancement oisif* (en anglais, idling). Nous disons aussi que l'ordonnancement se fait avec insertion de *temps creux* (en anglais, idle times) c'est-à-dire des intervalles de temps pendant lesquel le processeur n'exécute aucune tâche. Il peut alors être mis en état de veille lorsque sa technologie le permet et ainsi consommer peu voire aucune énergie. Au contraire, un ordonnancement sera qualifié de *non oisif* (en anglais, non-idling ou work-conservative) lorsque le processeur devient nécessairement actif dès lors qu'il y a au moins une tâche prête à s'exécuter.

- Ordonnancement clairvoyant vs non clairvoyant : Nous disons qu'un ordonnanceur est *non-clairvoyant* lorsqu'il ne sait rien des tâches qu'il doit ordonnancer, en dehors de leur dates d'arrivées, au moment de leur arrivée. Au contraire, il est *clairvoyant* lorsqu'il connait a priori les caractéristiques des tâches qui arriveront dans le futur.

1.3.2 Terminologie de l'ordonnancement temps réel

- Ordonnancement faisable : Nous disons qu'un ordonnancement est *faisable* pour une configuration de tâches donnée s'il existe au moins un ordonnanceur capable de produire une séquence où toutes les contraintes temporelles de cette configuration sont respectées.

- Séquence valide : Une séquence où toutes les tâches terminent leur exécution avant échéance est dite *valide*.

- Configuration de tâches ordonnançable vs non ordonnançable : Nous disons qu'une configuration de tâches est *ordonnançable* s'il existe au moins un ordonnanceur capable de satisfaire toutes les contraintes temporelles de cette configuration. S'il n'existe aucun ordonnancement faisable pour une configuration, nous disons que celle-ci est *non ordonnançable*. C'est notamment le cas de toutes les configurations de tâches qui nécessitent plus de cent pour cent de temps processeur et donc qui *surchargent* le processeur.

- Configuration de tâches fiablement ordonnancée : Une configuration de tâches est dite *fiablement ordonnancée* par un algorithme d'ordonnancement donné si ce dernier construit une séquence valide pour la configuration.

- Condition d'ordonnançabilité : Nous appelons *condition d'ordonnançabilité*, l'ensemble des conditions qui lorsqu'elles sont vérifiées permettent de garantir qu'une configuration de tâches est ordonnançable. Une telle condition doit être testée hors-ligne pour décider de la faisabilité d'une application temps réel à contraintes dures avant son lancement.

- Ordonnanceur optimal : Nous qualifions un ordonnanceur d'*optimal* s'il ordonnance fiablement toutes les configurations de tâches ordonnançables. En d'autres termes, si une configuration n'est pas fiablement ordonnancée par un ordonnanceur optimal, alors elle ne peut l'être par aucun autre ordonnanceur.

- Optimalité de classe : Nous pouvons définir l'optimalité dans une *classe d'ordonnan-*

ceurs. Par exemple, nous disons qu'un ordonnanceur est optimal dans la classe des ordonnanceurs à priorité fixe s'il est le meilleur ordonnanceur parmi ceux à priorité fixe. Il se peut donc qu'il ne soit pas le meilleur de tous.

- Surcoût d'exécution d'un ordonnanceur : Tout algorithme d'ordonnancement est codé sous forme d'un programme séquentiel comportant une successions d'opérations élémentaires (additions, multiplications, etc.). Le temps mis par celles-ci pour s'exécuter engendre une perte de temps pour l'application appelé *surcoût d'exécution* (en anglais, overhead).

1.3.3 Ordonnancement de tâches périodiques

Comme la quasi-totalité des applications temps réel se composent de tâches périodiques, l'ordonnancement d'une configuration de tâches périodiques a été largement étudié, conduisant à proposer des algorithmes d'ordonnancement et les conditions d'ordonnançabilité associées. Dans la suite de ce document, nous ne présenterons que les ordonnancements les plus notoires parmi ceux conduits par la priorité et sous les hypothèses suivantes : ordonnancement monoprocesseur préemptif en-ligne appliqué à une configuration de tâches indépendantes. Nous appelons ici, *tâche indépendante* une tâche qui n'a aucun lien de précédence, de synchronisation et/ou de communication avec une autre. Une telle tâche est amenée à suspendre son exécution uniquement lorsqu'une tâche de priorité plus élevée est réveillée.

1.3.3.1 Problématiques

Dans toute application temps réel à contraintes dures, la problématique de l'ordonnancement se ramène d'une part à trouver un algorithme optimal et d'autre part à trouver l'ensemble des conditions qui lorsqu'elles sont verifiées nous permettent de garantir le respect des échéances des tâches sur une durée infinie. Toute mise en oeuvre d'une application passe ainsi par une phase de test hors-ligne, dite d'*analyse d'ordonnançabilité*.

La conclusion de cette analyse permettra de lancer l'application sans modification si le test conduit à une réponse positive concernant son ordonnançabilité avec un algorithme donné. Au contraire, s'il conduit à une réponse négative et si la condition testée est suffisante mais pas nécessaire alors cette analyse devra être complétée par un test de simulation sur un intervalle temps suffisant. Si par contre, la condition est à la fois nécessaire et suffisante, il faudra alors nécessairement modifier les paramètres temporels de l'application (augmentation des périodes des tâches tant que possible, modification du code pour réduire les durées d'exécution des tâches) de façon à réduire la charge de traitement imposé au processeur et ainsi pouvoir respecter les contraintes temporelles. Une autre possibilité souvent imposée par l'application (période de tâche non modifiable sans nuire à la stabilité du procédé commandé par exemple) consistera à opter pour une architecure matérielle plus performante c'est-à-dire choisir un processeur cadencé à une plus grande fréquence pour diminuer les durées d'exécution.

Dans une application temps réel à contraintes fermes, l'objectif de l'ordonnanceur sera d'optimiser le pourcentage d'échéances respectées, c'est-à-dire le pourcentage de tâches qui terminent leur exécution dans le respect de leur contrainte temporelle. Ce ratio détermine représente donc la Qualité de Service d'un tel système. Dans ce contexte, l'analyse d'ordonnançabilité hors-ligne

n'est pas impérative. Mais elle s'avère trés utile car si la réponse au test est négatif, nous pouvons être assuré que des échéances ne pourront pas être respectées. Ainsi, nous pouvons décider à l'avance de la façon dont nous traiterons ce type de situation en-ligne.

Qu'elle que soit le type d'application visée, l'ordonnanceur se devra d'être le moins pénalisant possible du point de vue des pertes de temps engendrées par son exécution du fait que le logiciel d'application pouvant être soumis à des contraintes temporelles trés fortes incompatibles avec ces pertes de temps. Ces dernières sont de deux ordres :

– overhead d'exécution à chaque décision d'ordonnancement ;
– nombre de préemptions, chaque préemption engendrant des changements de contexte.

Etudier la performance d'un ordonnanceur, c'est mesurer d'une part sa performance en termes de quantités de configurations ordonnançables qu'il peut fiablement ordonnancer ou de pourcentages d'échéances satisfaites dans le cas de configurations non ordonnançables. C'est d'autre part évaluer ses surcoûts d'exécution.

1.3.3.2 Algorithmes d'ordonnancement classiques

Ci-dessous, nous citons les ordonnanceurs les plus connus ou les plus mis en oeuvre en raison de leur performance et de leur simplicité d'implémentation, destinés aux applications temps réel à contraintes fermes ou à contraintes dures. Nous donnons pour chacun d'eux une ou plusieurs conditions d'ordonnançabilité, également connues pour leur implémentation aisée.

EDF (Earliest Deadline First) est un algorithme d'ordonnancement à priorité dynamique connu initialement sous le nom d'algorithme de Jackson [18]. Sous EDF, une tâche est d'autant plus prioritaire que son urgence est forte c'est-à-dire que sa date d'échéance est proche de la date courante. Dans le cas particulier où les tâches sont synchrones, EDF est parfois appelé EDD (Earliest Due Date). Il peut aussi porter le nom de Relative Urgency (RU) [27].

Théorème 1.1. *L'algorithme EDF est optimal. [13] [27].*

L'optimalité de EDF signifie que si une configuration de tâches périodiques indépendantes ne peut être fiablement ordonnancée par EDF, alors elle ne pourra l'être par aucun autre ordonnanceur, qu'il soit préemptif ou non-préemptif, oisif ou non oisif, clairvoyant ou non clairvoyant. Si nous supposons les tâches non préemptables, le problème d'ordonnancement associé s'avère NP-difficile [43] signifiant l'absence d'un ordonnanceur optimal dans ce cas. Toutefois, si nous restreignons la recherche d'optimalité à la classe des ordonnanceurs non oisifs appliqué à des tâches non préemptables, l'algorithme EDF reste optimal comme démontré par George *et al.* dans [15].

Nous citons ci-dessous les deux conditions d'ordonnançabilité les plus connues et les plus simples à tester, associées à EDF :

Théorème 1.2. *Une configuration de n tâches périodiques est fiablement ordonnançée par EDF si son facteur de charge donné par* $Ch = \sum_{i=1}^{n} \frac{C_i}{D_i}$*, satisfait [27] :*

$$Ch \leq 1 \tag{1.1}$$

Cette condition est uniquement suffisante. C'est pourquoi, d'autres études ont permis de proposer par la suite des CNS (Condition Nécessaire et Suffisante) utilisant l'approche de la *demande processeur* (en anglais, processor demand). Le test consiste à calculer la quantité de traitement noté $h(L)$ à effectuer dans tout intervalle de temps de longueur L et vérifier que cette quantité de traitement requise est inférieure à L. La première étude utilisant cette approche est reportée dans [5], conduisant au test suivant :

Théorème 1.3. *Une configuration de tâches périodiques est fiablement ordonnançée par EDF si et seulement si :*

$$\forall L > 0, \sum_{i=1}^{n} \lfloor \frac{L + T_i - D_i}{T_i} \rfloor C_i \leq L \tag{1.2}$$

Cette condition a donc l'avantage d'être à la fois nécessaire et suffisante. Mais cet avantage est contrebalancé par son inconvénient majeur : une forte complexité de mise en oeuvre. Si les tâches sont ER (période de tâche égale à échéance), alors nous disposons d'une CNS trés simple :

Théorème 1.4. *Une configuration de tâches périodiques ER est fiablement ordonnançée par EDF si et seulement si son facteur d'utilisation donné par $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$, satisfait [27] :*

$$U \leq 1 \tag{1.3}$$

Outre son optimalité, l'ordonnanceur EDF possède de nombreuses qualités dont :

- Une mise en oeuvre aisée car utilisant un seul paramètre pour ordonner les tâches prêtes, l'échéance.
- Cent pour cent de la capacité de traitement du processeur peut être utilisée tout en garantissant un ordonnancement fiable aux tâches.
- De par son fonctionnement, EDF convient aussi bien à l'ordonnancement de tâches périodiques qu'apériodiques critiques.
- Des variantes de cet ordonnanceur existent de façon à pouvoir ordonnancer des tâches qui ne sont pas nécessairement indépendantes, notamment des tâches accèdant à des ressources critiques, ou des tâches périodiques fermes pouvant sous certaines conditions manquer certaines échéances.
- EDF est non clairvoyant : il produit une séquence optimale sans nécessiter la connaissance des événements futurs.

LLF (Least Laxity First) est un algorithme d'ordonnancement à priorité dynamique pour lequel une tâche est d'autant plus prioritaire que sa laxité est plus faible à la date courante. La laxité est définie par la longueur de temps séparant l'instant courant de l'échéance diminuée de la durée d'exécution restante de la tâche. LLF est aussi un algorithme optimal [48], avec les mêmes conditions d'ordonnançabilité que EDF. Il perd son optimalité en version non préemptive [16]. Son principal défaut tient dans le nombre prohibitif de préemptions engendrées et donc de changements de contexte. Son surcoût d'exécution (overhead) peut ainsi s'évérer inacceptable.

RM (Rate Monotonic) est un algorithme d'ordonnancement à priorité fixe conçu spécialement pour des tâches périodiques ER. Sous RM, une tâche est d'autant plus prioritaire que sa période est plus petite.

Théorème 1.5. *L'algorithme RM est optimal dans la classe des ordonnanceurs à priorité fixe pour des configurations de tâches périodiques indépendantes ER. [27].*

Théorème 1.6. *Une configuration de n tâches périodiques ER est fiablement ordonnançée par RM si son facteur d'utilisation satisfait [27] :*

$$U \leq n(2^{1/n} - 1) \tag{1.4}$$

Notons que le terme $n(2^{1/n} - 1)$ tend vers ln2(=0.693) lorsque n tend vers l'infini. Cela signifie qu'il n'est pas possible avec un tel test de décider de l'ordonnançabilité par RM d'une configuration ayant un facteur d'utilisation compris entre la valeur de ce terme et 1. Cela restreint donc considérablement le champ de son utilisation qui a toutefois le mérite d'être trés simple à mettre en oeuvre. Dans [23], les auteurs montrent que l'algorithme RM peut ordonnancer fiablement une configuration de tâches avec un facteur d'utilisation du processeur pouvant atteindre 88%.

DM (Deadline Monotonic) est un algorithme d'ordonnancement à priorité fixe pour lequel une tâche est d'autant plus prioritaire que son délai critique est plus petit. Nous l'appelons aussi "Inverse Deadline". Il a été proposé initialement dans la littérature par [25]. L'algorithme Deadline Monotonic se ramène à RM quand le délai critique de toute tâche est égal à sa période.

Théorème 1.7. *L'algorithme DM est optimal dans la classe des ordonnanceurs à priorité fixe [25].*

Le principal avantage de l'ordonnanceur DM ou RM réside dans une implémentation trés simple tout en étant optimal dans sa classe. C'est d'ailleurs pour cela qu'il séduit les industriels et en fait certainement l'ordonnanceur le plus couramment intégré dans les systèmes d'exploitation temps réel. De plus, DM se comporte de façon déterministe en cas de surcharge de traitement. En effet, à tout moment, la tâche la moins prioritaire est celle qui a le délai critique le plus grand. C'est donc d'abord celle-ci qui risquera de violer son échéance si du temps processeur venait à manquer puisque la dernière à s'exécuter parmi les tâches prêtes. Avec l'ordonnanceur EDF, il est impossible de connaître a priori quelle tâche pourrait pâtir d'une surcharge, les priorités évoluant dynamiquement.

De nombreux tests d'ordonnançabilité ont par ailleurs été proposés pour fournir des conditions qui sont à la fois nécessaires et suffisantes et permettent ainsi une analyse exacte de l'ordonnançabilité. Comme ces études dépassent le cadre de cette thèse, nous renvoyons le lecteur à des ouvrages de synthèse tel que [7].

1.3.4 Un résultat central : la cyclicité des séquences

Bien qu'intuitif, le résultat certainement le plus fondamental concernant l'ordonnancement de tâches périodiques concerne la forme de la séquence construite. Ce résultat établi par Leung et Merril en 1982 démontre qu'il existe aussi une répétion dans l'activité du processeur lorsque celui-ci doit exécuter des tâches périodiques [24] :

Théorème 1.8. *La séquence produite par tout algorithme d'ordonnancement préemptif sur une configuration de tâches périodiques est elle-même périodique de période égale au Plus Petit Commum Multiple (PPCM) des périodes des tâches de la configuration, noté P.*

Autrement dit, si le processeur est inactif à un instant t donné, il le sera aussi à tout instant $t + kP$ ($k \in \mathbb{N}$). De même, s'il a travaillé x_i unités de temps sur une instance de τ_i à l'instant t, il aura travaillé aussi x_i unités de temps sur une autre instance de τ_i à l'instant $t + kP$.

Ainsi, lorsqu'une condition suffisante d'ordonnançabilité n'est pas valide, il suffit de simuler la séquence produite par l'ordonnanceur sur une fenêtre temporelle appelée *hyper-période* de durée égale au PPCM des périodes.

1.4 Réseaux de capteurs sans fil

1.4.1 Caractéristiques d'un capteur

Un capteur sans fil représente un exemple significatif de système embarqué avec des caractéristiques temps réel. Sa spécificité est son autonomie d'où le qualificatif de *système autonome communicant*. Comme les capteurs sans fil peuvent être destinés à relever des informations en environnements hostiles auxquels l'homme n'a pas toujours accès, nous parlons aussi de *systèmes autonomes abandonnés*. Nous considérons qu'une fois déployés, les capteurs deviennent entièrement autonomes. S'ils sont alimentés par une batterie, la durée de vie et la durée de charge de celle-ci déterminera donc la durée de vie du capteur. L'aspect énergétique est donc au centre des préocupations du concepteur : comment fournir la plus longue autonomie au capteur tout en respectant les limitations d'encombrement et de poids. Cette problématique est d'autant plus sévère que les applications se sophistiquent : plus de traitements, plus de communications et donc davantage d'énergie consommée. Ce constat a suggéré de ne pas embarquer l'énergie nécessaire pour toute la durée de vie d'une application mais à la place de tirer cette énergie de l'environnement en fonction des besoins.

Le concept de réseaux de capteurs sans fil (Wireless Sensor Network, en anglais) a été initialement proposé à partir de la fin des années 70, par l'Université de Carnegie-Mellon où a commencé la recherche sur ce domaine sous le financement du Département de la Défense de U.S.A. Un réseau de capteurs sans fil introduit une nouvelle technologie d'acquisition et de traitement des données, qui intègre trois principales fonctionalités dans un même système : la technologie de capteur, la technologie MEMS (micro-electro-mechanical system) et le réseau sans fil.

1.4.2 Applications

Le réseau de capteurs sans fil peut comprendre différents types de capteurs s'il a en charge de mesurer des paramètres de l'environnement tels que la température, l'humidité, la force, le mouvement, la lumière, le bruit, la vitesse, le volume, l'acido-alcalinité, etc. [1]. En tant que système autonome distribué, il a une grande flexibilité, de la robustesse et de l'intelligence. Il s'applique dans de nombreux secteurs tels que la défense militaire, la sécurité civile, la protection et la surveillance de l'environnement, les soins de santé [1, 42], etc.

Citons les exemples concrets suivants :

– Application militaire : comme une partie intégrée du système C4ISRT (Command, Control, Communications, Computers, Intelligence, Surveillance, Reconnaissance, and Targeting),

il peut réaliser la surveillance des forces d'ennemis, la surveillance du champ de bataille, l'évaluation des dommages de bataille, etc. ;
- Application environnementale : la détection des incendies de forêt, la surveillance des trajectoires des animaux, la détection d'inondation, la surveillance d'atmosphère et de sol, etc. ;
- Santé : le médecin peut faire l'acquisition de nos données physiologiques grâce aux capteurs embarqués sur ou dans le corps humain, et réaliser la télésurveillance ou télédiagnostic ;
- Domotique : les capteurs peuvent être présents dans toutes les pièces d'une maison et servir aussi bien au contrôle d'intrusion, la régulation du chauffage, l'aide à la personne, etc.

1.4.3 Architecture interne d'un noeud

Chaque noeud du réseau de capteurs sans fil constitue un micro système embarqué temps réel constitué de :
- une *unité d'acquisition* utilisant un capteur qui va obtenir des mesures numériques sur les paramètres environnementaux et d'un convertisseur Analogique/Numérique qui convertit l'information relevée et la transmet à l'unité de traitement.
- une *unité de traitement* munie d'une interface avec l'unité d'acquisition et d'une interface avec l'unité de transmission. Cette unité met donc en oeuvre l'application temps réel constituée de tâches telles que décrites précèdemment et un noyau de système d'exploitation temps réel. Elle acquiert les informations en provenance de l'unité d'acquisition et les envoie à l'unité de transmission.
- une *unité de transmission* responsable de toutes les émissions et réceptions de données via un support de communication radio.

Un tel noeud est schématisé sur la figure 1.7.

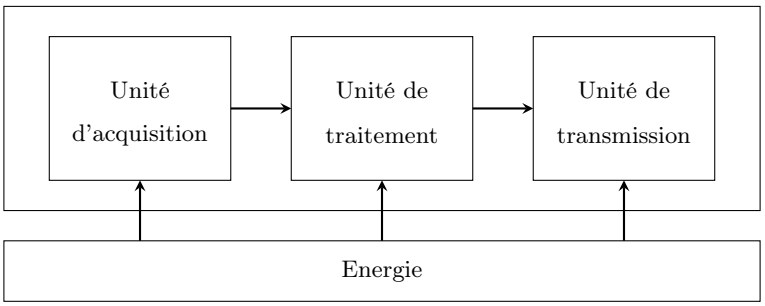


Figure 1.7: – Architecture interne d'un noeud

La figure 1.8 illustre une architecture de communication d'un réseau de capteurs sans fil [1]. Un grand nombre de capteurs (noeuds) sont déployés dans un environnement dynamique, ils peuvent s'auto-configurer en un réseau de communication. Chaque noeud se connecte à Internet par un élément "Sink". Tous les noeuds se trouvent au même niveau : il n'y a pas de précédence, ni de noeud central. Si un noeud est en panne ou se sépare, il n'influe pas sur le reste du réseau.

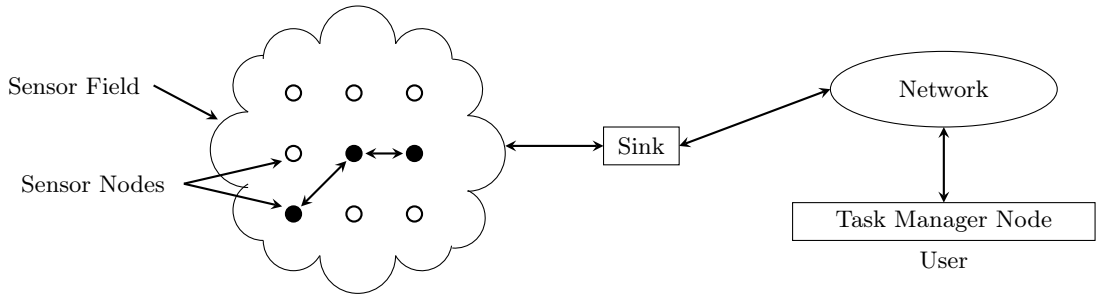


Figure 1.8: – Architecture d'un réseau de capteurs sans fil

1.4.4 Problématique énergétique

Un grand souci relatif à la conception d'un réseau de capteurs sans fil concerne donc la source d'alimentation pour assurer son autonomie sinon perpétuelle au moins pendant plusieurs années. Bien que le réseau de capteur sans fil utilise déjà la technologie MEMS pour diminuer sa consommation d'énergie, la batterie classique ne satisfait pas pleinement le besoin énergétique d'un tel système. En effet, une batterie classique ne peut pas assurer l'alimentation sur une durée "infinie" ni même de plusieurs dizaines d'années ce qui, pourtant correspond à la durée de vie de ces systèmes. De plus une batterie contient des matériaux chimiques. Sa présence dans l'environnement entraine donc des nuisances notamment lors de sa destruction en fin de vie.

La plupart de réseaux de capteurs sans fil trouvent et prouvent leur grande utilité dans des secteurs hostiles où l'humain ne peut accèder facilement (fonds marins, environnements nucléaires ou forêts par exemple). Changer la batterie usagée devient soit impossible soit possible mais à des coûts trop élevés. Nous voyons ainsi que c'est la durée de vie de la batterie qui conditionne la durée du capteur et donc de l'application dans sa globalité. C'est en grande partie dû au constat que les batteries traditionnelles ne peuvent à elles seules suffire à l'alimentation d'un capteur sans fil, que nous nous sommes penchés sur l'utilisation d'une énergie alternative : *l'énergie environnementale* dite aussi *énergie ambiante*.

Désormais, savoir exploiter l'énergie ambiante (énergie solaire, énergie thermique, etc.) devient une problématique abordée non pas par les énergéticiens mais par les informaticiens désireux d'alimenter les systèmes embarqués et particulièrement les réseaux de capteurs sans fil grâce à cette énergie durable et gratuite produite par l'environnement. Les avantages sont alors indéniables : une alimentation perpétuelle des systèmes à moindre côut à la fois d'exploitation et de maintenance et une nuisance sur l'environnement limitée.

1.5 Conclusion

L'objectif de ce chapitre a été de présenter les notions de base nécessaire à la compréhension du contexte de cette thèse. Ce chapitre regroupe les généralités sur l'informatique temps réel. Nous y définissions les caractéristiques générales des systèmes temps réel. Dans le cadre de cette thèse, nous nous intéresserons particulièrement à un système temps réel caractérisé par une configuration de tâches périodiques, préemptables, indépendantes sur une architecture monoprocesseur.

C'est pourquoi, nous avons concentré nos rappels par rapport à ces hypothèses.

Nous avons terminé ce chapitre en introduisant les réseaux de capteurs qui constituent des systèmes temps réel embarqués de plus en plus répandus et dans tous les secteurs. Leur fonctionnement autonome entraine des problèmes liés à leur alimentation énergétique. Indéniablement, pour satisfaire les exigences d'une informatique verte, nous constatons que nous devrons de plus en plus faire appel à l'énergie environementale pour les alimenter.

La conception d'un système embarqué récupérant l'énergie embiante va donc poser un certain nombre de problèmes que jusqu'à maintenant la recherche en informatique temps réel n'avait eu à résoudre. Ces problèmes peuvent se résumer par les questions clé suivantes :

- Comment convertir l'énergie environnementale en énergie électrique ?
- Comment stocker l'énergie électrique pour pouvoir l'utiliser ultérieurement ?
- Comment s'assurer qu'à tout moment, l'énergie électrique disponible est suffisante pour satisfaire au besoin du système embarqué c'est-à-dire exécuter toutes les tâches temps réel dans le respect de leur échéance ?

Dans le chapitre 2, nous allons donc présenter les principales sources d'énergies renouvelables et systèmes de stockage d'énergie. Ce chapitre permettra ainsi de décrire l'état d'avancement de toute la technologie connue sous le nom "Energy Harvesting" et maintenant disponible pour la conception des systèmes embarqués de nouvelle génération.

Chapitre 2

Gestion de l'énergie renouvelable

2.1 Introduction

La plupart des systèmes embarqués sont conçus comme un assemblage de composants communiquant par un ou plusieurs réseaux, alimentés soit, par une batterie primaire (ou rechargeable), par un condensateur (ou supercondensateur) ou plus rarement par le secteur. Traditionnellement, ils sont conçus pour fonctionner en "basse consommation". Cependant, il existe une différence essentielle entre la technologie basse consommation et celle que nous pouvons qualifier de "conduite par la consommation". Selon cette dernière, l'objectif est de faire le meilleur usage de la puissance disponible, ce qui revient pour l'ordonnanceur, à exécuter les tâches temps réel dans des intervalles de temps tel que toutes les contraintes temporelles et énergétiques soient satisfaites.

Dans les systèmes dits à basse consommation d'énergie, nous minimisons massivement la consommation d'énergie avec des technologies innovantes. Celles-ci consistent, soit à diminuer la vitesse de fonctionnement du processeur (méthodes connues sous le nom de DVFS [28] : Dynamic Voltage Frequency Scaling), soit à mettre hors tension l'ensemble ou une partie des circuits électroniques (méthodes connues sous le nom de DPM [47] : Dynamic Power Management). Malgré ces améliorations technologiques, la quantité maximum d'énergie embarquée c'est-à-dire disponible sans nécessiter une recharge de batterie ne peut pas garantir un fonctionnement de l'application sur une durée infinie ni même sur plusieurs dizaines d'années. Il va de soi que tout système embarqué devra mettre en oeuvre ces techniques de récupération d'énergie environnementale pour gagner significativement en longévité.

D'autre part, dans la grande majorité des applications sans fil tels que les réseaux de capteurs, recharger ou remplacer une batterie se révèle délicat voire impossible. Le dimensionnement en termes de volume et de poids est aussi un facteur déterminant dans la conception d'un système. Embarquer une batterie la plus petite possible ou n'en embarquer aucune conduirait à des économies indéniables. L'idée de la technologie "Energy Harvesting" consiste donc à n'embarquer une batterie (ou un supercondensateur) que pour le stockage temporaire de l'énergie récoltée en "temps réel" sur l'environnement et permettant ainsi une durée de vie théoriquement infinie, limitée par la seule durée de vie de cette batterie ou supercondensateur. Reste à prouver toutefois l'applicabilité de cette nouvelle technologie aux systèmes temps réel caractérisés principalement par des impératifs de fonctionnement dans le respect plus ou moins strict de contraintes de temps

de réponse [11].

Les systèmes autonomes que constituent les réseaux de capteurs (WSN : Wireless Sensor Network [1,42]) représentent des utilisateurs privilégiés de la récupération d'énergie renouvelable en vue de les alimenter. En effet, ces derniers sont en général difficilement accessibles, doivent être de taille réduite et doivent avoir une très grande longévité. Chaque noeud de WSN contient ainsi une source d'énergie, un capteur, une unité de transmission et un contrôleur (cf. figure 1.7). Chaque noeud est considéré comme un système autonome s'il peut récupérer l'énergie de son environnement. Les systèmes autonomes doivent contenir les composants nécessaires pour transférer l'énergie environnementale vers une unité de stockage d'énergie. Un des grands défis actuels se porte sur la manipulation de signaux électriques de faibles amplitudes qui varient de quelques dizaines à quelques centaines de millivolts en fonction de l'amplitude de l'excitation de la source. Les énergies associées sont très faibles, de quelques centaines de nanowatts à quelques microwatts.

Dans ce chapitre, nous allons rappeler les principales sources d'énergie renouvelables présentes dans l'environnement, et les microgénérateurs d'énergie disponibles actuellement. Ensuite nous présenterons les diverses technologies liées au stockage de l'énergie.

2.2 Production de l'énergie renouvelable

Les différents types de sources d'énergie sont disponibles dans l'environnement : photoélectrique, mécanique, vibratoire, thermique, et des rayonnements électromagnétiques, etc. Les sources d'énergie renouvelables présentent certains avantages qui apparaissent aujourd'hui de plus en plus déterminants, en particulier leur caractère inépuisable et leur faible impact sur l'environnement par rapport aux sources d'énergie traditionnelles telle que fossiles. Dans la section suivante, nous ne décrirons que les sources d'énergie renouvelables les plus couramment rencontrées. Nous allons aussi comparer ces différentes sources d'énergie afin de déterminer les plus intéressantes à employer dans les futurs microsystèmes autonomes.

2.2.1 Sources d'énergie photoélectrique

Une cellule photovoltaïque, basée sur une jonction P-N en silicium a le fonctionnement suivant : quand elle est exposée à la lumière, la jonction libère une paire électron-trou au travers du circuit électrique associé. Le champ électrique dirige l'électron vers la région N tandis que le trou se dirige vers la région P. Un courant est créé si l'électron ne se recombine pas au travers de la jonction.

- Soleil

Le rayonnement solaire constitue l'origine principale des énergies renouvelables dont font partie aussi l'énergie éolienne, l'énergie océanique, l'énergie biologique, etc. Depuis l'antiquité, l'humain sait profiter de la puissance du soleil. Mais c'est seulement après le déclenchement d'un conflit armé au Moyen-Orient en 1973, et avec l'émergence de la crise du pétrole à cette même période que beaucoup de pays ont démarré des projets de recherche et de développement portant

sur l'énergie solaire.

À l'aide de matériaux semi-conducteurs, en particulier le silicium, il devient possible de réaliser des dispositifs, nommés cellules photovoltaïques, qui transforment le rayonnement solaire en électricité et sont donc des convertisseurs d'énergie solaire en énergie électrique. En disposant une cellule photovoltaïque face au soleil, une tension électrique apparaît. Actuellement, les meilleures cellules photovoltaïques ont un rendement d'environ 15%. Cela signifie donc que 85% de l'énergie qui arrive sur la surface de la photopile n'est pas transformée en électricité. Même si cela représente une déperdition importante, l'énergie récupérée à partir de la lumière reste toutefois accessible à un coût raisonable.

En 2007, la puissance mondiale installée s'élève à $9400MW$ selon les chiffres de l'agence internationale de l'énergie. Et ce volume total a augmenté de 40% par rapport à celui de 2006 et ne cesse d'augnmenter chaque année ($13500MW$ pour 2008). La figure 2.1 nous montre les puissances installées des principaux pays en 2007. Sur cette figure, nous noterons que l'Allemagne et le Japon ont représenté 60% du total. A noter également que les technologies innovantes relatives à la capture de l'énergie solaire émanent trés majoritairement d'entreprises issues de ces deux pays.

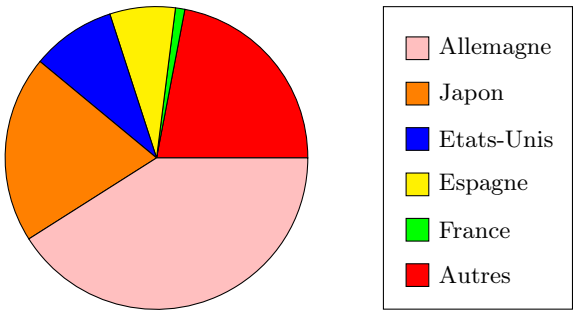


Figure 2.1: – Les puissances installées des pays principaux en 2007

Bien que l'énergie solaire offre des qualités indéniables telles qu'universelle, infinie, propre et durable, elle présente cependant des insuffisances : le rendement est bas, la production énergétique est instable et varie en fonction des rythmes saisonniers et de la localisation, le coût d'installation est élevé. Même si l'énergie solaire ne peut pas remplacer le pétrole dans le future proche et dans des secteurs d'applications tel que l'automobile, l'avionique, au contraire certains produits en profitent déjà. Voir la figure 2.2.

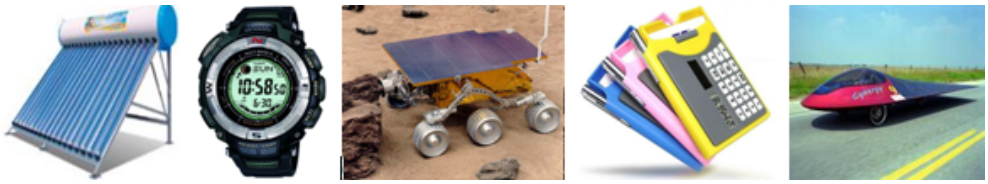


Figure 2.2: – Produits profitant de l'énergie solaire

Le silicium est devenu le principal matériau pour fabriquer les cellules photovoltaïques. Une cellule photovoltaïque, exposée au soleil, présente une densité maximale de $100mW/cm^2$. Pour alimenter les petits systèmes (calculatrice, montre, PDA, etc.), il est suffisant d'utiliser une ou plusieurs cellules photovoltaïques. Si nous voulons construire une centrale électrique, une matrice de panneaux photovoltaïques deviendra nécessaire pour fournir une puissance de l'ordre de plusieurs kilowatts.

- Source lumineuse en environnement intérieur

La puissance lumineuse disponible en environnement intérieur est bien plus faible qu'en environnement extérieur. Il existe deux sortes de sources de lumière. Les sources primaires et les sources secondaires. Les sources primaires sont les sources de lumière qui produisent et émettent leur propre lumière. Ce sont par exemple le soleil, les étoiles, les lampes, le feu, la bougie, etc. Les sources lumineuses secondaires diffusent, dans toutes les directions, une partie de la lumière qu'elles reçoivent. Tous les corps éclairés sont donc sources secondaires, par exemple, la lune, les planètes, l'écran de cinéma, etc.

La source lumineuse en environnement intérieur peut être une source artificielle si nous nous situons loin d'une fenêtre. Elle peut s'agir d'une lampe à incandescence ou une lampe fluorescente. L'énergie est mesurée de manière différente suivant que nous nous trouvons sous un rayonnement solaire ou sous un éclairage artificiel. Généralement, le rayonnement solaire est mesuré en prenant la valeur de l'irradiance exprimée en W/m^2 ou $\mu W/cm^2$. Mais la puissance lumineuse incidente d'une source artificielle est quant à elle mesurée en prenant la valeur de l'éclairement E exprimé en lm/m^2 (ou en lux) et ne prend en compte que le spectre du visible.

$$E = A \int I(\lambda) y(\lambda) d\lambda \tag{2.1}$$

où $A = 683.002\ lm/W$, $y(\lambda)$ est une fonction décrivant la sensibilité spectrale de l'oeil humain, et $I(\lambda)$ est l'irradiance du visible exprimée en W/m^2.

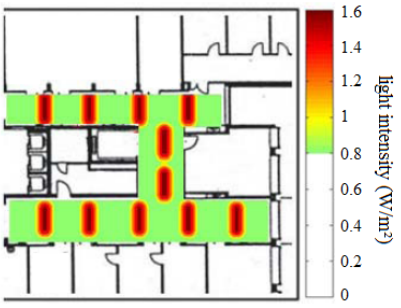


Figure 2.3: – Modélisation de l'énergie rayonnée dans un local [41]

Un modèle permettant d'estimer l'énergie disponible en intérieur en prenant en compte toutes les sources lumineuses présentes a été développé par Randall [41]. De plus, des mesures ont été prises dans un couloir pour avoir un ordre de grandeur de la puissance récupérable (cf. figure

2.3). Une interpolation a permis d'obtenir la carte d'éclairement du couloir. Nous y voyons bien les îlots dus aux sources fluorescentes qui s'y trouvent.

Ces mesures sont données à titre indicatif car il faut savoir qu'un bureau est en moyenne quatre fois plus éclairé qu'un couloir. Nous pouvons ainsi définir la quantité d'énergie disponible suivant la localisation de la cellule solaire (cf. tableau 2.1).

Environnement	Intérieur				Extérieur	
	éclairage tamisé	éclairage de couloir	éclairage de travail	éclairage par incandescence	temps couvert	très ensoleillé
E(lux)	80	150	740	1200	3030	100000
I($\mu W/cm^2$)	67	125	616.7	1000	2525	100000

Tableau 2.1: – Energie disponible pour la récupération photovoltaïque [53]

Prenons l'hypothèse pessimiste suivante, où le capteur photovoltaïque est positionné au nord et soumis à un temps pluvieux d'hiver pendant 6 heures par jour, ce qui représente une densité de puissance solaire disponible de 20 W/m^2. En théorie, une surface de 1 cm^2 de cellule solaire avec un rendement de 12% permettrait de récupérer 5.2 Joules pendant cette journée. Sachant qu'un système autonome actuel nécessite une énergie de l'ordre de 4 Joules pour mesurer et transmettre cette donnée, nous remarquons qu'il est possible d'autoalimenter le capteur autonome quelles que soient les conditions de l'environnement [53].

2.2.2 Sources d'énergie mécanique

Nous générons continuellement de l'énergie mécanique, que ce soit par l'intermédiaire de nos objets ou que ce soit directement par nos mouvements.

Il existe à ce jour trois moyens de convertir l'énergie mécanique :

– par conversion électromagnétique,
– par conversion piézoélectrique,
– par conversion électrostatique.

- Générateurs électromagnétiques

L'induction électromagnétique, découverte par Faraday en 1831, consiste en la génération d'un courant électrique dans un conducteur placé dans un champ magnétique. Dans la plupart des cas, le conducteur est sous la forme d'une bobine et l'électricité est générée par le mouvement d'un aimant dans la bobine grâce à la variation du flux du champ magnétique (cf. figure 2.4). Le courant ainsi généré dépend de l'intensité du flux du champ magnétique, de la rapidité du déplacement de l'aimant et du nombre de tours de la bobine. Nous allons introduire ici deux exemples relatifs au générateur électromagnétique.

Un transducteur d'énergie avec miro résonateur (cf. figure 2.5) a été proposé par J.M.H. Lee et *al*. dans [22]. Ce transducteur est conçu sous la forme de pile AA, qui transforme l'énergie

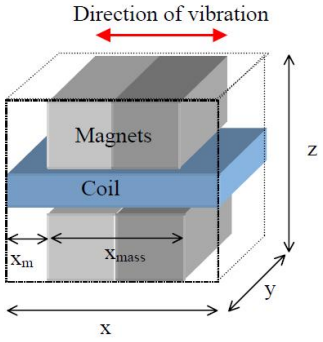


Figure 2.4: – Schéma du générateur électromagnétique [35]

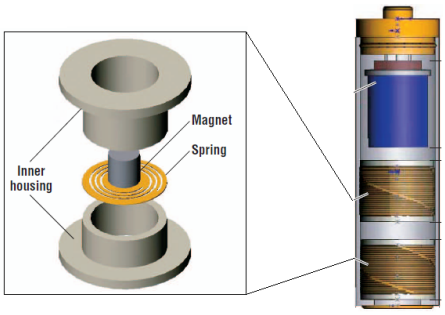


Figure 2.5: – Architecture interne d'un transducteur d'énergie [22]

mécanique causée par la vibration mécanique (fréquence de $100Hz$) en énergie électrique par la loi de Faraday. Les applications potentielles concernent surtout l'alimentation des systèmes de transmission sans fil (IR et RF).

L'architecture de la figure 2.6 a été proposée par les chercheurs de l'université de Southampton [17]. Ce générateur s'applique également dans le champ de vibration, est capable de convertir l'énergie de vibration en énergie électrique pour alimenter les systèmes intelligents de capteurs. Ce dispositif est aussi particulièrement utilisé pour l'alimentation électrique d'un microsystème en zone inaccessible. Les résultats expérimentaux indiquent que la valeur de crête de sortie est $3.9mW$, et la valeur de sortie moyenne est $157uW$.

- Générateurs piézoélectriques

La piézoélectricité, est une propriété particulière des corps possédant une anisotropie cristalline. La piézoélectricité se définit par la propriété que possèdent certains corps à se polariser électriquement sous l'action d'une force mécanique (effet direct) et, réciproquement, de se déformer lorsque nous leur appliquons un champ électrique (effet inverse). La découverte de cette propriété a était faite par Pierre Curie bien qu'il semble que le premier à avoir observé ce phénomène soit l'abbé René Just Haüy (1743-1822) [3].

Certains cristaux soumis à une contrainte mécanique se polarisent. Cette polarité est proportionnelle à la contrainte subie. Inversement, ces matériaux se déforment s'ils sont soumis

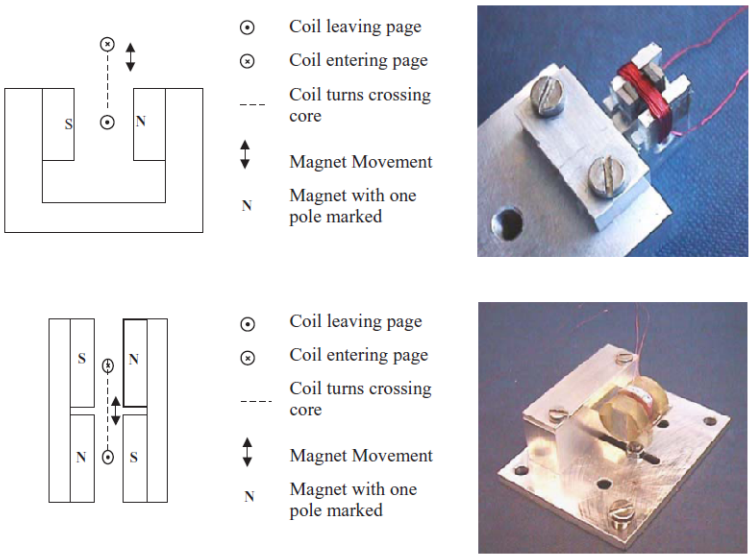


Figure 2.6: – Architectures de générateurs miniatures [17]

à un champ électrique. Il existe un grand nombre de matériaux piézoélectriques, incluant des cristaux simples comme le quartz, les piézocéramiques comme le plomb zirconium de titane (PZT), des films fins de nitrure d'aluminium (AlN) ou d'oxyde de zinc, des films épais obtenus à l'aide de poudres de piézocéramiques ou des matériaux polymères tel que le polyvinylidenefluoride (PVDF). Les matériaux piézoélectriques ont des propriétés anisotropes qui diffèrent selon la direction et l'orientation des forces et de la polarisation. Malheureusement les propriétés piézoélectriques évoluent avec le vieillissement du matériau, les stimuli et la température. Ces changements sont dépendants du mode de fabrication et du type de matériau. La température est un facteur limitant car si nous dépassons le point de Curie, le matériau se dépolarise et perd toutes ses propriétés piézoélectriques.

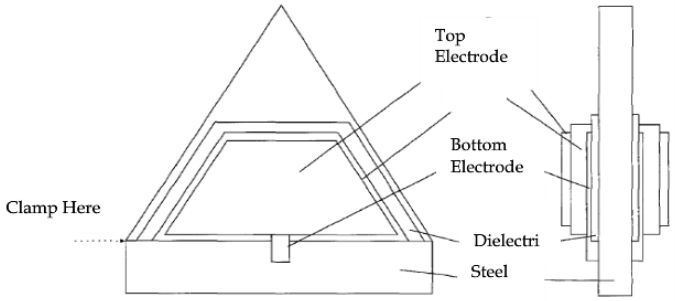


Figure 2.7: – Architecture d'un micro générateur piézoélectrique sur un cantilever [54]

En 2001, White et al. ont proposé une simple architecture d'un micro générateur piézoélectrique sur un cantilever dans [54], comme illustré sur la figure 2.7. Les auteurs ont effectué les travaux expérimentaux sur un matériau piézoélectrique de type $PZH\ 5H$. Ils ont mis ce géné-

rateur dans un champ résonant, ont trouvé qu'il peut donner environ $2uW$, quand la fréquence de vibration est fixée à $80Hz$ et l'amplitude de déplacement à $0.9mm$.

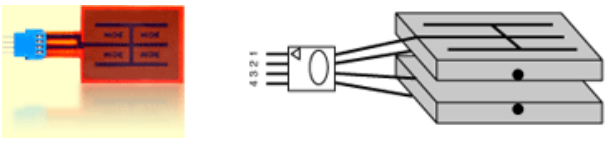


Figure 2.8: – Capteur piézoélectrique de type QuickPack-QP20w [36]

Geffrey K. Ottman et son équipe ont réalisé un système optimisé de récupération d'énergie à l'aide d'un capteur piézoélectrique du commerce(cf. figure 2.8), qui récolte de l'énergie à partir d'une source de vibration et recharge simultanément une batterie [36]. D'ailleurs, un convertisseur $DC-DC$ est proposé et appliqué dans ce système optimisé, conduisant à un rendement de transfert de courant électrique 4 fois supérieur à celui obtenu sans ce convertisseur $DC-DC$.

- Générateurs électrostatiques

Le générateur électrostatique est essentiellement un condensateur (comme illustré sur la figure 2.9) dont la capacité est variable.

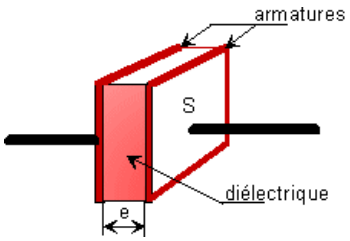


Figure 2.9: – Modèle d'un condensateur

Le modèle d'un condensateur est représenté sur la figure 2.9. Il existe une relation entre C, U et Q, où C désigne la capacité du condensateur exprimée en Farad (F), U désigne la tension aux bornes du condensateur exprimée en Volt (V) et Q représente la charge (ou quantité d'électricité)

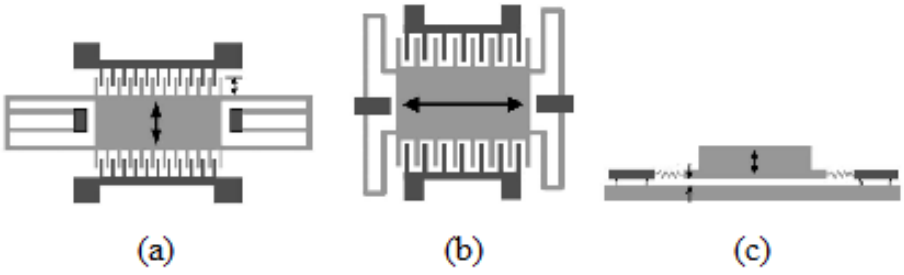


Figure 2.10: – Trois prototypes de générateur électrostatique [44]

exprimée en Coulomb (C). Si la charge du condensateur est constante, la tension va augmenter à mesure que la capacité diminue. Si la tension du condensateur est constante, la charge va diminuer quand la capacité diminue. Dans n'importe quel cas, l'énergie mécanique qui cause la variation du condensateur peut être convertie en énergie électrique. Dans l'article [44], les auteurs énumèrent trois prototypes typiques (cf. figure 2.10), et comparent les caractéristiques techniques (cf. tableau 2.2).

Type	**Avantages**	**Inconvénients**
(a)	Pas de butées mécaniques ; Le plus grand facteur Q.	Mauvaise stabilité ; La capacité est la plus petite.
(b)	Avoir la plus grande capacité.	Exister des butées mécaniques.
(c)	Avoir la plus grande capacité ; Bonne stabilité.	Le plus grand facteur d'amortissement mécanique ; Surface d'adhérence.

Tableau 2.2: – Comparaison de trois prototypes de générateur électrostatique

2.2.3 Sources d'énergie thermique

Le générateur thermoélectrique est associé à l'effet Seebeck qui a été découvert par un physicien allemand Thomas Johann Seebeck en 1821. Si un matériau est soumis à un flux de chaleur généré par une différence de température, alors une différence de potentiel électrique se forme aux extrémités de la jonction (cf. figure 2.11). Cette différence de potentiel est définie par l'équation : $V = S_{AB} \cdot (T_0 - T_1)$, où S_{AB} est le coefficient de Seebeck ($\mu V/K$).

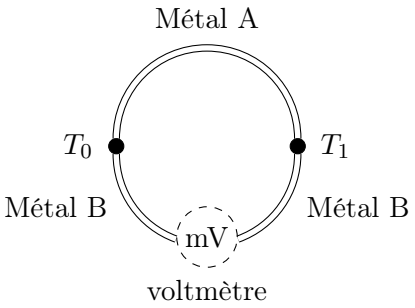


Figure 2.11: – Principe de l'effet Seebeck

Les thermogénérateurs sont constitués d'un assemblage de jonctions, et utilisent l'effet Seebeck pour générer de l'électricité. La figure 2.12 nous illustre l'architecture détaillée d'un générateur thermoélectrique. Les deux conducteurs de nature différente sont reliés par les deux jonctions J_1 et J_2, qui sont mis aux deux champs de températures différentes. Alors dans ces conducteurs, il y a du courant faible engendré, proportionnel à la différence de température.

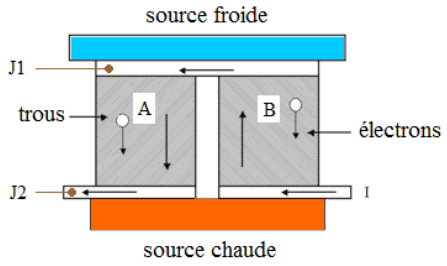


Figure 2.12: – Fonctionnement d'un générateur thermoélectrique

Il en résulte qu'un bon matériau thermoélectrique possède, d'une part, un coefficient Seebeck élevé, et d'autre part une forte conductivité électrique mais une faible conductivité thermique [53].

Nous retrouvons dans le commerce un bon nombre de produits constitués de ce type de matériau. La différence entre les différents produits réside d'une part dans la densité du nombre de jonctions permettant d'augmenter la conversion d'énergie par unité de surface et d'autre part par la conductivité thermique et la conductivité électrique du matériau. Deux applications typiques sont décrites dans les articles [31, 50].

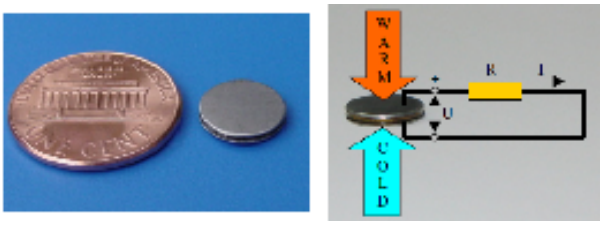


Figure 2.13: – Prototype d'un générateur thermoélectrique "Thermo Life" [50]

Un générateur thermoélectrique appelé "Thermo Life" est proposé par Stark dans [50]. Sa forme est petite et compacte, cf. figure 2.13. Il peut délivrer facilement une puissance de l0 à $100uW$ en fonction de la différence de température (p.ex. $3V$, $30uW$ pour ΔT=5 K). Si nous demandons une haute tension électrique, nous utilisons plusieurs modules identiques connectés en série.

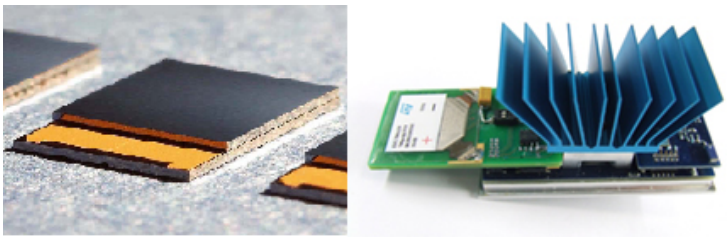


Figure 2.14: – Prototype d'un générateur thermoélectrique "Micropelt" [31]

En outre, pour répondre aux besoins de ce marché, la société allemande Micropelt, fabricant

de composants en couche mince aux propriétés thermoélectriques, vient de présenter un petit générateur capable de transformer une différence de température en énergie électrique [31]. Intégré sur la tête d'une vis $M24$ en acier, ce composant, baptisé $TE - Power - Bolt$ est capable de récupérer $15mW$ avec une différence de température de $10^{\circ}C$ à $20^{\circ}C$ entre le matériau et l'air ambiant.

2.2.4 Source d'énergie éolienne

L'énergie éolienne constitue une des énergies les plus propres. La première utilisation remonte à l'Antiquité. L'énergie éolienne est une énergie mécanique que nous pouvons capturer de plusieurs manières. Simplement avec une voile, l'énergie du vent peut servir à propulser un voilier. Aujourd'hui, le vent est à nouveau exploité par des moulins plus modernes communément appelés éoliennes. Nous commençons à construire des centrales éoliennes dans les régions riches en vent telles que les zones côtières ou les plaines.

Généralement,deux étapes s'avèrent nécessaires pour transformer l'énergie éolienne en l'énergie électrique. D'abord l'énergie cinétique du vent se transforme en énergie mécanique qui se transforme en énergie électrique. Bien que la ressource du vent soit inépuisable et propre, son exploitation est confrontée à des limites d'ordre saisonnier et climatique. La figure 2.15 illustre l'architecture d'une éolienne.

Figure 2.15: – Architecture d'une éolienne

Aujourd'hui, nous rencontrons des éoliennes portables, développées pour faciliter notre vie quotidienne. Ces dispositifs possèdent des propriétés de haut rendement et de basse vitesse de rotation. Nous pouvons penser que les microéoliennes spécifiques à l'alimentation de microsystèmes vont largement se développer dans un très proche avenir dans le domaine des systèmes autonomes communicants.

2.2.5 Sources d'énergie dans l'environnement humain

Les ressources ambiantes, notamment celles présentes dans l'environnement humain, ne sont pas nécessairement renouvelables mais constituent des ressources "quasi-gratuites". Ces ressources s'avèrent limitées mais toutefois suffisantes pour certaines catégories d'applications. Des applications ne demandant que très peu d'énergie pour fonctionner ont déjà vu le jour et le marché se trouve en pleine émergence. Alors que le développement des applications mobiles sans

fil connaît un essor remarquable, un défi, apparu dans les années 90, consiste à pouvoir alimenter ces materiels portables en exploitant les ressources énergétiques présentes sur l'homme.

- Sources mécaniques

Les puissances dissipées par le corps humain sont liées aux puissances consommées par le corps pour réaliser une action mécanique donnée que multiplie le rendement de conversion métabolique. En effet, qu'il soit au repos ou durant une activité quelconque, le corps humain produit de l'énergie, exprimée en calories par heure. Starner, dans [51], a répertorié la puissance calorifique consommée pour différentes activités comme dormir, être debout, jouer du piano, nager, etc. A titre d'exemple, la puissance calorifique consommée est d'environ $160W$ lorsque nous conduisons une voiture, et de $580W$ lorsque nous nageons. Pour la production de travail mécanique, le rendement énergétique humain n'est pas unitaire, c'est-à-dire qu'il s'accompagne de pertes se traduisant par un dégagement de chaleur. Le rendement de conversion correspondant, évalué à 25% [51], nous permet donc d'estimer la puissance récupérable. En reprenant les exemples précédents, pendant la conduite d'une voiture, l'énergie récupérable sera de $40W$, et de $150W$ pour la nage.

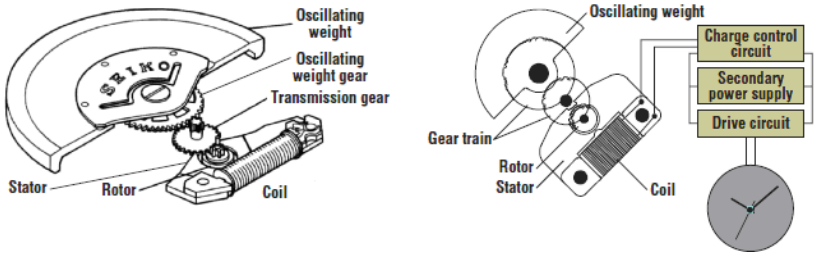


Figure 2.16: – Architecture interne de la montre Seiko AGS [37]

L'AGS (Automatic Generating System) de Seiko correspond à une série de montres autoalimentées (cf. figure 2.16) par le mouvement du bras. Lorsque la montre est portée normalement au poignet, le générateur produit en moyenne $5\mu W$. Quand nous secouons fortement la montre,le générateur peut produire $1mW$ [37].

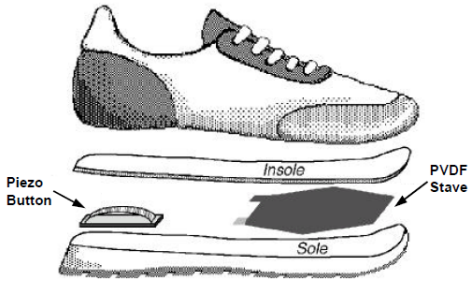


Figure 2.17: – Capteurs piézoélectriques embarqués dans une chaussure [46]

Au cours de la dernière décenniees, les chercheurs du MIT (Massachusettes Institute of Techno-

logy) ont effectué de grandes recherches sur l'exploitation de l'énergie "parasitaire" de chaussure. Plusieurs méthodes sont déjà proposés pour récupérer cette énergie, soit en utilisant le principe électronique, soit en utilisant le principe piézoélectrique [20, 46]. La figure 2.17 illustre les deux méthodes. Les deux capteurs piézoélectriques, un film de PVDF et un PZT unimorphe, sont embarqués séparément dans la selle et le talon de la chaussure. Bien que les deux capteurs soient différents, tous les deux profitent de l'effet piézoélectrique causé par la force entre le pied et la terre au cours de la marche.

- Sources thermiques

Si nous prenons l'exemple du corps humain, celui-ci nécessite, pour son bon fonctionnement, une température interne constante. Ceci oblige le corps humain à réguler sa température interne quelles que soient les conditions extérieures en dissipant ainsi énormément d'énergie (cf. figure 2.18).

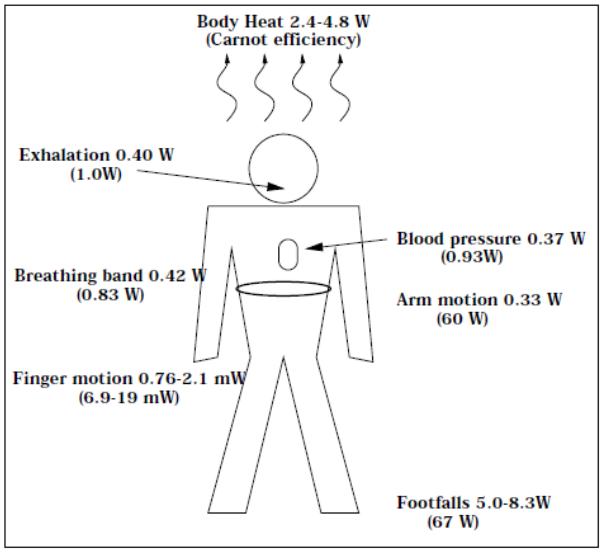


Figure 2.18: – Energie dissipée par le corps humain [52]

La temperature du corps humain est regulée aux environs de $T_b = 37^{\circ}C$. Ainsi, si la temperature de l'air ambiant est plus faible, par exemple $T_a = 20^{\circ}C$, un flux de chaleur, exprime en $8W/(m^2 \cdot {}^{\circ}C)$, se crée de la source chaude vers la source froide tel que : $Ps = h(T_b - T_a)$.

Le coefficient h est un coefficient d'échange thermique estimé à $8W/(m^2 \cdot {}^{\circ}C)$ au repos. En considerant un écart de temperature entre le corps et l'air ambiant de $17^{\circ}C$, nous en déduisons approximativement que la peau nue dissipe environ $140W/(m^2)$ soit $14mW/(cm^2)$. Cependant, les vêtements limitent la puissance dissipee en reduisant sensiblement l'écart de temperature entre la peau et l'intérieur des vêtements et ainsi le coefficient d'échange équivalent avec l'extérieur. Quoi qu'il en soit, pour les zones ou la peau est à l'air libre, les échanges thermiques dépendent directement de la temperature exterieure.

La première application concerne une montre de marque Seiko. A l'intérieur de la montre

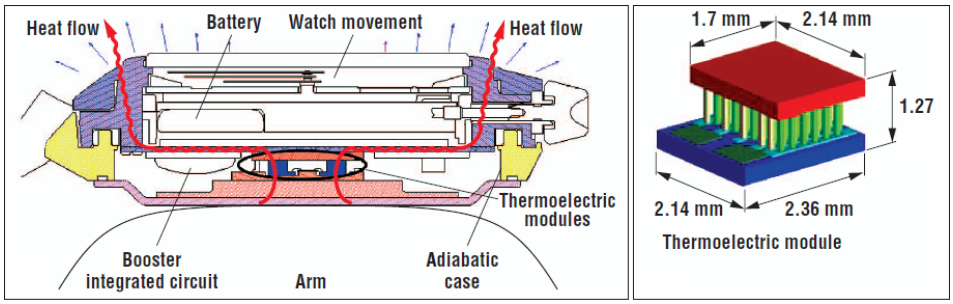


Figure 2.19: – (a). Profil de montre (b). Diagramme 3D de module thermoélectrique [37]

(cf. figure 2.19), nous embarquons un générateur thermoélectrique qui profite de la différence de température entre le corps et l'environnement pour convertir l'énergie thermique en énergie électrique servant ainsi à alimenter le micromoteur de la montre.

2.2.6 Sources d'énergie renouvelable : synthèse

Il existe un grand nombre de sources d'énergie renouvelables qui restent encore très peu exploitées. Le faible impact sur l'environnement de leur exploitation devrait nous inciter à les développer. Outre les sources d'énergie énumérées précédemment, il en existe encore d'autres qui toutefois ne conviennent peut-être pas toutes aux microsystèmes autonomes. C'est notamment le cas de l'énergie géothermique, l'énergie hydraulique, l'énergie marémotrice, etc.

Le tableau 2.20 permet de comparer les performances des différentes sources de récupération d'énergie pour les microsystèmes autonomes. Il faut être attentif au fait que les sources d'énergie ne délivrent pas une puissance linéaire avec la variation de leurs dimensions. L'analyse des rendements montre que les systèmes de récupération vibratoire sont les plus performants, mais ils sont également les plus sélectifs sur les caractéristiques de leurs stimuli. Ainsi, une vibration dont la fréquence et l'amplitude évoluent sur une large plage ne sera que très difficilement convertie en énergie électrique. Par contre, les systèmes photovoltaïques et thermiques affichent les meilleures densités de puissance, et malgré leur rendement plus faible, ils permettent de récupérer une quantité bien plus importante d'énergie électrique [53].

2.3 Exemples de prototypes

2.3.1 Smart Dust

L'université de Berkeley fait partie des premières universités à avoir travaillé sur ce domaine avec notamment un projet nommé "Smart Dust" (cf. figure 2.21). Le but était de développer un système autonome millimétrique capable de communiquer et de faire partie intégrante d'un réseau de capteurs distribués [39].

Ils ont réalisé un demonstrateur autonome de 138 mm^3 capable de mesurer des variations d'accélération, ainsi que des variations de lumière incidente. Le systeme possède pour cela deux capteurs, un capteur capacitif pour l'accélération et une photodiode. Les données sont ensuite

Source d'énergie	Dimensions, notes	Densité de puissance (1)	Excitation	Rendement
Gradient de T°C :				
Thermoélectrique : Thermo Life™	0.2 cm^2	60 µW.cm^{-2}	5°C	12%
Micromoteur : Stirling	5 mm*15 mm	225 mW.cm^{-2}	NC	NC
Vibrations & Chocs				
Piézo-électrique :	800 µm *1200 µm 1 mm^3, poutre MEMS	40 µW.cm^{-2}	39 ms^{-2}, 1.3 kHz	NC
Electrostatique :	104 g	333 µw.cm^{-2}	50 Hz	60%
Electromagnétique :	0.15 cm^3	307 µW.cm^{-2}	0.59 ms^{-2}	30%
Champs électromagnétiques				
Télé alimentation :	15 tours, 46*46 mm^2	142 µW.cm^{-2}	13.56 MHz	25%
Déplacement de gaz				
Microturbine		1 mW.cm^{-2}	30 $l.min^{-1}$	NC
Soleil & Lumière				
Photovoltaïque : Extérieur	très ensoleillé	10-20 mW.cm^{-2}	Soleil	10-20%
Intérieur	fenêtre à proximité	1-2 mW.cm^{-2}	Soleil	10-20%
Intérieur, lumière artificielle	1000 lux, source fluo - source halogène	10-60 µW.cm^{-2}	Sans soleil	10%
Photosynthèse Biopile	Implanté dans un cactus	9 µW.cm^{-2}	Halogène	NC

Figure 2.20: – Comparaison des différents systèmes de récupération d'énergie [53]

transmises par un canal optique. Le système consomme 75 μW et a été testé avec deux sources différentes, une batterie $Mn - Ti - Li$ et une cellule solaire de 2 mm^2 de surface.

Le but principal de ces études était de permettre des avancées significatives dans le domaine des systèmes autoalimentés par l'élaboration de sources récupératrices d'énergie de densités de puissance supérieures à 100 $\mu W/cm^3$.

Ces travaux ont été poursuivis en 2008 avec la présentation du système "Picocube" (cf. figure 2.22), un système communiquant de récupération vibratoire tridimensionnel dont le récupérateur piézoélectrique est externe dû à sa taille importante. Un nouveau transmetteur a été développé ainsi qu'un système de gestion d'énergie. Le circuit de gestion d'énergie presente un rendement global de 74% pour une puissance de 450 μW [8].

2.3.2 Heliomote

Le système Heliomote a été développé par le laboratoire "Networked and Embedded Systems Laboratory" de UCLA. Il se compose principalement d'un panneau solaire et de deux accumulateurs Ni-MH. La tension à puissance maximum du panneau solaire est égale à 3 volts. La tension des accumulateurs varie entre 2,2 et 2,8 volts. Le panneau solaire est connecté à deux accumulateurs au moyen d'une diode en série qui empêche la décharge éventuelle des accumulateurs dans le panneau (cf. figure 2.23). La diode présente une chute de tension de 0,7 volts, ce qui fixe la tension aux bornes du panneau solaire au voisinage du point de fonctionnement à puissance

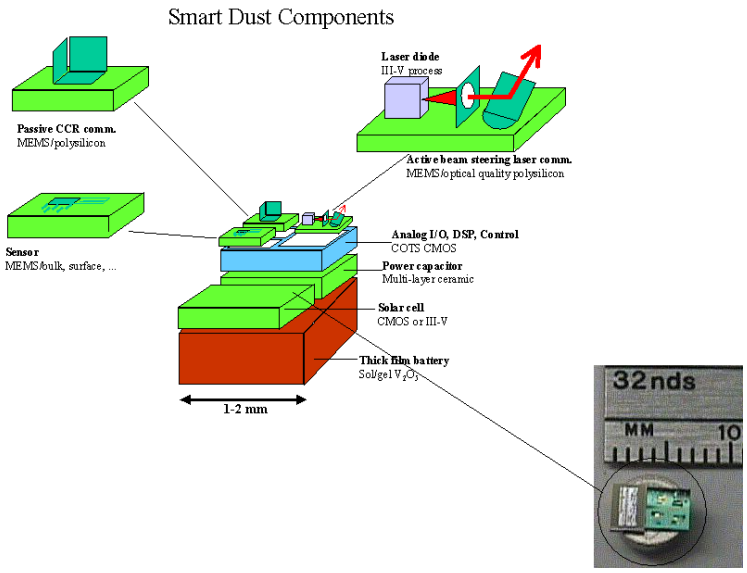


Figure 2.21: – Système autonome SMART DUST [http ://robotics.eecs.berkeley.edu/]

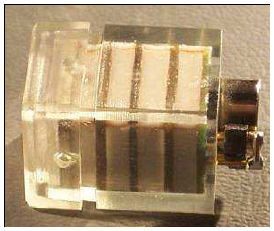


Figure 2.22: – Le système Picocube [8]

maximum.

Si la tension des panneaux reste inférieure à la tension des accumulateurs additionnée à la tension de la diode polarisée en directe, les accumulateurs ne peuvent pas se charger, les réseaux de capteurs sans fil ne sont alors alimentés que par les accumulateurs. Les accumulateurs ont un nombre limité de cycles de charge et décharge. La durée de vie des réseaux de capteurs sans fil est donc affectée par celle des accumulateurs [26, 40].

Figure 2.23: – Le système Heliomote

2.3.3 Prometheus

Le système Prometheus, développé par UC Berkeley se compose d'un accumulateur Li-Polymère rechargeable, de deux supercondensateurs de 2,5 volts, 22 farads et d'un panneau solaire. L'ensemble alimente un réseau de capteurs sans fil qui s'appelle Telos. Les supercondensateurs sont utilisés comme un premier tampon pour stocker l'énergie. La figure 2.24 présente la vue d'ensemble du système Prometheus [19].

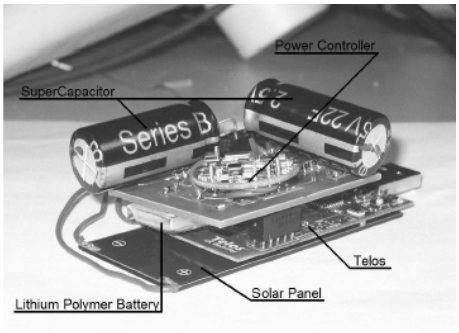


Figure 2.24: – Le système Prometheus [19]

Cependant, les supercondensateurs ont une capacité faible et leur tension chute quand ils se déchargent pour alimenter le circuit. Ils ne permettent donc pas d'assurer l'alimentation du réseau de capteur sans fil dans la durée. Par conséquent, l'accumulateur est ajouté car il a une densité d'énergie plus élevée et une tension moins variable.

2.3.4 WATS : Wireless Autonomous Transducer Solutions

L'IMEC (Interuniversity Microelectronics Center) travaille sur un projet nommé "WATS" pour Wireless Autonomous Transducer Solutions. Le but de ce projet est de développer un microsystème autonome, basé sur la récupération de l'énergie et la conception de circuits à très faible consommation. Les principaux sujets de recherche sont : la transmission sans fil avec le développement d'un transmetteur faible puissance, et la récupération d'énergie thermique avec le développement d'un circuit de gestion d'énergie.

Figure 2.25: – Exemples de systèmes développés par L'IMEC [38]

Le système micro énergétique est basé sur l'utilisation d'un générateur thermoélectrique. Une électronique de conversion adaptée a été conçue avec un transmetteur très faible puissance montée

sur substrat flexible (cf. figure 2.25). L'énergie générée par le circuit est de 0.1 mW avec une tension supérieure à $1V$. Ils indiquent que cette puissance est suffisante pour charger une petite pile bouton et pour transférer la valeur de la température du corps (toutes les deux secondes) à une station de réception proche. Pour cela, un module a été developpé, basé sur les technologies MEMS. L'application d'une variation de temperature de $8^{\circ}C$ (correspondant à la différence entre une temperature ambiante de $22^{\circ}C$ et une temperature de peau de $30^{\circ}C$) permet finalement d'obtenir une variation de $5^{\circ}C$ aux bornes du générateur thermoélectrique, délivrant ainsi une puissance de l'ordre de 100 μW.

2.4 Les réservoirs d'énergie

2.4.1 Introduction

Nous entendons par réservoir d'énergie, l'équipement permettant de stocker temporairement de l'énergie électrique. L'électricité est une forme d'énergie dite secondaire en ce sens qu'une fois crée, elle devient plus difficile à stocker par rapport aux autres formes d'énergie. Ainsi, l'électricité produite est soit instantanément consommée, soit perdue, ou stockée dans un condensateur. Afin de pouvoir exploiter au mieux les ressources énergétiques et assurer des pertes minimum, il s'avère incontournable de stocker l'énergie drainée de l'environnement. Stocker l'énergie est un défi technique fondamental car cela implique souvent un processus compliqué, qui consomme lui-même de l'énergie.

Le cycle de fonctionnement d'un réservoir d'énergie comprend deux opérations : le stockage et la restitution. Dans un tel cycle, l'énergie se convertit deux fois. Nous devons donc apporter une attention particulière au rendement du système de stockage, qui dépend énormément de la nature du stockage et des systèmes physiques mis en oeuvre.

Les matériaux de stockage d'énergie se classent en six catégories selon ses mécanismes de fonctionnement (cf. figure 2.26). Avec la découverte de nouveaux matériaux et l'avancement des technologies de fabrication, de nouveaux procédés de stockage d'énergie voient régulièrement le jour.

2.4.2 Batteries rechargeables

Les batteries rechargeables (batteries secondaires) jouent un très important rôle dans les applications industrielles et civiles. Selon les données reportées dans la littérature [4, 12, 21, 45], nous pouvons faire un comparatif des six types différents de batteries rechargeables(cf. tableau 2.3). Ces données sont très générales et la performance précise d'une batterie dépend de sa taille et des conditions de service.

Outre les six types énoncés ci-dessous, il existe également d'autres types de batteries rechargeables disponibles sur le marché, incluant la batterie NaS et la batterie Reusable Alkaline.

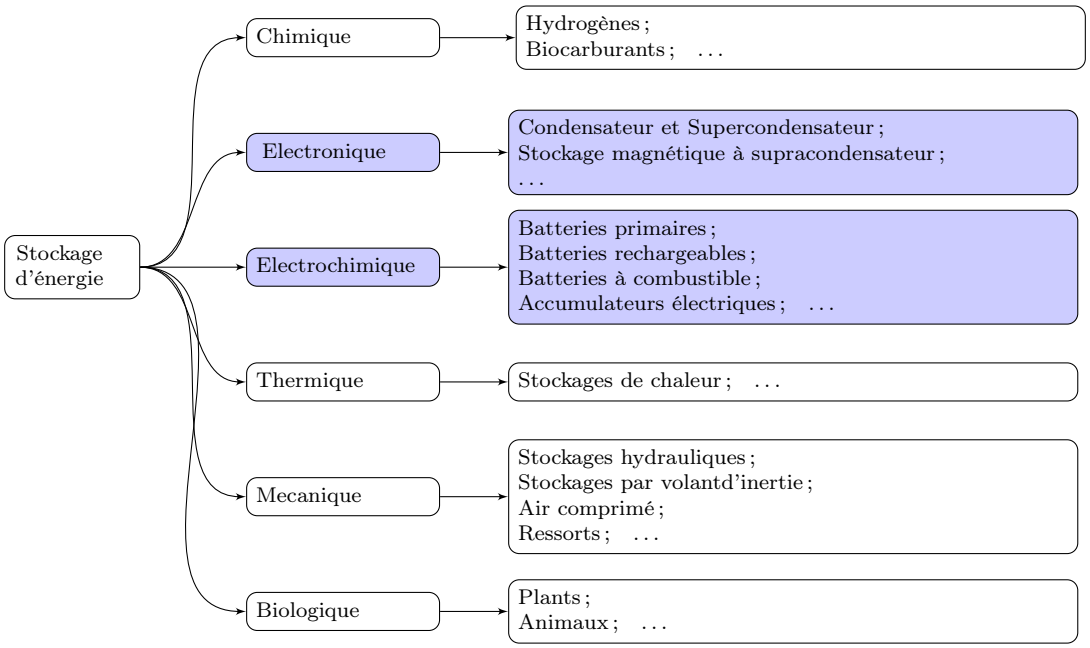


Figure 2.26: – Les grandes familles de stockage d'énergie

2.4.3 Supercondensateurs

Les batteries sont caractérisées par une forte densité énergétique et une faible densité de puissance. Leur durée de vie en nombre de cycles charge/décharge est relativement limitée. Le développement technologique et la maîtrise de fabrication de nouveaux matériaux ont permis la réalisation d'autres nouveaux types de stockages d'énergie électrique comme les supercondensateurs. Ceux-ci peuvent s'utiliser de façon complémentaire aux batteries ou à la pile à combustible. Une complémentarité en termes de puissance instantanée disponible et de quantité d'énergie stockée permettra d'augmenter les performances des systèmes d'alimentation. En fait, le supercondensateur se définit comme un condensateur électrochimique aussi, mais renforcé par une architecture à deux couches, cf. la figure 2.27.

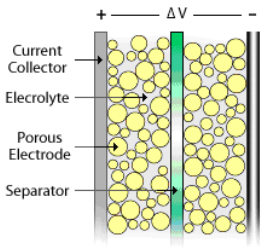


Figure 2.27: – Description schématique d'un supercondensateur

Par rapport à une batterie rechargeable, le supercondensateur possède les qualités suivantes :
- Le processus de charge/décharge est rapide ;
- Il peut délivrer des impulsions fréquentes d'énergie sans l'effet d'endommagement ;

Type	**Tension nominale** (V)	**Densité d'énergie** $Wh/kg - Wh/l$	**Prix**	**Commentaire**
Pb-acid	2	35-70	Petit	Utilisée dans les véhicules, les systèmes d'alarme et les systèmes d'alimentation sans interruption.
Ni-Ca	1.2	40-100	Petit	Utilisée dans les applications domestiques, sa vie est longue (>1500 cycles), avoir l'effet mémoire, va être remplacée par Li-ion and Ni-MH.
Ni-MH	1.2	90-245	Petit	était utilisée dans les ordinateurs portables, sa vie est plus coute que Ni-Ca, mais plus chère, de plus en plus remplacée par Li-ion.
Ag-Zn	1.5	110-220		Utilisée dans les applications militaires, plus sécurisée que Li-ion, mais sa vie est très courte, bonne perspective d'avenir.
Li-ion	3.6	125-440	Moyen	Très populaire, 1^{er} choix pour les produits digitaux, pas d'effet mémoire.
Li-ion polymer	3.1	400-800	Grand	Peut être formée en film, meilleure performance que Li-ion, chargée rapidement, 2^{eme} génération de Li-ion, va remplacer Li-ion traditionnelle.

Tableau 2.3: – Comparaisons des batteries rechargeables

- Sa vie est plus longue : il peut se charger/décharger jusqu'à des milliers fois ;
- La résistance interne est petite : son efficacité peut atteindre 84-95% ;
- Il peut être chargé à n'importe quel niveau, sans effet mémoire ;
- La bande de température de fonctionnement est large, min.-40°C ;
- Il n'y a pas de limitation chimique à l'intérieur.

Cependant, il n'est pas parfait car il a une forte autodécharge et la tension sortie varie selon le niveau d'énergie stockée. En outre, la taille d'un supercondensateur dépasse celle d'une batterie rechargeable pour stocker la même quantité d'énergie.

2.4.4 La pile à combustible

La pile à combustible se définit comme une batterie qui transforme directement l'énergie d'une réaction chimique en énergie électrique de façon continue. Elle fonctionne à l'inverse de l'électrolyse de l'eau. Il s'agit plutôt d'un générateur d'électricité que d'un stockeur d'électricité, qui de plus possède un très haut rendement énergétique.

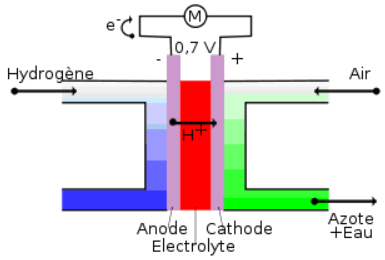


Figure 2.28: – Schéma d'une pile à combustible

Comme illustré sur la figure 2.28, le fonctionnement d'une pile à combustible est particulièrement propre puisqu'il ne produit que de l'eau et consomme uniquement des gaz. Les taux d'émission théorique en CO_2 et autres polluants nuisibles à l'environnement et à la santé sont nuls.

2.4.5 Stockage magnétique à supraconducteur

Le principe du stockage magnétique à supraconducteur consiste simplement à faire passer un courant électrique dans une bobine supraconductrice. Le courant circule sans perte dans le supraconducteur, et une certaine quantité d'énergie est stockée pour une durée théoriquement infinie.

Pour récupérer l'énergie, il suffit d'ouvrir le circuit et le brancher sur un récepteur électrique à alimenter. Le retard de la charge et la décharge est très court. La puissance est disponible presque instantanément avec un rendement de puissance très élevé sur une brève période. La résistance du supraconducteur étant nulle, les pertes du stockage magnétique à supraconducteur sont très faibles. Ce procédé se montre donc très efficace par rapport aux autres techniques de stockage d'énergie. Globalement, ses avantages se résument dans les points suivants :

- Charges/décharges très rapides ;
- Longue durée de vie ;
- Pas d'électrolyte liquide corrosif ;
- Etat de charge vérifiable très facilement (via le courant dans la bobine).

2.4.6 Stockage d'énergie : synthèse

Ci-dessus, nous avons introduit les différents procédés de stockage d'énergie. Nous vous présentons le diagramme de Ragone (cf. figure 2.29) très largement répandu pour comparer les performances de ces procédés.

Une autre comparaison (cf. tableau 2.4) peut être faite entre les batteries Li-ion, Li-ion polymer et un supercondensateur et ce, sur la base de six paramètres.

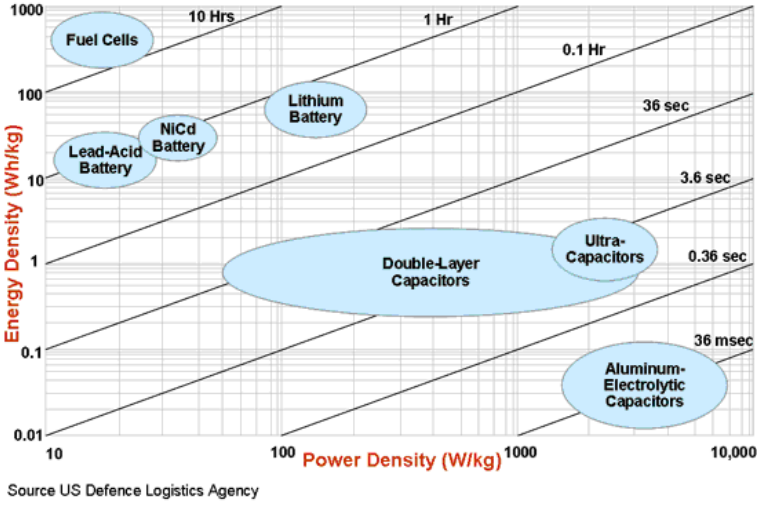


Figure 2.29: – Diagramme de Ragone

	Batterie Li-ion	**Supercondensateur**	**Batterie Li-ion polymer**
Cycle rechargé	Bas (100)	Très haut	Haut (>5000)
Autodécharge	Moyen	Haut	Très bas
Temps de charge	Lent	Rapide	Rapide
Taille	Moyenne	Grande	Petite
Variation de sortie		Variée	Stable
Capacité	0.3-10$mAHr$	10-100$uAHr$	12-85$uAHr$

Tableau 2.4: – Comparaison des types de stockages d'énergie

2.5 Conclusion

Les énergies renouvelables n'engendrent pas ou peu de déchets ou d'émissions polluantes. Elles participent à la lutte contre l'effet de serre et les rejets de CO_2 dans l'atmosphère. En effet, protéger l'environnement tout en assurant l'approvisionnement en énergie est l'un des défis majeurs du 21^e siècle. C'est pourquoi ce domaine de recherche se montre trés actif depuis un peu plus d'une décennie.

Dans cette thèse, nous nous interessons aux systèmes embarqués qui doivent fonctionner de façon autonome, tirant leur énergie à partir de l'environnement dans lequel ils évoluent. Leur mise en oeuvre suppose que soient installés différents dispositifs pour capturer l'énergie, stocker l'énergie et enfin restituer l'énergie sous forme d'électricité afin d'alimenter le système embarqué.

Nous avons donc présenté un état de l'art synthétique des principales technologies existantes même si celles-ci évoluent extrêmement rapidement.

Ce chapitre a permis de montrer que l'exploitation de l'énergie environnementale et en particulier l'énergie solaire peut servir à l'alimentation des systèmes embarqués et ainsi garantir un fonctionnement perpétuel sous réserve d'y associer des dispositifs de stockage d'énergie tels que batterie ou supercondensateur.

Dans le chapitre suivant, nous introduirons un modèle destiné à l'étude d'un système temps réel qui utilise l'énergie ambiante pour son alimentation, d'où la nécessité de résoudre des problèmes d'ordonnancement faisant intervenir la dimension énergétique.

Chapitre 3

Ordonnancement temps réel et énergie ambiante : problématique

3.1 Introduction

3.1.1 Particularités des systèmes autonomes

L'alimentation en énergie électrique est une problématique cruciale en particulier dans la conception des systèmes nomades qui par nature doivent être autonomes du point de vue énergétique. Aujourd'hui, cette problématique est principalement traitée par des méthodes de type DVS (Dynamic Voltage Scaling) et/ou DPM (Dynamic Power Management) qui visent à faire baisser la consommation d'énergie des circuits électroniques dont la technologie permet désormais de mettre en oeuvre ces méthodes. Les solutions proposées permettent ainsi d'étendre les durées séparant deux recharges successives d'une batterie. Ces recharges demeurent toutefois impératives.

Mais les systèmes embarqués de nouvelle génération, particulièrement ceux fonctionnant en environnement hostile ou inaccessible, limitent les interventions humaines. Ils fonctionnent grâce à des batteries (ou tout autre type de réservoirs d'énergie), qui se rechargent continûment au cours du temps à partir d'une source d'énergie renouvelable. Nul doute que les techniques DVS et DPM sont utiles aux systèmes autonomes : elles vont permettre d'utiliser des batteries de plus faible capacité, des panneaux photovoltaïques de plus petite taille, etc. Mais ces techniques ne permettent à elles seules d'assurer un fonctionnement à l'infini et qualifié de *neutre énergétiquement*. La neutralité énergétique se définit ici par la propriété que posséde le système embarqué à fonctionner dans le respect de toutes ses contraintes temporelles et ce, en n'utilisant que l'énergie disponible dans le réservoir et sans jamais en manquer.

La problématique liée à la conception des systèmes autonomes porte alors sur chacune de ses trois composantes :

- Le *récupérateur d'énergie* (harvester, en anglais) dont le choix dépend de la nature de l'énergie environnementale (voir chapitre précédent), de la quantité d'énergie requise, etc.
- Le *réservoir d'énergie* (appelé aussi stockage d'énergie) tel que batterie et/ou supercondensateur dont le choix dépend des dynamiques du système (voir diagramme de Ragone, figure 2.29), de contraintes de dimmensionnement, de coût, etc.

– Le *consommateur d'énergie* que représente ici le support d'exécution des tâches temps-réel. Dans cette thèse, nous considérons que l'énergie consommée par la partie opérative du système embarqué (actionneur, capteur, led, etc.) fait l'objet d'une alimentation séparée de même que le module d'émission/réception radio. Le consommateur d'énergie est donc synonyme de carte électronique construite autour d'un microcontrôleur ou d'un microprocesseur.

A ces composantes de base, peuvent s'ajouter d'autres composantes annexes mais parfois indispensables tels que des circuits DC/DC pour effectuer des conversions de tensions.

Dans ce travail de thèse, nous nous focalisons sur le consommateur d'énergie. D'un point de vue énergétique et de façon simplifiée, celui-ci peut être considéré comme une machine qui a des besoins énergétiques variables au cours du temps. Ceux-ci sont requis par l'ensemble des programmes et en particulier des tâches temps réel qui demandent à s'exécuter sur cette machine pendant certains intervalles de temps. Le besoin en énergie n'est donc pas constant et continu au cours du temps. Ce besoin dépendra donc du profil temporel des tâches trés généralement caractérisé par les périodes et/ou les échéances de celles-ci.

Un système embarqué et notamment un capteur intelligent autonome est en général conçu pour fonctionner plusieurs années voire plusieurs dizaines d'années sans possibilité d'intervention. C'est pourquoi, le fait de pouvoir garantir hors-ligne que celui-ci satisfera ses contraintes (même si elles ne sont pas dures) revêt une importance cruciale. Contrairement aux systèmes temps réel conventionnels, une des difficultés sera liée à l'incertitude qui peut porter sur la quantité d'énergie effectivement tirée de l'environnement à tout instant et éventuellement des estimations qui peuvent en être faîtes. Ce dernier problème lié aux techniques de prédiction associées à la production de l'énergie environnementale dépasse le cadre de cette thèse. Nous restreindrons notre étude à des modèles de prédiction simplistes en particulier où l'énergie est produite à puissante constante (par exemple, énergie lumineuse in-door) ou à puissance périodique connue (par exemple, énergie piézoélectrique par suite de la déformation d'un matériau).

3.1.2 Problématique liée à l'ordonnancement : aperçu

Nous constatons ainsi que la conception d'un système autonome renvoie à plusieurs questions fondamentales :

En supposant que l'énergie d'alimentation soit parfaitement caractérisée (profil de la source énergétique, taille de batterie de stockage, etc.), comment vérifier et garantir avant la mise en opération du système que celui-ci aura une autonomie perpétuelle avec un niveau de performance toujours acceptable ?

Cela suppose donc en premier lieu de définir ce niveau de performance souvent appelé Qualité de Service, caractérisé à partir des contraintes applicatives. Dans cette thèse, nous considèrerons un système temps-réel ferme. Ce niveau de performance se rapportera donc en premier lieu au pourcentage d'échéances satisfaites, mesuré sur la totalité des tâches dont la fin d'exécution est soumise au respect d'une échéance.

D'un point de vue logiciel, un système temps réel se compose des tâches applicatives et du système d'exploitation temps réel (communément appelés RTOS) en charge d'assurer, entre autres

fonctionalités, l'ordonnancement de ces dernières. Dans le chapitre 1, nous avons rappelé les notions de base relatives aux ordonnanceurs temps réel typiquement implémentés dans les RTOS actuels, qu'ils soient faits maison, propriétaires ou libres. Ces ordonnanceurs ont, dans la plupart des cas la particularité d'être en-ligne, donc non oisifs, conduits par la priorité et préemptifs. Leur implémentation n'engendre aucune difficulté : il s'agit de gérer une ou plusieurs structures de données organisées en listes, chacune associée à l'état des tâches. Le rôle de l'ordonnanceur consiste à ordonner ces listes et les mettre à jour, qu'il s'agisse d'une gestion à priorité fixe avec Rate Monotonic ou à priorité dynamique avec Earliest Deadline First. Mais ces ordonnanceurs optimaux offrent leur performance sans prendre en compte aucune contrainte d'ordre énergétique. En effet, leur optimalité est démontrée sous l'hypothèse centrale suivante : le processeur dispose à tout instant de l'énergie requise à l'exécution de toute tâche. Nous voyons ainsi que la seule contrainte à gérer par l'ordonnanceur est d'ordre temporel. Des conditions d'ordonnançabilité associées à ces ordonnanceurs portent ainsi sur le taux d'utilisation du processeur, ce dernier calculé à partir de la périodicité des tâches et de leur besoin en temps d'éxécution. D'autres conditions portent sur des bornes majorantes de traitement requis dans des intervalles de temps critiques (approche de la demande processeur).

Dans un système autonome alimenté par l'énergie ambiante, la problématique de l'ordonnancement sera liée au fait de prendre en compte conjointement deux contraintes physiques : le temps qui se mesure en secondes et l'énergie qui se mesure en joules. Les questions fondamentales qui se posent alors sont les suivantes : un ordonnanceur aussi efficace et performant que Earliest Deadline First peut-il convenir aux systèmes soumis en plus de ses contraintes temporelles à des contraintes énergétiques ? Existe t-il, dans ce nouveau contexte de fonctionnement propre à l'énergie ambiante, un ordonnanceur à la fois optimal et implémentable aisément ?

La problématique qui vient d'être résumée a fait l'objet de premières études publiées en 2001 [2], soit il y a seulement une dizaine d'années. Il s'agit d'un sujet encore très peu étudié actuellement (comparé à celui de l'ordonnancement visant à minimiser l'énergie consommée) et auquel nous proposons d'apporter notre contribution dans cette thèse.

Pour décrire plus précisément et plus formellement le problème traité, nous proposons dans ce chapitre, de suivre le plan suivant :

- Description des travaux de recherche initiaux traitant de ce sujet [2],
- Description du modèle précis faisant l'objet de notre étude,
- Mise en évidence du comportement d'un algorithme d'ordonnancement temps réel conventionnel tel que EDF, sous le modèle décrit.

3.2 Travaux de recherche initiaux

Nous reportons dans cette section le premier travail de recherche lié à l'ordonnancement temps réel sur un système monoprocesseur alimenté par l'énergie ambiante, réalisé à l'université de Pittsburg [2].

3.2.1 Modèle étudié

Le système étudié considère :

– Un processeur unique monovitesse, c'est-à-dire fonctionnant avec une seule fréquence possible. Ce processeur possède seulement deux états : marche et arrêt.

– Une configuration de tâches périodiques de périodes identiques (connue sous le nom anglais de *Frame Based System*) et d'échéance commune (délai critique commun), notée D_i. Nous appellerons ces tâches, *des tâches basiques* et cette période commune, *une fenêtre*. La préemption est autorisée et les coûts engendrés par celle-ci en temps et en énergie sont considérés comme négligeables. Nous supposons que la puissance instantanée consommée par la tâche τ_i en exécution est constante, notée par P_c. Alors si la tâche τ_i est exécutée continuement sur un intervalle de temps donné $[t_1,\ t_2]$, sa consommation d'énergie sera donnée par $E = \int_{t_1}^{t_2} P_c \cdot dt = P_c(t_2 - t_1)$.

– Un réservoir d'énergie rechargeable dont le niveau doit toujours rester entre deux bornes E_{min} et E_{max}, donnant donc une capacité égale à $E_{max} - E_{min}$, voir figure 3.1. Le taux de charge du réservoir d'énergie, noté P_r correspond à la puissance produite par la source d'énergie environnementale et qui sert à recharger le réservoir. Nous supposons une absence totale de pertes ou bien intégrées déjà dans la valeur de P_r.

  Si le réservoir devient plein c'est-à-dire son niveau d'énergie égal à E_{max} et si nous continuons à le charger, l'énergie superflue sera alors perdue. Si au contraire le réservoir d'énergie devient vide c'est-à-dire son niveau d'énergie égal à E_{min}, le processeur devra rester inactif sans possibilité d'exécution de tâche.

  Le taux instantané de recharge du réservoir d'énergie pendant l'exécution de la tâche τ_i est constant et donné par $w_i = P_r - P_c$. Selon la valeur de w_i, la tâche τ_i est soit de type *Rechargeante* (dite aussi productrice d'énergie) ou de type *Déchargeante* (dite aussi dissipatrice d'énergie). Nous pouvons ainsi distinguer deux sous-ensembles de tâches : le groupe $R = \{\tau_i | w_i \geq 0\}$ et le groupe $D = \{\tau_i | w_i < 0\}$.

  Par suite, nous pouvons introduire les deux variables notées $|R|$ et $|D|$, définies comme suit : $|R| = \sum w_i \cdot t_i, \tau_i \in R$ et $|D| = -\sum w_i \cdot t_i, \tau_i \in D$. Pour chaque groupe de tâches, le paramètre $|R|$ (respectivement $|D|$) donne une mesure de l'énergie reçue (respectivement perdue) lorsque toutes les tâches du groupe s'exécutent dans une fenêtre.

3.2.2 Position du problème

Le problème soulevé ici consiste à trouver un ordonnancement faisable c'est-à-dire satisfaisant les contraintes suivantes :

– Contrainte d'ordre temporel : Dans chaque fenêtre, toutes les tâches s'exécutent avant

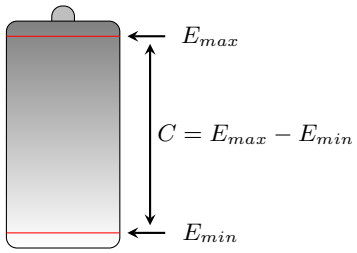


Figure 3.1: – Modèle de batterie rechargeable

l'échéance commune D_i,

– Contrainte d'ordre énergétique : Le réservoir est plein à l'instant de démarrage (supposé 0) et doit être plein à la fin de chaque fenêtre.

La recherche d'une solution se restreint donc à la fenêtre initiale. L'ensemble des tâches ayant la même échéance, ces tâches peuvent donc s'exécuter dans n'importe quel ordre dans cette fenêtre.

3.2.3 Algorithme d'ordonnancement

3.2.3.1 Principe de l'algorithme d'Allavena et Mossé

Le principe de base de l'ordonnanceur consiste à exécuter successivement les tâches d'un même ensemble. Il sélectionne donc les tâches déchargeantes de sorte que le niveau d'énergie disponible en batterie baisse le plus possible jusqu'à atteindre le niveau minimum E_{min}. Puis il sélectionne les tâches rechargeantes jusqu'à ce le niveau d'énergie remonte le plus possible jusqu'à atteindre le niveau maximum E_{max}, maintenant ainsi le niveau d'énergie entre les deux bornes E_{min} et E_{max}. Une préemption se produit donc chaque fois que le niveau d'énergie atteint l'une des deux bornes pour commuter sur mode de recharge en exécutant des tâches rechargeantes ou de décharge de la batterie en exécutant des tâches déchargeantes.

L'algorithme d'Allavena et Mossé distingue les tâches par le paramètre w_i et les regroupe en deux listes de tâches : une liste de tâches de type rechargeantes R et une liste de tâches de type déchargeantes D. Il évalue les valeurs relatives de $|R|$ et de $|D|$. Si $|R| < |D|$; cela signifie alors que le besoin en énergie réclamé par les tâches déchargeantes restant à s'exécuter avant échéance est plus important que l'apport en énergie des tâches rechargeantes restant à s'exécuter. Afin d'assurer l'équilibre entre la production d'énergie et la consommation, l'ordonnanceur doit insérer un intervalle de temps inactivité du processeur n'engendrant ainsi ni consommation ni production d'énergie liée à l'exécution d'une tâche. Le rajout de cet intervalle de temps creux se fait dès lors que la condition $|R| < |D|$ est vérifiée et la longeur de cet intervalle d'inactivité est alors donnée par : $t_{idle} = \frac{|D|-|R|}{P_r}$.

La condition $|R| < |D|$ est vérifiée initialement hors-ligne puis ensuite testée en-ligne à chaque fois que le niveau d'énergie atteint l'une des deux bornes. Si initialement, cette condition est vraie et si la somme des durées d'exécution des tâches rajoutées à ce temps d'inactivité dépasse l'échéance commune c'est-à-dire $\sum_{i=1}^{n} C_i + t_{idle} > D_i$, nous pouvons conclure à la non faisabilité du système. La condition $\sum_{i=1}^{n} C_i + t_{idle} \leq D_i$ constitue donc une condition nécessaire et suffisante

d'ordonnançabilité.

3.2.3.2 Pseudo-code de l'algorithme d'Allavena et Mossé

Le pseudo-code de l'algorithme d'Allavena et Mossé est fourni ci-dessous. Il représente l'ensemble des opérations mises en oeuvre pour construire en-ligne la séquence dans chaque fenêtre.

Données : maintenir un ensemble d'indices $i \in N$ pour toutes les tâches déjà préparées mais pas encore terminées τ_i.
Résultats : sortir une séquence valide.

Tant que (liste de tâches en attente n'est pas vide) **faire**
  Pour i **de** 1 **à** n **faire**
    $P_c \leftarrow$ puissance instantanée consommée par la tâche τ_i ;
    $P_r \leftarrow$ puissance constante de l'énergie ambiante ;
    $w_i \leftarrow P_r - P_c$;
    Si $(w_i \geq 0)$ **Alors**
      ajouter τ_i dans la liste R, nommée τ_k ;
    Sinon
      ajouter τ_i dans la liste D, nommée τ_j ;
    Fin Si
  Fin Pour
  calculer $|R|$ et $|D|$;
  Si $(|R| < |D|)$ **Alors**
    insérer un intervalle de temps idle t_{idle} $(= \frac{|D|-|R|}{P_r})$ dans la liste R ;
  Fin Si
  Si $(E_{min} < E)$ **Alors**
    exécuter τ_j jusqu'à la fin de la liste D ou $E_c = E_{min}$;
  Sinon
    exécuter τ_k jusqu'à la fin de la liste R ou $E_c = E_{max}$;
  Fin Si
Fait

Tableau 3.1: – Pseudo-code de l'algorithme d'Allavena et Mossé

3.2.4 Exemple illustratif

Exemple 3.1.

Le comportement de l'algorithme d'Allavena et Mossé est illustré sur la figure 3.2 sur une configuration de tâches $\mathscr{T}=\{\tau_1, \tau_2\}$ constitué de 2 tâches périodiques, concrètes et ER. Chaque tâche τ_i est caractérisée par un 3-uplets (a_i, C_i, T_i). Pour simplifier, nous mettons les deux tâches identiques : $\tau_1 = \tau_2 = (0, 1, 5)$. Nous supposons aussi que la puissance instantanée consommée par toute tâche τ_i est égale à la puissance du processeur, soit 8, et la puissance de l'énergie

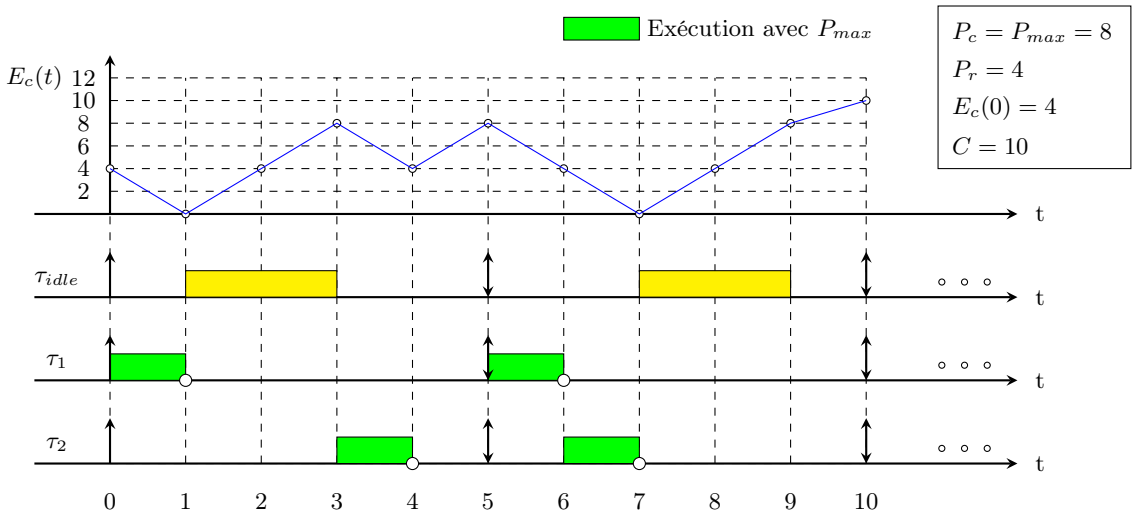


Figure 3.2: – Séquence produite par l'algorithme d'Allavena et Mossé

ambiante P_r est constante, égale à 4. Parce que $w_i = P_r - P_c = 4 - 8 = -4 < 0$, ces deux tâches sont donc les tâches déchargeantes. Evidemment $|R| < |D|$, car il n'y a pas de tâches rechargeantes dans la configuration de tâches. Il faudrait alors, pour assurer l'équilibre entre la production d'énergie et la consommation, insérer un intervalle de temps d'inactivité du processeur (temps creux), avec $t_{idle} = 2$ ($= \frac{|D|-|R|}{P_r} - \sum_{i=1}^{2} C_i = \frac{16-0}{4} - 2 = 2$). Par conséquent, d'après $\sum_{i=1}^{2} C_i + t_{idle} = 2+2 = 4 \leq D_i$, la configuration de tâches $\mathscr{T}$ peut être fiablement ordonnançée par l' algorithme d'Allavena et Mossé.

Concernant l'énergie, nous utilisons un réservoir d'énergie de capacité $C = 10$, avec $E_{min} = 0$. Initialement, le niveau d'énergie $E_c(0)$ est égal à 4.

3.2.5 Commentaires

Dans ce modèle, les auteurs effectuent des hypothèses très restrictives : une période et une échéance commune pour toutes les tâches de l'application et la puissance de la source d'énergie environnementale constante au cours du temps. La solution proposée a le mérite d'être optimale et simple à mettre en oeuvre. En effet, la complexité de cet ordonnanceur est clairement linéaire : aucun tri le liste n'est nécessaire. Cet algorithme a été ensuite modifié, dans le même article, pour s'appliquer à un processeur à vitesse variable mais toujours restreint à des tâches basiques.

Nous proposons dans le paragraphe suivant de définir précisément le modèle de tâches périodiques sur lequel va s'appuyer notre étude.

3.3 Modèle étudié dans cette thèse

3.3.1 Description schématique

Le modèle étudié dans cette thèse correspond à celui décrit dans les travaux suivants : [32–34]. Il constitue une avancée significative dans la modélisation des systèmes embarqués alimentés par l'énergie ambiante par rapport au premier modèle étudié en 2001 et rapporté dans le paragraphe

précédent. A noter qu'entre 2001 et 2006, aucune étude sur ce sujet spécifique n'a fait l'objet de publications.

Nous considérons ici un système embarqué autonome construit autour de trois éléments principaux que constituent le récupérateur d'énergie, le réservoir d'énergie et le microprocesseur, consommateur d'énergie (cf. figure 3.3). Ces trois éléments doivent donc être modélisés le plus finement possible de façon à construire pour les tâches temps réel, le meilleur ordonnancement possible.

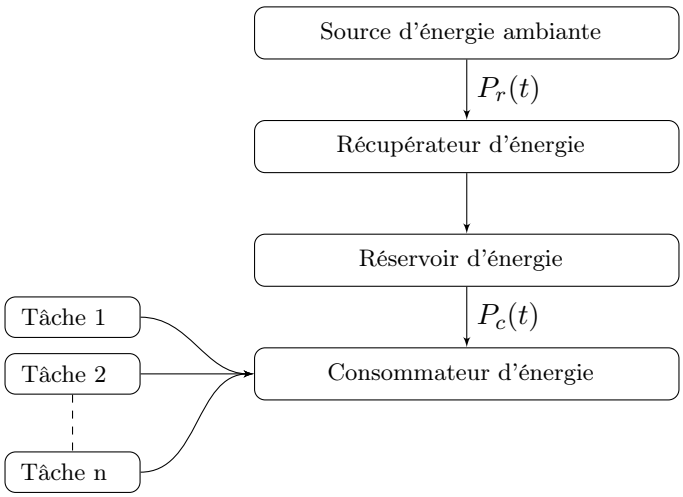


Figure 3.3: – Modèle de configuration d'un système rechargeable

3.3.2 Modélisation de la source d'énergie

Le récupérateur d'énergie (par exemple un panneau photovoltaïque) récupère l'énergie à partir d'une source d'énergie (par exemple le soleil ou une lampe) et la convertit en énergie électrique. Nous noterons $P_r(t)$, la puissance instantanée de récupération d'énergie. Nous supposerons que cette grandeur inclut toutes les pertes d'énergie occasionnées par la mise en oeuvre matérielle. Nous appelerons *profil énergétique de la source*, la descrition de la fonction $P_r(t)$ pour toute la durée de vie de l'application, soit dans l'idéal un temps infini.

L'énergie récupérée dans un intervalle $[t_1, t_2]$, notée $E_r(t_1, t_2)$ se calcule donc de la façon suivante :

$$E_r(t_1, t_2) = \int_{t_1}^{t_2} P_r(t)dt \tag{3.1}$$

3.3.3 Modélisation du consommateur d'énergie

Le coeur du système embarqué, un processeur unique, est caractérisé par une puissance de consommation instantanée notée $P_c(t)$, exprimée en watts, et par une puissance de consommation maximum notée P_{max}, soit $0 \leq P_c(t) \leq P_{max}$. Pour un processeur n'utilisant pas une technique DVS, la puissance $P_c(t)$ sera supposée nulle lorsque le processeur se trouve en état de veille, et

égale à P_{max} lorsqu'il exécute une tâche. Nous noterons $E(t_1, t_2)$ l'énergie consommée au cours de l'intervalle $[t_1, t_2]$.

Nous supposons que le processeur décrit ci-dessus est destiné à l'exécution d'une configuration de tâches périodiques, indépendantes, préemptables à tout instant et toutes synchrones à l'instant initial égal à zéro. Nous noterons $\mathscr{T}$ une telle configuration et P le PPCM des périodes de ses tâches. Les surcoûts en temps et en énergie, occasionnés par une préemption, sont supposés nuls. Chaque tâche τ_i est caractérisée par les paramètres suivants :

- sa *période*, T_i, exprimée en secondes ;
- son *délai critique*, D_i, avec $D_i \leq T_i$, exprimée en secondes ;
- sa *consommation énergétique*, E_i, exprimée en joules.

La consommation énergétique de τ_i sera mesurée dans le pire cas et donc correspondant à la plus grande quantité d'énergie qu'elle peut consommer en s'exécutant sur un processeur dont la puissance instantanée de consommation est donnée par P_{max}.

Lorsque la tâche τ_i s'exécute sur le processeur avec une puissance P_{max}, sa durée d'exécution pire cas sans préemption est donc donnée par : $C_i = E_i/P_{max}$, exprimée en secondes. Cela signifie qu'une tâche τ_j qui aurait une consommation d'énergie égale au double de celle de τ_i aurait aussi une durée d'exécution égale au double de celle de τ_i sur le même processeur. Autrement dit, si $E_j = k \times E_i$ alors $C_j = k \times C_i$.

En résumé, la configuration de tâches décrite ici correspond au modèle classique de Liu et Layland avec des données supplémentaires : la consommation énergétique requise par toute exécution de tâche et la puissance de consommation instantanée du processeur qui relie les deux grandeurs : consommation énergétique et durée d'exécution de toute tâche. Et ce rapport entre les deux grandeurs est le même pour toutes les tâches de la configuration.

Nous appellerons :

- **taux d'utilisation énergétique**, la quantité donnée par $U_e = \sum_{i=1}^{n} \frac{E_i}{T_i}$
- **taux d'utilisation processeur**, la quantité donnée par $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ soit aussi $U = (\frac{1}{P_{max}}) \sum_{i=1}^{n} \frac{E_i}{T_i}$ ou $U = (\frac{1}{P_{max}})U_e$

3.3.4 Modélisation du réservoir d'énergie

L'énergie récupérée est stockée dans un réservoir d'énergie tel que ceux décrits au chapitre 2. Ce réservoir de type batterie ou supercondensateur est caractérisé par sa capacité, notée C. Nous le supposons idéal en ce sens qu'il peut toujours être chargé à sa capacité maximmale. Nous considérons que cette grandeur est constante au cours du temps. A tout instant t, l'énergie stockée dans le réservoir est notée $E_c(t)$ et vérifie donc $0 \leq E_c(t) \leq C$. Le réservoir n'est pas nécessairement plein à l'instant initial, soit $E_c(0) \leq C$.

Nous considérons que si le réservoir devient plein et qu'aucune tâche ne s'exécute, alors l'énergie récupérée sera *gaspillée*. De ce qui précède, nous pouvons établir que : $E_c(t_2) \leq E_c(t_1) + E_r(t_1, t_2) - E(t_1, t_2)$, pour tout $t_1 \leq t_2$.

De plus, le processeur ne peut pas consommer plus, dans l'intervalle $[t_1, t_2]$, que l'énergie disponible dans le réservoir à l'instant t_1 à laquelle s'ajoute la quantité d'énergie produite par la source entre t_1 et t_2, ce qui correspond à la relation : $E(t_1, t_2) \leq E_c(t_1) + E_r(t_1, t_2)$.

3.4 Nouvelle terminologie

Nous proposons, pour ce nouveau modèle de système temps réel, d'introduire une nouvelle terminologie adaptée aux caracteristiques énergétiques des tâches, du processeur et de la source d'alimentation.

- Ordonnancement totalement non clairvoyant : Nous disons qu'un ordonnanceur est *totalement non clairvoyant* lorsqu'il ne sait rien ni des tâches qu'il doit ordonnancer, ni du profil energétique de la source qui alimente le système.

- Ordonnancement avec clairvoyance énergétique : Nous disons qu'un ordonnanceur a *une clairvoyance énergétique* lorsqu'il connait a priori le profil energétique de la source qui alimente le système. En d'autres termes, il sait quand et en quelle quantité, l'énergie sera produite. Dans le cas contraire, nous pourrons dire que l'ordonnanceur est non clairvoyant du point de vue énergétique.

- Ordonnancement avec clairvoyance de traitement : Nous disons qu'un ordonnanceur est *une clairvoyance de traitement* lorsqu'il connait a priori quand vont les tâches vont arriver pour demander à s'exécuter.

- Ordonnancement totalement clairvoyant : Un ordonnanceur *totalement clairvoyant* est celui qui a à la fois une clairvoyance de traitement et une clairvoyance énergétique.

- Ordonnancement temporellement faisable : Nous disons qu'un ordonnancement est *temporellement faisable* pour une configuration de tâches donnée s'il existe au moins un ordonnanceur capable de produire une séquence où toutes les contraintes temporelles de cette configuration sont respectées sans prendre en compte ses contraintes énergétiques.

- Configuration de tâches ordonnançable vs non ordonnançable : Nous disons qu'une configuration de tâches est *ordonnançable* s'il existe au moins un ordonnanceur capable de satisfaire toutes les contraintes temporelles et énergétiques de cette configuration étant données les caractéristique du réservoir d'énergie et de la source d'énergie. Sinon, elle est dite *non ordonnançable* soit à cause de ses contraintes temporelles (échéances trop strictes) soit à cause de ses contraintes énergétiques (réservoir trop petit ou puissance de la source trop petite).

- Surcharge de traitement : Nous disons qu'une configuration de tâches est en *surcharge de traitement* lorsqu'elle a un taux d'utilisation du processeur trop élevé pour pouvoir respecter ses échéances indépendemment de ses besoins énergétiques.

- Surcharge énergétique : Nous disons qu'une configuration de tâches est en *surcharge énergétique* lorsqu'elle a des besoins en énergie trop élevés pour pouvoir respecter ses échéances indépendemment de la quantité de traitement requise.

- Demande processeur : Nous appelons *demande processeur* d'une configuration de tâches sur un intervalle de temps $[t_1, t_2]$, la quantité de temps de traitement requis par son exécution entre t_1 et t_2, notée $h(t_1, t_2)$.

- Demande énergétique : En parallèle à la demande processeur, nous appelons *demande énergétique* d'une configuration de tâches sur un intervalle de temps $[t_1, t_2]$, la quantité d'énergie requise par son exécution entre t_1 et t_2, notée $g(t_1, t_2)$.

- Système en puissance constante : Un système est dit en *puissance constante* lorsque la source délivre son énergie avec la même puissance sur toute la durée de vie de l'application.

La puissance constante sera notée P_r. Sinon, le système est dit en *puissance variable.*

Les autres définitions énoncées dans le chapitre 1 restent valables pour ce nouveau modèle.

3.5 Tests d'ordonnançabilité

Pour retourner au modèle de base, il faudrait considérer que $P_{max} = 0$ et que pour toute tâche τ_i, $E_i = 0$. Et sous cette hypothèse, les tests d'ordonnançabilité rappelés dans le chapitre 1 s'appliqueraient. Nous proposons ci-dessous des conditions d'ordonnançabilité pour ce nouveau modèle.

3.5.1 Condition nécessaire d'ordonnançabilité

Vérifier qu'une configuration de tâches peut être ordonnançable dans ce nouveau modèle revient à vérifier qu'elle n'est jamais, ni en surcharge de traitement ni en surcharge énergétique sur toute la durée de vie de l'application. Mettre en oeuvre un test d'ordonnançabilité hors ligne requiert donc de connaître a priori tous les paramètres de l'application de façon précise. En ce qui concerne les caractéristiques des tâches, si nous considérons qu'elles sont périodiques, alors il sera possible d'évaluer hors ligne son taux d'utilisation processeur et vérifier l'absence de surcharge de traitement. Concernant l'aspect énergétique, nous pouvons dégager deux cas de figure :

1. Le profil énergétique présente une puissance constante au cours du temps (système en puissance constante). C'est par exemple le cas de l'énergie obtenue par un convertisseur thermo-électrique utilisant le gradient de température entre le corps humain et la température ambiante.
2. Le profil énergétique est caractérisé par une puissance d'émission qui varie au cours du temps (système en puissance variable). C'est par exemple le cas des fonctions périodiques sinusoïdales (énergie solaire) où la période est de 24 heures. Lorque cette puissance d'émission n'est pas cyclique, il faudra donc faire des observations sur une durée suffisamment longue pour en déduire une puissance moyenne d'émission de la source d'énergie. Par exemple, considérons l'énergie émise par un matériau piezoélectrique placé dans la semelle d'une chaussure. En supposant que lorsque la personne marche à allure normale, nous récupérons 12 milliWatts et que cette personne marche au minimum 2 heures par jour. Nous en déduisons alors que la puissance moyenne récupérée est de 1 milliwatt dans le pire cas.

Supposons que le réservoir d'énergie est de taille finie, quelque soit son niveau de remplissage au temps initial. En effet, si celui-ci était de taille infinie et plein au temps initial, alors le système ne serait pas contraint par l'énergie quelque soit le profile énergétique de la source. Et la seule condition d'ordonnançabilité serait d'être en sous-charge de traitement (taux d'utilisation processeur inférieur ou égal à 1).

Si nous notons $\tilde{P}_r$ la puissance moyenne récupérable de la source d'énergie, alors nous tirons le test de faisabilité suivant :

Théorème 3.1. *Une configuration de tâches périodiques est ordonnançable seulement si*

$$U \leq \min(1, \tilde{P}_r/P_{max}) \tag{3.2}$$

Preuve 3.1. *Pour qu'une configuration de tâches périodiques soit ordonnançable sur le processeur, son taux d'utilisation processeur doit être inférieur ou égal à 1 soit* $U \leq 1$. *Du point de vue de l'énergie, la configuration ne doit pas consommer plus d'énergie qu'elle n'en reçoit sur une durée infinie. En d'autres termes, la puissance moyenne consommée par les tâches sur une durée infinie ne doit pas être supérieure à la puissance moyenne fournie par la source d'énergie sur une durée infinie. Ce qui revient à ce que son taux d'utilisation énergétique,* U_e *donnée par* $U_e = \sum_{i=1}^{n} \frac{E_i}{T_i}$ *soit inférieur ou égal à* $\tilde{P}_r$.

D'où la condition nécessaire d'ordonnançabilité : $U \leq 1$ *et* $U_e \leq \tilde{P}_r$. *Comme* $U_e = (P_{max}) \times U$, *il faut donc que* $U \leq 1$ *et* $U_p \leq \tilde{P}_r / P_{max}$, *c'est-à-dire que* $U \leq \min(1, \tilde{P}_r / P_{max})$. □

Exemple 3.2. *Sachant qu'une configuration de deux tâches périodiques ER (Echéance sur Requête) caractérisée par :* $\mathscr{T} = \{ \tau_i | 1 \leq i \leq 2, avec\ \tau_i = (a_i, C_i, T_i) \}$. *Supposons que* $\tau_1 = (0, 3, 10)$ *et* $\tau_2 = (0, 2, 5)$. *Nous notons que* $PPCM(\tau_1, \tau_2) = 10$. *Nous supposons aussi que la puissance de la source d'énergie est constante :* $P_r = 6$, *donc* $\tilde{P}_r = P_r = 6$. *De plus, la capacité du réservoir* $C = 12$, *l'énergie initiale du réservoir* $E_c(0) = 8$ *et la puissance du processeur* $P_c = P_{max} = 8$.

Vérifions ici la condition nécessaire d'ordonnançabilité selon le théorème 3.1. Nous avons

$$\begin{aligned} U &= \sum_{i=1}^{2} \frac{C_i}{T_i} = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{3}{10} + \frac{2}{5} = 0.7 \\ \tilde{P}_r / P_{max} &= 6/8 = 0.75 \end{aligned}$$

Donc $U = 0.7 \leq \min(1, 0.75)$, la condition nécessaire d'ordonnançabilité est satisfaite. La séquence d'ordonnancement produite par EDF est illustrée dans la figure 3.4. Nous trouvons que la configuration de tâches est fiablement ordonnancée car aucune tâche ne viole son échéance.

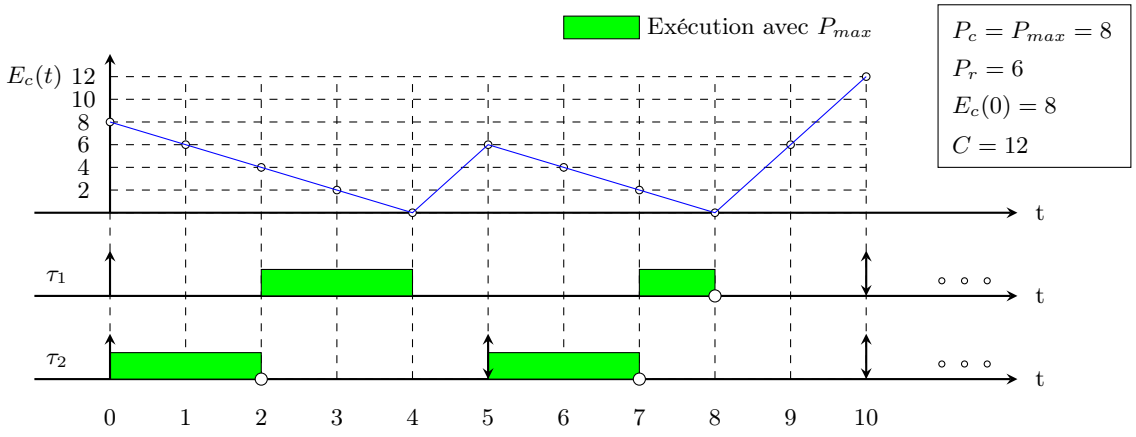


Figure 3.4: – Séquence produite par EDF sous contraintes énergétiques

3.5.2 Commentaires

Le test précédent s'interprète de la façon suivante :

- Notons que si le système est idéal c'est-à-dire que le processeur a une puissance de consommation instantanée qui tend vers 0, alors ce test devient $U \leq 1$, celui employé habituellement en l'absence de contraintes énergétiques. Et si les tâches sont de plus ER, ce test devient en plus une CNS d'ordonnançabilité.
- Si $\tilde{P_r}$ est plus grand que P_{max}, alors cela signifie qu'en moyenne la source produit plus d'énergie que le processeur ne peut en consommer. Le système est sous-chargé du point de vue énergétique. Et dans ce cas, le taux d'utilisation du processeur est contraint uniquement du point de vue du temps d'exécution. Et ce taux ne doit pas excéder 1 pour éviter une surcharge de processeur. Une idée serait de faire en sorte que $\tilde{P_r}$ soit le plus grand possible comparé à P_{max} la puissance de consommation instantanée du processeur. Dans une application, faire augnmenter $\tilde{P_r}$ revient à faire augnmenter la taille du récupérateur d'énergie. Pour un panneau photovoltaique, $\tilde{P_r}$ varie avec sa surface. De même que pour un matériau piezo-électrique, $\tilde{P_r}$ pourra varier avec la longueur ou la surface du matériau. Pour un système embarqué, l'aspect dimmensionnement prime. Et nous souhaiterons intégrer un récupérateur d'énergie qui soit le plus petit (souvent le moins cher) possible tout en garantissant la faisabilité de l'application.
- Si P_{max} est plus grand que $\tilde{P_r}$, alors cela signifie que lorsque le processeur est en activité pour exécuter une tâche, la consommation d'énergie est plus forte que la production. D'où la nécessité que le facteur d'utilisation ne soit pas trop élevé pour permettre une recharge du réservoir. En effet, ce sera uniquement pendant les périodes d'inactivité du processeur que le niveau d'énergie augnmentera dans le réservoir. Plus le rapport $\tilde{P_r}/P_{max}$ sera petit, plus cette contrainte énergétique aura un impact sur le taux d'utilisation.

Compte tenu de la forme du test, nous constatons clairement que le mieux est de concevoir une application qui d'une part optimise l'utilisation du processeur en ayant un taux d'utilisation qui tend vers 1 et d'autre part, garantit un apport énergétique tout juste suffisant pour s'ajuster à la demande de la configuration de tâches. Nous constatons de plus que la capacité du réservoir n'intervient pas dans le test. En effet, ce test considère la charge de traitement et la charge énergétique moyennes sur une trés longue période de fonctionnement. C'est donc une condition relative à une vue macroscopique du système.

3.5.3 Condition suffisante d'ordonnançabilité

Ci-dessous, considérons une configuration de tâches périodiques $\mathscr{T} = \{ \tau_i | 1 \leq i \leq n \text{ tel que } \tau_i = (C_i, D_i, T_i) \}$, un système en puissance constante donnée par P_r, un réservoir de capacité finie C et un processeur qui consomme P_{max}. Nous démontrons les résultats suivants :

Théorème 3.2. *S'il existe un ordonnancement faisable pour $\mathscr{T}$ sur un intervalle $[(k-1)P, kP], k \geq 1$ tel que $E_c(kP) \geq E_c((k-1)P)$ alors le même ordonnancement faisable existe pour $\mathscr{T}$ sur $[kP, (k+1)P]$ tel que $E_c((k+1)P) \geq E_c(kP)$.*

Preuve 3.2. *La configuration est ordonnançable sur $[(k-1)P, kP]$, donc toutes les instances de tâches réveillées dans cet intervalle ont terminé leur exécution avant échéance. A l'instant kP, le système, du point de vue de l'activité du processeur se retrouve dans le même état qu'à*

(k-1)P. Du point de vue énergétique, le système se retrouve à l'instant kP dans une situation plus favorable qu'à (k-1)P car avec un surplus d'énergie dans le réservoir à l'instant kP, égal à $E_c(kP) - E_c((k+1)P)$*. Donc la configuration sera aussi ordonnançable sur* $[kP, (k+1)P]$*. Montrons maintenant que nous pouvons appliquer aux instances réveillées sur* $[kP, (k+1)P]$ *le même ordonnancement que celui appliqué aux instances réveillées sur* $[(k-1)P, kP]$ *et nous aurons alors nécessairement* $E_c((k+1)P) \geq E_c(kP)$*.*

Soient deux instants t_1 *et* t_2 *tels que* $(k-1)P \leq t_1 \leq kP$ *et* $t_2 = t_1 + P$*. Supposons que les deux séquences soiet identiques respectivement jusqu'à* t_1 *et* t_2 *avec* $E_c(t_2) \geq E_c(t_1)$*. Montrons qu'elles peuvent être identiques dans l'unité de temps suivante et que nous aurons aussi* $E_c(t_2+1) \geq E_c(t_1+1)$*. Entre* t_1 *et* t_1+1*, voici les différentes situations possibles :*

- *Le processeur est actif. Alors ayant plus d'énergie dans le réservoir qu'à* t_1*, il peut aussi être actif entre* t_2 *et* t_2+1*. Et le niveau d'énergie dans le réservoir a diminué de la même quantité donnée par* $P_{max} - P_r$*. D'où* $E_c(t_2+1) = E_c(t_1+1) + E_c(kP) - E_c((k-1)P)$*. Et donc* $E_c(t_2+1) \leq E_c(t_1+1)$*.*
- *Le processeur est inactif. Il peut évidemment aussi être inactif entre* t_2 *et* t_2+1*. Trois situations sont toutefois à considérer du point de vue du réservoir :*
  1. *Le réservoir est plein à* t_1 *et de l'énergie est gaspillée entre* t_1 *et* t_1+1 *d'une quantité égale à* P_r*. Alors nécessairement le réservoir est La même quantité sera donc aussi gaspillée entre* t_2 *et* t_2+1 *et donc avec la même différence de quantité d'énergie dans le réservoir. Soit* $E_c(t_2+1) = E_c(t_1+1) + E_c(kP) - E_c((k-1)P)$*. Et donc* $E_c(t_2+1) \geq E_c(t_1+1)$*.*
  2. *Le réservoir n'est pas plein ni à* t_1*, ni à* t_2*. Dans ce cas, dans les deux séquences, le réservoir se remplit de la même quantité égale à* P_r *et nous avons aussi* $E_c(t_2+1) = E_c(t_1+1) + E_c(kP) - E_c((k-1)P)$*. Et donc* $E_c(t_2+1) \geq E_c(t_1+1)$*.*
  3. *Le réservoir n'est pas plein à* t_1 *mais plein à* t_2*. Soit* $E_c(t_1) < C$ *et* $E_c(t_2) = C$*. Nous avons donc* $E_c(t_2+1) = C$ *car l'énergie récupérée entre* t_2 *et* t_2+1 *est gaspillée. Et nous avons* $E_c(t_1+1) \leq C$*. Et donc* $E_c(t_2+1) \geq E_c(t_1+1)$*.*

Nous appliquons ce raisonnement par récurrence et montrons que $E_c((k+1)P) \geq E_c(kP)$*.* □

Le théorème précédent montre que si nous trouvons une hyperpériode dans laquelle un algorithme d'ordonnancement produit une séquence valide et tel que le niveau d'énergie à la fin de l'hyperpériode est au moins aussi grand qu'au début de l'hyperiode, alors il pourra produire la même séquence valide dans l'hyperpériode suivante et le niveau d'énergie sera au moins aussi grand à la fin qu'au début de cette hyperpériode.

Nous pouvons donc donner le théorème qui en découle :

Théorème 3.3. *S'il existe un ordonnancement faisable pour* $\mathscr{T}$ *sur* $[0, P]$ *et si* $E_c(P) \geq E_c(0)$*, alors cet ordonnancement est faisable et cyclique de période P sur une durée infinie.*

Preuve 3.3. *D'après le théorème précèdent, s'il existe un ordonnancement faisable pour* $\mathscr{T}$ *sur* $[0, P]$ *et si* $E_c(P) \geq E_c(0)$*, alors le même ordonnancement faisable existe pour sur* $[P, 2P]$ *tel que* $E_c(2P) \geq E_c(P)$ *et donc tel que* $E_c(2P) \geq E_c(0)$*. Et par récurrence, il existe le même*

ordonnancement faisable sur $[(k-1)P, kP]$ *pour tout k, tel que* $E_c(kP) \geq E_c(0)$. *Donc, il existe un ordonnancement faisable qui est cyclique de période P.* □

Pour montrer qu'un algorithme d'ordonnancement ordonnance fiablement une configuration de tâches, nous simulons la séquence produite par cet ordonnanceur sur la première hyperpériode. Si le niveau d'énergie dans le réservoir à la fin de cette hyperpériode est au moins égale à celui du début de l'hyperpériode, nous savons que la séquence produite sera cyclique. Nous savons aussi qu'à la fin de chaque hyperpériode, le niveau d'énergie dans le réservoir sera au moins égal au niveau d'énergie initial et toujours évidemment au plus égal à la capacité du réservoir.

Par contre si le niveau d'énergie dans le réservoir au terme de la première hyperpériode est inférieur au niveau d'énergie initial alors il faudra continuer la simulation jusqu'à trouver la première hyperpériode $[(k-1)P, kP]$ tel que $E_c(kP) \geq E_c((k-1)P)$. Et de par le premier théorème, nous en déduirons la validité de la séquence sur une durée infinie. Cela prouve donc que dans ce type de système il existe une phase transitoire pendant laquelle le niveau du réservoir peut diminuer jusqu'à se stabiliser.

Exemple 3.3. *Sachant qu'une configuration de deux tâches périodiques ER (Echéance sur Requête) caractérisée par :* $\mathscr{T} = \{\ \tau_i | 1 \leq i \leq 2, avec\ \tau_i = (a_i, C_i, T_i)\ \}$. *Supposons que* $\tau_1 = (0, 1, 2)$ *et* $\tau_2 = (0, 1, 4)$. *Nous notons que* $PPCM(\tau_1, \tau_2) = 4$. *En outre, la puissance de la source d'énergie* $P_r = 6$, *la capacité du réservoir* $C = 12$, *l'énergie initiale du réservoir* $E_c(0) = 8$ *et la puissance du processeur* $P_c = P_{max} = 8$.

Nous utilisons aussi la politique EDF dans cet exemple, l'ordonnanceur laisse le processeur en mode veille pendant une seule unité de temps quand le réservoir d'énergie devient vide. La séquence d'ordonnancement produite est illustrée dans la figure 3.5.

La hyperpériode P est égale à 4. Dans la première hyperpériode $[0, 4]$, la configuration de tâches est bien ordonnancée, et nous constatons que $E_c(4) = E_c(0) = 8$. Dans les hyperpériodes successives, nous obtenons les résultats identiques, $E_c(8) = E_c(4) = E_c(0) = 8$. La trace rouge présente l'évaluation de l'énergie du réservoir sur les points hyperpériodiques.

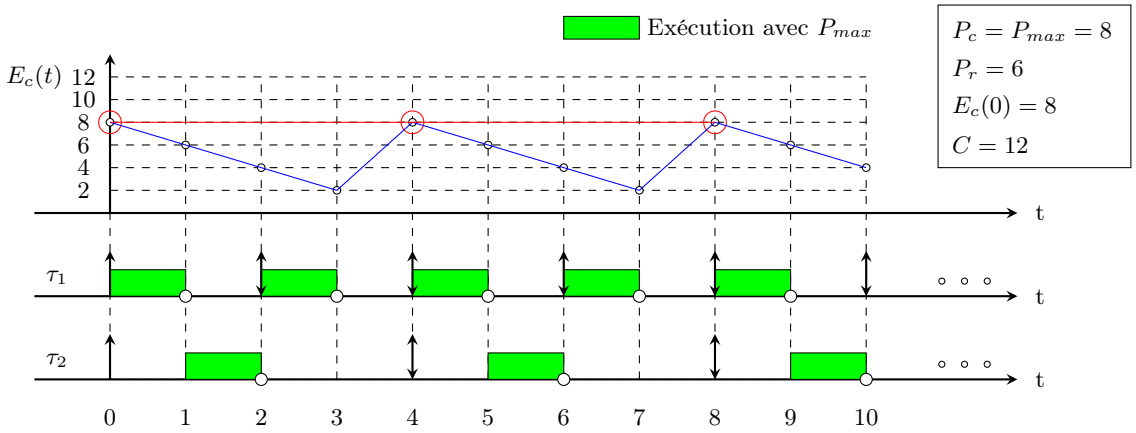


Figure 3.5: – Séquence produite par EDF sous contraintes énergétiques

Exemple 3.4. *Nous reprenons l'exemple 3.3. Mais nous mettons la puissance de la source d'énergie* $P_r = 4$.

Comme la trace rouge, bien que la configuration de tâches est bien ordonnancée dans la première hyperpériode, mais $E_c(4) < E_c(0)$, c'est-à-dire nous utilisons une partie de l'énergie stockée du réservoir comme supplément dans la cas où l'énergie récupérable n'est pas suffisante. Alors, dans ce cas-là, si la puissance de la source d'énergie est constante, la configuration de tâches n'est plus ordonnancée fiablement à cause de la limite d'énergie dans les hyperpériodes prochaines.

Nous voyons que dans cet exemple la tâche τ_1 est bien ordonnancée, mais l'ordonnancement de τ_2 échoue.

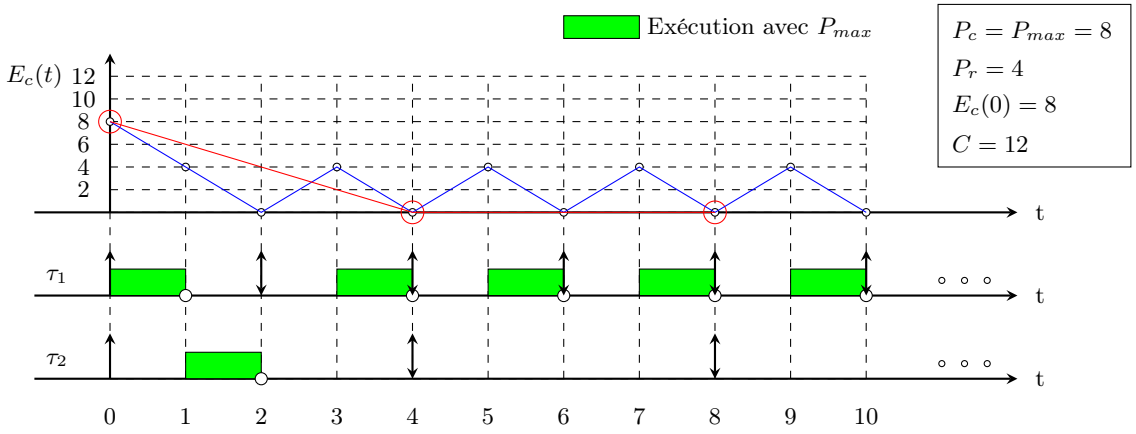


Figure 3.6: – Séquence produite par EDF sous contraintes énergétiques

3.6 Comportement de l'ordonnancement EDF classique

Dans ce paragraphe, nous voulons, sur un exemple simple, montrer qu'un ordonnanceur conventionnel, conduit par la priorité ne peut construire un ordonnancement optimal dans ce contexte.

Considérons l'ordonnanceur EDF qui peut être implémenté sous deux variantes bien distinctes, respectivement appelées ASAP (As Soon As Possible) et ALAP (As Late As Possible). Ces deux variantes ont aussi été appelées dans la littérature EDS (Earliest Deadline as Soon as possible) et EDL(Earliest Deadline as Late as possible) [9, 10].

En l'absence de contraintes énergétiques et lorque le processeur est sous-chargé ($U \leq 1$), l'ordonnanceur EDF est optimal. Son optimalité demeure, y compris pour l'ordonnancement de tâches sporadiques et/ou de tâches apériodiques critiques. Il possède en outre de très grandes qualités : implémentation aisée, exécution rapide (faibles overheads), taux de prémptions réduit.

Dans la plupart des cas, EDF est implémenté comme un ordonnanceur en-ligne : les tâches en attente d'exécution sont rangées selon leur échéance et systématiquement exécutées au plus tôt, c'est-à-dire dès qu'elles possèdent la plus haute priorité parmi celles en attente.

Toutefois, retarder le plus possible l'exécution des tâches tout en garantissant le respect de leurs échéances peut s'avérer indispensable. En effet, cette problématique se rencontre lorsque les tâches périodiques et apériodiques se partagent le même processeur. Nous montrons que dans ces circonstances, retarder au maximum l'exécution des tâches périodiques lors de l'arrivée d'une tâche apériodique permet alors d'exécuter au plus tôt la tâche apériodique et ainsi minimiser son temps de réponse effectif. Dans cette situation, il s'agira donc de calculer la longueur de temps pendant lequel retarder les tâches périodiques à partir de l'instant courant. Dans la suite de ce document, nous appelons laxité (en anglais, Slack time), ce temps maximum pendant lequel nous pouvons laisser le processeur oisif à partir de l'instant courant sans remettre en question le respect des échéances des tâches périodiques.

3.6.1 Faiblesses d'un ordonnancement ASAP par EDF

La variante de EDF qui consiste à appliquer la politique ASAP ou EDS consiste à rendre immédiatement exécutable la tâche prête la plus prioritaire (celle dont l'échéance est la plus proche). Nous illustrons ci-dessous cette politique d'ordonnancement non-oisive.

Exemple 3.5. *Soit une configuration de deux tâches apériodiques critiques caractérisée par :* $\mathscr{T} = \{\ \tau_i | 1 \leq i \leq 2, avec\ \tau_i = (a_i, C_i, D_i)\ \}$. *Supposons que* $\tau_1 = (0, 4, 9)$ *et* $\tau_2 = (2, 3, 3)$.

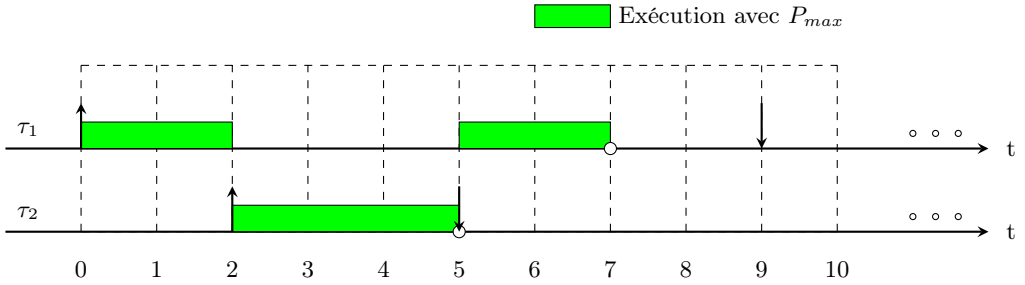


Figure 3.7: – Séquence produite par EDF en ASAP

La figure 3.7 illustre la séquence produite. Dans cet exemple, nous constatons que la configuration de tâches est fiablement ordonnancée car aucune tâche ne viole son échéance. Il existe donc pour celle-ci un ordonnancement temporellement faisable.

Dans l'exemple donné ci-dessus, l'ordonnanceur ASAP fonctionne au mieux car n'ayant à gérer que des limitations d'ordre temporel.

Exemple 3.6. *Afin de vérifier le comportement de l'ordonnanceur ASAP sous contraintes énergétiques, nous reprenons la même configuration de tâches que précédemment. Nous considérons une source d'énergie qui produit avec la même puissance au cours du temps égale à 6 et un réservoir d'énergie de capacité égale à 12 avec une charge initiale* $E_c(0) = 8$ *(cf. figure 3.8).*

A partir de la séquence produite, nous constatons que la configuration de tâches ne peut plus être fiablement ordonnancée, la tâche τ_2 violant son échéance à l'instant 5. Le point faible de cette

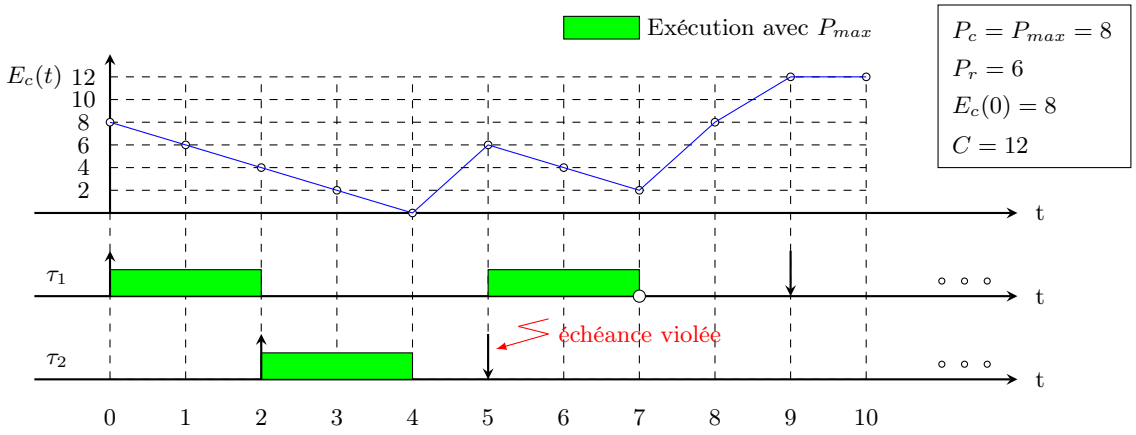


Figure 3.8: – Séquence produite par ASAP sous contraintes énergétiques

variante de EDF est d'être goulu, donc de consommer du temps processeur dés que possible et par voie de conséquence de consommer de l'énergie tant qu'il en existe dans le réservoir conduisant au vidage complet de ce réservoir.

3.6.2 Faiblesses d'un ordonnancement ALAP par EDF

Par symétrie avec la variante ASAP, examinons le comportement de la variante ALAP. Celle-ci consiste à exécuter les tâches le plus tard possible c'est-à-dire dérammer leur exécution le plus tard possible tout en respectant leur échéance. Cette variante s'adapte particulièrement aux situations imposant de mettre le processeur en veille le plus longtemps possible tout en préservant le respect des échéances [9]. L'idée de retarder l'exécution des tâches périodiques critiques au plus tard pour récupérer la disponibilité du processeur (pour libérer des temps creux au plus tôt) est apparue pour ordonnancer conjointement des tâches apériodiques non critiques. En retardant l'exécution des tâches périodiques, à l'instant d'arrivée d'une tâche non critique, nous minimisons ainsi son temps de réponse [10]. ALAP peut être considérée comme une stratégie d'ordonnancement oisive qui laisse délibérément le processeur inactif malgré la présence de tâches en attente d'exécution.

Exemple 3.7. *Nous reprenons la même configuration de tâches que précédemment et nous appliquons la variante ALAP de EDF, c'est-à-dire EDL. (cf. figure 3.9). EDF ordonnance fiablement cette configuration que ce soit en version ASAP ou ALAP.*

Exemple 3.8. *Examinons le comportement de ASAP en présence de contraintes énergétiques. Nous faisons les mêmes hypothèses que dans l'exemple précédent concernant les caractéristiques énergétiques du système.*

La figure 3.10 donne la séquence ASAP. A l'instant d'arrivée de τ_1, sa date de début d'exécution au plus tard par ALAP est calculée et donne 5, égale à son échéance minorée de sa durée d'exécution. Le processeur, en état de veille à partir de zéro ne consomme donc aucune énergie ce qui permet au réservoir de se recharger jusqu'à l'instant 1. A cet instant, le réservoir a atteint

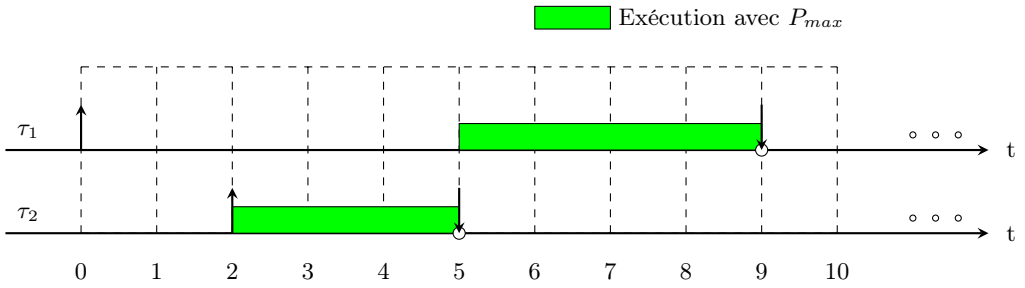


Figure 3.9: – Séquence produite par la variante ALAP de EDF

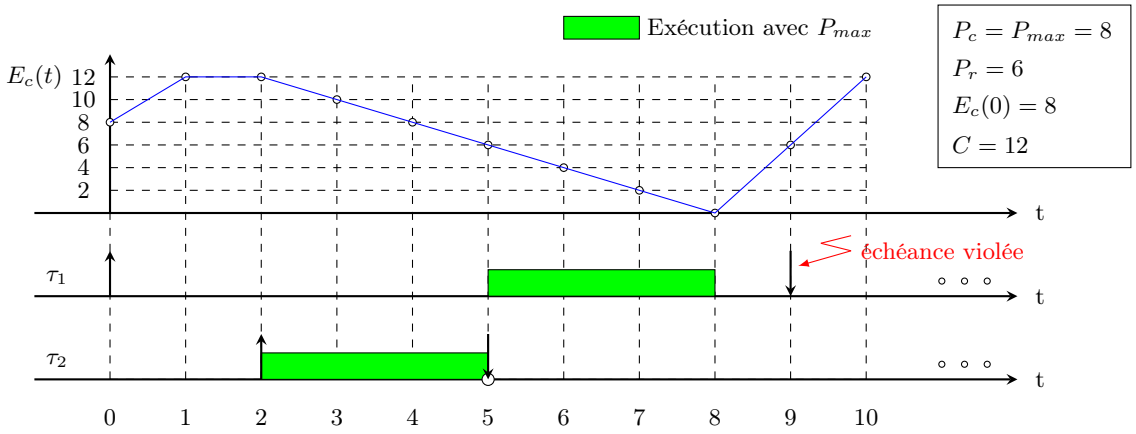


Figure 3.10: – Séquence produite par ALAP sous contraintes énergétiques

sa capacité maximale et donc l'énergie produite en provenance de la source se trouve gaspillée. A l'instant 2, la tâche τ_2 arrive. Sa date de sémarrage au plus tard, compte tenu de sa durée d'exécution correspond à sa date d'arrivée. La tâche τ_2 s'exécute de l'instant 2 à l'instant 5, d'où une décharge linéaire du réservoir qui, à l'instant 5, contient 6 unités d'énergie. τ_2 démarre son exécution conduisant à une décharge de la batterie de 2 unités d'énergie par unité de temps. A l'instant 8, le réservoir est épuisé, d'où l'impossibilité pour le processeur de terminer l'exécution de τ_2 qui viole nécessairement son échéance par insuffisante non pas de temps mais d'énergie.

Cet exemple montre bien que, malgré une suffisance en termes d'énergie et de capacité de traitrement, les ordonnanceurs classiquement utilisés s'avèrent indaptées à faire face aux contraintes énergétiques. Nous pourrions dire que la version ASAP ou EDS est trop goulue puisqu'elle conduit à consommer au plus vite l'énergie et à épuiser le réservoir ignirant l'arrivée d'énergie dans le futur. Au contraire, la version ALAP ou EDL n'est pas assez goulue puisqu'elle conduit à différer sa consommation énergétique d'où un gaspillage de l'énergie du fait de la capacité limitée du réservoir qui ne peut toute l'énergie récupérée mais non consommée. Et cela se traduit ensuite par une famine en énergie ne permettant pas de terminer l'exécution d'une tâche même si cette tâche avait suffisament de temps avant son échéance.

3.7 Conclusion

Dans ce chapitre, nous avons décrit un modèle en vue d'étudier le comportement d'un système embarqué temps réel qui tire son énergie de l'environnement. Ce modèle fait apparaître trois composants de base que sont l'unité de récupération de l'énergie, l'unité de stockage de l'énergie et l'unité de traitement qui consomme cette énergie et qui a de plus des contraintes temporelles à respecter.

Dans ce modèle, nous supposons que les tâches ont des échéances strictes, sont périodiques. Elles consomment une quantité d'énergie proportionnelle à leur durée d'exécution et le facteur de proportionnalité correspond à la puissance de consommation instantanée du processeur. Ce modèle, introduit pour la première fois en 2006, correspond à une extension d'un modèle élémentaire formé de tâches basiques (frame based system, en anglais) proposé initialement en 2001.

De façon à accompagner ce modèle, nous avons introduit une nouvelle terminologie spécifique de l'ordonnancement temps réel sous contraintes de récupération de l'énergie renouvelable. Les concepts de surcharge énergétique, clairvoyance énergétique et demande énergétique nous permettent ainsi de prendre en compte la dimension énergétique du système et pas uniquement la dimension temporelle.

Comme nous nous interessons ici à des systèmes temps réel à contraintes fermes, l'ordonnanceur vise à maximiser le taux d'échéances satisfaites. Toutefois, mettre en place un test d'ordonnançabilité s'avère impératif pour décider de la faisabilité d'une application quelque soit l'ordonnanceur sélectionné. Nous avons donné une condition nécessaire d'ordonnançabilité, simple à tester mais qui suppose de connaître sinon le profil énergétique exact de la source au moins la puissance moyenne délivrée par celle-ci sur la durée de vie de l'application. Puis nous avons montré que lorsqu'il existe un ordonnancement faisable pour une application, celui-ci est cyclique lorsque la source d'énergie produit avec une puissance constante. Nous donnons aussi dans ce contexte, une condition suffisante d'ordonnançabilité qui se ramène à un test de faisabilité réduit à la première hyperpériode.

Enfin, dans une dernière partie nous avons voulu mettre en évidence que les ordonnanceurs classiquement utilisés depuis de nombreuses années, en particulier EDF ne sont plus utilisables que ce soit en ordonnançant les tâches au plus tôt ou au plus tard.

C'est pourquoi, dans le chapitre suivant, nous proposons de décrire en détail un ordonnanceur optimal de type oisif introduit en 2006 dans [32] pour pallier aux déficiences de EDF. Nous en proposerons une analyse détaillée de ses points forts et de ses points faibles avant de donner les résultats d'une étude de performance comparative.

Chapitre 4

L'algorithme d'ordonnancement LSA : de la théorie à l'intégration

4.1 Introduction

Les travaux de recherche portant sur l'ordonnancement de tâches temps réel dans les systèmes alimentés par une source d'énergie renouvelable ont démarré en 2001. L'algorithme proposé alors par Allavena et Mossé que nous avons décrit dans le chapitre précédent apporte une solution efficace mais uniquement lorsque les tâches périodiques ont une période et une échéance communes. D'autre part, il s'agit d'une stratégie d'ordonnancement hors ligne qui fonde ses décisions d'ordonnancement sur les paramètres assignés aux tâches et le profil d'énergie avant leur activation.

Quant aux ordonnancements ASAP et ALAP de EDF, nous avons constaté que ces derniers sont inefficaces dès lors que que le processeur est limité dans son alimentation énergétique. Ils ne sont pas suffisament flexibles pour adapter l'activité du processeur en fonction d'une part des contraintes temporelles des tâches et d'autre part de la quantité d'énergie disponible ou à venir. De plus, ces ordonnanceurs ne prévoient pas comment gérer une situation de famine énergétique qui se caractérise par un réservoir d'énergie vide à un moment donné.

Pour pallier à cette absence d'ordonnanceur adapté, un nouvel algorithme d'ordonnancement monoprocesseur appelé LSA (Lazy Scheduling Algorithm) a été introduit en 2006 par Moser et *al.* de l'Institut Polytechnique de Zurich [33]. Il s'agit là du premier résultat significatif sur cette thématique après les travaux d'Allavena et *al.* de l'Université de Pittsburg.

Dans ce chapitre, nous donnerons d'abord les principes de cet ordonnanceur que nous accompagnerons de nombreuses illustrations. L'objectif de ce chapitre est de mettre en exergue les qualités mais aussi les défauts de LSA, justifiant les travaux de simulation présentés dans le chapitre suivant.

4.2 Principes de LSA

LSA est un algorithme d'ordonnancement en ligne. Il permet d'ordonnancer toute configuration de tâches, qu'elles soient périodiques ou apériodiques critiques, c'est-à-dire munies d'une échéance stricte. A l'arrivée d'une tâche, l'ordonnanceur calcule pour celle-ci, une date dite *date de démarrage*, notée s_i, à partir de laquelle la tâche commencera à s'exécuter sur le processeur

en utilisant sa puissance de consommation maximum. La détermination de cette date est faite de telle sorte que le processeur ne commence à l'exécuter ni trop tôt, ni trop tard. Entre la date d'arrivée et la date de démarrage, le processeur est volontairement laissé en veille pour permettre au réservoir de se recharger autant que possible, voire consommer l'énergie au même rythme que celle-ci est produite dans le cas où le réservoir d'énergie est plein. Cet intervalle de temps de recharge est calculé tel que, lorsque la tâche commencera à s'exécuter, le processeur disposera de suffisament d'énergie pour fonctionner de façon continue et sans risque de famine énergétique jusqu'à la date d'échéance de la tâche. En résumé, nous pouvons considérer que LSA offre un compromis entre ASAP (EDS) et ALAP (EDL), compromis basé uniquement sur la connaissance de l'énergie disponible dans le réservoir à tout instant et sur la connaissance du profil énergétique de la source. En effet, LSA suppose que nous connaissons au moins pour un futur proche, la quantité d'énergie qui pourra être tirée de la source. Nous pouvons donc dire que LSA est un ordonnanceur clairvoyant du point de vue énergétique. Il n'est cependant pas totalement clairvoyant car il ne nécessite pas de connaître a priori les dates d'arrivée des tâches.

4.2.1 Hypothèses

Nous rappelons ci-dessous le modèle précis considéré dans [33]. Ce modèle se compose de :

- Un processeur unique.
- Un réservoir d'énergie rechargeable avec une capacité $C = E_{max} - E_{min}$, cf. figure 3.1. Le taux de charge du réservoir, noté par P_r, correspond à la puissance produite par la source d'énergie environnementale. Nous supposons l'absence totale de pertes, ou bien intégrées déjà dans la valeur de P_r.
- Une configuration de tâches périodiques (ou apériodiques) pouvant être préemptées. Les coûts engendrés par ces préemptions en temps et en énergie sont considérés comme négligeables. Dans ce système, le système de traitement est supposé consommer la même puissance instantanée quelque soit la tâche exécutée. Cela signifie donc que la durée d'exécution et l'énergie consommée par toute tâche sont reliés par un unique facteur, la puissance maximale de consommation du processeur. Donc nous avons $C_i = E_i/P_{max}$.
- Nous supposons que la tâche τ_i peut être exécutée soit avec la puissance maximale du processeur P_{max} (donc la puissance instantanée consommée est constante et notée par $P_c = P_{max}$), soit avec la puissance instantanée de production de l'énergie renouvelable (donc la puissance instantanée consommée est notée par $P_c(t) = P_r(t)$). Ce modèle considère donc que le processeur est idéal en ce sens qu'il peut adapter dynamiquement sa puissance de consommation instantanée.
- A tout moment, la puissance instantanée consommée par une tâche est toujours supérieure ou égale à la puissance produite par la source d'énergie environnementale, c'est-à-dire, $P_{max} \geq P_r(t), \forall t$.

4.2.2 Calcul des dates de début d'exécution s_i

Dans l'exemple illustrant l'algorithme ALAP sous contraintes énergétiques (cf. exemple 3.8), nous avons trouvé que la date de début d'exécution de la tâche τ_1 est donnée par $s_1 = d_1 - C_1 =$

$9 - 4 = 5$. Si l'énergie disponible est toujours supérieure à l'énergie requise par τ_1, cette tâche est terminée juste à son échéance. Mais malheureusement à l'instant 8, le réservoir d'énergie est épuisé. La tâche τ_1 a dû nécessairement être préemptée pendant une unité de temps pour recharger le réservoir. Et alors l'échéance de τ_1 est dépassée après le cycle de charge. Cependant, nous constatons qu'un gaspillage d'énergie apparaît pendant les intervalles de temps d'inactivité du processeur de l'instant 0 à l'instant 2. Pour résoudre ce problème, nous constatons que si nous laissons $s_1 = 1$, alors le viol d'échéance est évité (cf. exemple 4.1).

Exemple 4.1.

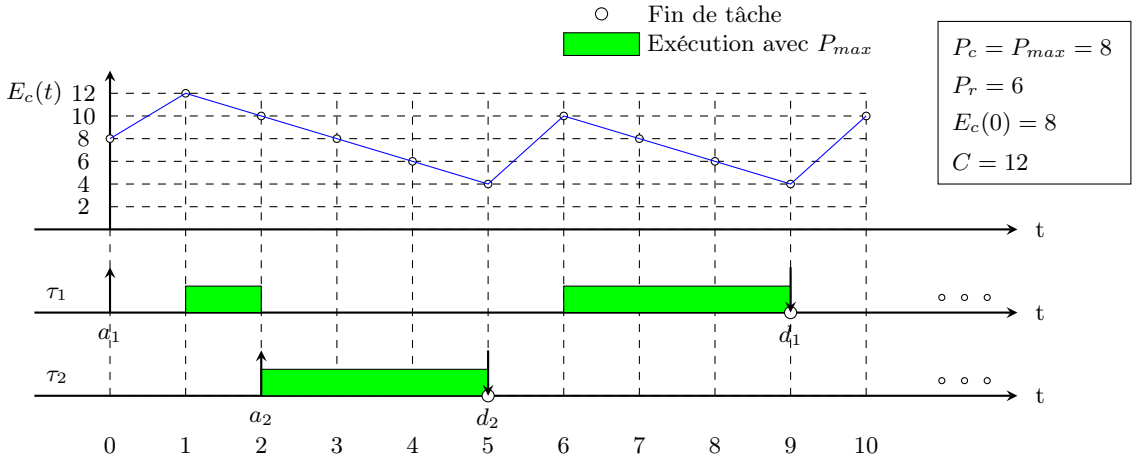


Figure 4.1: – Séquence produite par un ordonnanceur ALAP amélioré

De ce qui précède, nous pouvons en déduire la conclusion suivante : le meilleur choix de la date de début d'exécution de tâche s_i doit assurer que le processeur puisse exécuter continuellement des tâches dans l'intervalle $[s_i, d_i]$ avec la puissance P_{max}, et toute l'énergie stockée doit être totalement consommée à l'instant d_i, c'est-à-dire $E_c(d_i) = 0$. Durant les périodes précédant s_i, le système stocke d'abord de l'énergie autant que possible et une fois le réservoir d'énergie plein, la tâche va être exécutée tout de suite avec la puissance instantanée $P_r(t)$ jusqu'à l'instant s_i. Donc durant cette période, le réservoir reste plein puisque l'énergie est consommée au même rythme que sa production. En utilisant cette technique, nous évitons le gaspillage d'énergie dans la période $[a_i, s_i]$.

Pour déterminer s_i, nous devons d'abord calculer la somme de l'énergie disponible dans l'intervalle $[a_i, d_i]$ donnée par $E_c(a_i) + E_r(a_i, d_i)$. Puis, nous calculons le temps minimum requis pour consommer sans interruption cette quantité d'énergie par l'équation suivante : $\frac{E_c(a_i)+E_r(a_i,d_i)}{P_{max}}$. Et ainsi, nous pouvons déduire la date optimale de début d'exécution de tâche τ_i comme suit :

$$s_i^* = d_i - \frac{E_c(a_i) + E_r(a_i, d_i)}{P_{max}} = d_i - \frac{E_c(a_i) + \int_{a_i}^{d_i} P_r(t) \cdot dt}{P_{max}} \tag{4.1}$$

Toutefois, nous devons prendre en compte le scénario où le réservoir d'énergie est rempli avant cette date s_i^*. Afin d'éviter le débordement d'énergie, il est raisonnable de commencer

immédiatement à exécuter la tâche avec la puissance $P_r(t)$ dès l'instant où le réservoir d'énergie est plein jusqu'à s_i^*. Cela signifie aussi que l'énergie disponible est peut-être épuisée à un instant appartenant à l'intervalle $[s_i^*, d_i]$. Alors dans cette situation, nous pouvons aussi calculer la date de début d'exécution $s_i^{'}$ de la tâche τ_i comme suit :

$$s_i^{'} = d_i - \frac{C + E_r(s_i^{'}, d_i)}{P_{max}} = d_i - \frac{C + \int_{s_i^{'}}^{d_i} P_r(t) \cdot dt}{P_{max}} \tag{4.2}$$

Il s'ensuit que la date de début d'exécution de tâche s_i est donnée par la plus grande des valeurs données précédemment, soit :

$$s_i = max(s_i^*, s_i^{'}) \tag{4.3}$$

Nous pouvons combiner les équations 4.1, 4.2, 4.3. La date s_i peut être exprimée aussi par une autre forme plus simplifié comme suit :

$$\begin{aligned} s_i &= d_i - \frac{min\left[E_c(a_i) + E_r(a_i, d_i), C + E_r(s_i, d_i)\right]}{P_{max}} \\ &= d_i - \frac{min\left[E_c(a_i) + \int_{a_i}^{d_i} P_r(t) \cdot dt, C + \int_{s_i}^{d_i} P_r(t) \cdot dt\right]}{P_{max}} \end{aligned} \tag{4.4}$$

Nous pouvons donc donner synthétiquement, les deux règles de base sur lesquelles se fonde l'ordonnanceur LSA :

Règle 4.1. L'algorithme EDF est appelé à l'instant t pour choisir la tâche plus prioritaire (celle avec la plus proche échéance) parmi les tâches prêtes. Pour tout instant t tel que $s_i \leq t$, la tâche est exécutée avec la puissance $P_c(t) = P_{max}$.

Règle 4.2. Si le réservoir d'énergie devient plein ($E_c(t) = C$) avant s_i, la tâche s'exécute sur le processeur avec la puissance de consommation instantanée, $P_c(t) = P_r(t)$.

4.3 Considérations d'implémentation

4.3.1 Pseudo-code de l'algorithme LSA

Le pseudo-code de l'algorithme d'ordonnancement LSA est fourni ci-dessous dans le tableau 4.1. Il représente l'ensemble des opérations mises en oeuvre pour construire en-ligne la séquence d'ordonnancement à chaque instant.

Ci-dessous, nous donnons les principales opérations à effectuer par l'ordonnanceur LSA :

- Tout d'abord, parmi les tâches arrivées (réveillées), l'ordonnanceur sélectionne la tâche la plus urgente dans la liste des tâches prêtes ordonnée selon EDF.
- A son arrivée, nous calculons la date optimale de début d'exécution s_j de tâche τ_j.
- Dès que le réservoir d'énergie est plein, la tâche s'exécute avec la puissance instantanée de récupération d'énergie $P_r(t)$.
- Pendant la période $t \geq s_j$, la tâche τ_j s'exécute avec la puissance maximale du processeur P_{max}.
- Une fois que la tâche τ_j se termine, elle est enlevée de la liste $\mathscr{L}$.

Donnée d'entrée : une liste à jour des tâches prêtes à s'exécuter, notée $\mathscr{L}$.
Résultat de sortie : la séquence LSA.

$P_c(t) \leftarrow 0$;
Tant que (liste $\mathscr{L}$ non vide) **faire**
  $d_j \leftarrow min\{d_i : \tau_i \in \mathscr{L}\}$;
  calculer s_j ;
  exécuter la tâche τ_j avec la puissance $P_c(t)$;
  $t \leftarrow$ instant courant ;
  Si $(t = a_k)$ **Alors**
    ajouter la tâche τ_k dans la liste $\mathscr{L}$;
  Fin Si
  Si $(t = f_j)$ **Alors**
    supprimer la tâche τ_j de la liste $\mathscr{L}$;
  Fin Si
  Si $(E_c(t) = C)$ **Alors**
    $P_c(t) \leftarrow P_r(t)$;
  Fin Si
  Si $(t \geq s_j)$ **Alors**
    $P_c(t) \leftarrow P_{max}$;
  Fin Si
Fait

Tableau 4.1: – Pseudo-code de l'algorithme LSA [33]

4.3.2 Exemples illustratifs

4.3.2.1 Etude d'un système sous-chargé

Examinons dans ce paragraphe, un exemple de fonctionnement de l'ordonnanceur LSA lorsqu'appliqué sur une configuration de tâches ordonnançable, c'est-à-dire à la fois en sous-charge de traitement et en sous-charge énergétique.

Exemple 4.2. *Soit une configuration de deux tâches apériodiques caractérisée par :* $\mathscr{T} = \{ \tau_i | 1 \leq i \leq 2, avec\ \tau_i = (a_i, E_i, D_i) \}$. *Supposons que* $\tau_1 = (1, 24, 9)$ *et* $\tau_2 = (5, 8, 8)$. *Nous supposons que le réservoir d'énergie a une capacité de 10 unités d'énergie (*$C = 10$*) et que la source d'énergie émet à puissance constante égale à 4 unités d'énergie (*$P_r = 4$*). De plus, nous supposons que la puissance de consommation instantanée du processeur est égale à 8 unités d'énergie (*$P_{max} = 8$*).*

D'abord, selon les équations 4.1, 4.2, 4.3, calculons la date de démarrage s_1 pour la tâche τ_1 à t=1 :

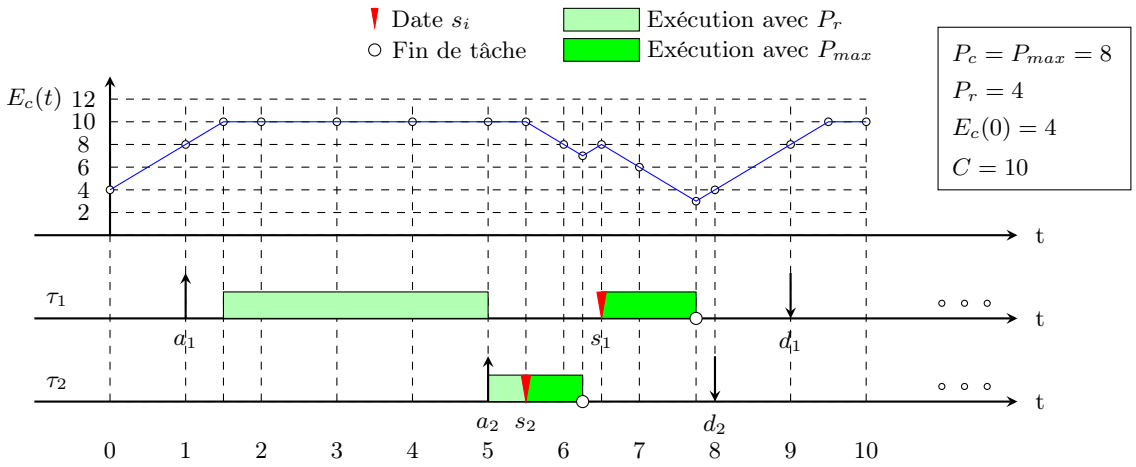


Figure 4.2: – Séquence LSA pour une configuration de tâches apériodiques ordonnançable

$$
\begin{aligned}
s_1^* &= d_1 - \frac{E_c(a_1) + E_r(a_1, d_1)}{P_{max}} = d_1 - \frac{E_c(a_1) + \int_{a_1}^{d_1} P_r \cdot dt}{P_{max}} \\
&= 9 - \frac{8 + \int_1^9 4 \cdot dt}{8} = 9 - \frac{8 + 4 \cdot (9-1)}{8} = 4 \\
s_1^{'} &= d_1 - \frac{C + E_r(s_1^{'}, d_1)}{P_{max}} = d_1 - \frac{C + \int_{s_1^{'}}^{d_1} P_r \cdot dt}{P_{max}} \\
s_1^{'} &= 9 - \frac{10 + \int_{s_1^{'}}^{9} 4 \cdot dt}{8} = 9 - \frac{10 + 4 \cdot (9 - s_1^{'})}{8}
\end{aligned}
\tag{4.5}
$$

Nous en déduisons que $s_1^{'} = 6.5$. Et donc $s_1 = max(s_1^*, s_1^{'}) = max(4, 6.5) = 6.5$.

Alors, nous en déduisons que dans l'intervalle [1, 1.5], le processeur reste en mode veille pour charger le réservoir. Une fois le réservoir d'énergie plein, le processeur commence à exécuter la tâche τ_1 dans [1.5, 5] avec la puissance P_r=4.

Puis, à t=5, la tâche τ_2 arrive, engendrant le calcul de s_2 :

$$
\begin{aligned}
s_2^* &= d_2 - \frac{E_c(a_2) + E_r(a_2, d_2)}{P_{max}} = d_2 - \frac{E_c(a_2) + \int_{a_2}^{d_2} P_r \cdot dt}{P_{max}} \\
&= 8 - \frac{10 + \int_5^8 4 \cdot dt}{8} = 8 - \frac{10 + 4 \cdot (8-5)}{8} = 5.25 \\
s_2^{'} &= d_2 - \frac{C + E_r(s_2^{'}, d_2)}{P_{max}} = d_2 - \frac{C + \int_{s_2^{'}}^{d_2} P_r \cdot dt}{P_{max}} \\
s_2^{'} &= 8 - \frac{10 + \int_{s_2^{'}}^{8} 4 \cdot dt}{8} = 8 - \frac{10 + 4 \cdot (8 - s_2^{'})}{8}
\end{aligned}
\tag{4.6}
$$

Nous en déduisons que $s_2^{'} = 5.5$. Et donc $s_2 = max(s_2^*, s_2^{'}) = max(5.25, 5.5) = 5.5$.

Alors, puisque le réservoir d'énergie est toujours plein depuis t=1.5, donc dans [5, 5.5], les tâches sont exécutées avec la puissance instantanée P_r=4. A partir de s_2=5.5, τ_2 est exécutée avec P_{max}=8 jusqu'à la fin, c'est-à-dire jusqu'à t=6.25.

A t=6.25, la tâche τ_2 se termine. Donc nous relançons la tâche τ_1 avant t=6.5, car le réservoir d'énergie n'est pas plein. Donc le processeur doit rester inactif de façon à permettre la recharge d'énergie jusqu'à t=6.5. A partir de t=6.5, la tâche τ_1 est exécutée avec P_{max} jusqu'à la fin à t=7.75.

Cet exemple montre que LSA n'utilise pas dans ses décisions d'ordonnancement, la connaissance de la durée d'exécution ou de la consommation énergétique des tâches.

Nous donnons ci-dessous un exemple illustratif de LSA pour une configuration de tâches périodiques.

Exemple 4.3. *Nous nous intéressons ici à une configuration de tâches périodiques ER (Echéance sur Requête) caractérisée par :* $\mathscr{T} = \{\ \tau_i | 1 \leq i \leq 2, avec\ \tau_i = (a_i, E_i, T_i)\ \}$. *Supposons que* $\tau_1 = (0, 24, 10)$ *et* $\tau_2 = (0, 8, 5)$. *Nous notons que* $PPCM(\tau_1, \tau_2) = 10$.

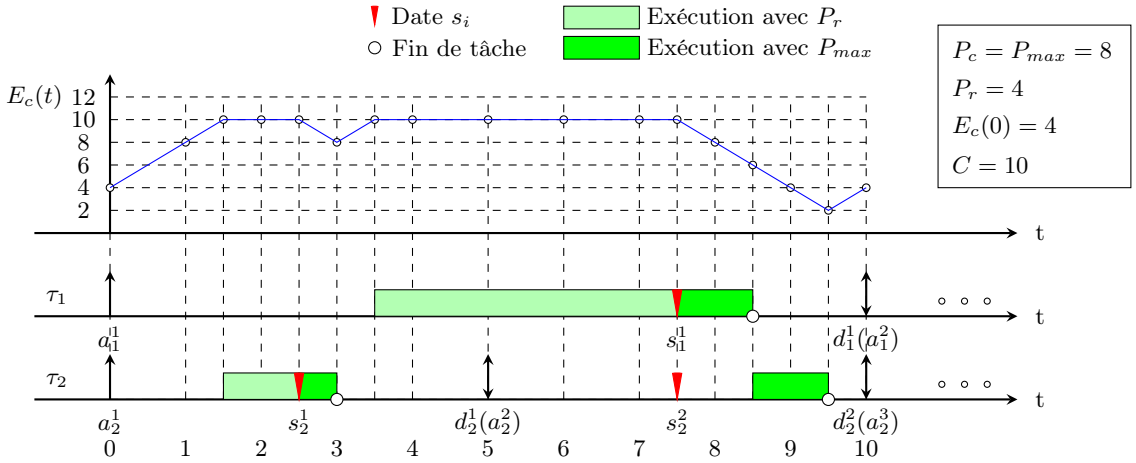


Figure 4.3: – Séquence LSA pour une configuration de tâches périodiques ordonnançable

Nous constatons que le taux d'utilisation processeur de la configuration est :

$$\begin{aligned} U_p &= \sum_{i=1}^{2} \frac{C_i}{T_i} = \sum_{i=1}^{2} \frac{E_i}{T_i} \cdot \frac{1}{P_{max}} \\ &= \sum \frac{E_1}{T_1} \cdot \frac{1}{P_{max}} + \frac{E_2}{T_2} \cdot \frac{1}{P_{max}} = \sum \frac{24}{10} \cdot \frac{1}{8} + \frac{8}{5} \cdot \frac{1}{8} = \frac{1}{2} < 1 \end{aligned}$$

Nous constatons aussi que la puissance moyenne consommée par $\mathscr{T}$ est donnée par :

$$\sum_{i=1}^{2} \frac{E_i}{T_i} = \frac{24}{10} + \frac{8}{5} = \frac{40}{10} = 4$$

Comme $P_r = 4$, nous constatons que la condition nécessaire de faisabilité est satisfaite.

A l'instant t=0, les instances τ_1^1 et τ_2^1 sont réveillées simultanément. Comme $d_1^1 > d_2^1$, la tâche τ_2^1 est plus prioritaire que τ_1^1. D'abord, nous calculons s_1^1 et s_2^1, les dates de démarrage pour les

instances réveillées à t=0 :

$$\begin{aligned} s_1^{1*} &= d_1^1 - \frac{E_c(a_1^1) + E_r(a_1^1, d_1^1)}{P_{max}} = d_1^1 - \frac{E_c(a_1^1) + \int_{a_1^1}^{d_1^1} P_r \cdot dt}{P_{max}} \\ &= 10 - \frac{4 + \int_0^{10} 4 \cdot dt}{8} = 10 - \frac{4 + 4 \cdot (10 - 0)}{8} = 4.5 \\ s_1^{1'} &= d_1^1 - \frac{C + E_r(s_1^{1'}, d_1^1)}{P_{max}} = d_1^1 - \frac{C + \int_{s_1^{1'}}^{d_1^1} P_r \cdot dt}{P_{max}} \\ s_1^{1'} &= 10 - \frac{10 + \int_{s_1^{1'}}^{10} 4 \cdot dt}{8} = 10 - \frac{10 + 4 \cdot (10 - s_1^{1'})}{8} \end{aligned} \tag{4.7}$$

Nous en déduisons que $s_1^{1'} = 7.5$. Et donc $s_1^1 = max(s_1^{1*}, s_1^{1'}) = max(4.5, 7.5) = 7.5$.

$$\begin{aligned} s_2^{1*} &= d_2^1 - \frac{E_c(a_2^1) + E_r(a_2^1, d_2^1)}{P_{max}} = d_2^1 - \frac{E_c(a_2^1) + \int_{a_2^1}^{d_2^1} P_r \cdot dt}{P_{max}} \\ &= 5 - \frac{4 + \int_0^{5} 4 \cdot dt}{8} = 5 - \frac{4 + 4 \cdot (5 - 0)}{8} = 2 \\ s_2^{1'} &= d_2^1 - \frac{C + E_r(s_2^{1'}, d_2^1)}{P_{max}} = d_2^1 - \frac{C + \int_{s_2^{1'}}^{d_2^1} P_r \cdot dt}{P_{max}} \\ s_2^{1'} &= 5 - \frac{10 + \int_{s_2^{1'}}^{5} 4 \cdot dt}{8} = 5 - \frac{10 + 4 \cdot (5 - s_2^{1'})}{8} \end{aligned} \tag{4.8}$$

Nous en déduisons que $s_2^{1'} = 2.5$. Et donc $s_2^1 = max(s_2^{1*}, s_2^{1'}) = max(2, 2.5) = 2.5$.

Alors, dans l'intervalle [0, 1.5], le processeur reste inactif de façon à permettre la recharge d'énergie. A partir de t=1.5, comme $1.5 < s_2^1$, τ_2^1 s'exécute avec la puissance P_r=4. Quand $t \geq s_2^1$, elle est exécutée en pleine vitesse avec P_{max}=8. A l'instant 3, s_2^1 est terminée et $E_c(3) = 8$. Pendant les périodes précédant s_i l'objectif principal est d'accumuler autant d'énergie que possible, donc le processeur retourne dans le mode inactif après l'instant 3. Quand le réservoir devient plein, le processeur commence à exécuter τ_1^1 avec P_r. A t=5, la nouvelle instance τ_2^2 arrive, provoquant le calcul de s_2^2 :

$$\begin{aligned} s_2^{2*} &= d_2^2 - \frac{E_c(a_2^2) + E_r(a_2^2, d_2^2)}{P_{max}} = d_2^2 - \frac{E_c(a_2^2) + \int_{a_2^2}^{d_2^2} P_r \cdot dt}{P_{max}} \\ &= 10 - \frac{10 + \int_5^{10} 4 \cdot dt}{8} = 10 - \frac{10 + 4 \cdot (10 - 5)}{8} = 6.25 \\ s_2^{2'} &= d_2^2 - \frac{C + E_r(s_2^{2'}, d_2^2)}{P_{max}} = d_2^2 - \frac{C + \int_{s_2^{2'}}^{d_2^2} P_r \cdot dt}{P_{max}} \\ s_2^{2'} &= 10 - \frac{10 + \int_{s_2^{2'}}^{10} 4 \cdot dt}{8} = 10 - \frac{10 + 4 \cdot (10 - s_2^{2'})}{8} \end{aligned} \tag{4.9}$$

Nous en déduisons que $s_2^{2'} = 7.5$. Et donc $s_2^2 = max(s_2^{2*}, s_2^{2'}) = max(6.25, 7.5) = 7.5$.

Comme $d_1^1 = d_2^2 = 10$ et $s_1^1 = s_2^2 = 7.5$, ces deux tâches sont de même priorité. Afin de réduire le temp de changement de contexte, le processeur toujours exécute l'instance τ_1^1 avec P_r jusqu'à

l'insant s_1^1. Dès t=s_1^1, le processeur exécute les instances en pleine vitesse avec P_{max}=8. τ_1^1 et τ_2^2 se terminent successivement aux instants 8.5 et 9.5.

4.3.2.2 Etude d'un système surchargé

Examinons dans ce paragraphe, un exemple de fonctionnement de l'ordonnanceur LSA lorsqu'appliqué à une configuration de tâches qui n'est pas ordonnançable.

Exemple 4.4. *Dans cet exemple, nous reprenons un exemple similaire à l'exemple 4.2, mais nous supposons que :* $\tau_1 = (1, 24, 9)$ *et* $\tau_2 = (5, 20, 8)$.

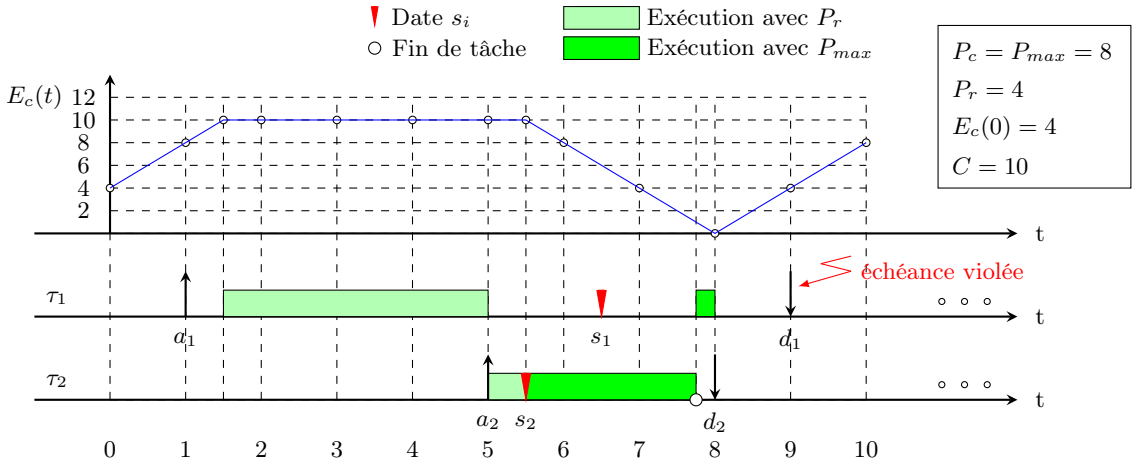


Figure 4.4: – Séquence LSA pour une configuration de tâches apériodiques non ordonnançable

Sur les équations du calcul de s_i (cf. équations 4.1 et4.2), nous pouvons déduire que la valeur de s_i dépend des paramètres suivants : a_i, d_i, $E_c(a_i)$, $P_r(t)$, P_{max} et C. Ce calcul ne fait pas intervenir le paramètre E_i, l'énergie consommée par tâche. Donc dans cet exemple, le résultat des calculs de s_i reste identique à celui de l'exemple 4.2.

$$
\begin{aligned}
s_1^* &= d_1 - \frac{E_c(a_1) + E_r(a_1, d_1)}{P_{max}} = d_1 - \frac{E_c(a_1) + \int_{a_1}^{d_1} P_r \cdot dt}{P_{max}} \\
&= 9 - \frac{8 + \int_1^9 4 \cdot dt}{8} = 9 - \frac{8 + 4 \cdot (9-1)}{8} = 4 \\
s_1^{'} &= d_1 - \frac{C + E_r(s_1^{'}, d_1)}{P_{max}} = d_1 - \frac{C + \int_{s_1^{'}}^{d_1} P_r \cdot dt}{P_{max}} \\
s_1^{'} &= 9 - \frac{10 + \int_{s_1^{'}}^9 4 \cdot dt}{8} = 9 - \frac{10 + 4 \cdot (9 - s_1^{'})}{8}
\end{aligned}
\tag{4.10}
$$

Nous en déduisons que $s_1^{'} = 6.5$. Et donc $s_1 = max(s_1^*, s_1^{'}) = max(4, 6.5) = 6.5$.

$$
\begin{aligned}
s_2^* &= d_2 - \frac{E_c(a_2) + E_r(a_2, d_2)}{P_{max}} = d_2 - \frac{E_c(a_2) + \int_{a_2}^{d_2} P_r \cdot dt}{P_{max}} \\
&= 8 - \frac{10 + \int_5^8 4 \cdot dt}{8} = 8 - \frac{10 + 4 \cdot (8-5)}{8} = 5.25 \\
s_2^{'} &= d_2 - \frac{C + E_r(s_2^{'}, d_2)}{P_{max}} = d_2 - \frac{C + \int_{s_2^{'}}^{d_2} P_r \cdot dt}{P_{max}} \\
s_2^{'} &= 8 - \frac{10 + \int_{s_2^{'}}^8 4 \cdot dt}{8} = 8 - \frac{10 + 4 \cdot (8 - s_2^{'})}{8}
\end{aligned}
\tag{4.11}
$$

Nous en déduisons que $s_2^{'} = 5.5$. Et donc $s_2 = max(s_2^*, s_2^{'}) = max(5.25, 5.5) = 5.5$.

Alors, dans [1, 1.5], le processeur reste en mode veille pour permettre la recharge d'énergie. Une fois le réservoir d'énergie plein, depuis t=1.5, le processeur commence à exécuter la tâche τ_1 avec la puissance P_r=4 (cf. figure 4.4).

Puisque le réservoir d'énergie est toujours plein depuis t=1.5 jusqu'à t=5, donc quand la tâche τ_2 est activée, le processeur continue à l'exécuter avec P_r=4. Depuis t=s_2=5.5, elle s'exécute avec P_{max}=8 jusqu'à la fin.

A l'instant t=7.75, la tâche τ_2 se termine. Donc le processeur commute de l'exécution de τ_2 à l'exécution de τ_1. τ_1 est exécutée avec P_{max}, car t=7.75 > s_1.

S'il n'y avait pas de contrainte d'énergie, la tâche τ_1 serait exécutée continuellement avec P_{max} pendant 1.25 ($(E_1 - P_r \cdot 3.5)/P_{max}$=1.25) unités de temps pour terminer à t=9. Mais malheureusement, quand la tâche τ_2 se termine à t=7.75, il reste très peu d'énergie dans le réservoir d'énergie. Cette quantité s'avère insuffisante pour l'exécution fiable de la tâche τ_1. Donc quand le réservoir d'énergie est épuisé à t=8, le processeur doit rester en mode veille pour permettre la recharge d'énergie. Malgré cette recharge, il reste 8 unités d'énergie qui ne sont pas exécutées. Evidemment l'échéance de la tâche τ_1 sera violée. (cf. figure 4.4).

Exemple 4.5. *Dans cet exemple, nous nous intéressons à une configuration de tâches périodiques ER caractérisée par :* $\mathscr{T} = \{\ \tau_i | 1 \leq i \leq 2, avec\ \tau_i = (a_i, E_i, T_i)\ \}$. *Supposons que* $\tau_1 = (0, 24, 10)$ *et* $\tau_2 = (0, 16, 5)$.

Nous constatons que le taux d'utilisation processeur est :

$$
\begin{aligned}
U_p &= \sum_{i=1}^{2} \frac{C_i}{T_i} = \sum_{i=1}^{2} \frac{E_i}{T_i} \cdot \frac{1}{P_{max}} \\
&= \sum \frac{E_1}{T_1} \cdot \frac{1}{P_{max}} + \frac{E_2}{T_2} \cdot \frac{1}{P_{max}} = \sum \frac{24}{10} \cdot \frac{1}{8} + \frac{16}{5} \cdot \frac{1}{8} = \frac{7}{10} < 1
\end{aligned}
$$

Nous constatons aussi que la puissance moyenne consommée par $\mathscr{T}$ est donnée par :

$$\sum_{i=1}^{2} \frac{E_i}{T_i} = \frac{24}{10} + \frac{16}{5} = \frac{56}{10} = 5.6$$

Comme $P_r = 4$, nous constatons que la condition nécessaire de faisabilité n'est pas satisfaite. Examinons comment se comporte l'ordonnanceur LSA en telle circonstance c'est-à-dire à quoi aboutit le calcul des dates s_i et quelle séquence est produite sur les tâches.

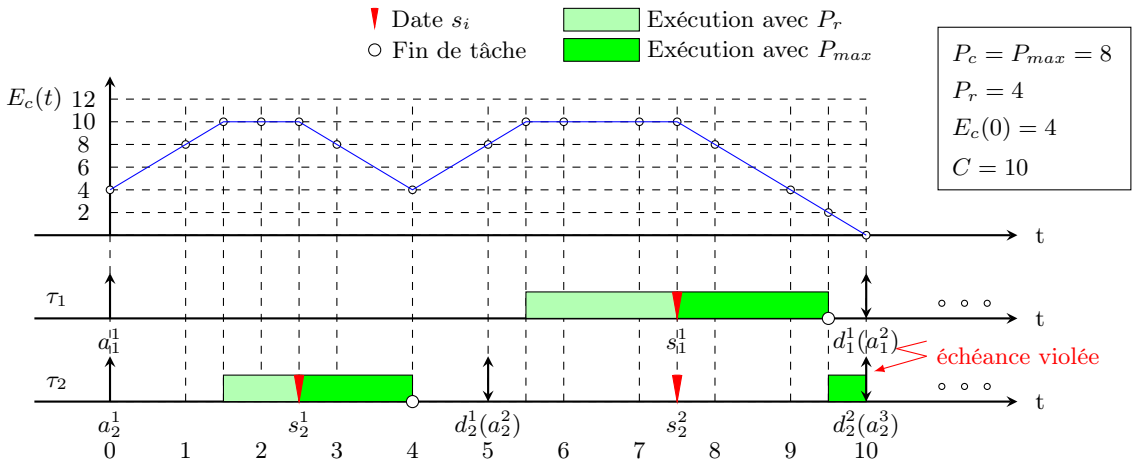


Figure 4.5: – Séquence LSA pour une configuration de tâches périodiques non ordonnançable

Dans cet exemple, le calcul des dates de début d'exécution s_i est similaire à celui de l'exemple 4.3. Nous calculons d'abord s_1^1 et s_2^1 par :

$$\begin{aligned}
s_1^{1*} &= d_1^1 - \frac{E_c(a_1^1) + E_r(a_1^1, d_1^1)}{P_{max}} = d_1^1 - \frac{E_c(a_1^1) + \int_{a_1^1}^{d_1^1} P_r \cdot dt}{P_{max}} \\
&= 10 - \frac{4 + \int_0^{10} 4 \cdot dt}{8} = 10 - \frac{4 + 4 \cdot (10 - 0)}{8} = 4.5 \\
s_1^{1'} &= d_1^1 - \frac{C + E_r(s_1^{1'}, d_1^1)}{P_{max}} = d_1^1 - \frac{C + \int_{s_1^{1'}}^{d_1^1} P_r \cdot dt}{P_{max}} \\
s_1^{1'} &= 10 - \frac{10 + \int_{s_1^{1'}}^{10} 4 \cdot dt}{8} = 10 - \frac{10 + 4 \cdot (10 - s_1^{1'})}{8}
\end{aligned} \tag{4.12}$$

Nous en déduisons que $s_1^{1'} = 7.5$. Et donc $s_1^1 = max(s_1^{1*}, s_1^{1'}) = max(4.5, 7.5) = 7.5$.

$$\begin{aligned}
s_2^{1*} &= d_2^1 - \frac{E_c(a_2^1) + E_r(a_2^1, d_2^1)}{P_{max}} = d_2^1 - \frac{E_c(a_2^1) + \int_{a_2^1}^{d_2^1} P_r \cdot dt}{P_{max}} \\
&= 5 - \frac{4 + \int_0^{5} 4 \cdot dt}{8} = 5 - \frac{4 + 4 \cdot (5 - 0)}{8} = 2 \\
s_2^{1'} &= d_2^1 - \frac{C + E_r(s_2^{1'}, d_2^1)}{P_{max}} = d_2^1 - \frac{C + \int_{s_2^{1'}}^{d_2^1} P_r \cdot dt}{P_{max}} \\
s_2^{1'} &= 5 - \frac{10 + \int_{s_2^{1'}}^{5} 4 \cdot dt}{8} = 5 - \frac{10 + 4 \cdot (5 - s_2^{1'})}{8}
\end{aligned} \tag{4.13}$$

Nous en déduisons que $s_2^{1'} = 2.5$. Et donc $s_2^1 = max(s_2^{1*}, s_2^{1'}) = max(2, 2.5) = 2.5$.

La séquence produite par LSA est illustrée sur la figure 4.5. Avant s_2^1=2.5, le processeur reste en veille jusqu'à t=1.5. Dans [1.5,2.5], la tâche s'exécute avec la puissance P_r. Dès l'instant 2.5, elle s'exécute avec P_{max}. Quand τ_2^1 se termine, le réservoir d'énergie n'est plus plein. Il se charge

encore avant la prochaine date de début d'exécution. A l'instant 5, l'instance τ_2^2 est réveillée, et nous calculons s_2^2 de la façon suivante :

$$\begin{aligned} s_2^{2*} &= d_2^2 - \frac{E_c(a_2^2) + E_r(a_2^2, d_2^2)}{P_{max}} = d_2^2 - \frac{E_c(a_2^2) + \int_{a_2^2}^{d_2^2} P_r \cdot dt}{P_{max}} \\ &= 10 - \frac{8 + \int_5^{10} 4 \cdot dt}{8} = 10 - \frac{8 + 4 \cdot (10-5)}{8} = 6.5 \\ s_2^{2'} &= d_2^2 - \frac{C + E_r(s_2^{2'}, d_2^2)}{P_{max}} = d_2^2 - \frac{C + \int_{s_2^{2'}}^{d_2^2} P_r \cdot dt}{P_{max}} \\ s_2^{2'} &= 10 - \frac{10 + \int_{s_2^{2'}}^{10} 4 \cdot dt}{8} = 10 - \frac{10 + 4 \cdot (10 - s_2^{2'})}{8} \end{aligned} \tag{4.14}$$

Nous en déduisons que $s_2^{2'} = 7.5$. Et donc $s_2^2 = max(s_2^{2*}, s_2^{2'}) = max(6.5, 7.5) = 7.5$.

Comme $d_1^1 = d_2^2 = 10$ et $s_1^1 = s_2^2 = 7.5$, ces deux instances sont de même priorité. Dans cet exemple, nous traitons d'abord τ_1^1. Alors quand le réservoir devient plein à l'instant t=5.5, d'après la règle LSA, τ_1^1 s'exécute avec P_r jusqu'à l'instant 7.5. A partir de l'instant 7.5, elle s'exécute avec P_{max} et se termine à l'instant t=9.5. A partir de t=9.5, τ_2^2 commence à s'exécuter, mais malheureusement il n'y a pas assez de temps, et son échéance est violée.

4.3.3 Propriétés

Quand nous observons le processus de calcul des dates s_i, nous constatons que ce calcul peut être simplifié dans certaines situations comme indiqué ci-après.

Propriété 4.1. *Soit deux tâches apériodiques τ_j et τ_k, $j, k \in N$, tel que $a_k \geq s_j$ et $d_k \leq d_j$, alors $s_k = a_k$.*

Preuve 4.1. *τ_k a une priorité supérieure à celle de τ_j car d'échéance inférieure. La tâche τ_j étant active à l'instant a_k, il est évident que $s_j < a_k$. Il s'ensuit par le calcul de s_j que le processeur peut rester actif continuellement entre s_j et d_j. Il peut donc rester actif continuellement entre a_k et d_k. Donc τ_k doit nécessairement préempter τ_j à l'instant a_k. Il s'ensuit que le démarrage de τ_k correspond à l'instant de son arrivée, soit a_k.* □

Cette propriété concerne la situation où une tâche apériodique arrive après le démarrage d'une autre tâche déjà présente mais moins prioritaire. Dans ce cas, il est inutile d'effectuer le calcul de la date de démarrage de cette tâche car celle-ci sera égale à sa date d'arrivée. En effet, le démarrage d'une tâche a lieu des lors que le système a siffisamment d'énergie pour laisser le processeur actif avec la puissance P_{max} jusqu'à l'échéance de la tâche. Comme ici $d_k \leq d_j$, alors il s'ensuit que le processeur pourra aussi être actif jusqu'à d_k puisqu'il peut l'être jusqu'à d_j. D'où s_k correspond à l'instant courant d'arrivée de τ_k. Dès son arrivée, la nouvelle tâche arrivante préemptera celle en cours d'exécution. Ce constat permet ainsi de limiter les surcoûts de mise en oeuvre de l'ordonnanceur LSA.

Nous préconisons donc, dès lors qu'une tâche arrive, de voir si elle devient la tâche la plus prioritaire parmi celle de la liste des tâches prêtes. Et si oui, ne pas lancer le calcul de sa date de démarrage.

Exemple 4.6. *Supposons deux tâches apériodiques* $\tau_1 = (1, 32, 9)$ *et* $\tau_2 = (7, 8, 9)$ *avec* $P_{max} = 8$, $P_r = 4$ *et un réservoir d'énergie de capacité égale à 10.*

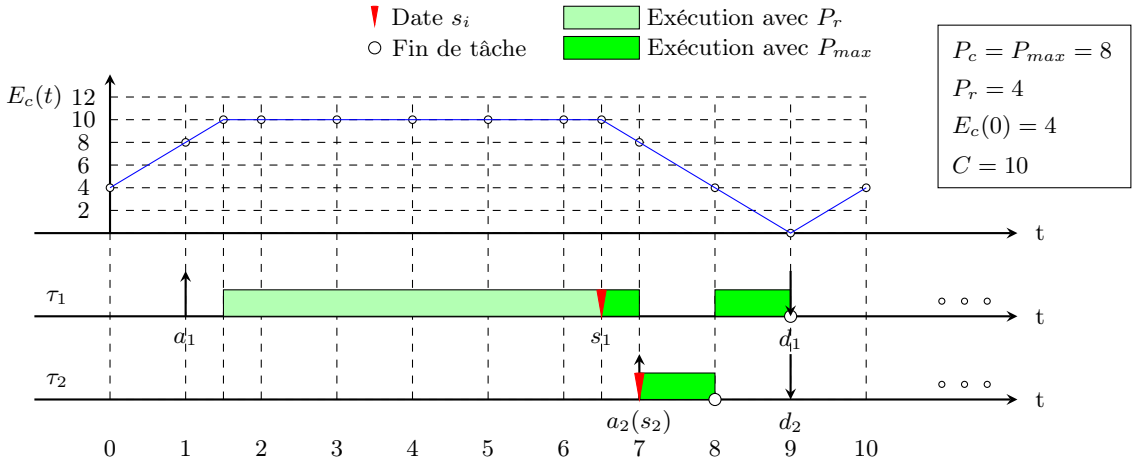


Figure 4.6: – Séquence LSA

Nous calculons d'abord s_1 par :

$$\begin{aligned} s_1^* &= d_1 - \frac{E_c(a_1) + E_r(a_1, d_1)}{P_{max}} = d_1 - \frac{E_c(a_1) + \int_{a_1}^{d_1} P_r \cdot dt}{P_{max}} \\ &= 9 - \frac{8 + \int_1^9 4 \cdot dt}{8} = 9 - \frac{8 + 4 \cdot (9-1)}{8} = 4 \\ s_1^{'} &= d_1 - \frac{C + E_r(s_1^{'}, d_1)}{P_{max}} = d_1 - \frac{C + \int_{s_1^{'}}^{d_1} P_r \cdot dt}{P_{max}} \\ s_1^{'} &= 9 - \frac{10 + \int_{s_1^{'}}^9 4 \cdot dt}{8} = 9 - \frac{10 + 4 \cdot (9 - s_1^{'})}{8} \end{aligned} \tag{4.15}$$

Nous en déduisons que $s_1^{'} = 6.5$. Et donc $s_1 = max(s_1^*, s_1^{'}) = max(4, 6.5) = 6.5$.

La séquence produite est illustrée sur la figure 4.6. Quand le réservoir d'énergie devient plein, τ_1 s'exécute avec P_r jusqu'à s_1. Dès s_1, elle s'exécute avec P_{max}.

A t=7, τ_1 se réveille. Comme $a_2 \geq s_1$ et $d_2 = d_1$, alors $s_2 = a_2 = 7$. Nous vérifions par l'équation 4.4 :

$$\begin{aligned} s_2^* &= d_2 - \frac{E_c(a_2) + E_r(a_2, d_2)}{P_{max}} = d_2 - \frac{E_c(a_2) + \int_{a_2}^{d_2} P_r \cdot dt}{P_{max}} \\ &= 9 - \frac{8 + \int_7^9 4 \cdot dt}{8} = 9 - \frac{8 + 4 \cdot (9-7)}{8} = 7 \\ s_2^{'} &= d_2 - \frac{C + E_r(s_2^{'}, d_2)}{P_{max}} = d_2 - \frac{C + \int_{s_2^{'}}^{d_2} P_r \cdot dt}{P_{max}} \\ s_2^{'} &= 9 - \frac{10 + \int_{s_2^{'}}^9 4 \cdot dt}{8} = 9 - \frac{10 + 4 \cdot (9 - s_2^{'})}{8} \end{aligned} \tag{4.16}$$

Nous en déduisons que $s_2^{'} = 6.5$. Et donc $s_2 = max(s_2^*, s_2^{'}) = max(7, 6.5) = 7$.

4.4 Performances théoriques

4.4.1 Optimalité en termes d'ordonnancement

La stratégie d'ordonnancement LSA est un algorithme d'ordonnancement conduit par priorité dynamique piloté intrinsèquement par l'énergie récupérée de l'environnement. Quand nous considérons une architecture matérielle de type monoprocesseur en charge d'exécuter des tâches soumises à des échéances strictes et qui requièrent des quantités d'énergie différentes pour leur exécution, il peut garantir perpétuellement leurs contraintes temporelles en exploitant de façon adéquate à la fois la ressource processeur et la ressource énergie ambiante.

Nous reportons ci-dessous des résultats fondamentaux concernant les performances de LSA.

Théorème 4.1. *Si LSA ne peut pas ordonnancer fiablement une configuration de tâches, alors aucun autre algorithme ne peut le faire, même s'il est totalement clairvoyant.*

Preuve 4.2. *voir [34]*

Ce théorème démontre donc l'optimalité de l'ordonnanceur LSA. Il prouve également que même si nous disposions d'une clairvoyance totale c'est-à-dire si nous connaissions à l'avance les caractéristiques des tâches à ordonnancer, nous ne pourrions pas faire mieux que LSA. L'optimalité de LSA se mesure ici en termes de nombre de configurations de tâches qui peuvent être fiablement ordonnancées, étant données les spécifications du réservoir d'énergie, de la puissance de consommation du processeur et du profil de la source énergétique.

Le théorème suivant établit aussi l'optimalité de LSA en termes de taille de réservoir d'énergie.

4.4.2 Optimalité en termes de taille du réservoir d'énergie

Théorème 4.2. *Soit une configuration de tâches ordonnançable. LSA est l'ordonnanceur qui peut ordonnancer fiablement cette configuration en demandant la capacité minimum de réservoir d'énergie C et la puissance minimum de consommation du processeur P_{max}.*

Preuve 4.3. *voir [34]*

Cette propriété s'avère fondamentale dans le cadre des applications embarquées. En effet, pouvoir ordonnancer fiablement une configuration de tâches tout en requiérant un réservoir d'énergie le plus petit possible a un impact aussi sur le coût et l'encombrement du système résultant. De même, comme cet ordonnanceur est celui qui exige un processeur ayant la plus petite puissance de consommation instantanée, cela permettra de sélectionner un processeur dit de technologie "ultra low power" avec des dissipations thermiques faibles.

4.5 Test d'ordonnançabilité

Théoriquement, le concepteur d'un système temps réel à contraintes strictes devrait être capable de prouver que les limites temporelles ne seront jamais dépassées quelle que soit la situation, soit grâce à un test d'ordonnançabilité (ou contrôle d'admission) basé sur les paramètres temporels des tâches, soit grâce à une simulation d'exécution sur une durée suffisante.

Lorsque les tâches de l'application sont apériodiques et que les paramètres des tâches sont a priori inconnues avant leur arrivée, il s'avère évidemment impossible de mettre en oeuvre un test de faisabilité hors-ligne. D'ailleurs ce type d'application ne tombe pas dans la catégorie des applications temps réel à contraintes strictes mais de celles à contraintes fermes. Il s'agit alors pour l'ordonnanceur de viser à respecter le maximum d'échéances possibles pour les tâches qui arrivent au cours du temps et lorsque cela se révèle possible, de respecter les contraintes temporelles de la totalité des tâches. Toutefois, nous pouvons extraire une condition d'ordonnançabilité associée à LSA applicable à une configuration de tâches dont nous connaîtrions a priori les caractéristiques.

Notation : Notons $g(t_1, t_2)$ la demande energétique des tâches entre les instants t_1 et t_2, donc des tâches τ_i tel que $a_i \geq t_1$ et $d_i \leq t_2$. Nous avons donc :

$$g(t_1, t_2) = \sum_{i=1}^{n} E_i, \quad a_i \geq t_1 \text{ et } d_i \leq t_2 \tag{4.17}$$

La demande énergétique entre deux instants t_1 et t_2 reflète ainsi la quantité d'énergie requise par les tâches pour leur exécution lorsque ces tâches ont une date de réveil postérieure à t_1 et une échéance antérieure à t_2.

4.5.1 Condition nécessaire et suffisante d'ordonnançabilité

Théorème 4.3. *Soit une configuration $\mathscr{T}$ de tâches apériodiques et/ou périodiques caractérisée par sa demande energétique g. Soit un réservoir d'énergie de capacité C initialement plein. Soit E_r le profil de la source energétique. $\mathscr{T}$ est fiablement ordonnancée par LSA si et seulement si*

$$g(t_1, t_2) \leq min(E_r(t_1, t_2) + C, P_{max} \cdot (t_2 - t_1)), \quad \forall t_1, t_2 \geq 0 \tag{4.18}$$

Preuve 4.4. *voir [34]*

Commentaires : La condition $g(t_1, t_2) \leq P_{max} \cdot (t_2 - t_1)$, c'est-à-dire, $\frac{g(t_1,t_2)}{P_{max}} \leq (t_2 - t_1)$ signifie que la demande processeur de l'ensemble des tâches réveillées à partir de t_1 et d'échéance au plus égale à t_2 doit être inférieure à la longueur de l'intervalle [t_1, t_2], soit (t_2-t_1). Cette condition suffirait pour tester la faisabilité en l'absence de contraintes d'ordre énergétique.

La condition $g(t_1, t_2) \leq E_r(t_1, t_2) + C$, signifie que la quantité maximum d'énergie requise entre t_1 et t_2 doit être inférieure ou égale à la quantité d'énergie produite par la source entre t_1 et t_2 ajoutée de la quantité maximale d'énergie disponible dans le réservoir, soit C.

Dans le cas d'une configuration de tâches périodiques, la demande énergétique s'exprime par la relation suivante :

$$g(t_1, t_2) = \sum_{i=1}^{n} E_i \cdot \left\lceil \frac{t_2 - t_1 - D_i}{T_i} \right\rceil \tag{4.19}$$

où $\left\lceil \frac{t_2 - t_1 - D_i}{T_i} \right\rceil$ représente le nombre maximum de tâches reveillées pendant l'intervalle de temps $(t_2 - t_1)$.

4.5.2 Test d'ordonnançabilité sous une source d'énergie à puissance constante

Nous faisons l'hypothèse dans ce paragraphe que la source environnementale émet son énergie de façon régulière au cours du temps, c'est-à-dire que la fonction $P_r(t)$ est constante.

4.5.2.1 Illustrations

Exemple 4.7. *Soit une tâche apériodique* $\tau_1 = (1, 32, 9)$, *avec* $a_1 = 1$, $d_1 = 9$ *et* $E_1 = 32$. *Supposons que* $P_{max} = 8$, $P_r = 4$, $C = 10$ *et* $E_c(0) = 4$.

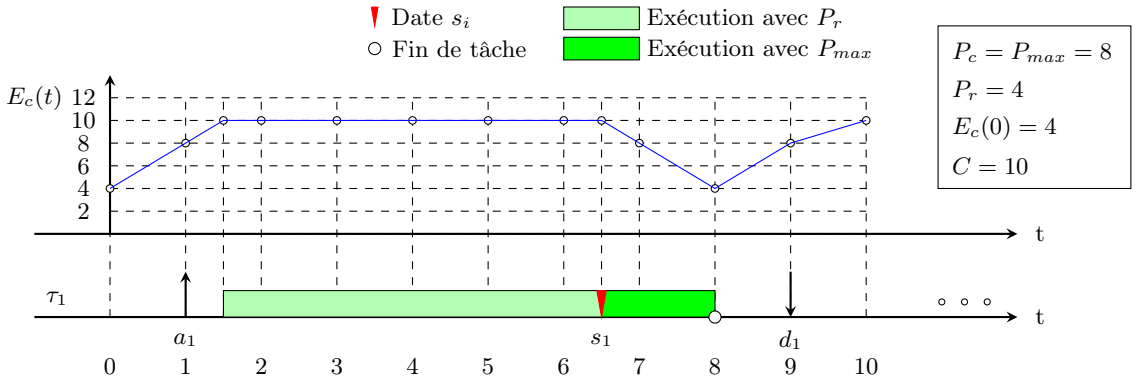


Figure 4.7: – Description de la séquence LSA

D'après la formule 4.17, $g(a_1, d_1) = E_1 = 32$. Et l'énergie récupérée pendant les périodes $[a_1, d_1]$: $E_r(a_1, d_1) = \int_{a_1}^{d_1} P_r(t) \cdot dt = \int_1^9 4 \cdot dt = 4 \cdot (9 - 1) = 32$. Donc $g(a_1, d_1) < E_r(a_1, d_1) + C$.

En outre, $P_{max} \cdot (d_1 - a_1) = 8 \cdot (9 - 1) = 64$.

Comme $g(a_1, d_1) \leq min(E_r(a_1, d_1) + C, P_{max} \cdot (d_1 - a_1))$, alors la tâche τ_1 sera fialement ordonnancée par LSA.

Exemple 4.8. *Ici, nous reprenons l'exemple 4.2 pour y appliquer le test d'ordonnançabilité précédent. Soit la configuration composée des deux tâches apériodiques* $\{\tau_1 = (1, 24, 9), \tau_2 = (5, 8, 8)\}$.

Nous devons effectuer les tests suivants :

$$
\begin{aligned}
g(a_1, d_1) &\leq min(E_r(a_1, d_1) + C, P_{max} \cdot (d_1 - a_1)) \\
g(a_1, d_2) &\leq min(E_r(a_1, d_2) + C, P_{max} \cdot (d_2 - a_1)) \\
g(a_2, d_1) &\leq min(E_r(a_2, d_1) + C, P_{max} \cdot (d_1 - a_2)) \\
g(a_2, d_2) &\leq min(E_r(a_2, d_2) + C, P_{max} \cdot (d_2 - a_2))
\end{aligned}
$$

Nous avons $g(1, 9) = 24+8 = 32$ d'une part, $E_r(1, 9)+C = 4 \cdot (9-1)+10 = 42, P_{max} \cdot (9-1) = 64$ d'autre part.

$$32 < min(42, 64) \tag{4.20}$$

Nous avons également :

$$g(1,8) = 8, E_r(1,8) + C = 4 \cdot (8-1) + 10 = 38, P_{max} \cdot (8-1) = 56. \quad (4.21)$$

$$g(5,9) = 8, E_r(5,9) + C = 4 \cdot (9-5) + 10 = 26, P_{max} \cdot (9-5) = 32. \quad (4.22)$$

$$g(5,8) = 8, E_r(5,8) + C = 4 \cdot (8-5) + 10 = 22, P_{max} \cdot (8-5) = 24. \quad (4.23)$$

Nous en déduisons que les inégalités 4.20, 4.21, 4.22 et 4.23 sont vérifiées. La condition nécessaire et suffisante d'ordonnançabilité étant satisfaite, nous pouvons affirmer que LSA construira une séquence valide c'est-à-dire une séquence qui respecte toutes les échéances.

4.5.2.2 Complexité algorithmique du test

La complexité du test d'ordonnançabilité de LSA est en $O(n^2)$ si la configuration contient n tâches. L'adjonction de contraintes énergétiques n'engendre donc aucune complexité supplémentaire dans le test d'ordonnançabilité.

4.5.3 Test sous une source d'énergie à puissance variable

Considérons maintenant que la source environnementale produit de l'énergie avec une puissance d'émission qui varie au cours du temps.

4.5.3.1 Illustrations

Exemple 4.9. *Soit une configuration de tâches apériodiques composée de* $\tau_1 = (1, 24, 9)$ *et* $\tau_2 = (5, 8, 8)$. *Supposons la puissance* $P_r(t)$ *définie comme suit :*

L'intervalle de temps	[0,1]	[1,2]	[2,3]	[3,4]	[4,5]	[5,6]	[6,7]	[7,8]	[8,9]	[9,10]
$P_r(t)$	4	2	6	4	2	6	4	2	6	4

Tableau 4.2: – Valeurs de P_r

Nous effectuons les tests suivants comme dans l'exemple 4.8 :

$$g(1,9) = 24 + 8 = 32, E_r(1,9) + C = (2+6+4+2+6+4+2+6) + 10 = 42, P_{max} \cdot (9-1) = 64. \quad (4.24)$$

$$g(1,8) = 8, E_r(1,8) + C = (2+6+4+2+6+4+2) + 10 = 36, P_{max} \cdot (8-1) = 56. \quad (4.25)$$

$$g(5,9) = 8, E_r(5,9) + C = (6+4+2+6) + 10 = 28, P_{max} \cdot (9-5) = 32. \quad (4.26)$$

$$g(5,8) = 8, E_r(5,8) + C = (6+4+2) + 10 = 22, P_{max} \cdot (8-5) = 24. \quad (4.27)$$

Nous en déduisons que les inégalités 4.24 - 4.27 sont vérifiées. Et donc la condition nécessaire et suffisante d'ordonnançabilité est satisfaite.

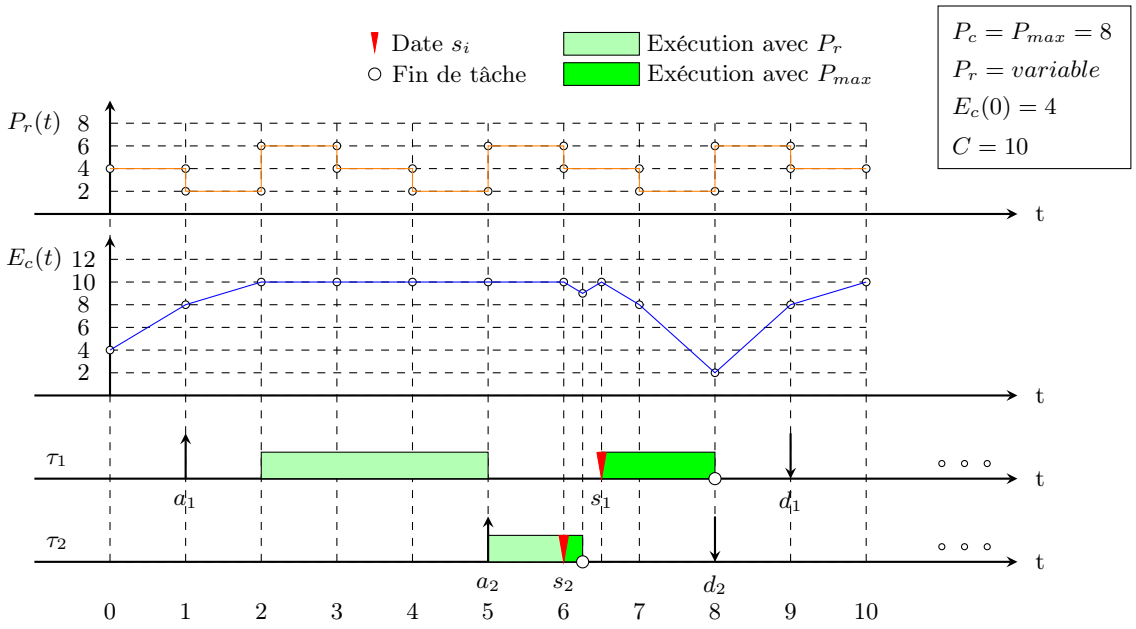


Figure 4.8: – Description de la séquence LSA

4.5.3.2 Complexité algorithmique du test

Nous pouvons donc affirmer que compte tenu du profil de la source énergétique, les tâches seront fiablement ordonnancées par LSA. En comparant avec la complexité algorithmique du test précédent, nous constatons ici que le test nécessite des opérations supplémentaires liées à l'évaluation de l'énergie drainée dans les différents intervalles de temps. Soit m, le nombre de valeurs distinctes caractérisant le profil énergétique. Il s'ensuit donc que la complexité du test est $O(n^2 + m)$.

4.5.4 Test d'une configuration de tâches périodiques

4.5.4.1 Problématique

Une tâche périodique peut être vue comme une infinité d'instances donc une infinité de tâches apériodiques. Il n'est pas envisageable de tester l'ordonnançabilité d'une infinité de tâches apériodiques sur une durée infinie. Ce constat est d'autant plus vrai lorsque nous ne pouvons prédire le profil énergétique de façon exacte sur une durée infinie. Nous constatons donc qu'hormis dans le cas où le profil énergétique est connu a priori et de façon précise, aucun test d'ordonnançabilité ne pourra être mis en oeuvre hors-ligne. Un tel test ne peut se mettre en oeuvre que dynamiquement au fur et à mesure que le système prend connaissance de son profil énergétique. En effet, nous pouvons imaginer que périodiquement, une méthode de prédiction permette de donner la connaissance de l'énergie qui pourra être drainée de la source sur un certain intervalle. C'est en particulier tout à fait réalisable pour l'énergie solaire. Comme vu dans le chapitre précédent, nous pouvons restreindre l'intervalle de test à une fenêtre temporelle plus réduite de l'ordre du PPCM des périodes des tâches. Lorsque l'énergie arrive à puissance variable, nous pouvons donc imaginer à chaque début de PPCM, d'effectuer un test d'ordonnançabilité. Si le test est véri-

fié, nous serons donc sûr de pouvoir ordonnancer toutes les instances réveillées dans la fenêtre temporelle. Sinon, nous savons que certaines de ces échéances ne pourront pas être respectées.

4.5.4.2 Illustration

Exemple 4.10. *Soit la configuration de tâches périodiques ER (Echéance sur Requête) suivante :* $\mathscr{T} = \{\ \tau_i | 1 \leq i \leq 2, avec\ \tau_i = (a_i, E_i, T_i)\ \}$. *Supposons que* $\tau_1 = (0, 24, 10)$ *et* $\tau_2 = (0, 8, 5)$. *Nous notons que* $PPCM(\tau_1, \tau_2) = 10$. *Nous prenons la même source d'énergie que dans l'exemple 4.9. Ses valeurs sont illustrées dans le tableau 4.2.*

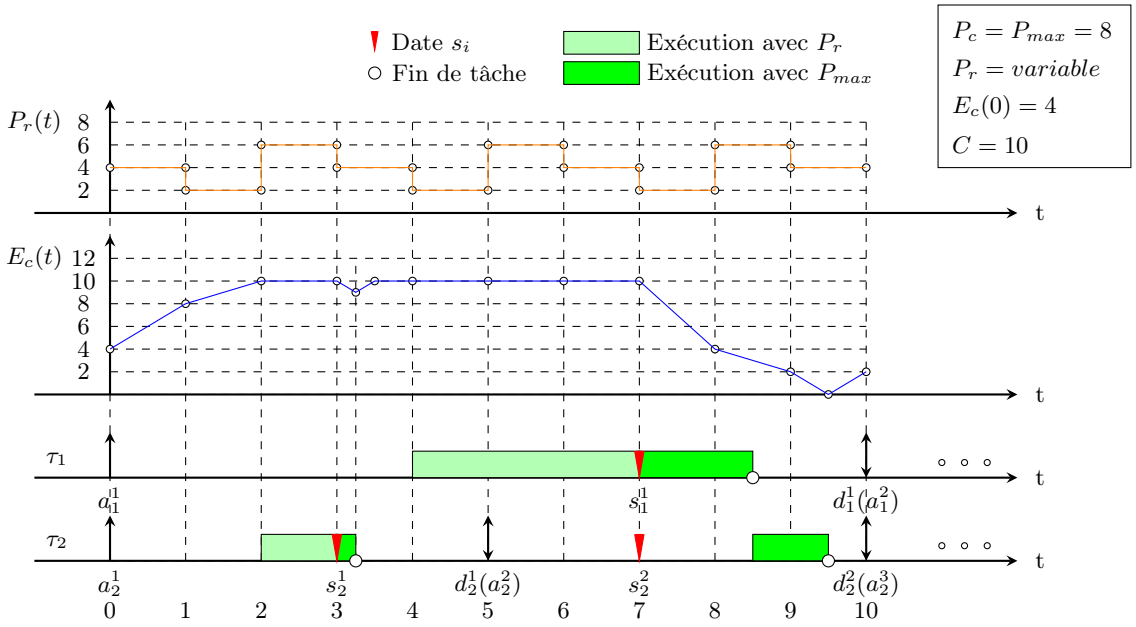


Figure 4.9: – Description de la séquence LSA

Pour vérifier l'ordonnançabilité de la configuration sur le premier PPCM, nous devons effectuer les tests suivants en utilisant la formule 4.19 :

$$
\begin{aligned}
g(a_1^1, d_1^1) &\leq min(E_r(a_1^1, d_1^1) + C, P_{max} \cdot (d_1^1 - a_1^1)) \\
g(a_1^1, d_2^1) &\leq min(E_r(a_1^1, d_2^1) + C, P_{max} \cdot (d_2^1 - a_1^1)) \\
g(a_1^1, d_2^2) &\leq min(E_r(a_1^1, d_2^2) + C, P_{max} \cdot (d_2^2 - a_1^1)) \\
g(a_2^1, d_1^1) &\leq min(E_r(a_2^1, d_1^1) + C, P_{max} \cdot (d_1^1 - a_2^1)) \\
g(a_2^1, d_2^1) &\leq min(E_r(a_2^1, d_2^1) + C, P_{max} \cdot (d_2^1 - a_2^1)) \\
g(a_2^1, d_2^2) &\leq min(E_r(a_2^1, d_2^2) + C, P_{max} \cdot (d_2^2 - a_2^1)) \\
g(a_2^2, d_1^1) &\leq min(E_r(a_2^2, d_1^1) + C, P_{max} \cdot (d_1^1 - a_2^2)) \\
g(a_2^2, d_2^1) &\leq min(E_r(a_2^2, d_2^1) + C, P_{max} \cdot (d_2^1 - a_2^2)) \\
g(a_2^2, d_2^2) &\leq min(E_r(a_2^2, d_2^2) + C, P_{max} \cdot (d_2^2 - a_2^2))
\end{aligned}
$$

Comme $a_1^1 = a_2^1$, $a_2^2 = d_2^1$ et $d_1^1 = d_2^2$, nous avons seulement besoin de calculer ce qui suit :

$$g(0,10) = 24 + 8 + 8 = 40, E_r(0,10) + C = 40 + 10 = 50, P_{max} \cdot (10 - 0) = 80. \quad (4.28)$$

$$g(0,5) = 8, E_r(0,5) + C = 18 + 10 = 28, P_{max} \cdot (5 - 0) = 40. \quad (4.29)$$

$$g(5,10) = 8, E_r(5,10) + C = 22 + 10 = 32, P_{max} \cdot (10 - 5) = 40. \quad (4.30)$$

Nous en déduisons que les inégalités 4.28, 4.29 et 4.30 sont satisfaites et donc que toutes les instances réveillées sur le premier PPCM respecteront leur échéance.

4.5.4.3 Complexité algorithmique du test

La complexité du test d'ordonnançabilité de LSA sur un PPCM est en $O(N^2 + m)$ si nous avons N instances de tâches périodiques à tester.

4.6 Surcoût d'exécution de l'ordonnancement LSA

Dans les paragraphes précédents, nous avons mis en évidence la supériorité de l'ordonnanceur LSA. Toutefois, il ne suffit pas qu'un ordonnanceur soit optimal pour pouvoir l'intégrer dans un système d'exploitation temps réel. Il faut aussi que cette ordonnanceur soit efficace en termes de complexité d'implémentation. En d'autres termes, nous devons vérifier que les surcoûts temporels qu'il engendre à chaque fois que nous l'appelons sont acceptables et n'entâche pas de manière prohibitive la performance du système.

Nous avons vu la formule mathématique pour calculer la date de démarrage s_i. La complexité de ce calcul dépend de la façon dont nous évaluons la quantité d'énergie environnementale produite entre deux instants. Selon que nous avons une source d'énergie à puissance constante ou à puissance variable, ce calcul sera donc plus ou moins complexe. Les différentes sources d'énergie que nous utilisons, se présentent sous différentes formes : par exemple l'énergie solaire varie selon la loi parabolique avec une très grande période, l'énergie piézoélectrique est souvent en forme de pulsations, l'énergie dissipée par le corps humain peut être considérée comme constante, etc.

Ci-dessous, nous donnons un aperçu de la complexité de LSA selon le type d'énergie environnementale utilisée.

4.6.1 Source d'énergie ambiante à puissance constante

Selon la formule de calcul des dates de début d'exécution 4.4, puisque la source d'énergie ambiante est constante, nous en déduisons que :

$$\begin{aligned} s_i^* &= d_i - \frac{E_c(a_i) + P_r \cdot (d_i - a_i)}{P_{max}} \\ s_i^{'} &= d_i - \frac{C + P_r \cdot (d_i - s_i^{'})}{P_{max}} = d_i - \frac{C}{P_{max} - P_r} \\ s_i &= max(s_i^*, s_i^{'}) \end{aligned} \quad (4.31)$$

D'après cette formule, s_i se calcule en $O(1)$ (complexité constante). Une complexité constante est la complexité algorithmique idéale, puisque peu importe le nombre de tâches à traiter, l'algorithme prendra toujours un nombre fixé à l'avance d'opérations.

4.6.2 Source d'énergie ambiante à puissance variable

Si la puissance d'énergie ambiante $P_r(t)$ est variable, elle peut être modélisée par les fonctions mathématiques diverses.

4.6.2.1 Cas d'une puissance de forme affine

Supposons la puissance d'énergie ambiante $P_r(t)$ soit modélisée par une fonction affine, par exemple $P_r(t) = a \cdot t + b$, où a et b sont deux constantes. Alors nous déduisons la formule de calcul des dates de début d'exécution 4.4 :

$$\begin{aligned} s_i^* &= d_i - \frac{E_c(a_i) + E_r(a_i, d_i)}{P_{max}} = d_i - \frac{E_c(a_i) + \int_{a_i}^{d_i} P_r(t) \cdot dt}{P_{max}} \\ &= d_i - \frac{E_c(a_i) + \int_{a_i}^{d_i} (a \cdot t + b) \cdot dt}{P_{max}} = d_i - \frac{E_c(a_i) + \frac{1}{2}(d_i^2 - a_i^2) + b \cdot (d_i - a_i)}{P_{max}} \quad (4.32) \\ s_i^{'} &= d_i - \frac{C + E_r(s_i^{'}, d_i)}{P_{max}} = d_i - \frac{C + \int_{s_i^{'}}^{d_i} P_r(t) \cdot dt}{P_{max}} \\ s_i^{'} &= d_i - \frac{C + \int_{s_i^{'}}^{d_i} (a \cdot t + b) \cdot dt}{P_{max}} = d_i - \frac{C + \frac{1}{2}(d_i^2 - s_i^{'2}) + b \cdot (d_i - s_i^{'})}{P_{max}} \end{aligned}$$

Nous en déduisons que

$$s_i^{'} = P_{max} - b \pm \sqrt{b^2 + P_{max}^2 - 4 \cdot b \cdot P_{max} + 2 \cdot C + d_i^2 + 2 \cdot b \cdot d_i - 2 \cdot P_{max} \cdot d_i}. \quad (4.33)$$

Et donc $s_i = max(s_i^*, s_i^{'})$.

Peu importe le nombre de tâches à traiter, le nombre d'opérations est fixe. Donc la complexité de calcul des s_i est en $O(1)$ (complexité constante).

4.6.2.2 Cas d'une puissance variable quelconque

Supposons la puissance d'énergie ambiante $P_r(t)$ modélisée par une fonction complexe ou aléatoire, par exemple une fonction aléatoire qui suit une loi normale. Le calcul de s_i devient plus compliqué.

En réalité, le système effectue en avance une prédiction de l'énergie à venir et mémorise le profil dans un tableau. Donc nous pouvons facilement obtenir la valeur de $\int_{a_i}^{d_i} P_r(t) \cdot dt$, égale à $\sum_{t=a_i}^{d_i} P_r(t)$. Concernant s_i^*, il n'est pas difficile d'obtenir sa valeur.

Mais concernant le calcul de $s_i^{'}$, nous utilisons une méthode énumérative pour résoudre une équation.

$$s_i^{'} = d_i - \frac{C + \int_{s_i^{'}}^{d_i} P_r(t) \cdot dt}{P_{max}}$$

d'où $P_{max} \cdot s_i^{'} = P_{max} \cdot d_i - C - \int_{s_i^{'}}^{d_i} P_r(t) \cdot dt$, et donc $P_{max} \cdot d_i - C - \int_{s_i^{'}}^{d_i} P_r(t) \cdot dt - P_{max} \cdot s_i^{'} = 0$.

Nous définissons la fonction

$$X(t) = P_{max} \cdot d_i - C - \int_{t}^{d_i} P_r(t) \cdot dt - P_{max} \cdot t, \quad pour \quad a_i \leq t \leq d_i \tag{4.34}$$

et nous cherchons la valeur de t telle que $X(t) = 0$.

Exemple 4.11. *Soit une tâche apériodique* $\tau_1 = (1, 24, 9)$. *Les valeurs de puissance* $P_r(t)$ *sont mémorisées dans le tableau 4.2.*

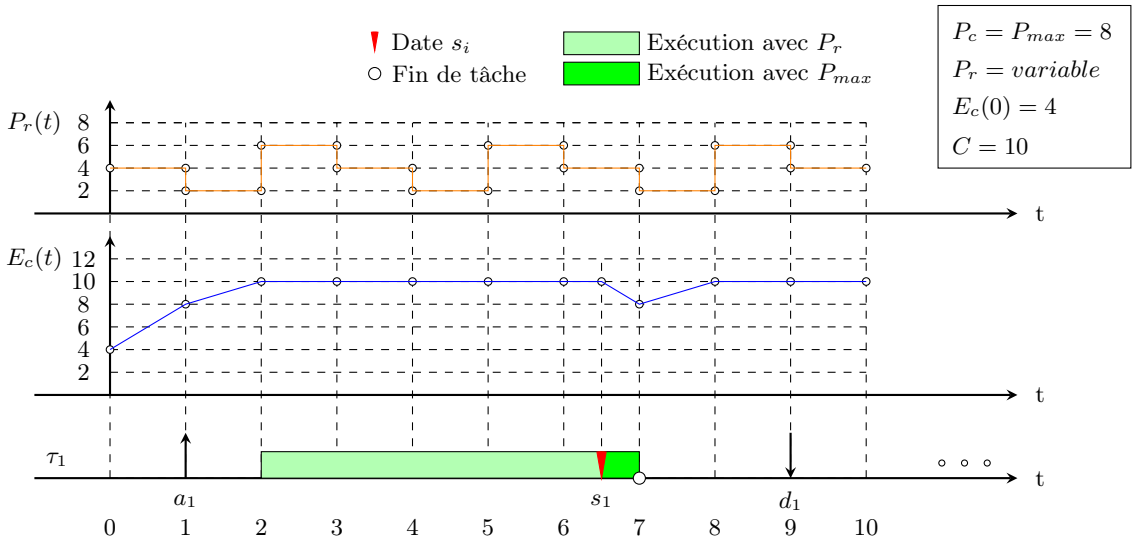


Figure 4.10: – Séquence LSA produite

$$\begin{aligned} s_1^* &= d_1 - \frac{E_c(a_1) + \int_{a_1}^{d_1} P_r(t) \cdot dt}{P_{max}} = d_1 - \frac{E_c(a_1) + \sum_{t=a_1}^{d_1} P_r(t) \cdot dt}{P_{max}} \\ &= 9 - \frac{8 + \sum_{t=1}^{9} P_r(t)}{8} = 9 - \frac{8+32}{8} = 4 \end{aligned} \tag{4.35}$$

Puis nous calculons $s_1^{'}$: La tâche τ_1 est activée à l'instant $t = 1$, nous vérifions la valeur de la formule 4.34 à chaque instant entre [1, 9]. Si la valeur de $X(t)$ est nulle à un instant, alors il est choisi comme la date de début d'exécution s_1, sinon nous choisissons le temps où la valeur est minimum comme s_1, cf. tableau 4.3.

Dans cet exemple, parmi ces valeurs, nous ne trouvons pas l'instant t où $X(t) = 0$, mais nous pouvons déterminer que $s_1^{'}$ se trouve dans [6, 7]. Donc nous revérifions $X(6.5) = 8 \cdot 9 - 10 - 10 - 8 \cdot 6.5 = 0$, et déterminons $s_1^{'} = 6.5$. Enfin, $s_1 = max(s_1^*, s_1^{'}) = max(4, 6.5) = 6.5$. La valeur optimale théorique de s_i est 6.5.

Ici nous avons choisi la valeur exacte 6.5, mais dans la réalité, une unité de temps ne peut pas être fractionnée. Il aurait donc fallu choisir soit 6 ou 7.

t	$X(t) = P_{max} \cdot d_i - C - \int_t^{d_i} P_r(t) \cdot dt - P_{max} \cdot t$
1	$X(1) = 8 \cdot 9 - 10 - 32 - 8 \cdot 1 = 22$
2	$X(2) = 8 \cdot 9 - 10 - 30 - 8 \cdot 2 = 16$
3	$X(3) = 8 \cdot 9 - 10 - 24 - 8 \cdot 3 = 14$
4	$X(4) = 8 \cdot 9 - 10 - 20 - 8 \cdot 4 = 10$
5	$X(5) = 8 \cdot 9 - 10 - 18 - 8 \cdot 5 = 4$
6	$X(6) = 8 \cdot 9 - 10 - 12 - 8 \cdot 6 = 2$
7	$X(7) = 8 \cdot 9 - 10 - 8 - 8 \cdot 7 = -2$
8	$X(8) = 8 \cdot 9 - 10 - 6 - 8 \cdot 8 = -8$
9	$X(9) = 8 \cdot 9 - 10 - 0 - 8 \cdot 9 = -10$

Tableau 4.3: – Exemple de calcul de s_i

En résumé, quand la loi de variation de la source d'énergie est complexe, nous devons effectuer D_i fois plus de calculs par rapport aux cas précédents pour déterminer la date de début d'exécution. Donc quand nous utilisons cette méthode pour s_i, le surcoût d'exécution de l'algorithme LSA devient prohibitif parce qu'il dépend du nombre de fois où nous devons résoudre l'équation 4.34.

4.7 Points faibles de l'algorithme LSA

Bien que l'algorithme LSA soit optimal parmi les stratégies d'ordonnancement monoprocesseur préemptives, il se fonde sur des hypothèse restrictives diverses, par exemple :

– la puissance de soure d'énergie ambiante doit être bien prévue ;

– la vitesse du processeur doit pouvoir varier de façon continue, etc.

En réalité, ces hypothèses sont particulièrement difficiles à accepter. Une fois que ces hypothèses fondamentales ne sont plus satisfaites, l'algorithme LSA n'est plus optimal. Ci-dessous, nous mettons en évidence les points faibles de l'algorithme LSA liées aux hypothèses irréalistes du modèle considéré :

4.7.1 Hypothèse irréaliste : vitesse continuellement ajustable

Le modèle de Moser et *al.* considère un processeur idéal dont la puissance de consommation peut varier de façon continue. Dans l'implémentation de l'algorithme LSA, tant que $t < s_i$ et le réservoir d'énergie est plein, il est supposé que la fréquence du processeur peut varier progressivement pour s'ajuster à la puissance instantanée de l'énergie ambiante $P_r(t)$. En fait, aujourd'hui nous ne pouvons pas trouver un tel type de processeur sur le marché. Même les processeurs multivitesse ne sont pas capables de suivre la puissance instantanée $P_r(t)$ de façon continue.

Solution proposée : Pour résoudre ce problème, quand nous implémentons l'algorithme LSA sur un processeur monovitesse, nous considérons que le processeur possède seulement deux états : marche et arrêt. Il fonctionne avec une seule puissance possible P_{max} et il ne peut suivre la puissance instantanée de l'énergie ambiante $P_r(t)$. Avant l'instant s_i, une fois que le réservoir

d'énergie est plein, la tâche s'exécute avec P_{max} au lieu de $P_r(t)$ pendant seulement une unité de temps. Puis, nous mettons le processeur en veille pour qu'il se recharge tout de suite, et une fois qu'il est rechargé à 100%, la tâche est exécutée avec P_{max} pendant une unité de temps. L'ordonnanceur répète le même comportement jusqu'à $t = s_i$.

4.7.2 Hypothèse irréaliste : puissance consommée nulle en mode veille

Habituellement, quand le processeur est en mode veille (mode shut down ou idle en anglais), de manière simplifiée, nous supposons que la puissance de consommation est nulle. Mais en réalité, ce n'est pas le cas. Il se révèle plus complexe à cause des consommations d'énergie statiques sur la mémoire flash, le module de communication, le support du réservoir d'énergie, etc. Elle est peut-être fluctuante en fonction de la circuiterie utilisée.

Bien que la consommation d'énergie en mode veille ne soit pas très grande, il est indispensable de la considérer pour analyser les applications, surtout dans les microsystèmes. Comment intégrer cette puissance en mode veille dans l'algorithme LSA ? Dans ce cas, nous proposons une variante de l'algorithme LSA que nous appelons l'*algorithme LSA-ESD* (Lazy Scheduling Algorithms with Energy of Shut Down) intégrant la consommation d'énergie en mode veille.

Solution proposée : Supposons qu'une tâche s'exécute par LSA avec P_{max}. Quand le processeur est mis en veille, la puissance de consommation de processeur est $P_{esd}(t)$ au lieu de 0 (cf. figure 4.11). Pendant la veille, sur un intervalle de temps donné $[t_1, t_2]$, sa consommation d'énergie sera donnée par $E_{esd} = \int_{t_1}^{t_2} P_{esd}(t) \cdot dt$.

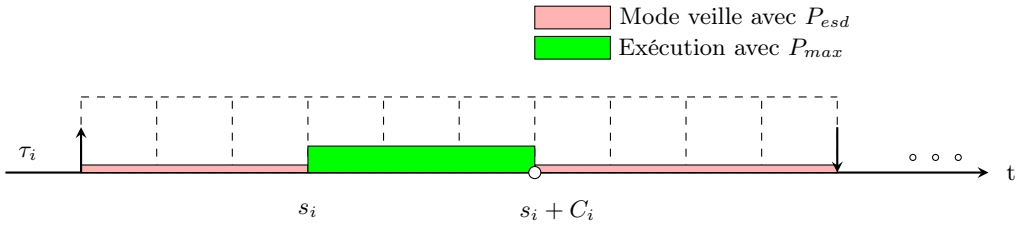


Figure 4.11: – Présentation de l'algorithme LSA-ESD

Selon l'équation 4.4, et si $P_{esd}(t)$ est constante, nous déduisons la date de début d'exécution s_i de la façon suivante :

$$\begin{aligned} s_i^* &= d_i - \frac{E_c(a_i) + E_r(a_i, d_i) - E_{esd}(a_i, s_i^*) - E_{esd}(s_i^* + C_i, d_i)}{P_{max}} \\ &= d_i - \frac{E_c(a_i) + \int_{a_i}^{d_i} P_r(t) \cdot dt - \int_{a_i}^{s_i^*} P_{esd}(t) \cdot dt - \int_{s_i^*+C_i}^{d_i} P_{esd}(t) \cdot dt}{P_{max}} \\ &= d_i - \frac{E_c(a_i) + \int_{a_i}^{d_i} P_r(t) \cdot dt - P_{esd} \cdot (s_i^* - a_i) - P_{esd} \cdot [d_i - (s_i^* + C_i)]}{P_{max}} \\ &= d_i - \frac{E_c(a_i) + \int_{a_i}^{d_i} P_r(t) \cdot dt - P_{esd} \cdot (d_i - a_i - C_i)}{P_{max}} \end{aligned} \tag{4.36}$$

$$
\begin{aligned}
s_i^{'} &= d_i - \frac{C + E_r(s_i^{'}, d_i) - E_{esd}(a_i, s_i^{'}) - E_{esd}(s_i^{'} + C_i, d_i)}{P_{max}} \\
s_i^{'} &= d_i - \frac{C + \int_{s_i^{'}}^{d_i} P_r(t) \cdot dt - \int_{a_i}^{s_i^{'}} P_{esd}(t) \cdot dt - \int_{s_i^{'}+C_i}^{d_i} P_{esd}(t) \cdot dt}{P_{max}} \\
s_i^{'} &= d_i - \frac{C + \int_{s_i^{'}}^{d_i} P_r(t) \cdot dt - P_{esd} \cdot (d_i - a_i - C_i)}{P_{max}}
\end{aligned} \tag{4.37}
$$

Nous déduisons $s_i = max(s_i^*, s_i^{'})$ en utilisant les formules 4.36 et 4.37.

Exemple 4.12. *Nous reprenons l'exemple 4.2, et supposons que* $P_{esd} = P_{max}/10$, *c'est-à-dire* $P_{esd} = 0.8$.

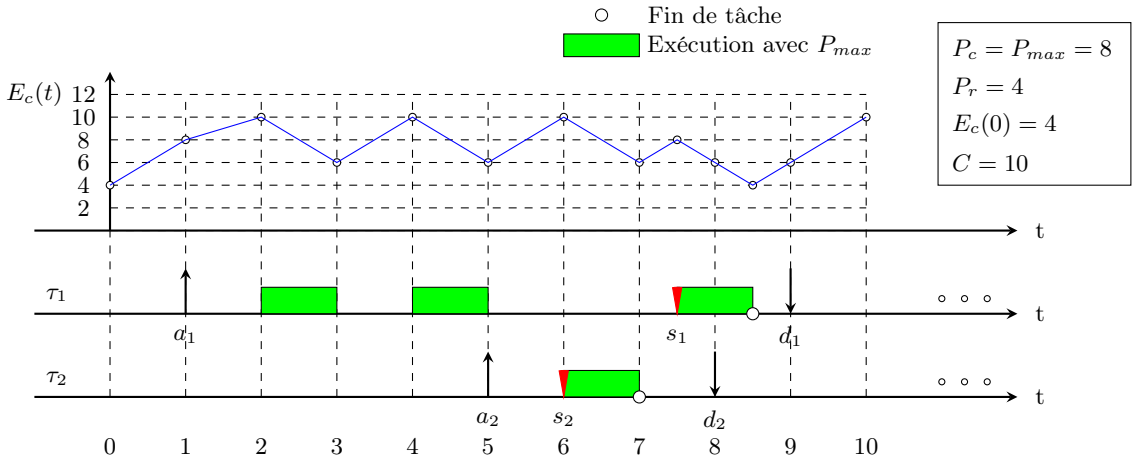


Figure 4.12: – Séquence LSA-ESD sur des tâches apériodiques

Selon les équations 4.36 et 4.37, calculons le s_1 pour la tâche τ_1 à t=1 :

$$
\begin{aligned}
s_1^* &= d_1 - \frac{E_c(a_1) + \int_{a_1}^{d_1} P_r(t) \cdot dt - P_{esd} \cdot (d_1 - a_1 - C_1)}{P_{max}} \\
&= 9 - \frac{8 + 4 \cdot (9-1) - 0.8 \cdot (9-1-3)}{8} = 4.5 \\
s_1^{'} &= d_1 - \frac{C + \int_{s_1^{'}}^{d_1} P_r(t) \cdot dt - P_{esd} \cdot (d_1 - a_1 - C_1)}{P_{max}} \\
s_1^{'} &= 9 - \frac{10 + 4 \cdot (9 - s_1^{'}) - 0.8 \cdot (9-1-3)}{8}
\end{aligned} \tag{4.38}
$$

Nous en déduisons que $s_1^{'} = 7.5$. Donc $s_1 = max(s_1^*, s_1^{'}) = max(4.5, 7.5) = 7.5$.

La figure 4.12 nous montre la séquence produite par LSA-ESD.

Puis, jusqu'à t=5, la tâche τ_2 est arrivée,d'où le calcul de s_2 :

$$\begin{aligned} s_2^* &= d_2 - \frac{E_c(a_2) + \int_{a_2}^{d_2} P_r(t) \cdot dt - P_{esd} \cdot (d_2 - a_2 - C_2)}{P_{max}} \\ &= 8 - \frac{6 + 4 \cdot (8-5) - 0.8 \cdot (8-5-1)}{8} = 5.95 \\ s_2^{'} &= d_2 - \frac{C + \int_{s_2^{'}}^{d_2} P_r(t) \cdot dt - P_{esd} \cdot (d_2 - a_2 - C_2)}{P_{max}} \\ &= 8 - \frac{10 + 4 \cdot (8-s_2^{'}) - 0.8 \cdot (8-5-1)}{8} = 5.9 \\ s_2 &= max(s_2^*, s_2^{'}) = max(5.95, 5.9) = 5.95 \end{aligned} \tag{4.39}$$

Dans cet exemple, la puissance de processeur ne suit pas la puissance de source d'énergie avant $t \leq s_i$ (cf. figure 4.12) : le processeur soit fonctionne avec la vitesse maximale, soit se met en mode veille. Par rapport à l'exemple 4.2, nous constatons que les dates de début d'exécution des tâches sont reportées.

4.7.3 Hypothèse irréaliste : profil energétique connu avec précision

Afin de calculer correctement s_i, nous supposons que la puissance de l'énergie ambiante est toujours bien connue sur une durée infinie. Dans [34], une idée de trace énergétique EVCC (en anglais, Energy Variability Characterization Curves) a été proposée. Elle décrit la loi de variation de la puissance $P_r(t)$. Le degré de précision de la trace énergétique EVCC influence beaucoup la fidélité de s_i. Mais en réalité, nous ne pouvons pas assurer la fidélité de la trace EVCC, et donc le test de faisabilité devient inutilisable car basé sur la connaissance précise de cette donnée. Mais nous pouvons faire tester la faisabilité intervalle par intervalle, par exemple, PPCM par PPCM.

4.7.4 Hypothèse irréaliste : puissance de consommation constante

Dans [34], le système de traitement est supposé consommé la même puissance instantanée quelque soit la tâche exécutée. Cela signifie donc que la durée d'exécution et l'énergie consommée par toute tâche sont reliés par un unique facteur, la puissance maximale de consommation du processeur, c'est-à-dire, $C_i = \frac{E_i}{P_{max}}$. En réalité, l'exécution de tâche est plus complexe que cette hypothèse. Bien que LSA considère P_{max}, c'est-à-dire la puissance maximum de consommation, cela introduit donc une erreur sur la détermination du paramètre s_i, du fait que cette puissance est en réalité variable.

4.7.5 Hypothèse irréaliste : puissance consommée supérieure à la puissance produite de la source renouvelable

Il existe une hypothèse extrêmement restrictive qui sous-tend l'optimalité de LSA : la puissance de consommation du processeur doit être à tout instant plus élevée que la puissance de la source ambiante, c'est-à-dire, $P_{max} > P_r(t)$. Ici nous allons montrer sur un exemple que, lorsque cette hypothèse ne tient plus, alors l'algorithme LSA n'est plus optimal.

Exemple 4.13. *Considérons une tâche apériodique τ_i à exécuter, caractérisée par $(0, 24, 10)$. Supposons que $C = 10$, $E_c(0) = 0$, $P_r = 6$ et $P_{max} = 4$. Notons ici $P_{max} < P_r$. Alors utilisons l'équation 4.4 pour calculer la date de début d'exécution s_i :*

$$\begin{aligned} s_i &= d_i - \frac{min\left[E_c(a_i) + E_r(a_i, d_i), C + E_r(s_i, d_i)\right]}{P_{max}} \\ &= 10 - \frac{min\left[0 + \int_0^{10} 6 \cdot dt, 10 + \int_{s_i}^{10} 6 \cdot dt\right]}{4} \\ &= -5 \end{aligned} \tag{4.40}$$

Evidemment, nous ne trouvons pas de solution pour l'équation précédente. Donc, quand $P_{max} \leq P_r(t)$, l'algorithme LSA ne fonctionne plus.

Exemple 4.14. *Reprenons l'exemple 4.11, un exemple avec une source d'énergie variable. Soit une tâche apériodique $\tau_1 = (1, 24, 9)$. Supposons que $C = 10$, $E_c(0) = 4$, $P_{max} = 8$. La source d'énergie est échantillonnée et mémorisée dans le tableau 4.4. Nous notons que $P_r[2, 3] = P_r[5, 6] = P_r[8, 9] = 10 > P_{max}$.*

L'intervalle de temps	[0,1]	[1,2]	[2,3]	[3,4]	[4,5]	[5,6]	[6,7]	[7,8]	[8,9]	[9,10]
$P_r(t)$	6	6	10	6	6	10	6	6	10	6

Tableau 4.4: – Valeurs de P_r

Nous calculons d'abord s_1^* :

$$\begin{aligned} s_1^* &= d_1 - \frac{E_c(a_1) + \int_{a_1}^{d_1} P_r(t) \cdot dt}{P_{max}} = d_1 - \frac{E_c(a_1) + \sum_{t=a_1}^{d_1} P_r(t) \cdot dt}{P_{max}} \\ &= 9 - \frac{10 + \sum_{t=1}^{9} P_r(t)}{8} = 9 - \frac{10 + 60}{8} = 0.25 \end{aligned} \tag{4.41}$$

t	$X(t) = P_{max} \cdot d_i - C - \int_t^{d_i} P_r(t) \cdot dt - P_{max} \cdot t$
1	$X(1) = 8 \cdot 9 - 10 - 60 - 8 \cdot 1 = -6$
2	$X(2) = 8 \cdot 9 - 10 - 54 - 8 \cdot 2 = -8$
3	$X(3) = 8 \cdot 9 - 10 - 44 - 8 \cdot 3 = -6$
4	$X(4) = 8 \cdot 9 - 10 - 38 - 8 \cdot 4 = -8$
5	$X(5) = 8 \cdot 9 - 10 - 32 - 8 \cdot 5 = -10$
6	$X(6) = 8 \cdot 9 - 10 - 22 - 8 \cdot 6 = -8$
7	$X(7) = 8 \cdot 9 - 10 - 16 - 8 \cdot 7 = -10$
8	$X(8) = 8 \cdot 9 - 10 - 10 - 8 \cdot 8 = -12$
9	$X(9) = 8 \cdot 9 - 10 - 0 - 8 \cdot 9 = -10$

Tableau 4.5: – Exemple de calcul de s_i

Puis calculons $s_1^{'}$. Nous vérifions $X(t)$ en utilisant la formule 4.34 dans [1, 9], cf. tableau 4.5. Parmi ces valeurs, la distribution de valeurs $X(t)$ est désordonnée, et nous ne trouvons pas l'instant t où $X(t) = 0$. Donc nous ne pouvons pas déterminer la valeur $s_1^{'}$ exacte.

4.8 Conclusion

Dans ce chapitre, nous avons présenté un algorithme d'ordonnancement LSA, applicable aux systèmes temps réel qui récupèrent l'énergie ambiante pour assurer leur autonomie énergétique. La question clé attachée à la gestion de puissance d'un système à récupération d'énergie n'est pas de minimiser la consommation de l'énergie pour maximiser la durée de vie du système comme dans les matériels portables traditionnels. La question est d'adapter l'activité du processeur en fonction de la quantité d'énergie disponible.

Premièrement, nous avons présenté la théorie de l'algorithme LSA et la méthode de calcul de la date de début d'exécution s_i de toute tâche. Puis nous avons analysé la faisabilité et le surcôut d'exécution de l'algorithme LSA sous des conditions différentes d'intégration. Nous avons finalement indiqué les points faibles de l'algorithme LSA, et proposé des solutions d'implémentation.

En résumé de ce chapitre, nous pouvons dégager la conclusion fondamentale : LSA est un ordonnanceur efficace et optimal uniquement sous une liste assez longue d'hypothèses trés restrictives et irréalistes.

Ce constat nous impose donc de simuler cet ordonnanceur et de comparer ses performances réelles à celles d'heuristiques plus simples d'implémentation. En effet, n'oublions pas que LSA est un ordonnanceur qui d'une part est oisif et d'autre part est clairvoyant du point de vue énergétique.

Une question à laquelle nous devons répondre : ne serait-il pas plus interessant d'employer une heuristique totalement non clairvoyante et non oisive au prix d'une baisse acceptable de la qualité de service c'est-à-dire du nombre d'échéances respectées ?

Chapitre 5

Systèmes temps réel alimentés en puissance constante

5.1 Introduction

L'objet de ce chapitre est d'étudier par voie de la simulation plusieurs strategies d'ordonnancement, incluant LSA, et ce lorsque le système temps réel se compose de tâches caractérisées par des contraintes temporelles et des contraintes énergétiques. Plus précisément, nous voulons comparer des heuristiques d'ordonnancement non oisives qui sont des variantes de EDF avec LSA. Nous allons faire des simulations en faisant varier la charge de traitement du processeur (U_p), la capacité du réservoir d'énergie (C) et la puissance de la source énergétique (P_r) et voir l'influence de ces paramètres sur la performance du système.

En premier lieu de ce chapitre, nous proposons de décrire les différentes heuristiques, variantes de EDF que nous avons choisies d'implémenter. Ensuite, nous considérerons successivement des études portant sur la variation des paramètres U_p, C et P_r et mesurerons les critères suivants :

- Le taux d'échéances respectées,
- L'énergie gaspillée pour cause de réservoir d'énergie plein,
- L'énergie gaspillée pour cause d'échéances violées,
- Le nombre d'épuisements du reservoir d'énergie,
- Le surcoût d'exécution.

Dans la dernière section du chapitre, nous établirons un bilan des résultats acquis par cette étude de simulation concernant la performance relative des différents ordonnanceurs envisagés.

5.2 Justification du modèle

Dans ce chapitre, nous réduisons notre étude aux applications temps réel alimentées par une source d'énergie à puissance constante ou quasi-constante. De façon à justifier cette hypothèse, nous montrons par deux exemples que celle-ci est assez réaliste pour un bon nombre d'applications.

Exemple 5.1.

Le corps humain produit constamment de l'énergie (chaleur, mouvement, respiration, etc.).

T. Starner analysait plusieurs sources potentielles d'énergie du corps humain qui peuvent être utilisées pour la conversion électrique en 1996 [51]. Parmis ceux-ci, nous concevons aisément que l'état d'équilibre est atteint lorsque le corps est capable d'équilibrer la quantité de chaleur échangée entre l'intérieur et l'extérieur. Pour cela, le corps humain fabrique continuellement de l'énergie. Il y a donc un transfert thermique entre l'intérieur du corps et l'ambiance extérieure à température variable (cf. figure 5.1).

Figure 5.1: – Convertir la chaleur corporelle en courant électrique

La puissance de l'énergie thermique produite dépend de la différence de temperature entre l'intérieur du corps et l'ambiance extérieure. La température du corps humain est presque constante et égale à 37°C environ. Bien que la température de l'ambiance extérieure soit variable, mais si le corps se trouve dans un local stable longtemps, nous pouvons également la considérer comme constante. Alors une telle source d'énergie renouvelable peut être appelée *source d'énergie renouvelable constante.*

Des scientifiques ont développé une méthode de production d'électricité à partir de la chaleur dégagée par le corps humain. Cette méthode repose sur l'utilisation de générateurs thermoélectriques constitués d'éléments semi-conducteurs. Ils pourraient alimenter notamment les appareils de mesure et de contrôle électronique connectés aux patients dans les services de réanimation (cf. figure 5.2).

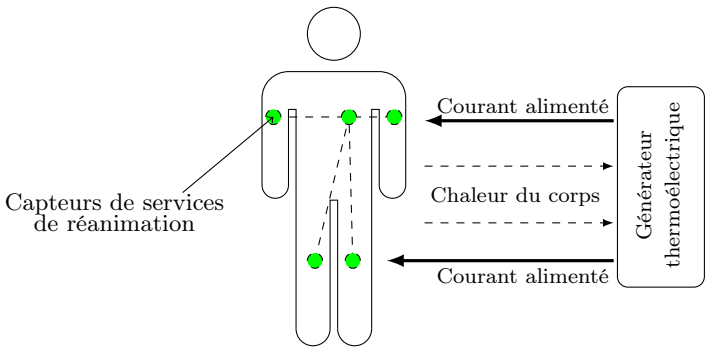


Figure 5.2: – Application à un service de réanimation

Exemple 5.2.

Comme le soleil, les sources lumineuses peuvent produire de l'énergie électrique à partir du rayonnement de lumière. Cette énergie photovoltaïque provient de la conversion de la lumière de source lumineuse en électricité au sein de composants photovoltaïques. Les performances d'une installation photovoltaïque dépendent de la taille des composants et des intensités lumineuses. Une fois les composants photovoltaïques bien orientés (cf. figure 5.3), si les sources lumineuses sont stables, donc l'énergie générée est aussi stable. Alors nous pouvons considérer l'énergie lumineuse "'in-door"' comme *une source d'énergie renouvelable constante.*

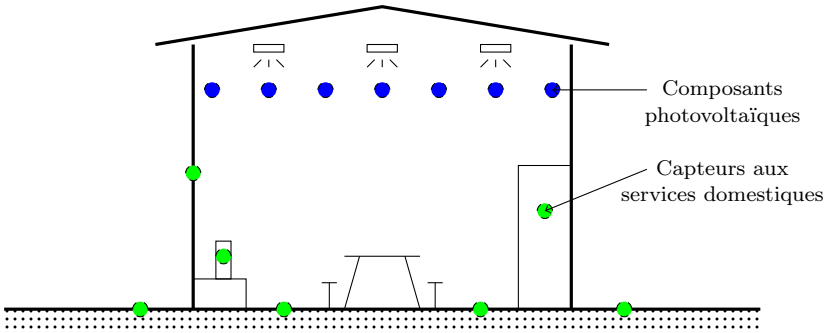


Figure 5.3: – Application de services domotiques

Actuellement, nous intégrons de plus en plus ce nouveau type de récupération d'énergie dans les réseaux domotiques (cf. figure 5.3) et en particulier dans les établissement utilisant la lumière artificielle 24/24.

5.3 Critères de performance mesurés

Afin d'estimer les performances des heuristiques d'ordonnancement et comparer celles-ci avec l'ordonnanceur LSA, nous proposons les six critères suivants :

– **Le taux d'échéances respectées :**
Il s'agit du pourcentage d'instances (requêtes) dont l'exécution s'est achevée avant échéance. Ce paramètre, noté $T_{\text{échéance}}$, représente la qualité de service observée globalement au niveau du système. Il s'agit donc de la mesure de performance la plus importante pour un système temps réel à contraintes fermes.

$$T_{\text{échéance}} = \frac{\text{nombre d'instances satisfaites}}{\text{nombre total d'instances}} \tag{5.1}$$

– **L'énergie gaspillée pour cause de réservoir d'énergie plein :**
Comme son nom l'indique, c'est la quantité d'énergie gaspillée pour cause de débordement du réservoir d'énergie durant toute la période de simulation. Il s'exprime en pourcentage,

calculé entre la somme d'énergie gaspillée et l'énergie totale disponible pendant la simulation. Ce facteur, noté G_1, permet d'évaluer l'efficacité énergétique de différents ordonnanceurs et mesurer l'impact de la capacité du réservoir d'énergie sur la qualité de service.

$$G_1 = \frac{\text{quantité d'énergie gaspillée pour cause de réservoir d'énergie plein}}{\text{énergie totale disponible pendant la simulation}} \tag{5.2}$$

– **L'énergie gaspillée pour cause d'échéances violées :**
Quant à la consommation d'énergie, elle est composée de deux parties essentielles : l'énergie consommée par les tâches qui respectent leurs échéances et celle consommée par les tâches n'ayant pas respecté leur échéance. Dans notre simulation, nous considérons que, si une tâche ne respecte pas son échéance, l'énergie consommée a été gaspillée. Cette mesure, notée G_2, s'exprime aussi en pourcentage. C'est le ratio entre la somme d'énergie gaspillée pour cause d'échéances violées et l'énergie totale disponible pendant la simulation. Ce facteur permet d'évaluer les détails de l'utilisation de l'énergie.

$$G_2 = \frac{\text{quantité d'énergie gaspillée pour cause d'échéances violées}}{\text{énergie totale disponible pendant la simulation}} \tag{5.3}$$

– **Le nombre d'épuisements du réservoir d'énergie :**
Il s'agit du nombre de fois où le reservoir devient vide. Selon ce paramètre, nous pouvons déterminer le nombre de cycles de décharge du réservoir d'énergie, et donc nous pouvons en déduire la durée moyenne de vie du réservoir par rapport aux spécifications du fabricant.

– **Le surcoût d'exécution (overhead en anglais) :**
Le surcoût de l'ordonnancement est lié au nombre d'opérations engendrées dans l'exécution du programme. L'algorithme se compose de différentes opérations mathématiques. Nous définissons des coefficients pondérants pour différentes opérations. A la fin de simulation, nous obtenons le surcoût en sommant les coefficients pondérants.

A propos du simulateur, la puissance de source d'énergie P_r, la capacité du réservoir d'énergie C et le facteur d'utilisation du processeur U_p sont les trois paramètres caractéristiques du point de vue énergétique. Nous faisons varier P_r pour simuler les diverses sources d'énergie, et aussi nous pouvons construire des applications virtuelles en changeant la valeur de U_p, la charge du processeur. Ainsi, nous considérerons des systèmes faiblement chargés, moyennement chargés et lourdement chargés. La capacité du réservoir d'énergie C peut affecter directement la taille de l'équipement. Normalement, la taille du réservoir définie par son poids et/ou son volume est proportionnelle à sa capacité de stockage. Pour les micro-systèmes embarqués, nous ne pouvons pas négliger la taille du réservoir d'énergie. La miniaturisation est la tendance pour les produits embarqués, alors le dimensionnement de la taille d'un réservoir d'énergie est un problème épineux.

Dans ce chapitre, nous proposons trois axes de simulation autour des paramètres P_r, C et U_p. Nous allons distinctement simuler différentes applications selon le plan suivant :

– En supposant P_r et C fixés, nous faisons varier la valeur U_p de 0 à 100%.
– En supposant P_r et U_p fixés, nous faisons varier la capacité du réservoir d'énergie C de 0 à une valeur finie.
– En supposant C et U_p fixés, nous faisons varier la puissance P_r de 0 à P_{max}.

5.4 Heuristiques étudiées

5.4.1 Introduction

Dans cette section, nous allons proposer et étudier plusieurs algorithmes d'ordonnancement non oisifs basés sur EDF en vue de les comparer à LSA. Ceux-ci ont la particularité d'être simples à implémenter dans un système d'exploitation temps réel. Ils sont non-clairvoyants, c'est-à-dire, ne necessitent pas la connaissance du profil de la source d'énergie. Ils vont se différencier les uns des autres par la stratégie mise en oeuvre pour gérer les situations où le réservoir d'énergie devient vide.

Les pseudo-codes des heuristiques sont fournis dans l'annexe A.

5.4.2 Ordonnancement EDt (Earliest Deadline with test)

EDt (*Earliest Deadline with test*) ordonnance les tâches au plus tôt selon Earliest Deadline. Avant de lancer l'exécution de la tâche la plus prioritaire, nous faisons d'abord un test pour savoir si l'énergie contenue dans le réservoir d'énergie est suffisante pour exécuter cette tâche. Si oui, nous l'exécutons. Sinon, nous mettons le processeur en mode veille jusqu'à ce que le réservoir contienne suffisamment d'énergie pour l'exécuter. Au cours de l'ordonnancement, nous testons le niveau d'énergie à chaque unité de temps.

Exemple 5.3. *Soit une configuration de tâches périodiques ER,* $\mathscr{T} = \{ \tau_i | 1 \leq i \leq 2, avec\ \tau_i = (a_i, E_i, T_i) \}$. *Supposons que* $\tau_1 = (0, 24, 10)$, $\tau_2 = (0, 8, 5)$. *Nous supposons aussi que* $C = 12$, $E_c(0) = 4$, $P_r = 4$ *et* $P_{max} = 8$.

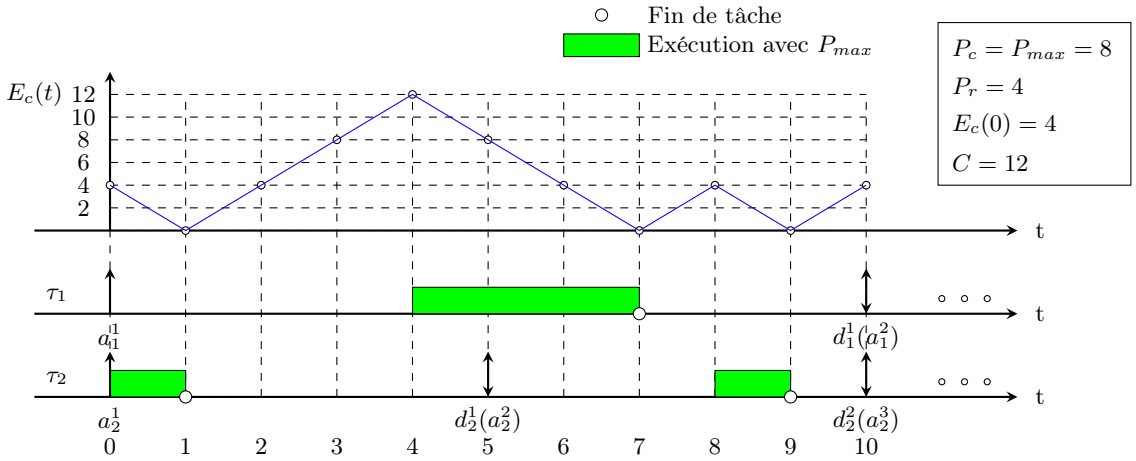


Figure 5.4: – Séquence produite par EDt

Au moment $t = 0$, les instances τ_1^1 et τ_2^1 sont réveillées simultanément, mais τ_2^1 est plus prioritaire avec $d_2^1 = 5$. $E_c(0) + E_r(0,1) = E_c(0) + \int_0^1 P_r \cdot dt = 4 + 4 \cdot (1 - 0) = 8$. Comme l'énergie contenue dans le réservoir d'énergie est suffisante pour exécuter cette instance, nous l'exécutons tout de suite. Lorsque cette instance termine, le réservoir est vide. Le processeur passe alors en mode veille pour se recharger jusqu'à ce que l'énergie contenue dans le réservoir soit suffisante. A $t = 4$, $E_c(4) + E_r(4,7) = E_c(4) + \int_4^7 P_r \cdot dt = 12 + 4 \cdot (7 - 4) = 24$. Cela suffit pour exécuter l'instance τ_1^1, $E_1 = 24$, donc nous l'exécutons dès l'instant 4. Ensuite, le système se charge et quand l'énergie contenue est suffisante, l'instance τ_2^2 commence à s'exécuter. La séquence produite par EDt est illustrée sur la figure 5.4.

5.4.3 Ordonnancement EDi (Earliest Deadline with inserted idle times)

EDi (*Earliest Deadline with inserted idle times*) ordonnance les tâches au plus tôt selon Earliest Deadline jusqu'à ce que le réservoir d'énergie soit vide. Lorsque le réservoir devient vide, la tâche est alors interrompue, nous mettons alors le processeur en mode veille jusqu'à la prochaine date de réveil. Puis nous reprenons l'exécution par Earliest Deadline.

Exemple 5.4. *Reprenons la même configuration de tâches de l'exemple 5.3.*

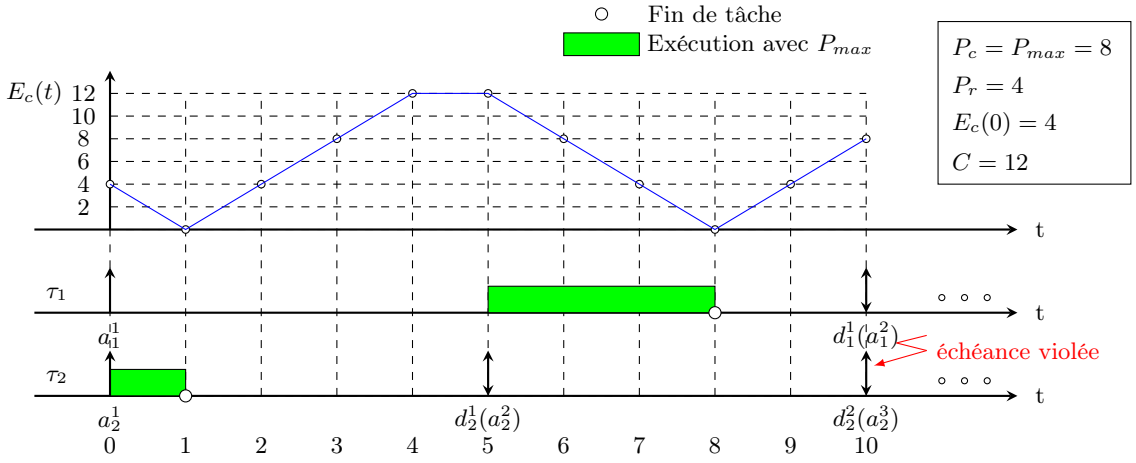


Figure 5.5: – Séquence produite par EDi

A $t = 0$, les instances τ_1^1 et τ_2^1 sont réveillées en même temps, mais τ_2^1 est plus prioritaire que τ_1^1. L'ordonnanceur traite l'instance τ_2^1 directement sans vérifier la quantité d'énergie, jusqu'à ce que le réservoir d'énergie soit vide. Puis le processeur doit rester inactif de façon à permettre la recharge d'énergie jusqu'à la prochaine date de réveil $t = 5$. A $t = 5$, τ_2^2 est réveillée, cependant elles possèdent la même priorité. Dans cet exemple, nous faisons d'abord ordonnancer l'instance τ_1^1. Alors τ_1^1 commence à s'exécuter dès $t = 5$, et se termine à $t = 8$ où le réservoir d'énergie est juste épuisé. L'ordonnanceur met alors le processeur en mode veille jusqu'à la prochaine date de réveil. Malheureusement τ_2^2 est rejetée, car son échéance est violée. La séquence produite par EDi est illustrée sur la figure 5.5.

5.4.4 Ordonnancement EDd (Earliest Deadline with discard)

EDd (*Earliest Deadline with discard*) ordonnance les tâches au plus tôt selon Earliest Deadline jusqu'à ce que le réservoir d'énergie soit vide. Quand le réservoir d'énergie est vide, nous supprimons toutes les instances réveillées et nous laissons le processeur en mode veille jusqu'à la prochaine date de réveil, pour permettre une recharge d'énergie.

Exemple 5.5. *Reprenons la même configuration de tâches de l'exemple 5.3.*

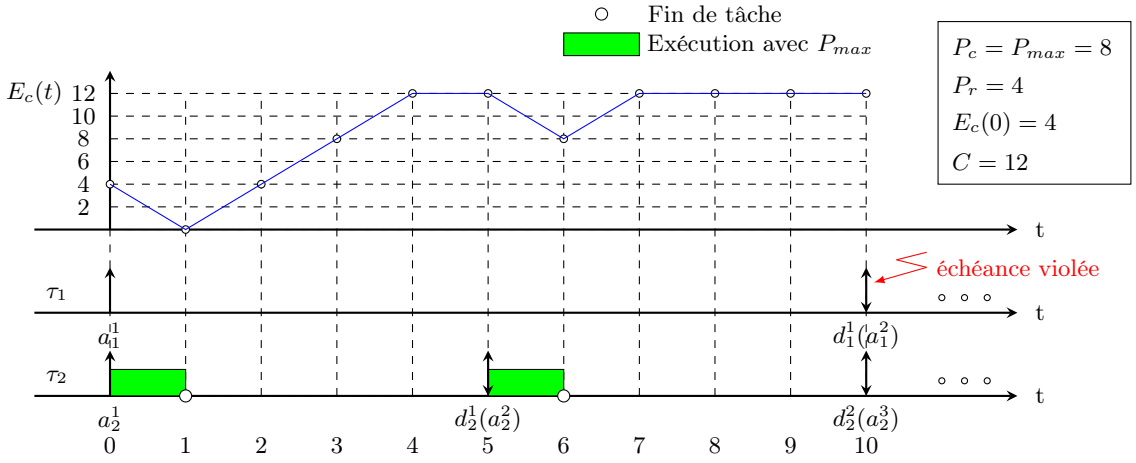


Figure 5.6: – Séquence produite par EDd

Comme dans les exemples 5.3 et 5.4, à $t = 0$, les instances τ_1^1 et τ_2^1 sont réveillées en même temps. Nous traitons l'instance τ_2^1 directement sans vérifier la quantité d'énergie. Quand τ_2^1 est terminée, le réservoir d'énergie est épuisé et τ_1^1 est supprimée. Alors le processeur doit rester inactif de façon à permettre la recharge d'énergie jusqu'à la prochaine date de réveil $t = 5$. A $t = 5$, τ_2^2 commence à s'exécuter. La séquence produite par EDd est illustrée sur la figure 5.6.

5.4.5 Ordonnancement EDu (Earliest Deadline with inserted unit time)

EDu (*Earliest Deadline with inserted unit time*) fonctionne de façon similaire à EDi. Par rapport à EDi, l'ordonnanceur EDu laisse le processeur en mode veille pendant une seule unité de temps quand le réservoir d'énergie devient vide. Puis nous reprenons l'exécution par earliest deadline. Comme avec EDi et contrairement à EDd, il n'y a pas de suppression de tâches sauf au moment où une tâche viole son échéance.

Exemple 5.6. *Reprenons la même configuration de tâches de l'exemple 5.3.*

τ_2^1 est ordonnancée jusqu'à ce que le réservoir d'énergie soit vide. Une fois que le réservoir d'énergie devient vide, nous laissons le processeur en mode veille juste une unité de temps pour se charger. Puis nous reprenons l'exécution par Earliest Deadline jusqu'à ce que le réservoir d'énergie soit vide. La séquence produite par EDu est illustrée sur la figure 5.7. Nous voyons que

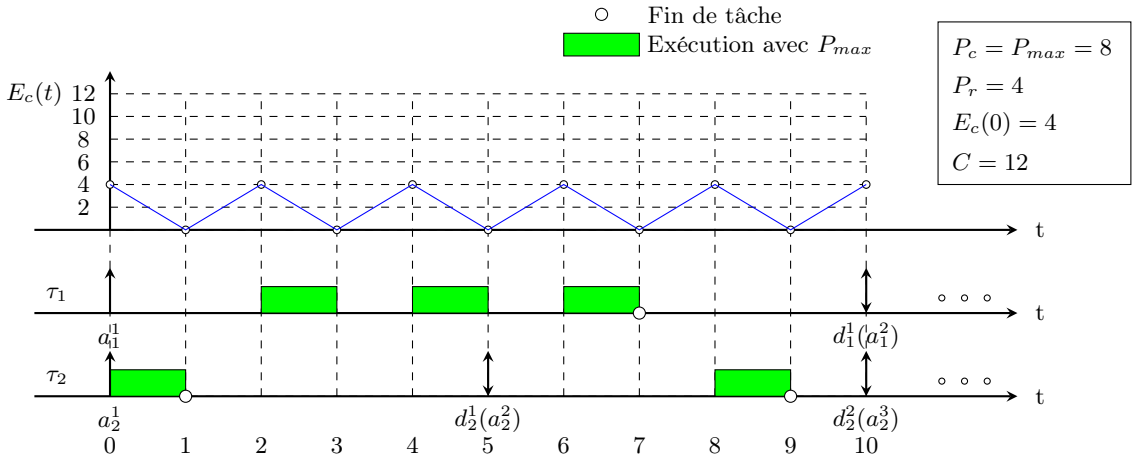


Figure 5.7: – Séquence produite par EDu

cet ensemble de tâches est ordonnançable fiablement, mais le niveau d'énergie E_c reste proche du niveau bas du réservoir.

5.4.6 Ordonnancement EDc (Earliest Deadline with discard the current task)

EDc (*Earliest Deadline with discard the current task*) fonctionne de façon similaire à EDd. Par rapport à EDd, quand le réservoir d'énergie est vide, l'ordonnanceur EDc juste supprime la tâche courante au lieu de jeter toutes les tâches réveillées, et nous laissons le processeur en mode veille jusqu'à la prochaine date de réveil. Puis nous reprenons l'exécution par Earliest Deadline.

Exemple 5.7. *Reprenons la même configuration de tâches de l'exemple 5.3.*

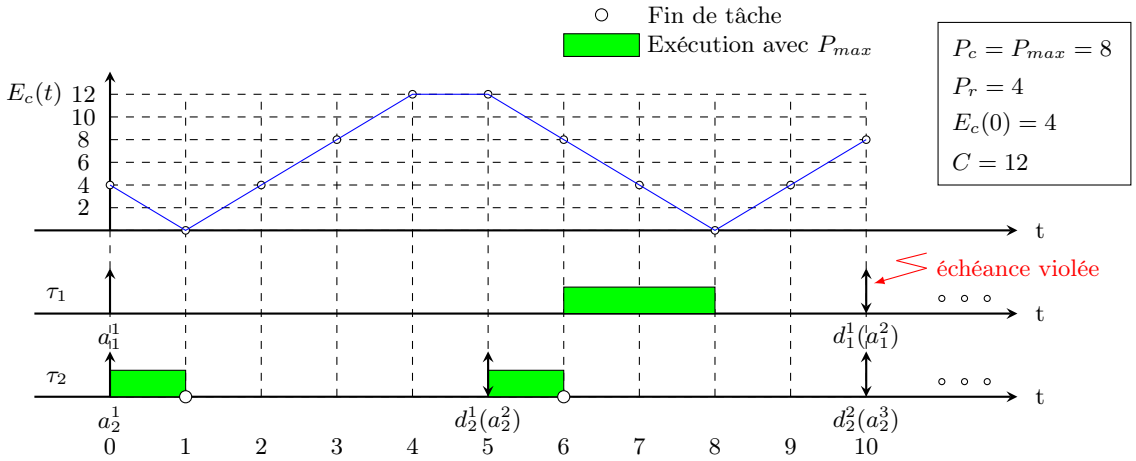


Figure 5.8: – Séquence produite par EDc

La figure 5.8 nous illustre la séquence produite par EDc. Nous voyons que le réservoir d'énergie devient vide à $t = 1$. Selon la politique EDc, l'ordonnanceur doit supprimer la tâche courante,

mais heureusement τ_2^1 est juste terminée à $t = 1$, donc l'ordonnanceur ne supprime aucune tâche. Nous laissons le processeur en mode veille jusqu'à la prochaine date de réveil $t = 5$, puis reprenons l'exécution de tâches par earliest deadline. A $t = 5$, τ_1^1 et τ_2^2 ont la même priorité. Afin d'expliquer le plus clairement possible la théorie EDc, nous laissons l'ordonnanceur exécuter τ_2^2 d'abord. τ_2^2 est terminée à $t = 6$, puis l'exécution de τ_1^1 est commencée tout de suite. A $t = 8$, le réservoir d'énergie est épuisé, mais τ_1^1 n'est pas encore terminée, donc elle est supprimée par l'ordonnanceur EDc.

5.5 Description du simulateur

L'environnement de simulation consiste en un noyau de simulation (ordonnanceur) accompagné d'un certain nombre de composantes impliquées dans la gestion et l'analyse des simulations. Le programme de simulation a été implémenté en langage *C*.

L'architecture fonctionnelle du simulateur est présentée dans la figure 5.9. Notre simulateur se compose principalement de :

- Un générateur de tâches périodiques.
- Un profil énergétique (modélisation d'une sources d'énergie renouvelable). Dans ce chapitre, nous considérerons que la puissance produite par la source énergétique sera invariable au cours du temps.
- Un ordonnanceur monoprocesseur, préemptif et non oisif.
- Un module display permettant d'illustrer la séquence produite et le niveau d'énergie du réservoir d'énergie.
- Un module d'enregistrement, qui enregistre toutes les données de simulation.

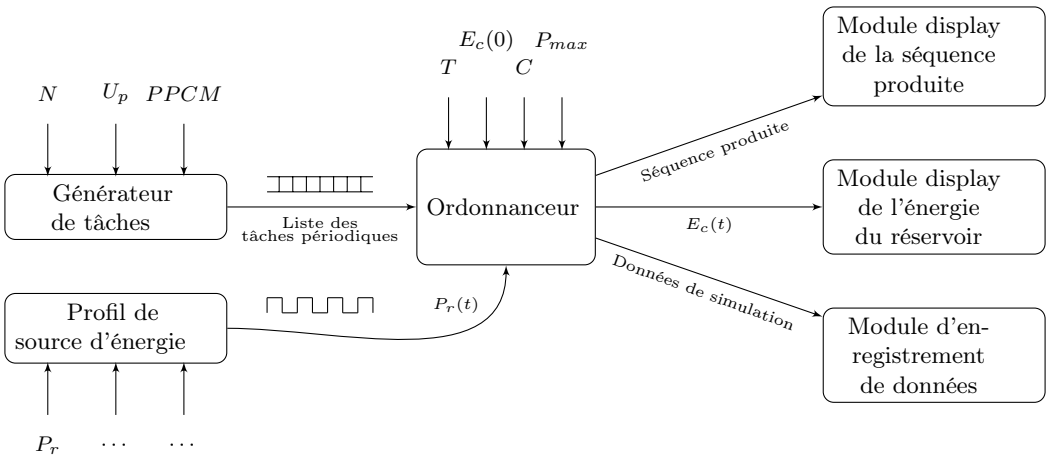


Figure 5.9: – Architecture fonctionnelle du simulateur

Un générateur de configurations de tâches périodiques a été conçu sur la base de celui décrit par Martineau dans [30]. Il accepte en entrée plusieurs paramètres : le nombre de tâches N souhaité pour la configuration, le $PPCM$ des périodes des tâches, le facteur d'utilisation du processeur U_p. En sortie, nous obtenons une configuration de tâches périodiques : $\mathscr{T} = \{ \tau_i | 1 \leq$

$i \leq n, avec\ \tau_i = (a_i, E_i, T_i)\ \}$.

Un générateur de source d'énergie, qui peut engendrer des valeurs de P_r pour simuler les diverse sources d'énergie. Les paramètres nécessaires en entrée par ce générateur dépend du type de source d'énergie souhaitée. Par exemple, dans ce chapitre nous voulons une source d'énergie constante, il est donc suffisant d'entrer une valeur de paramètre P_r. En sortie, nous obtenons un profil de source d'énergie virtuel, ses valeurs de puissance instantanée $P_r(t)$ sont enregistrées dans un tableau.

Une applicaion simulée se caractérise par les paramètres suivants : le nombre de configurations de tâches, la puissance maximale du processeur P_{max}, la durée de simulation T, la capacité du réservoir d'énergie C et son énergie initiale $E_c(0)$. Pour chaque application, 30 configurations de tâches différentes ont été générées, chacune d'elles comprenant 6 tâches avec un PPCM de périodes égal à 300. Les simulations ont été effectuées sur 5 hyperpériodes ($T = 5 \cdot PPCM$). Les performances des algorithmes d'ordonnancement ont été évaluées en faisant varier le facteur d'utilisation du processeur U_p ainsi que la capacité du réservoir d'énergie C et la puissance de source d'énergie P_r appliqués aux systèmes.

Enfin, les résultats de simulation sont affichées de manière graphique. Il permet de présenter la séquence produite en figurant le comportement des différentes tâches (réveils, exécutions, préemptions, etc.) et la variation de l'énergie du réservoir d'énergie.

5.6 Analyse des résultats expérimentaux

L'objectif de ce paragraphe est de décrire les résultats de simulations effectuées dans le but d'évaluer comparativement la performance des stratégies précédemment décrites. Les expérimentations en simulation tendent également à évaluer l'impact des paramètres P_r et C sur les performances relatives des différentes stratégies. Dans toutes les simulations, nous utilisons un processseur de puissance maximale égale à 8.

5.6.1 Etude avec puissance de source et capacité du réservoir d'énergie fixés

Nous effectuons dans ce paragraphe une étude avec les caractéristiques suivantes :
- La puissance de source d'énergie P_r est fixée à 6.
- La capacité du réservoir d'énergie C est fixé à 10.
- Le réservoir d'énergie est plein initialement, $E_c(0) = C = 10$.
- Nous générons des configurations de tâches ayant un taux d'utilisation du processeur U_p compris entre 0 et 1.

- Evaluation du taux d'échéances respectées

U_p varie dans [0,1]. De la figure 5.10, nous trouvons facilement que tous les ordonnanceurs ont une performance qui se dégrade avec l'augmentation de U_p. Quelle est la raison provoquant la dégradation des performances d'un système ? Nous pensons que la pénurie d'énergie est la raison fatale. A cause de l'augmentation du facteur d'utilisation des tâches, la demande d'énergie augmente aussi. L'ordonnanceur est obligé de récolter de l'énergie de l'environnement en sacrifiant

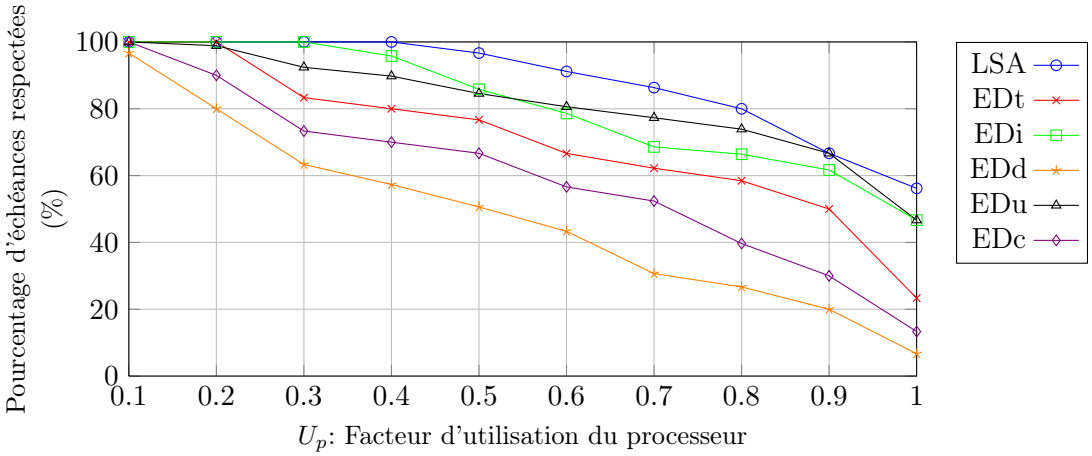


Figure 5.10: – Evaluation du taux d'échéances respectées

une partie de la ressource du processeur ; Il cause donc l'ascension du taux de défaillance des tâches.

Pendant la simulation, la performance de LSA se trouve toujours au plus haut niveau ce qui confirme son optimalité. Par contre, la performance d'EDd est toujours au plus bas niveau. Quand nous mettons $U_p = 0.6$ ce qui correspond à un processeur moyennement chargé, l'ordonnanceur LSA peut satisfaire 90% des échéances des tâches. Cependant, l'ordonnanceur EDd peut seulement satisfaire environs 40% des échéances des tâches. Pour $U_p = 1$, le taux d'échéances respectées de LSA atteint 60% alors que la performance d'EDd se dégrade à 10%. Globalement, plus le système est lourdement chargé, plus les gains de performance de LSA par rapport aux heuristiques sont significatifs.

Nous pouvons conclure ici, à la hierarchisation suivante des stratégies d'ordonnancement : $LSA > EDu \approx EDi > EDt > EDc > EDd$ en termes de taux d'échéances satisfaites.

- Evaluation du taux d'énergie gaspillée

L'ordonnanceur idéal est celui qui doit optimiser l'utilisation de l'énergie récupérée dans l'environnement, c'est-à-dire que l'énergie ne doit être gaspillée que lorsque le réservoir est plein et qu'aucune tâche ne demande à s'exécuter. Comme nous l'avons expliqué dans les paragraphes du dessus, il y a deux possibilités envisageables pour analyser les circonstances de la dissipation d'énergie : l'énergie gaspillée pour cause de réservoir d'énergie plein (cf. figure 5.11) et l'énergie gaspillée pour cause d'échéances violées (cf. figure 5.12).

Sur la figure 5.11, nous décrivons l'évolution de l'énergie gaspillée pour cause de réservoir d'énergie plein. Les courbes décroissent en suivant U_p, sauf celle d'EDt. La courbe la plus haute correspond à EDt. Elle est différente des autres. Parce qu'il vérifie l'énergie requise avant de pouvoir exécuter une tâche, le temps d'attente devient donc de plus en plus long, puis le réservoir d'énergie est plus facile à déborder, il a donc besoin d'un réservoir d'une plus grande capacité pour diminuer le gaspillage d'énergie.

Les ordonnanceurs EDd, EDi, EDc et LSA ont une caractéristique commune : le taux d'énergie gaspillée pour cause de réservoir plein décroit pour $U_p = [0.1, 0.6]$. A partir de $U_p = 0.6$, ces courbes restent quasiment stables. L'ordonnanceur EDu est celui qui perd de moins en moins d'énergie pour cause de réservoir plein avec l'accroissement de U_p. A partir de $U_p = 0.8$, le réservoir d'énergie ne déborde plus.

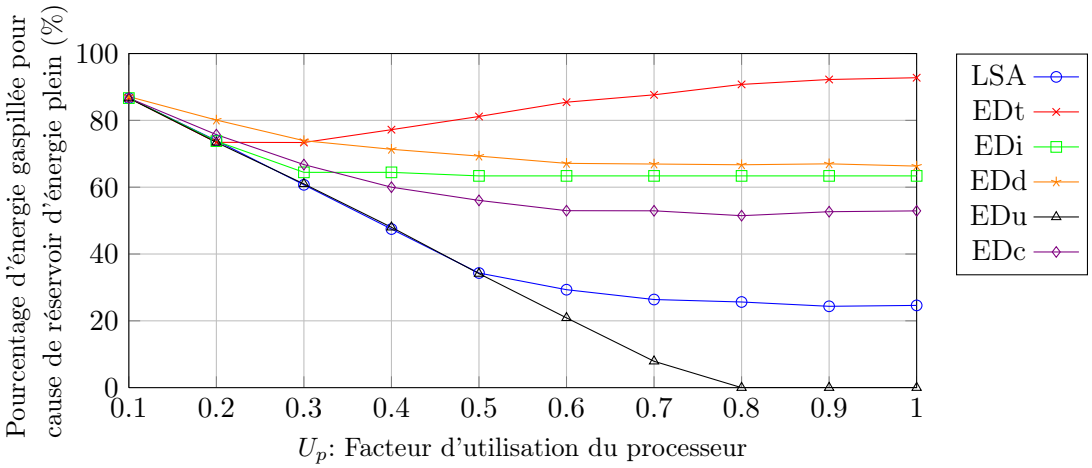


Figure 5.11: – Evaluation du taux d'énergie gaspillée pour cause de réservoir d'énergie plein

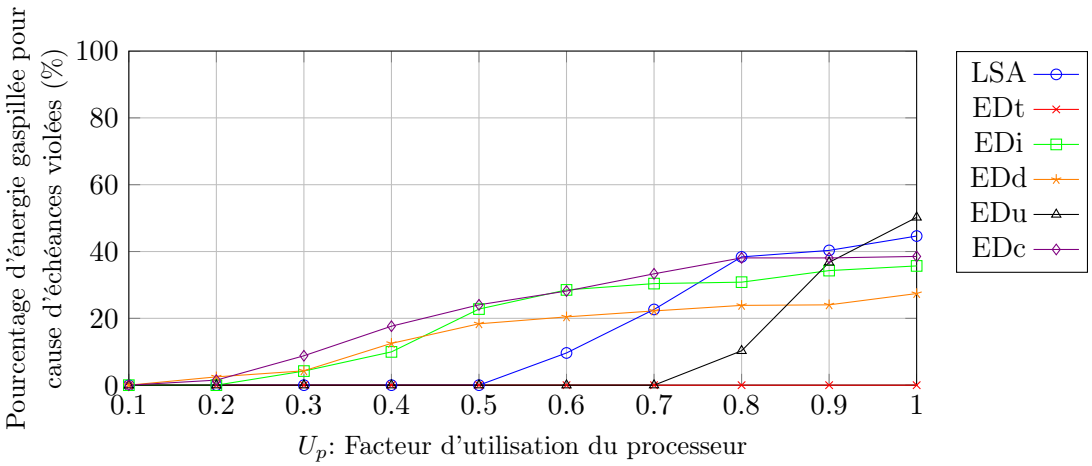


Figure 5.12: – Evaluation du taux d'énergie gaspillée pour cause d'échéances violées

La figure 5.12 nous présente l'évaluation du taux d'énergie gaspillée pour cause d'échéances violées. En raison de l'accroissement du taux d'utilisation des tâches, nous voyons que l'énergie gaspillée pour cause d'échéances violées est devenue plus grande. Les courbes des ordonnanceurs EDc, EDi et EDd évoluent en suivant une pente douce. L'énergie gaspillée à cause d'échéances violées d'EDu est toujours nulle avant $U_p = 0.7$. A partir de $U_p = 0.7$, sa performance se dégrade

rapidement : beaucoup de tâches ne peuvent pas être terminées avant l'échéance. Les courbes suivent donc une pente rapide. Le comportement de LSA fait un compromis entre ces deux cas. En contraste, le comportement d'EDt est toujours très stable : il est indépendant de U_p. En effet, EDt fait d'abord la vérification d'énergie pour tester si l'énergie contenue dans le réservoir est suffisante pour exécuter une tâche. Si oui, nous l'exécutons. Si non, nous ne faisons pas commencer l'exécution de la tâche et donc aucune énergie n'est gaspillée.

- Evaluation du nombre d'épuisements du réservoir d'énergie

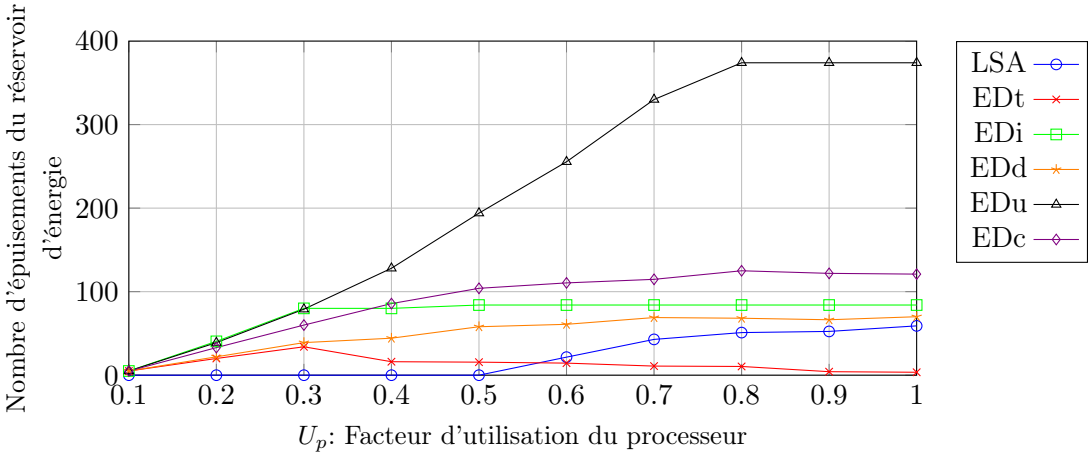


Figure 5.13: – Evaluation du nombre d'épuisements du réservoir d'énergie

Nous comptons le nombre d'épuisements du réservoir d'énergie en vue d'estimer la vie du réservoir. En effet, la vie du réservoir d'énergie souvent dépend du nombre de cycles charge/décharge. Le grand nombre d'épuisements signifie que le réservoir d'énergie se décharge fréquemment, et sa performance en termes de longévité va être dégradée.

Nous observons sur la figure 5.13, que pendant la durée de la simulation, le nombre d'épuisements du réservoir d'énergie de tous les ordonnanceurs, sauf EDu, augmentent bien mais lentement lorsque $U_p < 0.6$. A partir de $U_p = 0.6$, les courbes sont à peu près stables, et donnent des valeurs inférieures à 120. Nous remarquons surtout que pour LSA, ce nombre est toujours maintenu à zéro avant $U_p = 0.5$. Nous constatons le cas particulier de l'ordonnanceur EDu, très spécial : son nombre d'épuisements du réservoir d'énergie est étroitement lié à la valeur de U_p. Il devient progressivement plus grand avec l'augmentation de U_p. Quand $U_p = 0.8$, il arrive au maximum avec une valeur égale à 374 fois. Cela signifie que le niveau d'énergie contenue dans le réservoir tend à toujours rester très bas c'est-à-dire qu'il ne reste toujours qu'un fond d'énergie en réserve, affectant beaucoup la vie du réservoir d'énergie.

5.6.2 Etude avec puissance de source et facteur d'utilisation du processeur fixés

Nous effectuons dans ce paragraphe une étude avec les caractéristiques suivantes :

- La puissance de source d'énergie P_r est fixée à 6.
- Le facteur d'utilisation du processeur U_p est successivement fixé à 0.2, 0.5 et 0.8. $U_p = 0.2$ correspond au système faiblement chargé, $U_p = 0.5$ correspond au système moyennement chargé et $U_p = 0.8$ correspond au système lourdement chargé.
- Le réservoir d'énergie est plein initialement, $E_c(0) = C$.
- Nous faisons varier la capacité du réservoir pour évaluer son impact sur la performance d'un système.

- Evaluation du taux d'échéances respectées

Le choix de la capacité du réservoir d'énergie est crucial dans la conception d'un système embarqué. Avec une grande capacité du réservoir, il peut augmenter le taux d'échéances respectées. Mais une telle méthode n'est pas acceptable sur les systèmes embarqués à cause de la taille et donc du poids du réservoir. Dans cette partie de travail, nous allons déterminer le dimensionnement optimal du réservoir d'énergie pour une tolérance aux échéances.

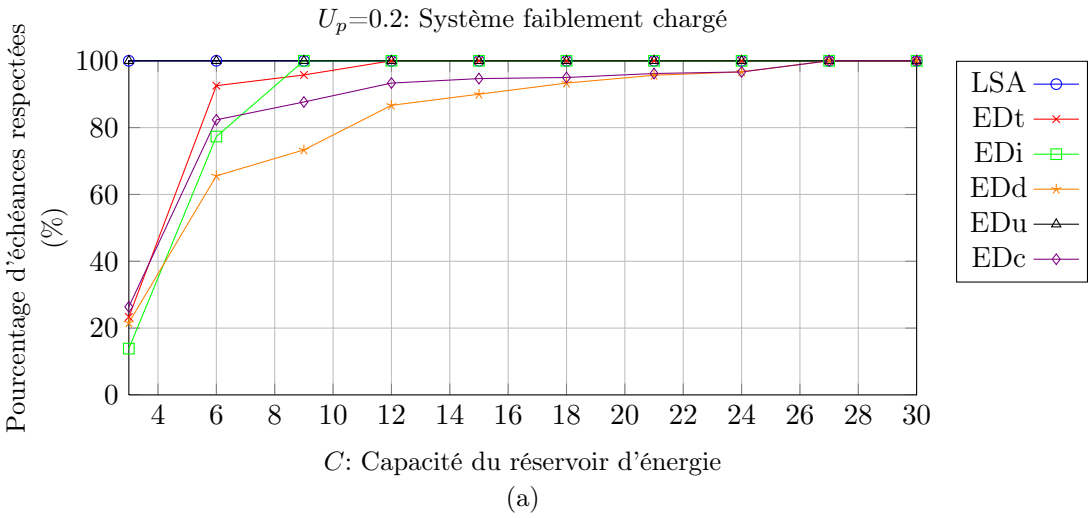
La figure 5.14 (a) nous illustre l'évaluation du taux d'échéances respectées pour un système faiblement chargé ($U_p = 0.2$). Quand $C < 15$, la différence entre les ordonnanceurs est très grande, mais avec l'augmentation de C, les écarts sont de plus en plus minces. Quand nous laissons $C = 30$, nous trouvons que LSA et EDu peuvent satisfaire 100% des échéances. Mais les autres sont capables de respecter 10% - 30% des échéances. Quand $C \geq 18$, les performances des ordonnanceurs sont quasi-identiques. Lorsque la capacité du réservoir est supérieur à 27, tous les ordonnanceurs peuvent conduire à satisfaire 100% des échéances.

La figure 5.14 (b) nous montre l'évaluation du taux d'échéances respectées pour un système moyennement chargé ($U_p = 0.5$). Avec cette plus grande valeur de U_p, les performances des ordonnanceurs se dégradent naturellement. Les performances de LSA et EDu sont optimales. Comme dans la figure précédente, il apparaît clairement que les comportements d'EDi, EDc, EDt et EDd sont très sensibles à la taille du réservoir d'énergie. Afin de garantir 40% des échéances, ils exigent un réservoir d'énergie d'une capacité minimum de 16, et nous conseillons un réservoir d'une capacité minimum de 40 pour garantir 90% des échéances satisfaites. Afin de respecter parfaitement les échéances pour tous les ordonnanceurs, la capacité du réservoir doit être supérieur à 72.

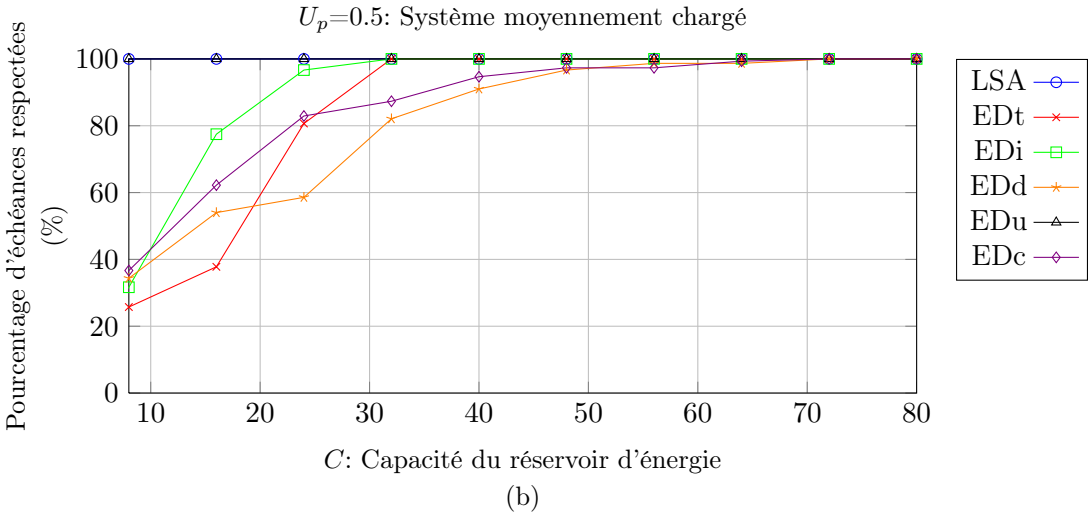
La figure 5.14 (c) nous présente l'évaluation du taux d'échéances respectées pour un système lourdement chargé ($U_p = 0.8$). En comparant avec les courbes précédentes, une plus grande capacité du réservoir est nécessaire pour assurer le même niveau de performance. Aucun ordonnanceur n'est capable de réaliser 100% des échéances satisfaites, même si la capacité du réservoir est supérieure à 150. Par conséquent, l'utilisation d'une plus grande capacité du réservoir pour améliorer la performance n'est pas une solution.

- Evaluation du taux d'énergie gaspillée

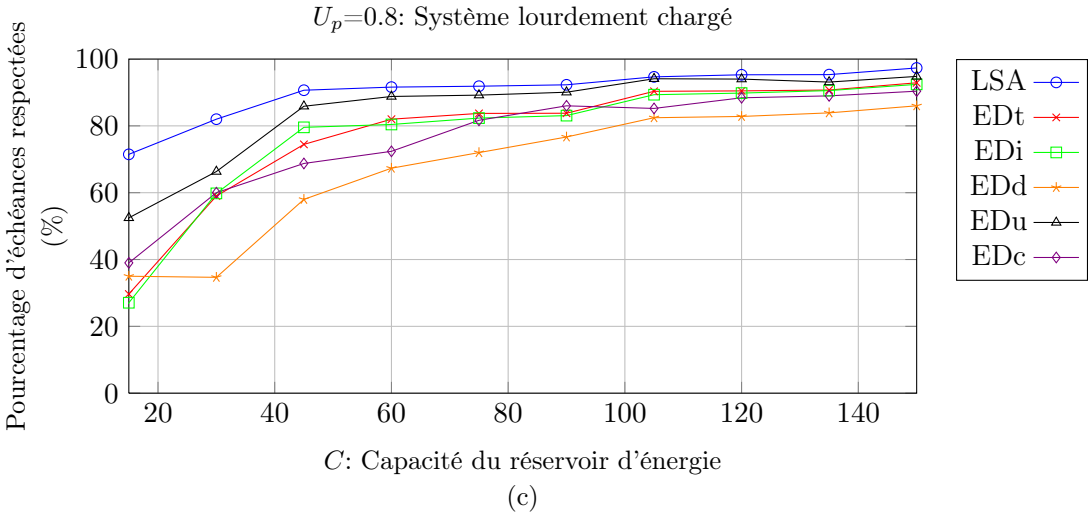
La figure 5.15 nous illustre les taux d'énergie gaspillée pour cause de réservoir d'énergie plein. Nous pouvons conclure que les courbes convergent uniformément vers une valeur définie avec l'augementation de C, par exemple, 75% pour $U_p = 0.2$, 35% pour $U_p = 0.5$ et 0% pour $U_p = 0.8$.



(a)



(b)



(c)

Figure 5.14: – Evaluation du taux d'échéances respectées

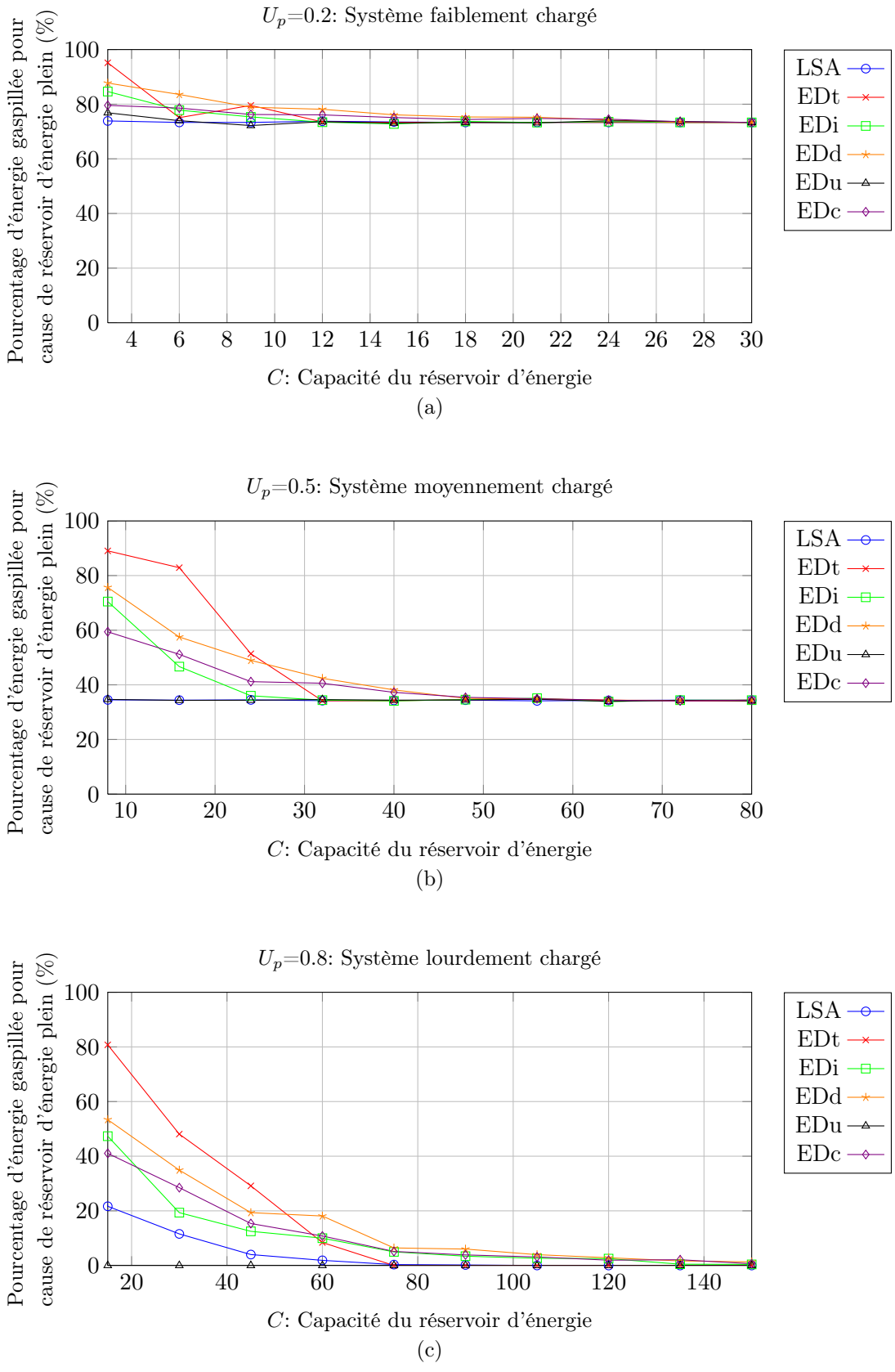


Figure 5.15: – Evaluation du taux d'énergie gaspillée pour cause de réservoir d'énergie plein

Sous cet aspect, le taux de gaspillage d'énergie d'EDt est le plus grand, tandis que celui d'EDu est le plus petit.

Sur la figure 5.15 (a), quand $C = 12$, les comportements des ordonnanceurs sont quasi-identiques. Sur la figure 5.15 (b), quand $C = 40$, les comportements des ordonnanceurs sont quasi-identiques. Sur la figure 5.15 (c), quand $C = 90$, les comportements des ordonnanceurs sont quasi-identiques.

Nous voyons aussi que l'impact de C est plus fort lorsque le système est faiblement chargé sur le taux d'énergie gaspillée pour cause de réservoir d'énergie plein.

La figure 5.16 nous présente les résultats sur l'évaluation du taux d'énergie gaspillée pour cause d'échéances violées. Les courbes sont convergentes progressivement vers une valeur définie avec l'augementation de C. Avec l'augmentation de U_p, ce n'est pas une solution efficace que d'augmenter indéfiniment la capacité du réservoir d'énergie pour diminuer le taux d'énergie gaspillée. Dans ce type d'évaluation, nous constatons que la performance d'EDd se dégrade de façon importante, tandis que celle d'EDt est la meilleure.

- Evaluation du nombre d'épuisements du réservoir d'énergie

La figure 5.17 présente les résultats sur l'évaluation du nombre d'épuisements du réservoir d'énergie. A cause de la spécificité de l'ordonnanceur EDu, le nombre d'épuisements du réservoir reste toujours le plus grand, et il est proportionnel au facteur d'utilisation U_p, mais n'est pas vraiment corrélé à C. Au contraire, celui de l'ordonnanceur EDt est le plus petit.

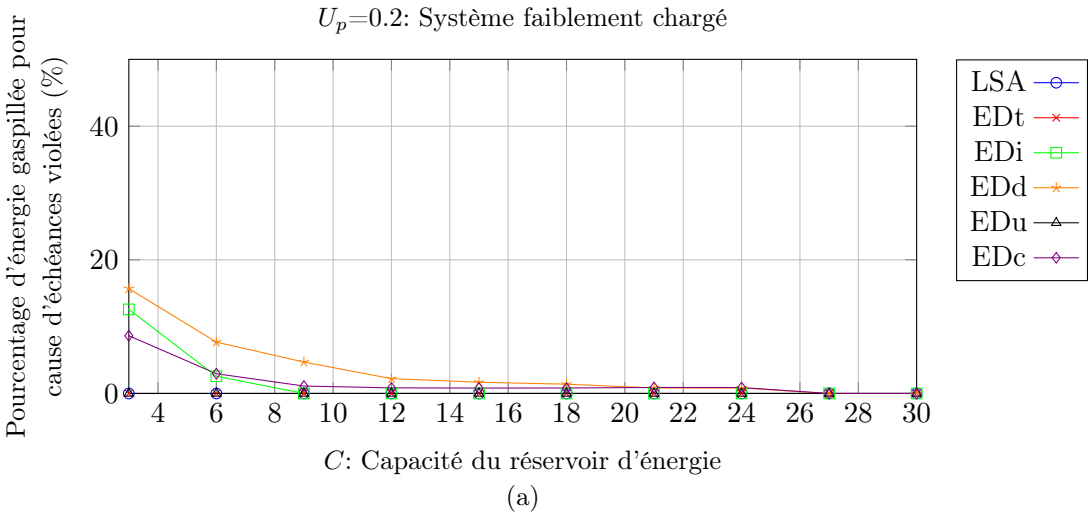
5.6.3 Etude avec facteur d'utilisation du processeur et capacité du réservoir d'énergie fixés

Nous effectuons dans ce paragraphe une étude avec les caractéristiques suivantes :
- La capacité du réservoir d'énergie C est fixé à 10.
- L'énergie initiale du réservoir d'énergie est pleine, $E_c(0) = C = 10$.
- Nous générons des configurations de tâches ayant un taux d'utilisation du processeur $U_p = 0.5$.
- Nous faisons varier la puissance de source d'énergie pour évaluer son impact sur la performance d'un système.

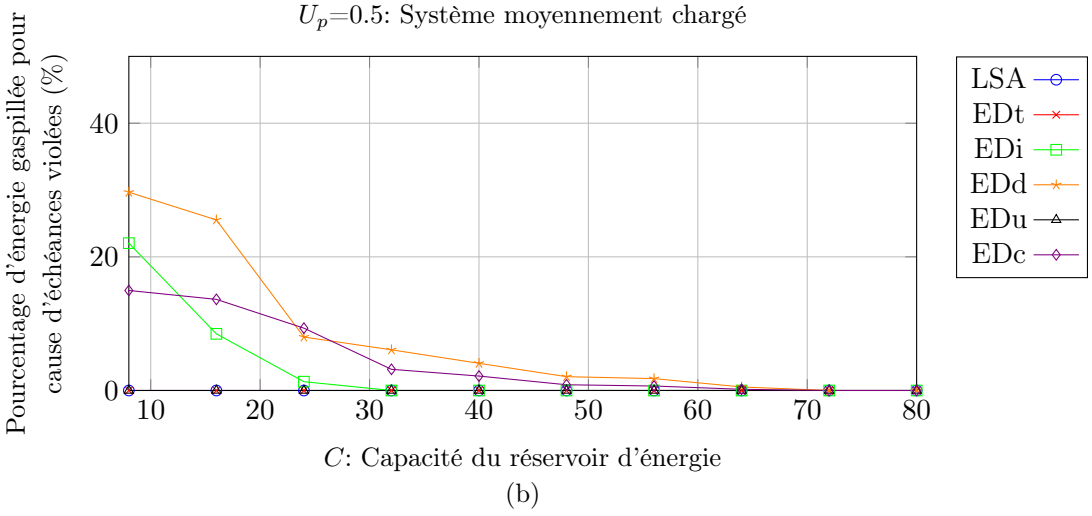
- Evaluation du taux d'échéances respectées

La puissance de source P_r influence beaucoup la durée de fonctionnement durable de système. Surtout pour un système autonome dont la capacité de stockage d'énergie est généralement fixée car limitée par la taille de la batterie ou du condensateur embarqué, donc nous devons exploiter au maximum son pouvoir de récupération d'énergie.

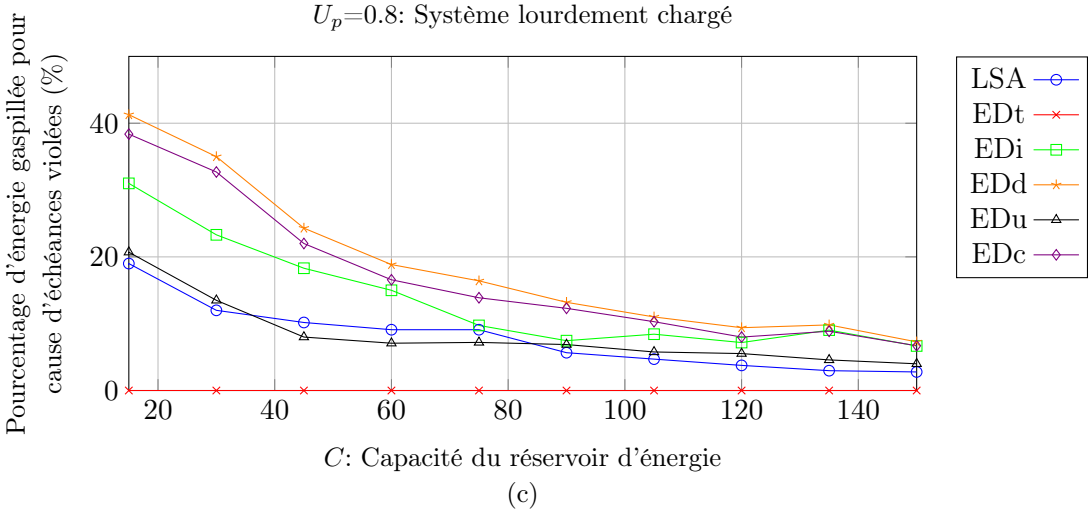
Comme l'illustre la figure 5.18, la valeur de P_r varie dans $[0, 8]$. Quand $P_r \geq 8$, le système ne possède plus de contrainte énergétique, car $P_r \geq P_{max} = 8$. Nous trouvons aussi que, les ordonnanceurs LSA et EDu possèdent le fort pouvoir à gérer l'énergie renouvelable. Le point $P_r = 4$ est un point critique. A partir de $P_r = 4$, 100% d'échéances sont respectées. Cette valeur de puissance source permet de répondre à toutes les demandes d'énergie : il y a seulement



(a)



(b)



(c)

Figure 5.16: – Evaluation du taux d'énergie gaspillée pour cause d'échéances violées

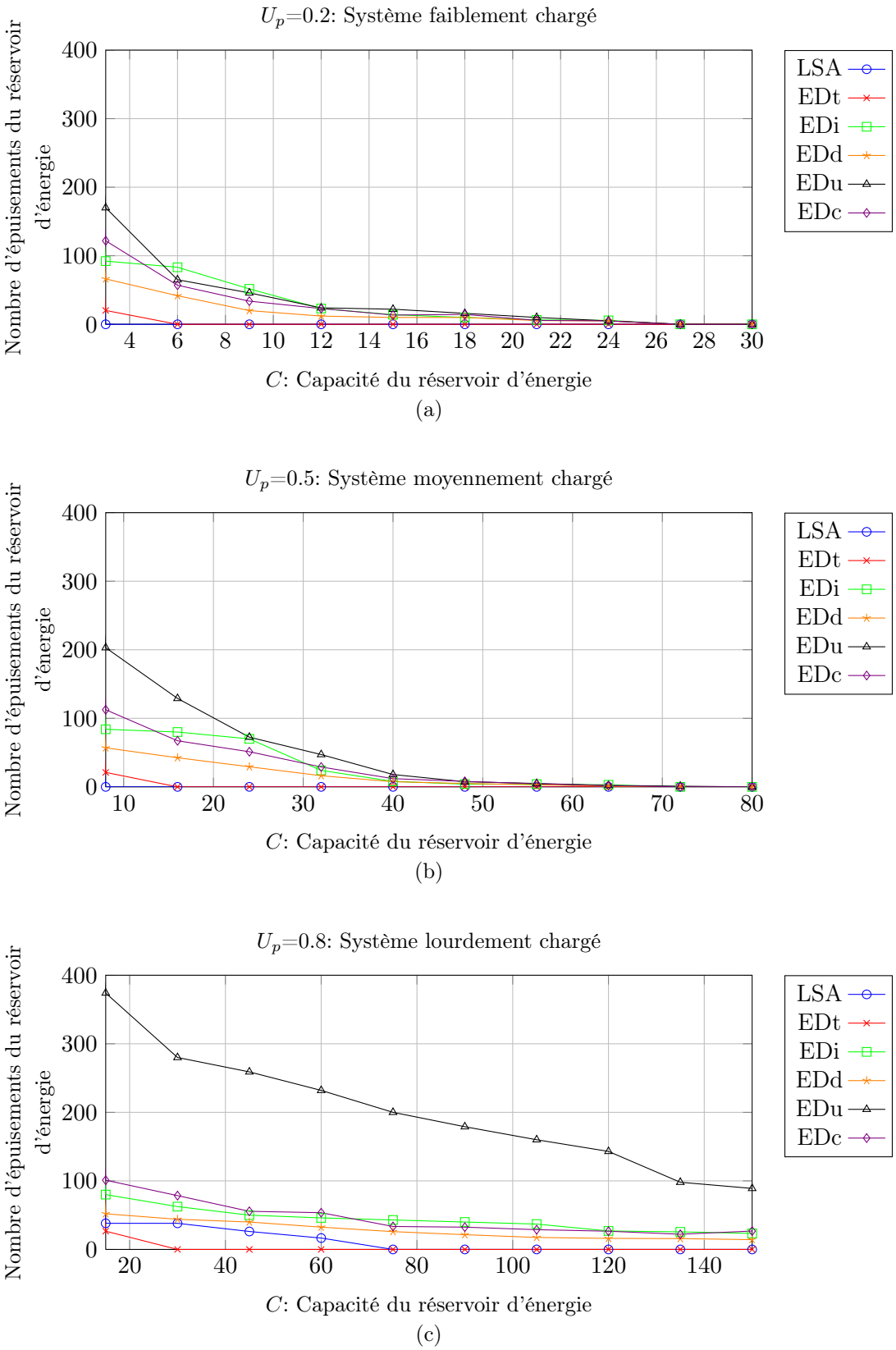


Figure 5.17: – Evaluation du nombre d'épuisements du réservoir d'énergie

besoin d'une faible capacité de stockage pour les mêmes configurations de tâches. En revanche, les ordonnanceurs autres que LSA et EDu sont moins performants.

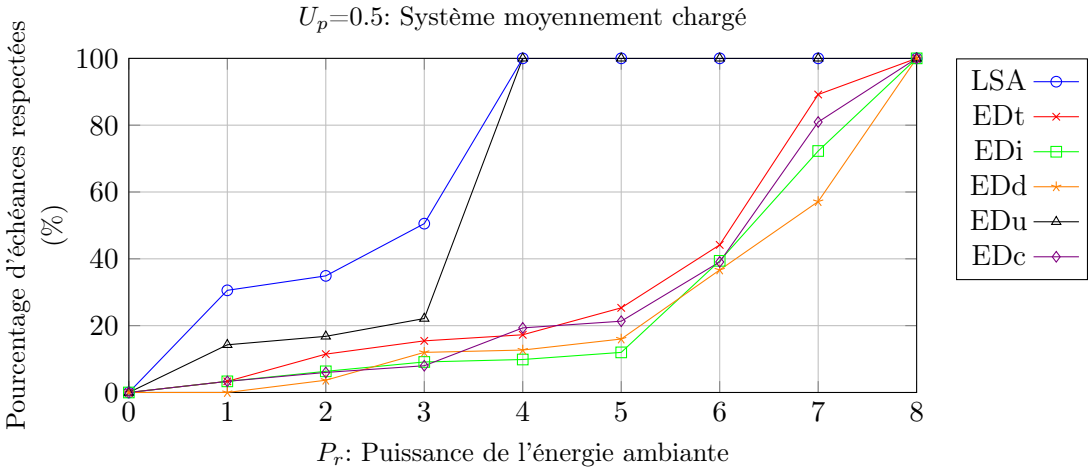


Figure 5.18: – Evaluation du taux d'échéances respectées

- Evaluation du taux d'énergie gaspillée

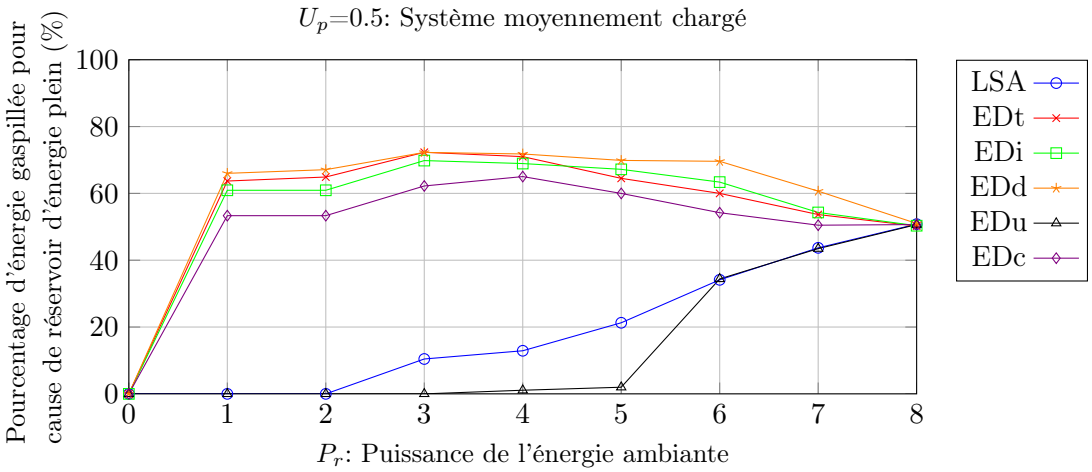


Figure 5.19: – Evaluation du taux d'énergie gaspillée pour cause de réservoir d'énergie plein

Selon l'analyse du dessus, puisque LSA et EDu ont un fort pouvoir de gestion d'énergie, alors leur taux d'énergie gaspillée pour cause de réservoir d'énergie plein sont plus petits (cf. la figure 5.19). Mais après le point critique $P_r = 4$, nous trouvons que les courbes (LSA et EDu) d'énergie gaspillée pour cause de réservoir d'énergie plein montent rapidement et se joignent aux autres au point $P_r = 8$. C'est parce que les tâches sont bien exécutées, et l'énergie est suffisante après $P_r = 4$. Donc avec l'augmentation de P_r, le problème d'énergie est résolu et l'énergie redondante fait déborder le réservoir d'énergie. A cause de la même raison, les courbes d'énergie gaspillée pour cause de tâches non terminées avant échéance deviennent nulles après le point $P_r = 4$ (cf. la figure 5.20), car il n'existe plus de contraintes énergétique au-delà.

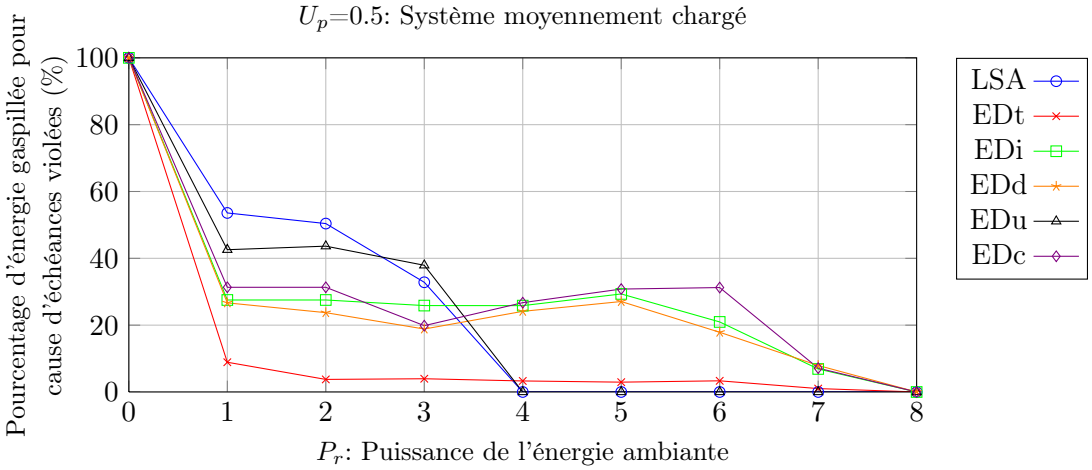


Figure 5.20: – Evaluation du taux d'énergie gaspillée pour cause d'échéances violées

- Evaluation du nombre d'épuisements du réservoir d'énergie

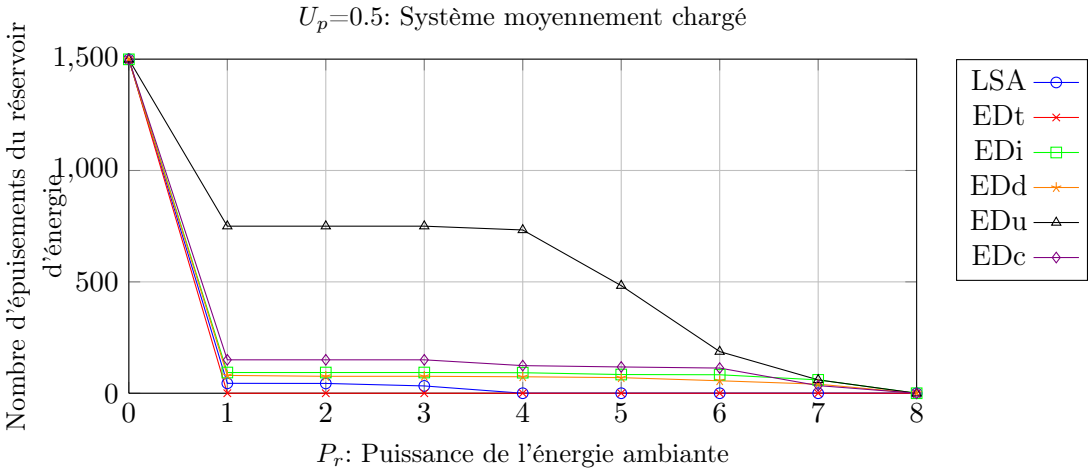


Figure 5.21: – Evaluation du nombre d'épuisements du réservoir d'énergie

Nous trouvons que tous les ordonnancements sauf EDu dont le nombre d'épuisements est maintenu à un plus bas niveau, donnent une valeur inférieure à 150. de par le fonctionnement de EDu, une fois le réservoir d'énergie devient vide, le niveau du réservoir se maintient à un plus bas niveau et il touche fréquemment le fond du réservoir, donc son nombre d'épuisements du réservoir d'énergie est plus grand.

5.6.4 Evaluation du surcoût d'exécution

Les différents algorithmes ont des surcoûts différents en terme de complexité d'exécution (nombre d'opérations effectuées par l'algorithme). Au début de ce chapitre, nous supposons que

la source d'énergie renouvelable est constante, donc la complexité de LSA est fortement diminuée.

Afin d'expliquer clairement ce problème, nous allons faire le test selon deux aspects : l'évaluation du surcoût d'exécution en fonction de la charge de système et l'évaluation du surcoût d'exécution par tâche. Le résultat du premier test est influencé non seulement par le taux d'échéances respectées, mais aussi par les tâches non terminées ; Donc nous faisons un autre test quantitatif, qui est réalisé sur l'ordonnancement d'une seule tâche.

5.6.4.1 Evaluation du surcoût d'exécution en fonction de la charge du système

D'abord, nous testons le surcoût d'exécution successivement sur un système faiblement chargé ($U_p = 0.2$), un système moyennement chargé ($U_p = 0.5$) et un système lourdement chargé ($U_p = 0.8$). Les surcoûts d'exécution sont illustrés sur la figure 5.22.

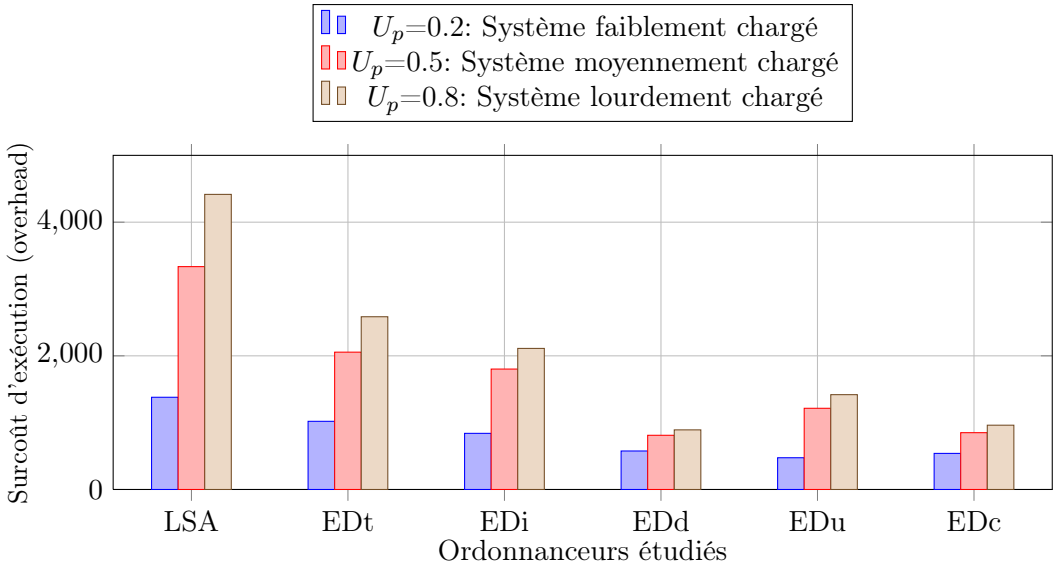


Figure 5.22: – Evaluation du surcoût d'exécution en fonction de la charge de système

Nous voyons que le surcoût de l'ordonnanceur LSA est le plus grand, car le calcul de la date de début d'exécution est une opération complexe. L'ordonnanceur EDt contient une opération plus complexe pour estimer si l'énergie contenue dans le réservoir d'énergie est suffisante pour exécuter une tâche, et pareillement l'ordonnanceur EDi possède une opération complexe pour mettre le processeur en mode veille jusqu'à la prochaine date de réveil lorsque le réservoir est vide, mais elles sont plus simples que l'opération de calcul de la date de début d'exécution. Donc EDt et EDi sont au niveau moyen. Le mécanisme des autres ordonnanceurs est plus simple : ils ne comportent pas d'opération complexe comme le calcul de la date de début d'exécution, etc. Donc les surcoûts d'EDd, EDu et EDc sont plus petits. Du point de vue des surcoûts pondérés, nous avons la hierarchie suivante : $LSA < EDt < EDi < EDu < EDc < EDd$ où "$<$" signifie "moins bon que".

5.6.4.2 Evaluation du surcoût d'exécution par tâche

En vue d'évaluer précisément le surcoût de chaque ordonnanceur, nous mettons une seule tâche à traiter pour chaque ordonnanceur. Soit une tâche indépendante $\tau_1 = (0,10,16)$, avec $a_1 = 0$, $d_1 = 10$ et $E_1 = 16$. Supposons aussi que $P_{max} = 8$, $P_r = 4$, C = 10 et Ec(0) = 0. Le résultat est présenté sur la figure 5.23. Du point de vue du surcoût, nous en déduisons la hierarchie suivante : $LSA < EDt < EDi < EDd < EDc < EDu$.

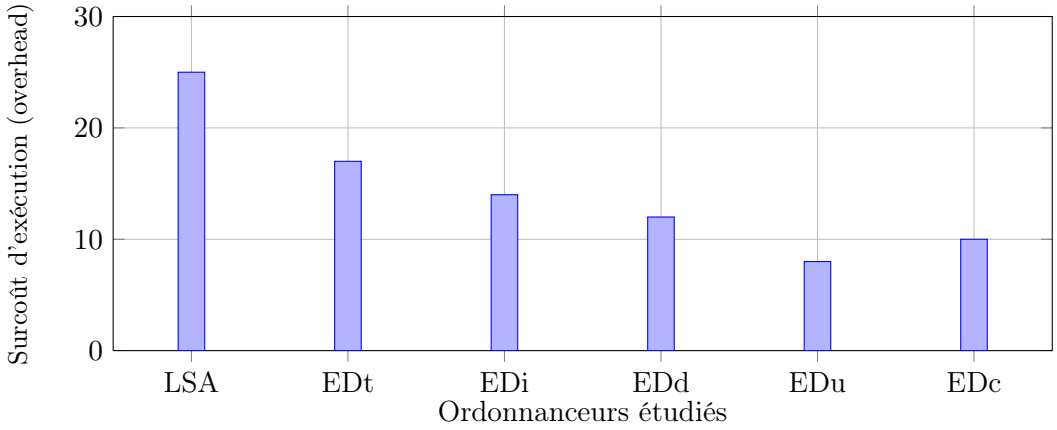


Figure 5.23: – Evaluation du surcoût d'exécution par tâche

5.7 Synthèse des résultats de simulation

	EDt	EDi	EDd	EDu	EDc	LSA
Qualité de service (taux d'échéances respectées)	☹	😐	☹	😐	☹	☺
Sourcoût d'exécution	😐	😐	☹	😐	☹	☹
Taille de réservoir d'énergie	😐	😐	☹	😐	☹	☺
Nombre de décharges complètes	😐	😐	☹	☹	😐	☺

Tableau 5.1: – Synthèse des performances des ordonnanceurs sous contrainte énergétique

Le but de ces simulations a été de nous permettre d'évaluer de façon comparative les performances de plusieurs heuristiques proposées dans ce chapitre et de l'ordonnanceur optimal LSA. Au vu d'une part des descriptions algorithmiques, et d'autre part des résultats de simulation, nous pouvons dresser le tableau récapitulatif suivant (cf. tableau 5.1) résumant les performances des différents ordonnanceurs en termes de performances et de complexités.

Dans ce tableau, le logo ☺ représente les bonnes performances de l'algorithme d'ordonnancement au niveau d'un critère donné, 😐 signifie que les performances de l'algorithme d'ordonnancement sont moyennes, et ☹ veut dire que les performances de l'algorithme d'ordonnancement sont mauvaises.

5.8 Conclusion

Dans ce chapitre, nous avons entrepris des simulations pour comparer les comportements des différents algorithmes en fonction d'une source d'énergie constante et de la charge périodique appliquée au système. Nous avons aussi proposé plusieurs heuristiques d'ordonnancement qui sont des variantes d'EDF.

Le principal critère de performance mesuré concerne le taux d'échéances respectées qui représente la qualité de service de tout système temps réel à contraintes fermes. Par cette étude, nous avons obtenu les résultats importants suivants :

- L'ordonnanceur LSA offre, conformément à l'étude théorique, la meilleur qualité de service, c'est-à-dire, nous permet d'obtenir le plus grand taux d'échéances respectées pour une même configuration de tâches utilisant le même réservoir d'énergie et la même source d'énergie. Par exemple, alors que LSA conduit à un taux de respect de 90% pour une simulation, ce taux descend à 80% pour EDi et EDu.
- L'ordonnanceur EDd offre la plus mauvaise qualité de service. Ce taux descend à 40% pour l'exemple précédent.
- Le surcoût d'exécution de LSA est le plus grand. Il est de l'ordre de 3 fois celui de EDu.
- Pour une même qualité de service obtenue, LSA et EDu demandent un réservoir d'énergie ayant la plus petite taille.
- Un système utilisant l'ordonnanceur EDu aura la vie de son réservoir d'énergie la plus courte compte tenu du nombre important de décharges complètes du réservoir d'énergie. Edu fonctionnera avec un réservoir dont le niveau est souvent proche du minimum.
- Pour une tolérance aux échéances, afin de diminuer le surcoût d'exécution, EDu représente un choix idéal comme remplaçant de LSA.
- LSA et EDt sont les seuls ordonnanceurs, qui nécessitent d'avoir accès au niveau d'énergie dans le réservoir. Ceci représente une trés forte contrainte technologique car il n'est pas simple voire impossible de mesurer un niveau d'énergie à un instant courant d'une manière exacte.

Cette étude a concerné un système temps réel utilisant une source d'énergie émettant avec une puissance toujours identique au cours du temps. En effet, cette hypothèse est parfaitement réaliste pour bon nombre d'applications. Par exemple, les applications médicales utilisant un convertisseur thermoélectrique qui génère l'énergie électrique proportionnelle à la différence de température entre l'environnement et le corps humain. C'est aussi le cas des applications utilisant l'énergie lumineuse in-door dans un bâtiment continnuellement allumé.

Les résultats reportés dans ce chapitre ont été publiés à la conférence internationale IEEE/ACM *GreenCom 2010*.

Dans le chapitre suivant, nous ferons une étude similaire lorsque la source d'énergie émet avec une puissance qui varie au cours du temps.

Chapitre 6

Systèmes temps réel alimentés en puissance variable

6.1 Introduction

Parmi les sources d'énergies renouvelables telles que l'énergie solaire et l'énergie éolienne, leur variabilité dépend considérablement de la météo ou de l'emplacement pouvant ainsi conduire à des profils énergétiques très spécifiques qu'il est nécessaire d'identifier. Les sources d'énergie solaire et éolienne s'opposent donc aux sources d'énergies constantes, objet du précédent chapitre (p. ex. l'énergie géothermique et l'énergie photovoltaïque indoor, cf. exemple 5.2).

Nous proposons dans ce chapitre, de mener une étude de simulation similaire sous l'hypothèse d'une source d'énergie à puissance variable au cours du temps. Dans ce chapitre, nous allons effectuer les mêmes tests relatifs respectivement à la qualité de service, au dimensionnement du réservoir d'énergie, au dimensionnement de la puissance de la source pour chacun des ordonnanceurs LSA, EDt, EDi, EDd, EDu et EDc. Notre objectif sera de mettre en lumière les avantages et les inconvénients de chaque ordonnanceur étudié. D'autre part, nous voulons particulièrement mesurer le surcoût d'exécution de l'ordonnanceur LSA dans ce contexte. Nous rappelons ci-dessous les critères mesurés :

- Le taux d'échéances respectées,
- L'énergie gaspillée pour cause de réservoir d'énergie plein,
- L'énergie gaspillée pour cause d'échéances violées,
- Le nombre d'épuisements du reservoir d'énergie,
- Le surcoût d'exécution.

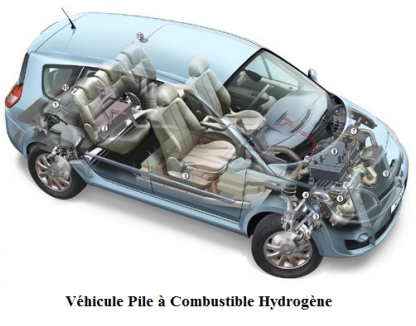
6.2 Justification du modèle

Avant de présenter cette évaluation de performance comparative, nous justifions pourquoi l'étude d'un tel modèle "source d'énergie à puissance variable" nous parait indispensable car conforme à la réalité des systèmes existants.

Exemple 6.1.

Le système SREC (Système de récupération de l'énergie cinétique) est de plus en plus connu et

appliqué dans l'industrie des voitures électriques. Il s'agit d'un système de freinage qui récupère une partie de l'énergie cinétique générée par la deccélération au lieu de la disperser sous forme de chaleur. Un des gros points forts de la motorisation électrique réside dans la possibilité qu'elle offre pour produire de l'électricité durant les phases de deccélération grâce à un frein spécifique.



a - Renault Scénic

b - Honda FCX Clarity

Figure 6.1: – Véhicules à moteur électrique avec pile à combustible hydrogène

La figure 6.1 montre les produits commerciaux du marché automobile. Le véhicule électrique fonctionne avec un moteur électrique, qui est alimenté en électricité par la pile à combustible hydrogène comme source d'électricité, et est utilisé comme générateur électrique au moment du freinage. Les moteurs électriques sont placés directement dans les roues arrières, et les moteurs électriques de l'essieu avant deviennent générateurs pendant le freinage pour recharger les batteries.

L'énergie récupérée peut alors être réutilisée, soit pour la propulsion du véhicule, soit pour toute autre fonction nécessitant une source d'énergie. Les systèmes de freinage à récupération d'énergie permettent d'augmenter l'efficacité énergétique et l'autonomie de ce type de véhicule. Clairement, l'énergie de la source alimente le réservoir d'énergie de façon totalement irrégulière dans le temps.

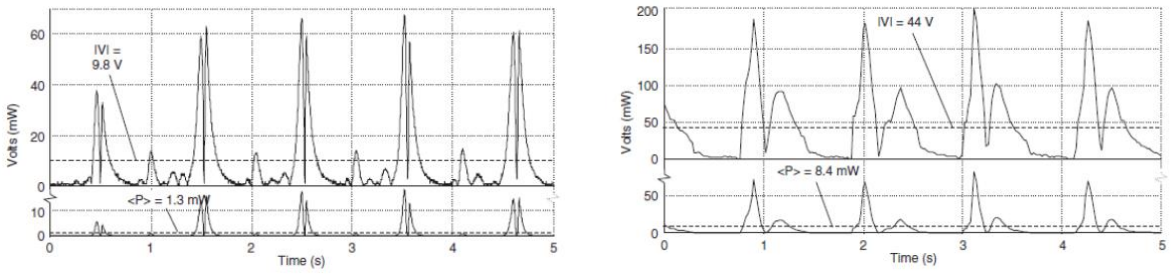
Exemple 6.2.

Les chercheurs du laboratoire Media du MIT (Massachusets Institute of Technology) ont développé un prototype de générateur piézoélectrique suffisamment petit et souple pour être inséré dans le talon et la semelle d'une chaussure. Une charge piézoélectrique est convertie en tension, soit pour charger des batteries ou utilisable directement pour alimenter des appareils électroniques, cf. la figure 6.2. Le simple fait de marcher permet alors de générer de l'énergie.

Ce type de chaussure avec système piezoélectrique intégré conduit à récupérer 17% de l'énergie utilisée lors de la marche. A chaque pas, les talons produisent $8.4mW$ et les doigts de pied produisent $1.3mW$. Ce système pourrait apporter à terme suffisament d'électricité pour recharger un smartphone en 2 heures. Les caractéristiques électriques tension/puissance sont illustrées sur la figure 6.3. L'énergie produite par ce type de système n'est clairement pas constante au cours du temps puisqu'aucune énergie n'est produite lorsque la personne se trouve immobile. Cependant,

Figure 6.2: – Nouvelle application de l'effet piézoélectrique dans les chaussures

lorsqu'une personne se met à marcher, il devient possible de prédire sur un court intervalle de temps, la quantité d'énergie récupérable de même que le profil de la fonction puissance. Nous pouvons constater que cette fonction est périodique avec une période de l'ordre de la seconde.



a - Semelle b - Talon

Figure 6.3: – Caractéristiques électriques tension/puissance

6.3 Description du simulateur

L'environnement de simulation présenté dans le chapitre 5 a été enrichi pour simuler les sources d'énergie variables dont la forme est soit impulsionnelle, soit distribuée selon une loi uniforme ou normale. La même architecture fonctionnelle étendue du simulateur est représentée sur la figure 5.9.

Le simulateur accepte en entrée plusieurs paramètres. Par exemple, si nous voulons obtenir une source d'énergie impulsionnelle, les paramètres nécessaires sont : la puissance maximale P_{rmax}, la puissance minimale P_{rmin} et la période T_r. Par exemple, si nous mettons $P_{rmax} = 6$, $P_{rmin} = 0$ et $T_r = 25$ pour établir une source d'énergie impulsionnelle, cela conduit au profil d'énergie illustré sur la figure 6.4. Et puis, afin de comparer les resultats, nous allons utiliser deux sources d'énergie impulsionnelles dans cette partie. Pour la deuxieme, nous considererons une plus grande période $T_r = 75$, mais avec une amplitude qui ne se change pas par rapport à la première. La figure 6.5 nous montre son profil énergétique. Outre les sources d'énergie impulsionnelles, nous avons aussi proposé une source d'énergie aléatoire qui obeit à une loi normale, cf. figure 6.4. Nous allons décrire en détail cette partie de l'étude dans la section 6.5.

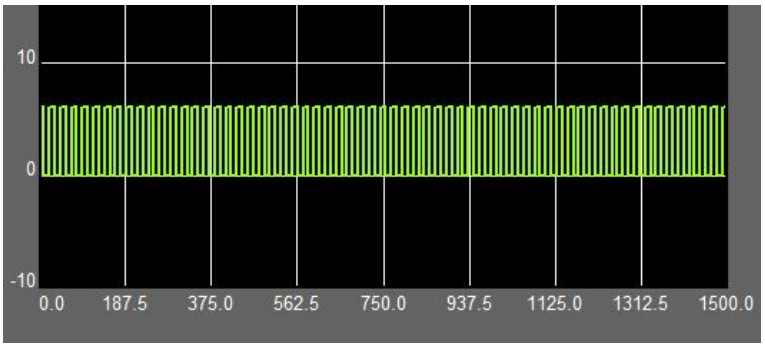


Figure 6.4: – Source d'énergie impulsionnelle de petite période

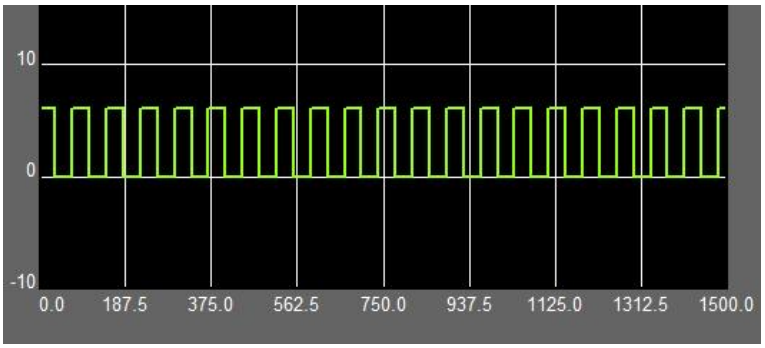


Figure 6.5: – Source d'énergie impulsionnelle de grande période

De façon similaire, après la génération de la source d'énergie et des tâches, le simulateur ordonnance la configuration de tâches périodiques en-ligne pour chaque ordonnanceur à comparer, ici LSA, EDt, EDi, EDd, EDu, EDc.

Pour chaque stratégie d'ordonnancement, 30 configurations de tâches ont été générées, chacune d'elles comprenant 6 tâches avec un PPCM de périodes égal à 300. Les simulations ont été effectuées sur 5 hyperpériodes ($T = 5 \cdot PPCM$). Les sources d'énergie simulées sont soit formées par des impulsions, dont la puissance maximale est égale à 6, cf. figure 6.4 6.5, soit formées par une loi normale, dont la puissance maximale est égale à 8, cf. figure 6.4.

6.4 Etude des sources d'énergie impulsionnelles

Les performances des algorithmes d'ordonnancement ont été évaluées en faisant varier le facteur d'utilisation du processeur U_p ainsi que la capacité du réservoir d'énergie C appliqué aux systèmes.

6.4.1 Etude avec puissance de source et capacité de réservoir fixés

Nous effectuons dans ce paragraphe une étude avec les caractéristiques :

– Les caractéristiques de chaque source d'énergie impulsionnelle sont énumérés dans le tableau 6.1.

Sources d'énergie impulsionnelles	P_{rmax}	P_{rmin}	T_r
source 1 (petite période)	6	0	25
source 2 (grande période)	6	0	75

Tableau 6.1: – Caractéristiques de sources d'énergie impulsionnelles

- La capacité du réservoir d'énergie C est fixée à 10.
- Le réservoir d'énergie est plein initialement, $E_c(0) = C = 10$.
- Nous générons des configurations de tâches ayant un taux d'utilisation du processeur U_p entre 0 et 1.

- Evaluation du taux d'échéances respectées

Les résultats de simulation sur l'évaluation du taux d'échéances respectées sont illustrés sur la figure 6.6, où la sous-figure (a) présente les résultats pour une source d'énergie impulsionnelle de petite période (T_r=25), alors que les courbes de la sous-figure (b) correspondent à une source de grande période (T_r=75).

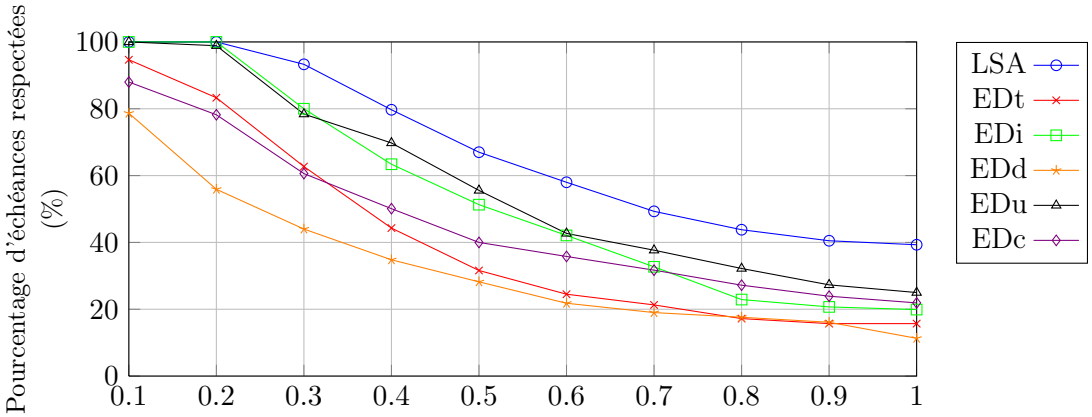
Premièrement, sur la figure 6.6, nous trouvons que les performances sont graduellement dégradées avec l'augmentation de U_p. Quand nous utilisons la source énergétique de petite période (source 1), les performances sont dégradées de façon régulière et lisse, presque linéairement. Mais il existe une présence du nodule au point U_p=0.5 en cas d'utilisation de la soure énergétique 2 (grande période). Quand U_p varie sur l'intervalle [0,0.5], les résultats de la soure énergétique 2 sont préférables, mais les circonstances deviennent pire quand $U_p \geq$0.5. Lorsque le système est lourdement chargé ($U_p \geq$0.8), les taux d'échéances respectées descendent de 5% par rapport aux résultats de la source 1. En effet, la période de la source énergétique 2 est trop longue et il n'y pas d'énergie récupérée pendant 37.5 unités de temps. Dans ce cas, le système exige une plus grande capacité de résevoir d'énergie pour obtenir des bons résultats.

La figure 6.6 (a) nous montre que le fonctionnement de LSA est le meilleur quand le système est alimenté par une source d'énergie impulsionnelle de petite période, et EDu et EDi fonctionnent sont les heuristiques qui fonctionnent le mieux parmi les heuristiques. EDd traite les tâches avec la pire performance. Quand aux performances de EDt et EDc, EDt fonctionne mieux que EDc quand $U_p \leq$0.33. A partir de ce point, la performance de EDt se dégrade de plus en plus vers celle de EDd.

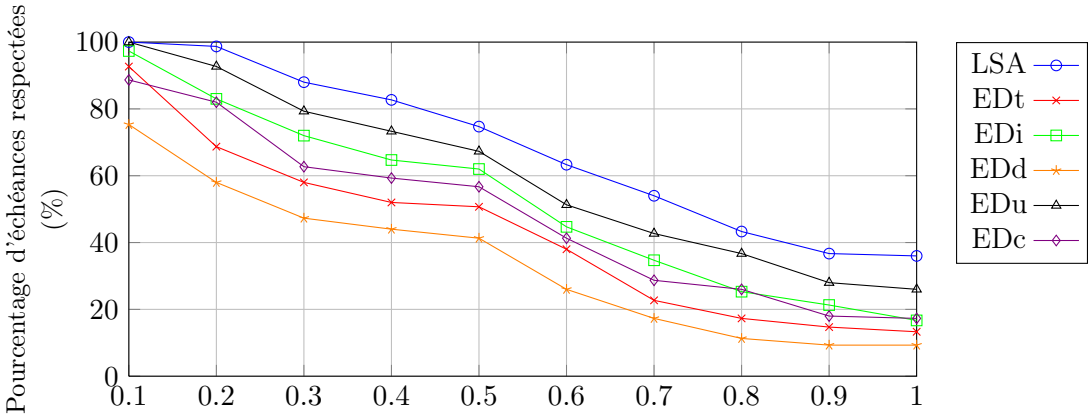
Quand le système est alimenté par la source énergétique 2, la période devient plus grande passant de 25 à 75. LSA reste toujours le meilleur ordonnanceur. EDu est la meilleure heuristique, cf. figure 6.6 (b).

Enfin, à travers la figure 6.6, nous conclurons que la période de la source d'énergie affecte beaucoup le fonctionnement des heuristiques, surtout EDi et EDc. En synthèse, nous concluons que l'ordre de supériorité des ordonnanceurs est le suivant :

Source d'énergie impulsionnelle	Ordre de supériorité des ordonnanceurs
source 1 (petite période)	$LSA > EDu \approx EDi > EDc > EDt > EDd$
source 2 (grande période)	$LSA > EDu > EDi \approx EDc > EDt > EDd$



(a): Source d'énergie impulsionnelle de petite période



(b): Source d'énergie impulsionnelle de grande période

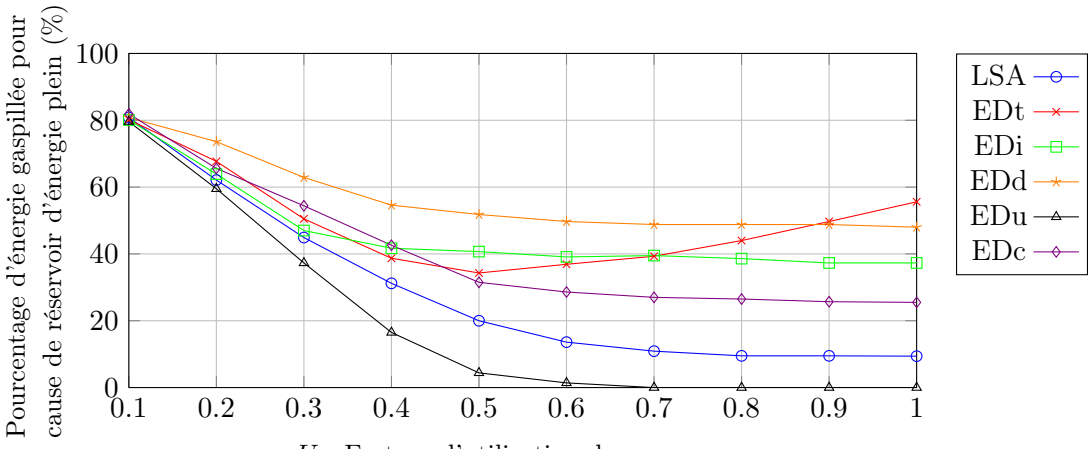
Figure 6.6: – Evaluation du taux d'échéances respectées

- Evaluation du taux d'énergie gaspillée

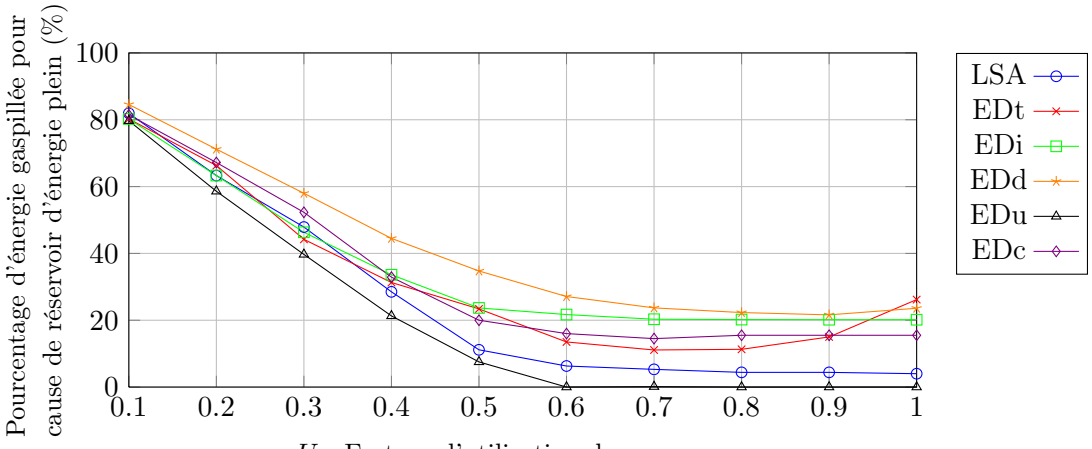
Dans cette étude, nous allons estimer le potentiel d'exploitation des sources d'énergie renouvelable à travers l'évaluation du taux d'énergie gaspillée. Nous considérons ici deux cas sur le taux d'énergie gaspillée : l'un effectué sur l'énergie gaspillée pour cause de réservoir d'énergie plein (cf. figure 6.7), l'autre effectué sur l'énergie gaspillée pour cause d'échéances violées (cf. figure 6.8).

En comparant les résultats de la figure 6.7 (a) et (b), nous observons que la différence sur l'énergie gaspillée pour cause de réservoir d'énergie plein pour les deux sources d'énergie proposées ci-dessus n'est pas très manifeste lorsque $U_p \leq 0.3$. C'est parce que les performances de chaque ordonnanceur sont approchantes avec n'importe quelle source d'énergie, cf. figure figure 6.6. Le même cas apparait sur la figure 6.8. Dès que $U_p = 0.5$, les courbes tendent à devenir stables, sauf pour l'ordonnanceur EDt. Ceci vient du fait que la capacité du réservoir d'énergie appliqué ($C = 10$) ne satisfait pas la demande énergétique du système avec l'augmentation de la charge

du système ; Le délai d'attente devient de plus en plus long, l'énergie gaspillée pour cause de réservoir d'énergie plein de EDt donc devient plus forte.



(a): Source d'énergie impulsionnelle de petite période

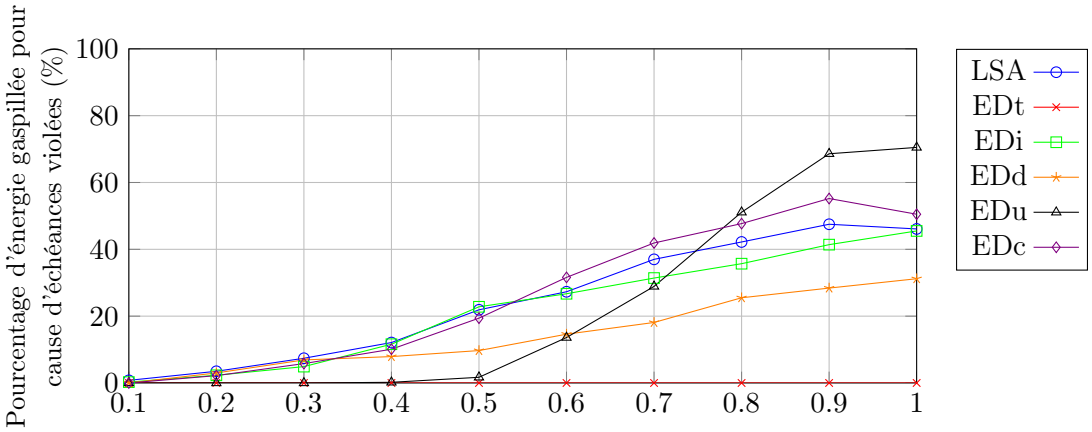


(b): Source d'énergie impulsionnelle de grande période

Figure 6.7: – Evaluation du taux d'énergie gaspillée pour cause de réservoir d'énergie plein

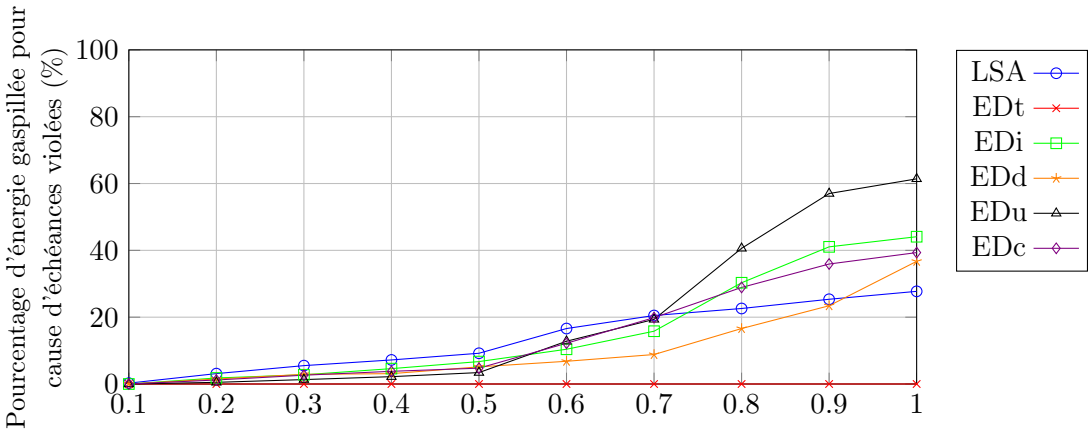
En outre, sur la figure 6.7, nous trouvons qu'avec une source d'énergie impulsionnelle de petite période, l'énergie gaspillée pour cause de réservoir plein avec les heuristiques EDd, EDi, EDt et EDc est plus grande qu'avec une source d'énergie impulsionnelle de grande période. Mais LSA et EDu sont exceptionnels, c'est-à-dire, ils s'adaptent facilement aux différentes sources d'énergie c'est-à-dire à la périodicité des impulsions.

L'évaluation du taux d'énergie gaspillée pour cause d'échéances violées se trouve illustrée sur la figure 6.8. Nous voyons que l'énergie gaspillée augment continuellement avec l'augmentation de la charge du processeur. Quand le système est alimenté par la source 1, exceptés EDt et EDd, les autres ordonnanceurs gaspillent plus d'énergie pour cause d'échéances violées qu'avec la source 2. C'est parce que beaucoup de tâches ne consomment pas du tout d'énergie pendant



U_p: Facteur d'utilisation du processeur

(a): Source d'énergie impulsionnelle de petite période



U_p: Facteur d'utilisation du processeur

(b): Source d'énergie impulsionnelle de grande période

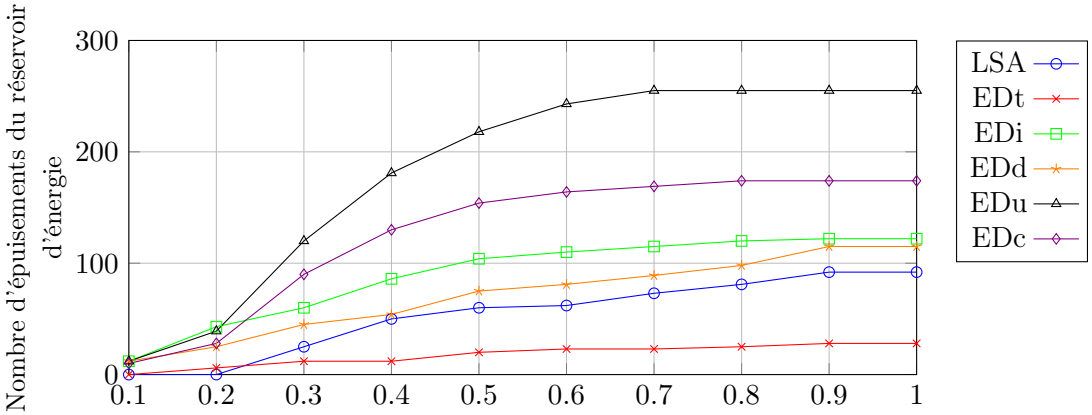
Figure 6.8: – Evaluation du taux d'énergie gaspillée pour cause d'échéances violées

le creux de la source d'énergie impulsionnelle avec grande période. Mais au contraire, beaucoup de tâches avec la source énergétique 1 ont souvent commencé leur exécution, mais ne peuvent pas être terminées avant l'échéance faute d'énergie.

- Evaluation du nombre d'épuisements du réservoir d'énergie

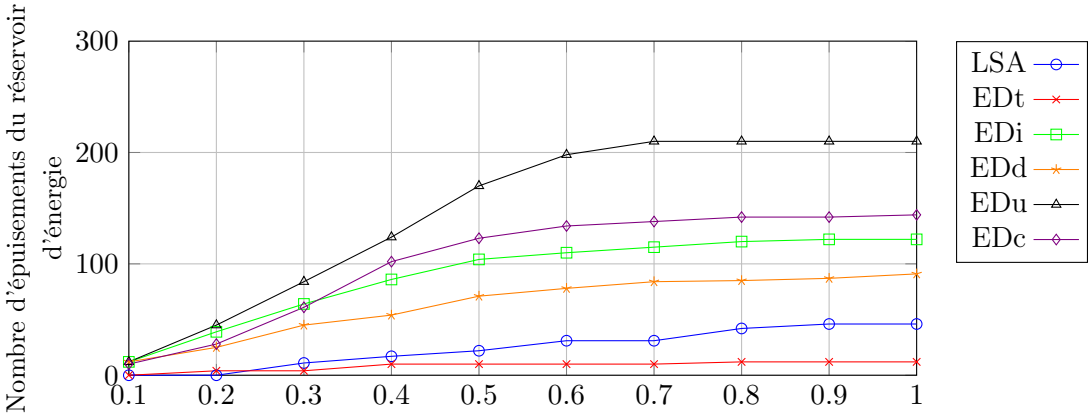
Habituellement, la durée de vie du réservoir d'énergie dépend de son cycle charge/décharge, surtout en ce qui concerne les batteries rechargeables. Le grand nombre d'épuisements conduit à ce que le réservoir d'énergie se charge fréquemment. Et sa performance va se dégrader avec la grande quantité de cycles charge/décharge, et sa vie sera raccourcie.

Les figures 6.9 (a) et (b) correspondent séparément aux sources d'énergie 1 et 2. Nous constatons que le nombre d'épuisements du réservoir varie de manière linéaire pour U_p dans [0,0.6]. Dès que $U_p = 0.7$, ce nombre atteint son maximum. Et nous trouvons ainsi qu'il existe une



U_p: Facteur d'utilisation du processeur

(a): Source d'énergie impulsionnelle de petite période



U_p: Facteur d'utilisation du processeur

(b): Source d'énergie impulsionnelle de grande période

Figure 6.9: – Evaluation du nombre d'épuisements du réservoir d'énergie

relation entre le nombre d'épuisements du réservoir et le nombre de temps creux produits par la source d'énergie. La plus petite période de source d'énergie engendre un plus grand nombre d'épuisements du réservoir, comme constaté sur la figure 6.9. Quand nous utilisons une source d'énergie de grande période, le nombre d'épuisements du réservoir augmente jusqu'à 25%.

6.4.2 Etude avec puissance de source et facteur d'utilisation fixés

Dans ce paragraphe, nous voulons faire varier la capacité du réservoir pour évaluer son impact sur la performance d'un système. Cet étude va être effectuée avec les caractéristiques suivantes :

- Les caractéristiques de sources d'énergie sont identiques à l'étude précédente, cf. tableau 6.1.
- Le facteur d'utilisation du processeur U_p est séparément fixé à 0.2, 0.5 et 0.8. $U_p = 0.2$ correspond au système dit faiblement chargé, $U_p = 0.5$ correspond au système dit moyennement chargé et $U_p = 0.8$ correspond au système dit lourdement chargé.

– La capacité du réservoir d'énergie C est fixé à 10 et nous considérons que le réservoir d'énergie est plein initialement, $E_c(0) = C = 10$.

- Evaluation du taux d'échéances respectées

Le dimensionnement, c'est-à-dire la détermination de la taille adéquate du réservoir d'énergie est une question clé lors de la conception de tout système autonome.

Les figures 6.10 et 6.11 nous présentent les résultats de simulation relatifs au taux d'échéances respectées pour un système alimenté par différentes sources d'énergie (cf. tableau 6.1). Pour chaque plan d'alimentation, nous avons distinctement évalué le taux d'échéances respectées à différents niveaux de charge du processeur : $U_p = 0.2$, 0.5 et 0.8. La même méthode est appliquée dans les études suivantes concernant l'énergie gaspillée et le nombre d'épuisements du réservoir.

Généralement, à partir de n'importe quelle source d'énergie que nous avons proposé, la performance de LSA est toujours meilleure, et l'ordonnanceur EDu s'avère la meilleure heuristique. En revanche, l'ordonnanceur EDd donne systématiquement la pire performance.

Les figures 6.10 (a) (b) et 6.11 (a) (b) illustrent le taux d'échéances respectées pour un système faiblement chargé ($U_p = 0.2$) et moyennement chargé ($U_p = 0.5$). A travers cette étude comparative, nous trouvons que les performances des ordonnanceus sur la source 1 sont meilleures que celles sur la source 2. En effet, le long temps creux de la source d'énergie provoque la dégradation de performance.

Afin d'obtenir un taux d'échéances respectées égal à 100%, le système alimenté par la source 1 demande une plus petite capacité de réservoir d'énergie. Par exemple, quand $U_p = 0.5$, si le système est alimenté par la source 1, le taux d'échéances respectées peut atteindre 100% avec un réservoir d'une capacité de 150. Mais, s'il est alimenté par la source 2, la source d'énergie impulsionnelle de grande période, la capacité de réservoir requise devient plus grande. Si nous prenons $C = 150$, aucun ordonnanceur ne peut conduire à un taux de respect de 100%, cf. figures 6.10 (b) et 6.11 (b).

Comparons les résultats obtenus pour un système faiblement chargé. Par exemple, si nous requiérons un taux d'échéances respectées de 90%, avec une source d'énergie 1, cela nécessite un réservoir de capacité $C = 30$. Mais si le système fonctionne avec la source d'énergie 2, la capacité minimale de réservoir requise est égale à 40.

Avec l'augmentation de U_p, la différence de performance entre les deux types de source d'énergie devient faible. Avec n'importe quelle source d'énergie utilisée, les résultats sont approchants : la supériorité de la source d'énergie 1 n'existe plus, cf. figures 6.10 (c) et 6.11 (c). Dès que $C = 150$, la performance de chaque ordonnanceur devient stable et l'impact de la capacité de réservoir d'énergie ne devient plus notoire.

Enfin, nous concluons que la soure d'énergie alimentée a un impact important sur le taux d'échéances respectées principalement pour les systèmes faiblement (ou moyennement) chargés.

- Evaluation du taux d'énergie gaspillée

Dans cette partie, nous analyserons l'énergie gaspillée par les ordonnanceurs. Pour un gaspillage d'énergie à cause du réservoir d'énergie plein, les résultats sont illustrés sur les figures

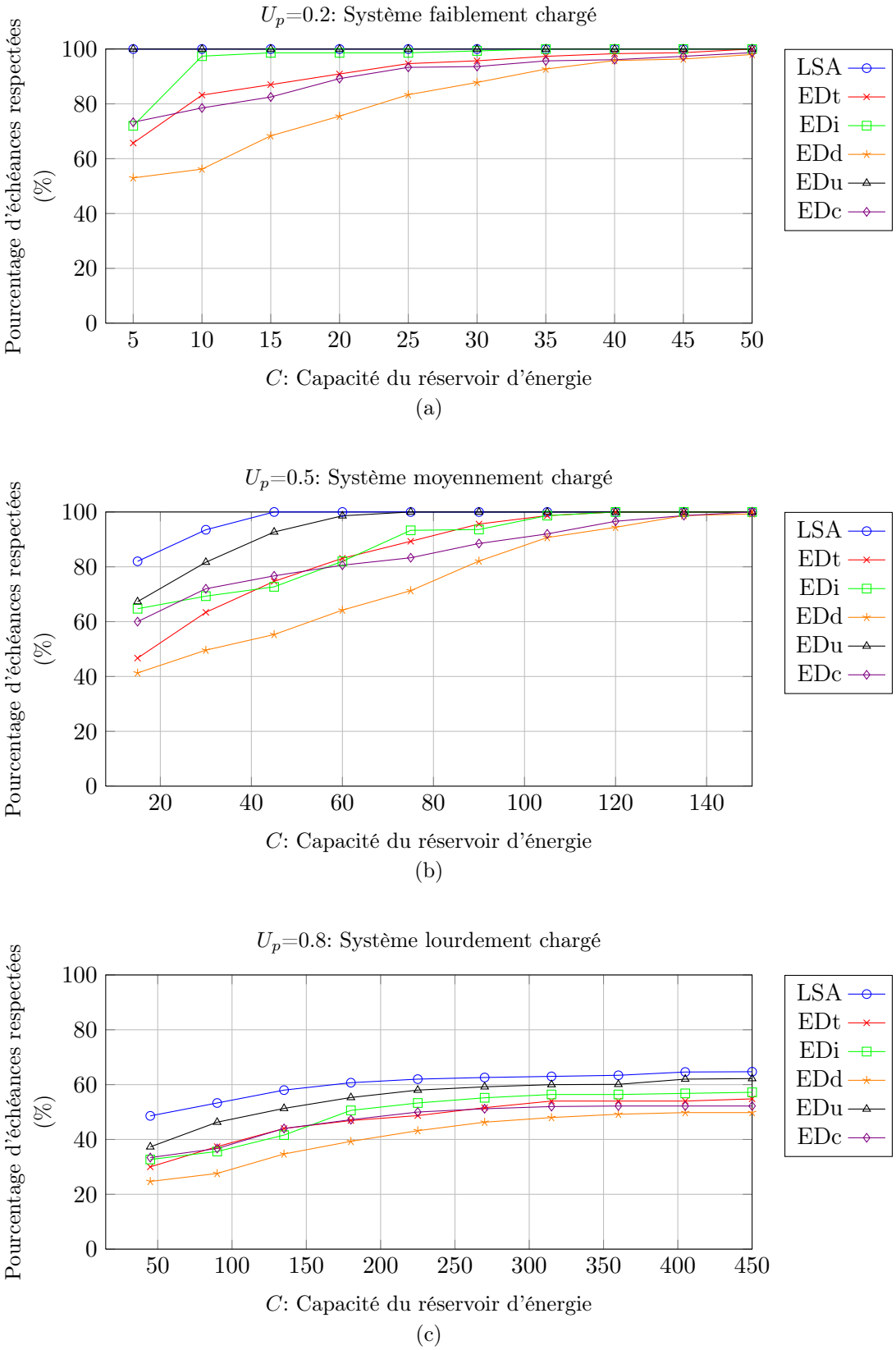


Figure 6.10: – Evaluation du taux d'échéances respectées avec une source d'énergie impulsionnelle de petite période

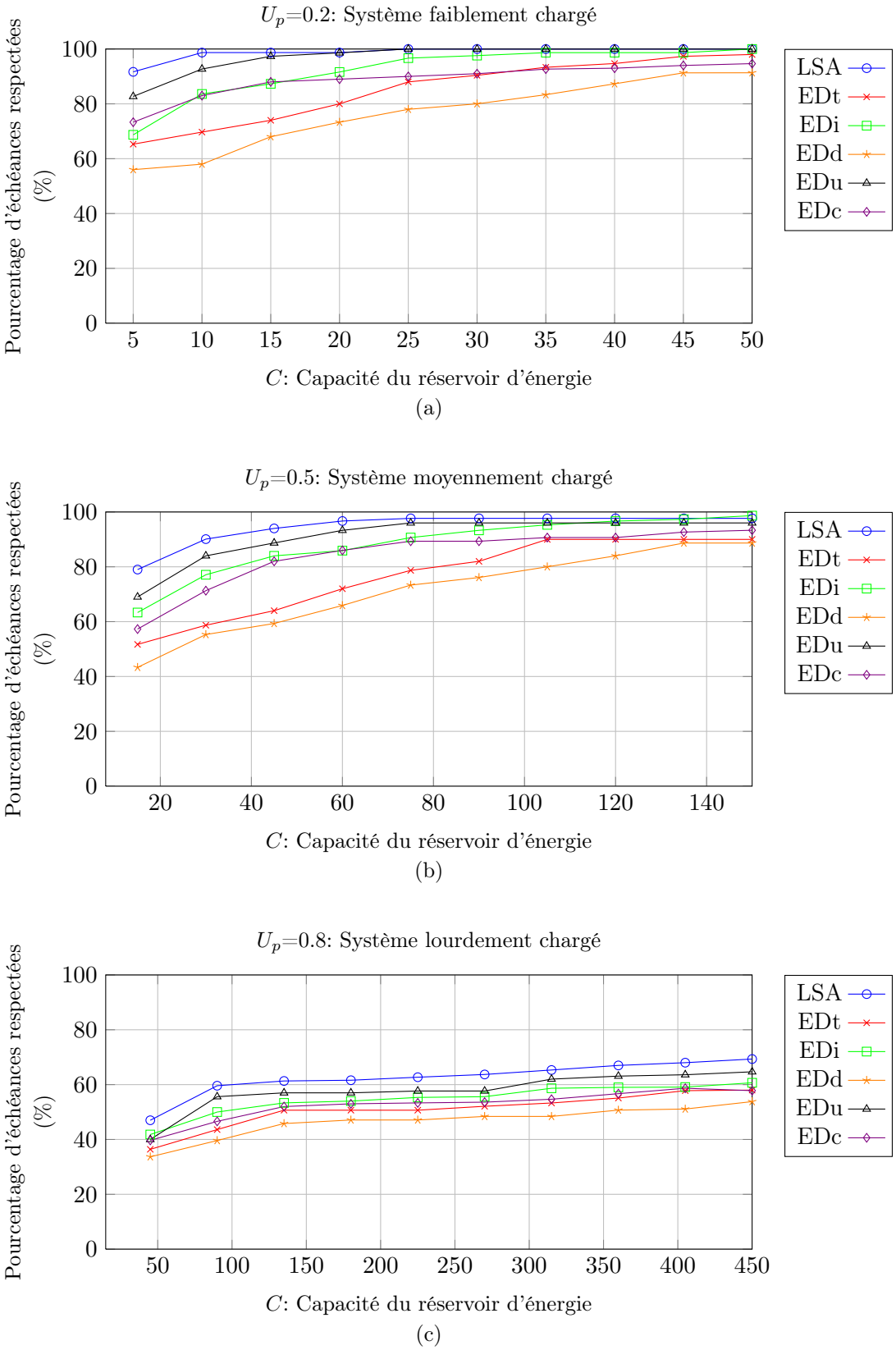
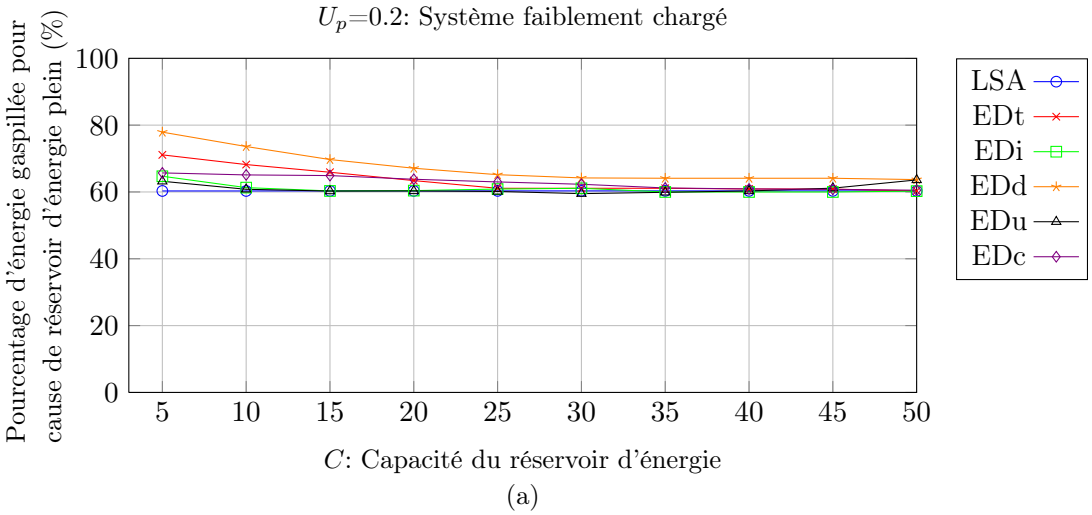
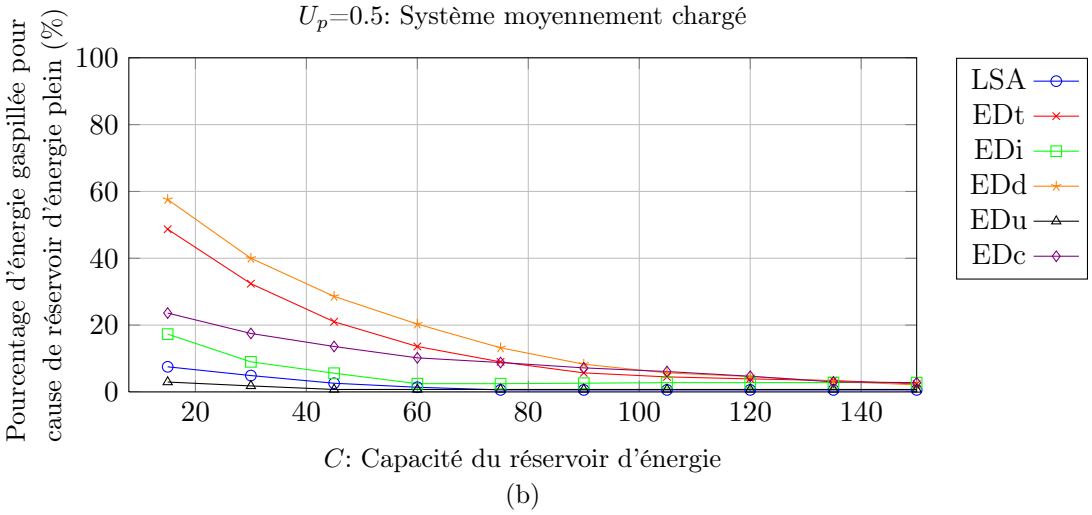


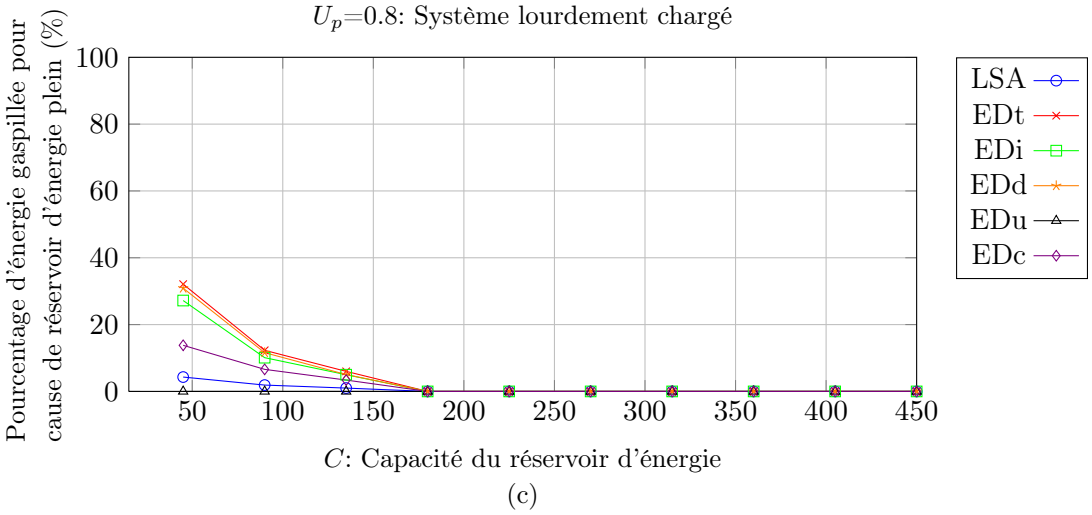
Figure 6.11: – Evaluation du taux d'échéances respectées avec une source d'énergie impulsionnelle de grande période



(a)

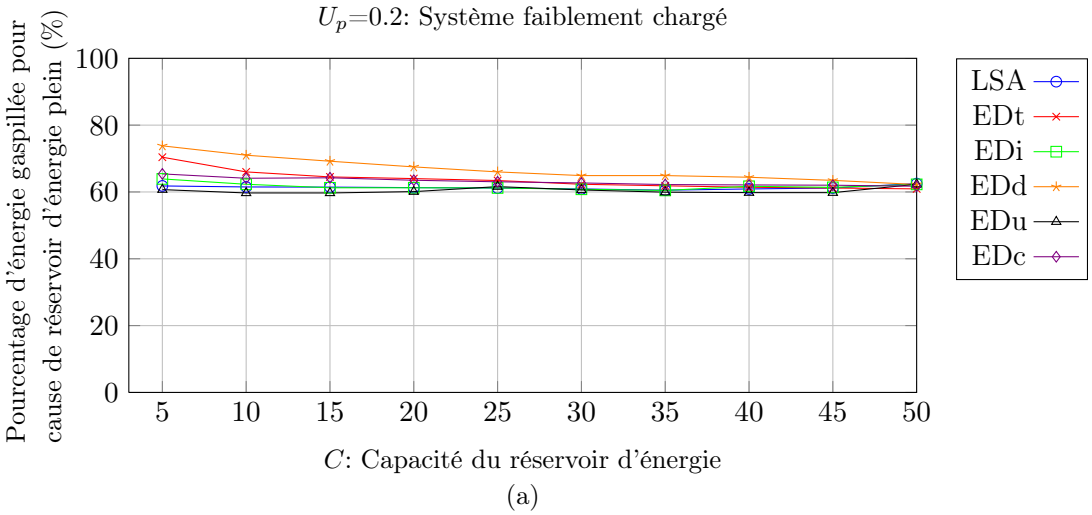


(b)

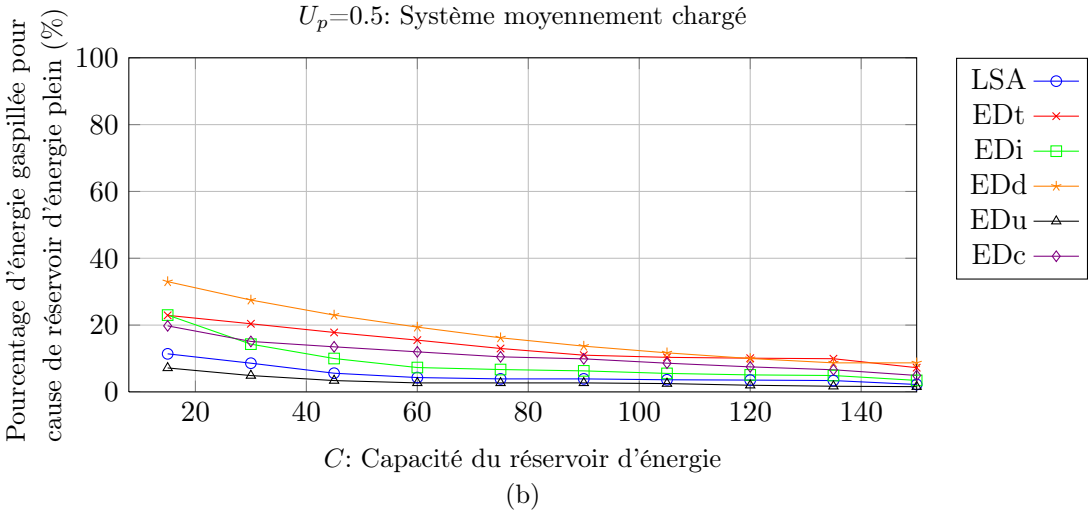


(c)

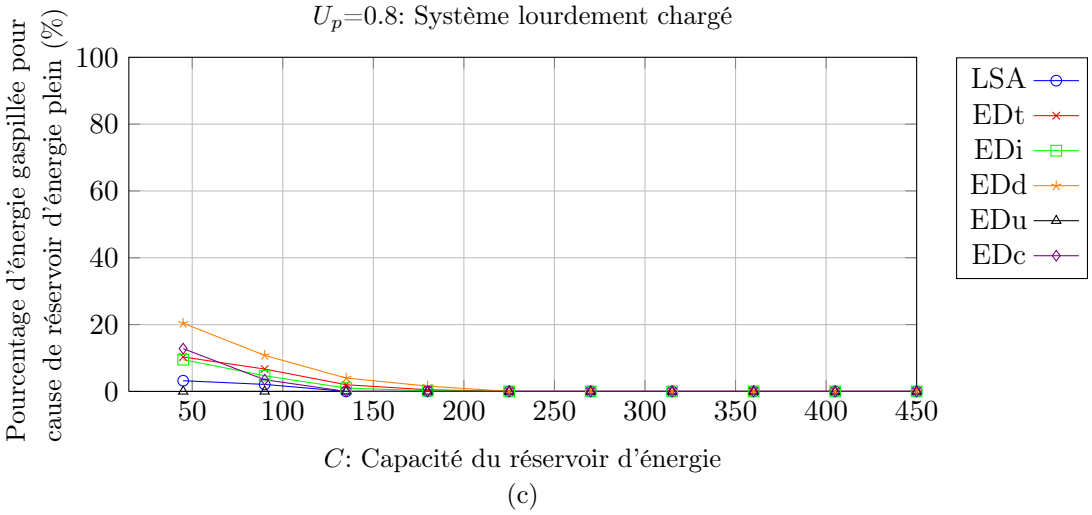
Figure 6.12: – Evaluation du taux d'énergie gaspillée pour cause de réservoir plein avec une source d'énergie impulsionnelle de petite période



(a)



(b)



(c)

Figure 6.13: – Evaluation du taux d'énergie gaspillée pour cause de réservoir plein avec une source d'énergie impulsionnelle de grande période

6.12 et 6.13. En outre, si une tâche n'est pas terminée avant son échéance, la part de l'énergie consommée déjà par une tâche non-terminée, est considérée aussi comme l'énergie gaspillée. Les résultats sont illustrés par les figures 6.14 et 6.15.

Parmi les résultats obtenus sur l'évaluation du taux d'énergie gaspillée pour cause de réservoir d'énergie plein, nous trouvons que l'ordonnanceur EDd est le plus mauvais, avec n'importe quelle source d'énergie et pour n'importe quel niveau de charge du processeur.

En général, les quantités d'énergie gaspillée pour cause de réservoir plein convergent uniformément vers une valeur définie avec l'augementation de C, par exemple, 60% pour $U_p = 0.2$ et 0% pour $U_p = 0.5$ et $U_p = 0.8$.

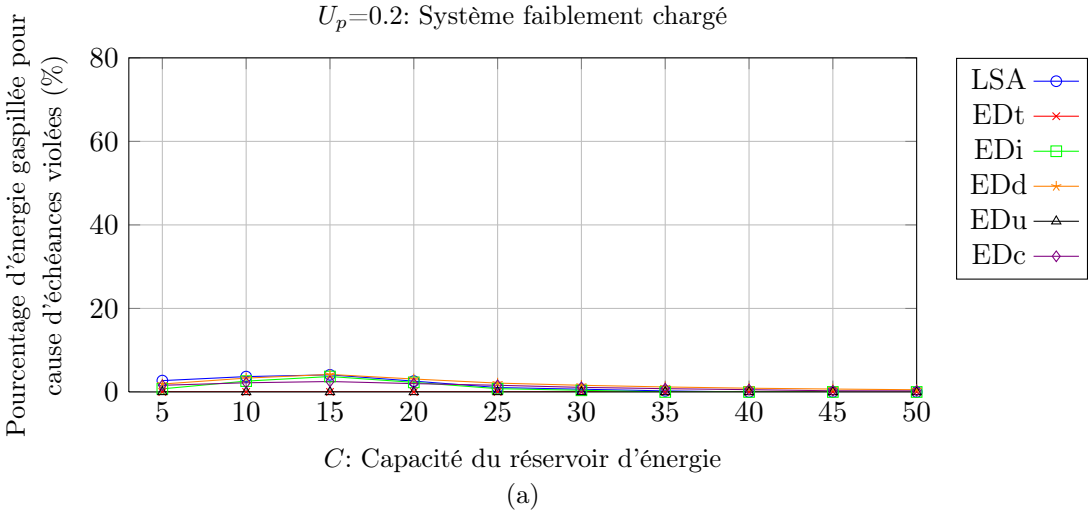
Lorsque le système est faiblement chargé ($U_p = 0.2$), les performances de chaque ordonnanceur sont approximativement identiques avec n'importe quelle source d'énergie, cf. figures 6.12 (a) et 6.13 (a).

Lorsque le système est moyennement ou fortement chargé ($U_p = 0.5$ et $U_p = 0.8$), nous trouvons que l'énergie gaspillée pour cause de réservoir d'énergie plein avec la source 1 est un peu plus élevée que celle avec la source 2. Mais ils convergent uniformément vers 0 avec l'augementation de C, cf. figures 6.12 (b) (c) et 6.13 (b) (c).

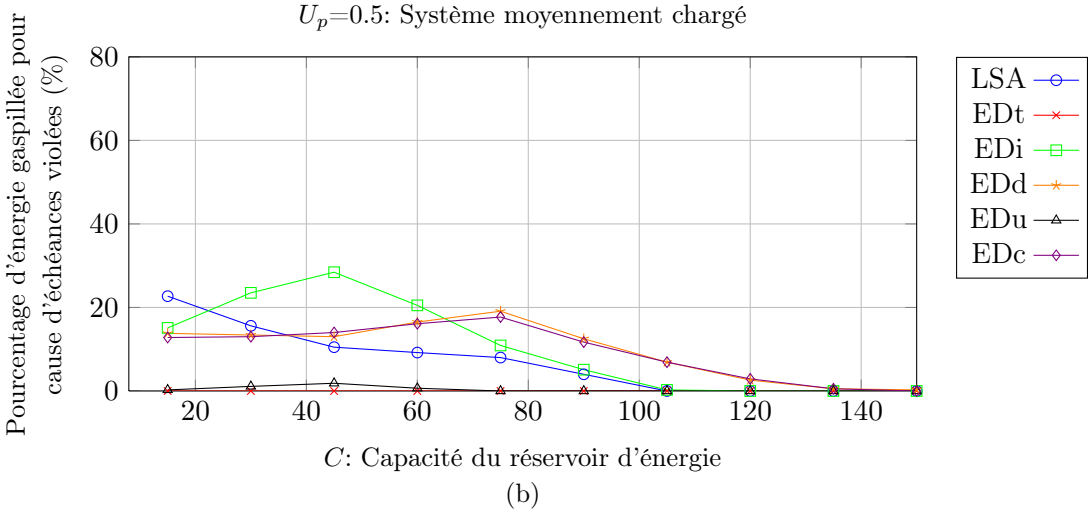
Concernant le gaspillage d'énergie pour cause d'échéances violées, les résultats de simulation sont présentés sur les figures 6.14 et 6.15.

Lorsque le système est faiblement chargé ($U_p = 0.2$), les quantités d'énergie gaspillée par les ordonnanceurs sont très bas ($\leq 5\%$) et aussi quasi-identiques. Les comportements des ordonnanceurs sont similaires avec n'importe quelle source d'énergie proposées, cf. figures 6.14 (a) et 6.15 (a).

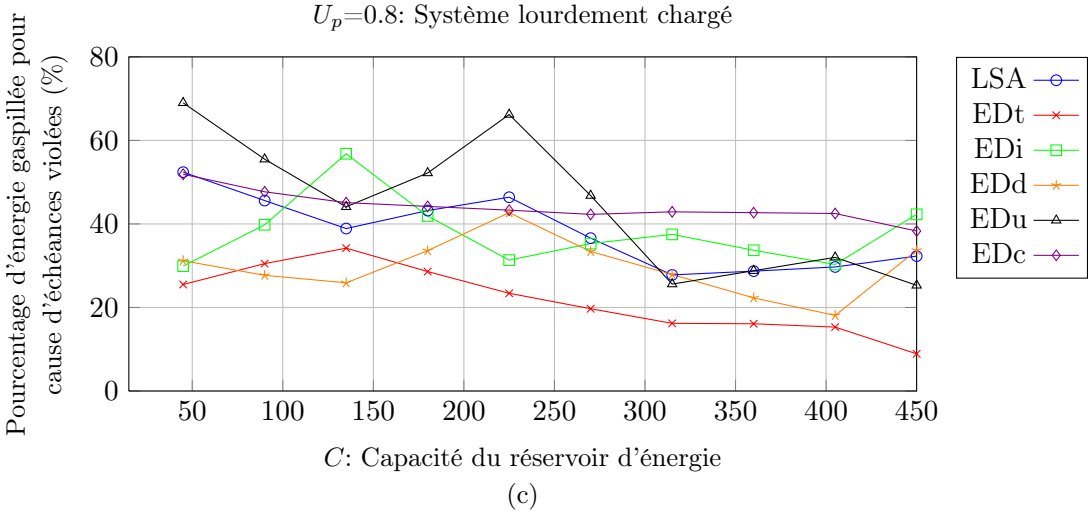
En augmentant la capacité du réservoir d'énergie C, le niveau de gaspillage d'énergie pour cause d'échéances violées augnmente, et de plus en plus, la variation de ce paramètre devient irrégulière, cf. figures 6.14 (b) (c) et 6.15 (b) (c).



(a)

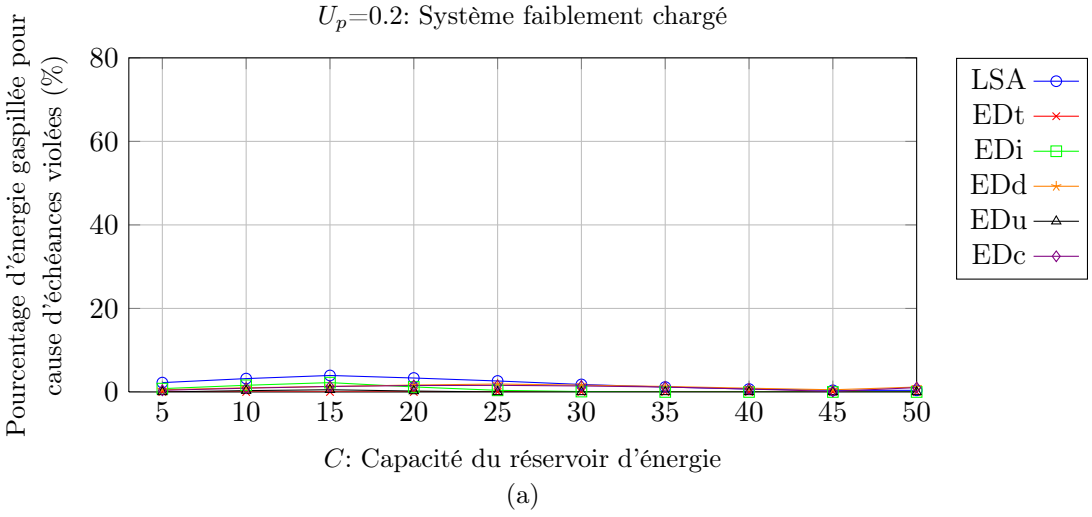


(b)

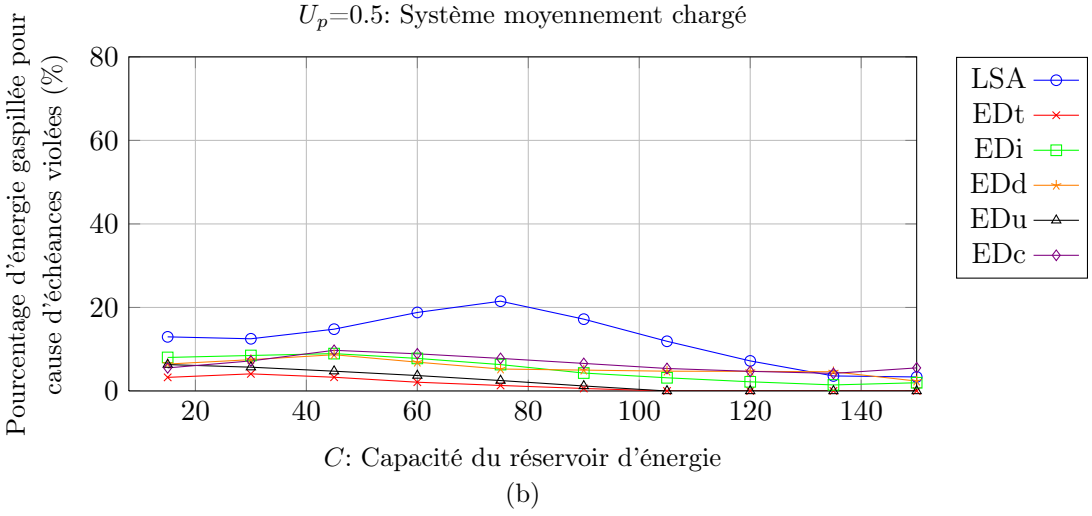


(c)

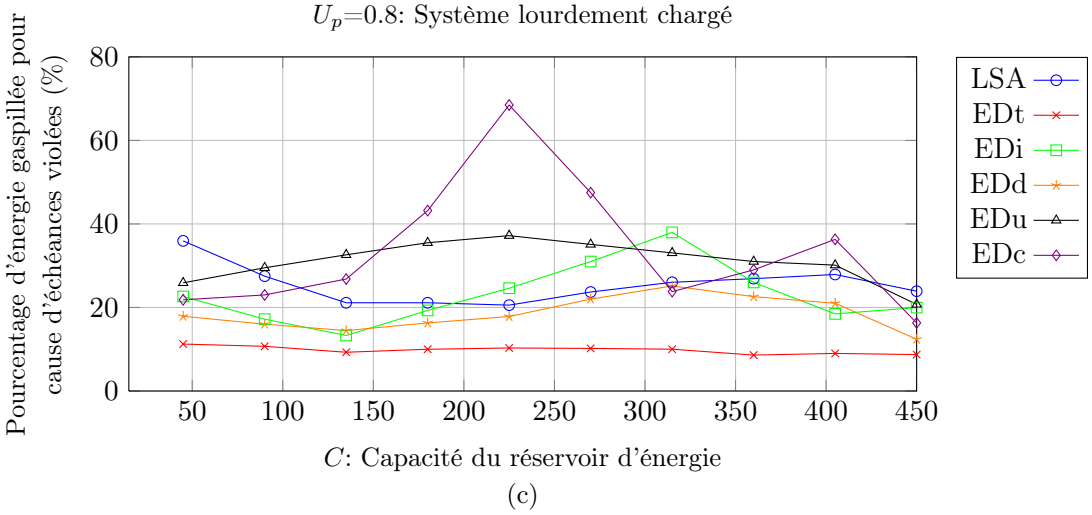
Figure 6.14: – Evaluation du taux d'énergie gaspillée pour cause d'échéances violées avec une source d'énergie impulsionnelle de petite période



(a)



(b)



(c)

Figure 6.15: – Evaluation du taux d'énergie gaspillée pour cause d'échéances violées avec une source d'énergie impulsionnelle de grande période

- Evaluation du nombre d'épuisements du réservoir d'énergie

Les figures 6.16 et 6.17 présentent les résultats sur l'évaluation du nombre d'épuisements du réservoir d'énergie.

Globalement, le nombre d'épuisements du réservoir se révèle proportionnel à la charge du processeur U_p, et est inversement proportionnel à la capacité du réservoir d'énergie C. Mais avec l'augmentation de C, son impact sur le nombre d'épuisements du réservoir devient de plus en plus faible. D'ailleurs, une source d'énergie de plus grande période va causer un plus grand nombre d'épuisements du réservoir.

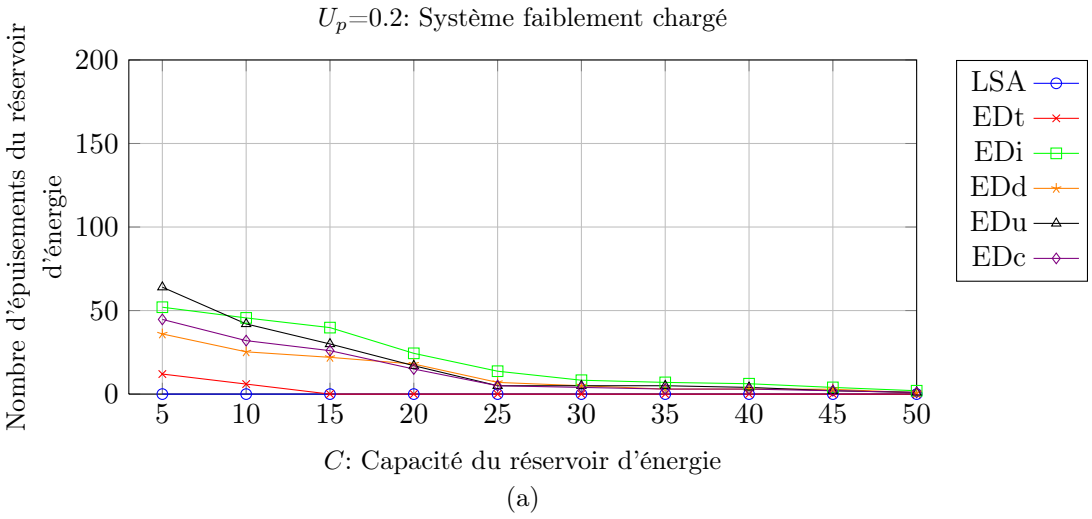
A cause de la spécificité de l'ordonnanceur EDu, le nombre d'épuisements du réservoir est toujours le plus grand. Au contraire, celui de l'ordonnanceur EDt s'avère le plus petit.

En comparant les figures obtenues, nous constatons que l'impact de la capacité du réservoir est évident pour les systèmes faiblement ou moyennement chargés. Mais pour les système lourdement chargés, cet impact de la capacité du réservoir devient très faible.

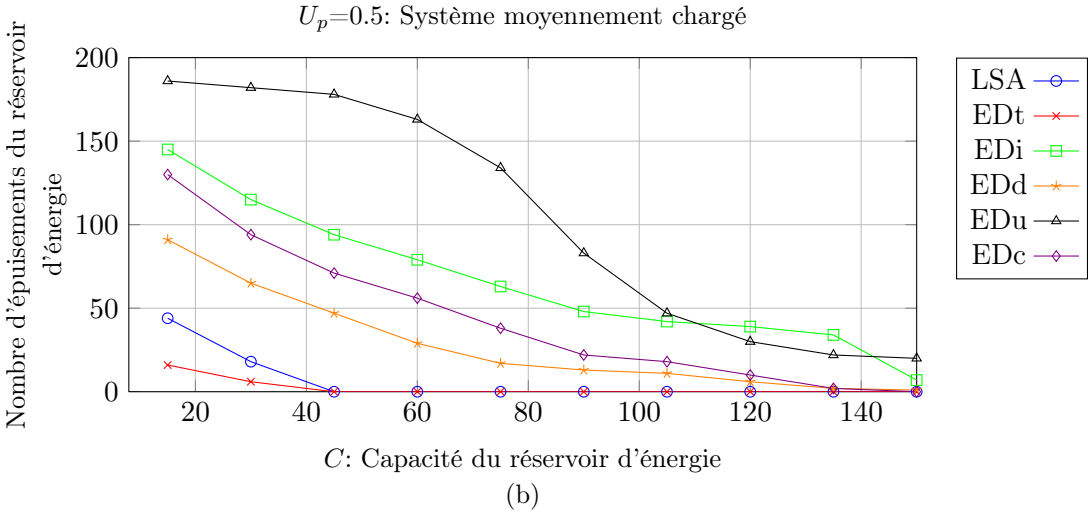
6.4.3 Evaluation du surcoût d'exécution (overhead)

Dans ce paragraphe, nous allons analyser les surcoûts d'exécution des ordonnanceurs pour les systèmes alimentés par une sourece d'énergie variable. Parce que les heuristiques sont des ordonnanceurs non clairvoyants, ceux-ci ne sont pas affectés par les différents types de source d'énergie. Ils n'ont pas besoin d'un grand nombre de calculs en-ligne. Cependant, en tant qu'ordonnanceur clairvoyant du point de vue énergétique, LSA va effectuer des calculs pour déterminer la date optimale d'exécution de toute tâche arrivante en fonction du profil énergétique à venir, complexifiant ainsi l'ordonnanceur.

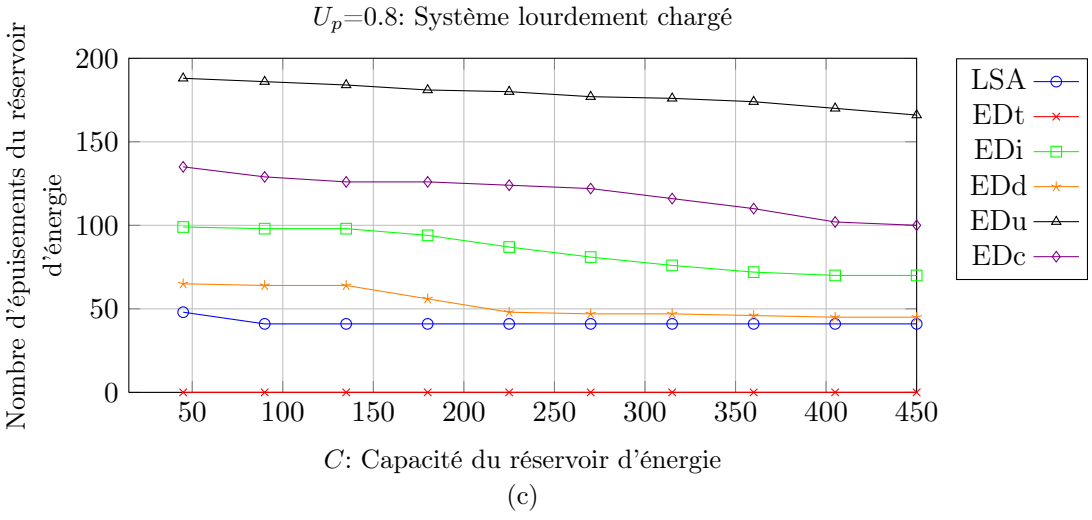
Ici, nous testons un système moyennement chargé ($U_p = 0.5$) avec un réservoir d'énergie donné ($C = 10$) et les sources d'énergie impulsionnelles proposées (cf. tableau 6.1). Les résultats sur le surcoûts d'exécution sont illustrés sur les figures 6.18 et 6.19. Nous constatons que le surcoût d'exécution dépend principalement de la quantité de tâches.



(a)



(b)



(c)

Figure 6.16: – Evaluation du nombre d'épuisements du réservoir d'énergie avec une source d'énergie impulsionnelle de petite période

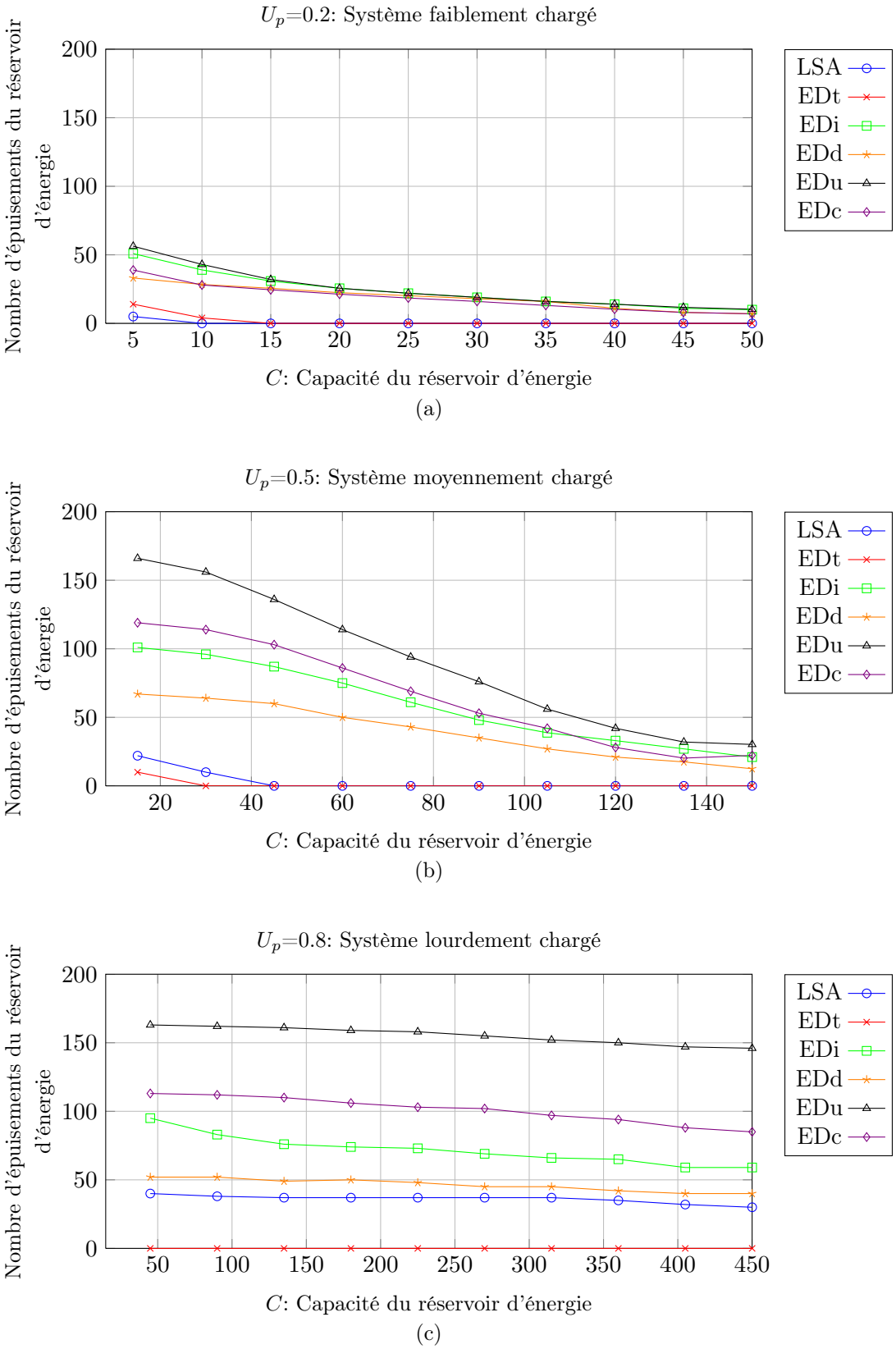


Figure 6.17: – Evaluation du nombre d'épuisements du réservoir d'énergie avec une source d'énergie impulsionnelle de grande période

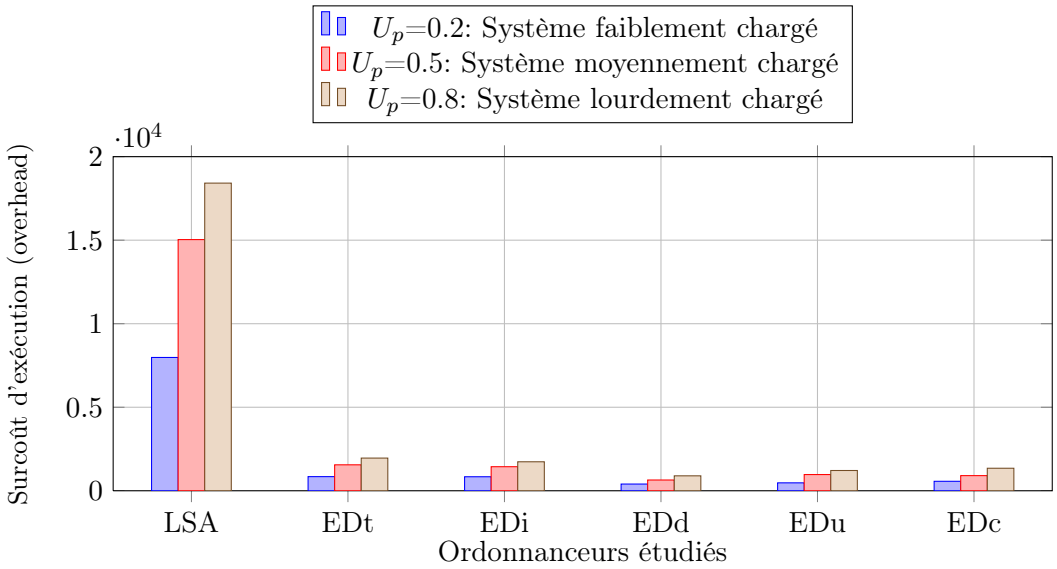


Figure 6.18: – Evaluation du surcoût d'exécution avec une source d'énergie impulsionnelle de petite période

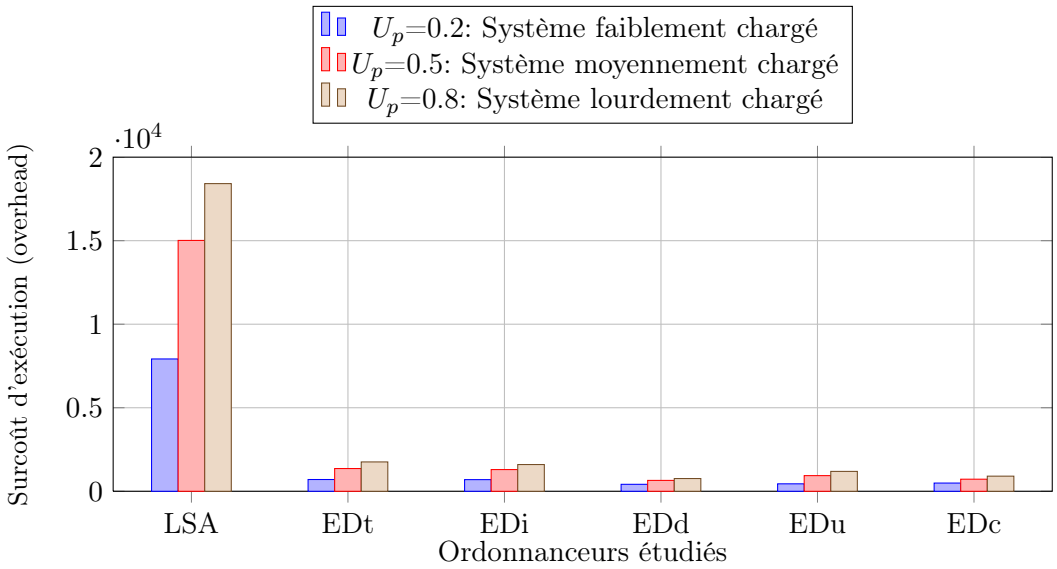


Figure 6.19: – Evaluation du surcoût d'exécution avec une source d'énergie impulsionnelle de grande période

6.5 Etude des sources d'énergie aléatoire en loi normale

Dans ce paragraphe, nous allons décrire les résultats de simulations effectuées sur une source d'énergie aléatoire obéit à une loi normale, les paramètres étant : l'espérance μ =0 et l'écart-type σ =1, c'est-à-dire $\mathscr{X} \sim \mathscr{N}$ (0,1). Alors une source d'énergie aléatoire virtuelle est générée selon :

$$P_r(t) = \left| P_{rmax} \cdot N(t) \cdot \cos(\frac{t}{70 \cdot \pi}) \cdot \cos(\frac{t}{100 \cdot \pi}) \right|,$$

cf. figure 6.20, où

- P_{rmax} est la puissance maximale de la source d'énergie ambiante ; Nous mettons P_{rmax}=8 dans cette étude ;
- $N(t)$ est une valeur aléatoire qui suit une loi normale $\mathscr{X} \sim \mathscr{N}$ (0,1).

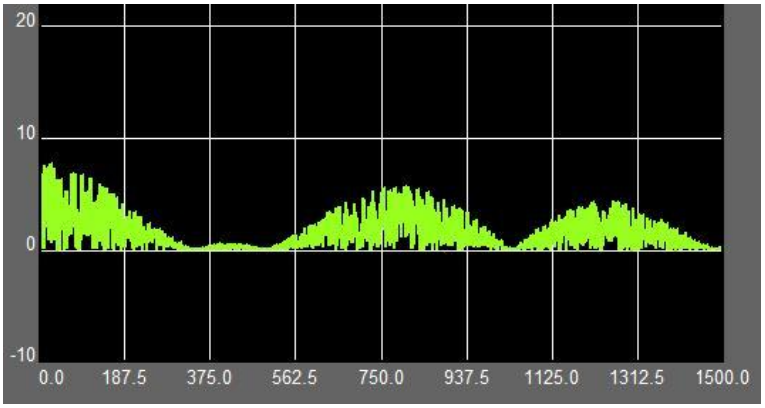


Figure 6.20: – Source d'énergie aléatoire suit à une loi normale

6.5.1 Etude avec puissance de source et capacité du réservoir d'énergie fixés

Nous effectuons dans ce paragraphe une étude avec les caractéristiques suivantes :

- La puissance maximale de source d'énergie P_{rmax} est fixée à 8.
- La capacité du réservoir d'énergie C est fixée à 10.
- Le réservoir d'énergie est plein initialement, $E_c(0) = C = 10$.
- Nous générons des configurations de tâches ayant un taux d'utilisation du processeur U_p compris entre 0 et 1.

- Evaluation du taux d'échéances respectées

Sur la figure 6.21, nous trouvons que tous les ordonnanceurs ont une performance dégradée à cause de l'augmentation du facteur d'utilisation du processeur. L'ordonnanceur est obligé de récolter de l'énergie de l'environnement en sacrifiant une partie de la ressource du processeur.

Commes dans les simulations précédentes, la performance de LSA se trouve toujours au plus haut niveau, ce qui confirme son optimalité. Par contre, la performance d'EDd est toujours au plus bas niveau. Cependant, à partir de U_p=0.7, ce qui correspond à un processeur lourdement chargé, la différence de performance entre les ordonnanceurs heuristiques, n'est plus évidente.

Quand le système est de légèrement à moyennement chargé, nous pouvons distinguer en général trois degrés différents : LSA est le meilleur, EDu et EDt se trouvent au niveau intermédiaire, EDi, EDc et EDd fonctionnent avec une performance excécrable. A U_p=0.3, les taux d'échéances respectées atteignent seulement 10%.

Pour $U_p = 1$, le taux d'échéances respectées de LSA atteint 14% alors que la performance d'EDu se dégrade à 9%.

Globalement, plus le système est légèrement chargé, plus les gains de performance de LSA par rapport aux heuristiques sont significatifs. Nous pouvons conclure ici, à la hierarchisation suivante des stratégies d'ordonnancement : $LSA > EDu \approx EDt > EDi > EDc > EDd$ en termes de taux d'échéances satisfaites.

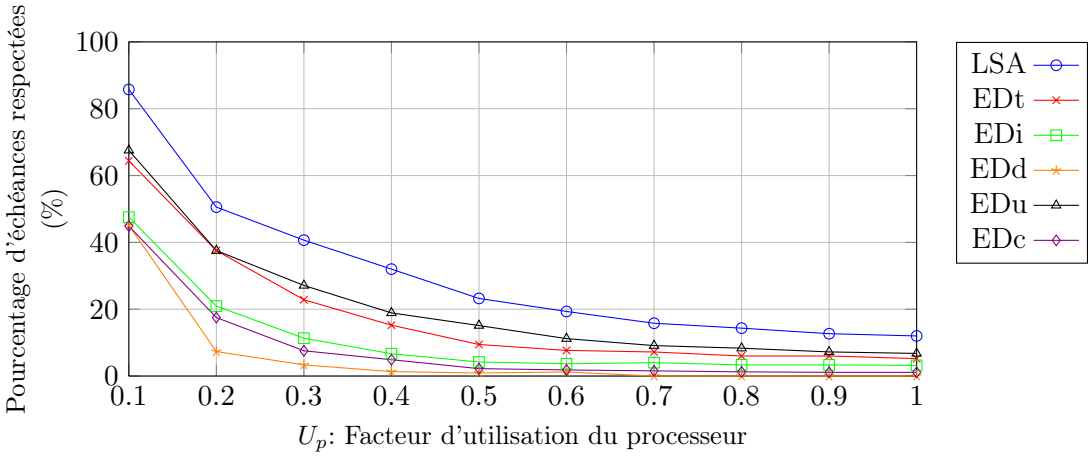


Figure 6.21: – Evaluation du taux d'échéances respectées

- Evaluation du taux d'énergie gaspillée

Comme nous l'avons expliqué dans les paragraphes du dessus, il y a deux possibilités envisageables pour analyser les circonstances de la dissipation d'énergie : l'énergie gaspillée pour cause de réservoir d'énergie plein (cf. figure 6.22) et l'énergie gaspillée pour cause d'échéances violées (cf. figure 6.23).

La figure 6.22 nous illustre l'évolution de l'énergie gaspillée pour cause de réservoir d'énergie plein. Nous voyons que les ordonnanceurs EDd, EDi, EDc et LSA ont une caractéristique commune : les niveaux d'énergie gaspillée pour cause de réservoir plein ne varient pas beaucoup en suivant U_p, sauf celui d'EDt.

La courbe la plus haute correspond à EDt : elle est différente des autres. Nous voyons qu'elle croit toujours jusqu'à U_p=0.5 et devient stable à 95% lorsque le système est très lourdement chargé. C'est parce qu'il vérifie toujours l'énergie requise avant de pouvoir exécuter une tâche, le temps d'attente devient donc de plus en plus long avec l'augmention de U_p. Et le réservoir d'énergie plus facile à déborder : il a donc besoin d'un réservoir d'une plus grande capacité pour diminuer le gaspillage d'énergie.

L'ordonnanceur EDu est celui qui perd de moins en moins d'énergie pour cause de réservoir plein avec l'accroissement de U_p. A partir de $U_p = 0.3$, le système est quand même faiblement chargé, le réservoir d'énergie ne déborde plus. C'est-à-dire, l'énergie produite par la source d'énergie ambiante n'est vraiment pas suffisante.

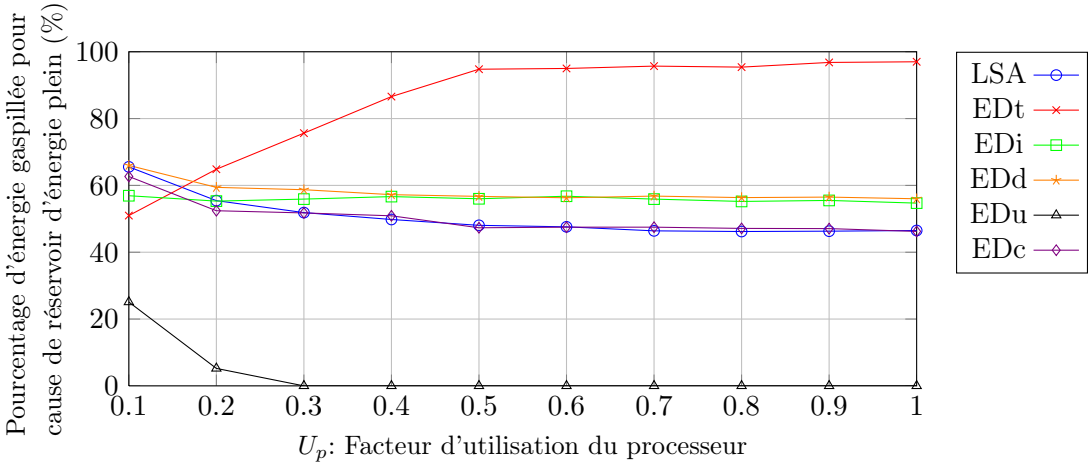


Figure 6.22: – Evaluation du taux d'énergie gaspillée pour cause de réservoir d'énergie plein

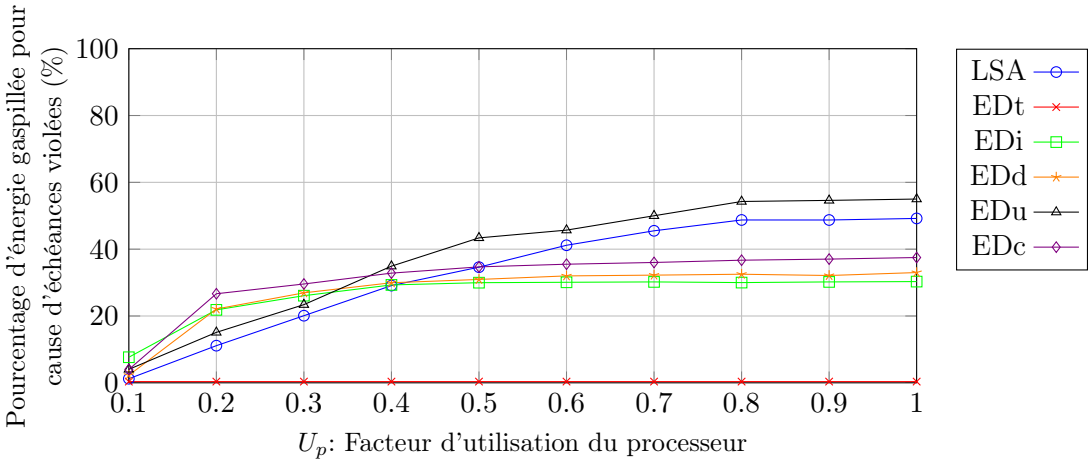


Figure 6.23: – Evaluation du taux d'énergie gaspillée pour cause d'échéances violées

La figure 6.23 nous présente l'évaluation du taux d'énergie gaspillée pour cause d'échéances violées. En raison de l'accroissement du taux d'utilisation des tâches, nous voyons que l'énergie gaspillée pour cause d'échéances violées est devenue plus grande, sauf pour EDt. C'est parce qu'il teste toujours l'énergie requise avant de pouvoir exécuter une tâche. Si l'énergie contenue dans le réservoir n'est pas suffisante pour assurer l'accomplissement complet d'une tâche, il ne commence pas l'exécution, et donc aucune énergie n'est gaspillée. Alors le comportement d'EDt est toujours très stable et est indépendant de U_p.

Les courbes des ordonnanceurs EDc, EDi et EDd évoluent en suivant une pente rapide lorsque le système est faiblement chargé ($U_p \leq 0.3$). A partir de $U_p = 0.3$, les courbes ne changent presque pas : elles restent globalement stables. C'est parce que ses performances concernant le taux d'échéances respectées se transforment en forme plus stable dès $U_p = 0.3$. Les performances ne se dégradent plus, cf. figure 6.21.

Au contraire, les performances des ordonnanceurs EDu et LSA se dégradent toujours en suivant une pente douce jusqu'à $U_p = 0.8$. Quand le système est lourdement chargé ($U_p \geq 0.6$), ses niveaux sont légèrement supérieurs à celui des ordonnanceurs EDc, EDi et EDd.

- Evaluation du nombre d'épuisements du réservoir d'énergie

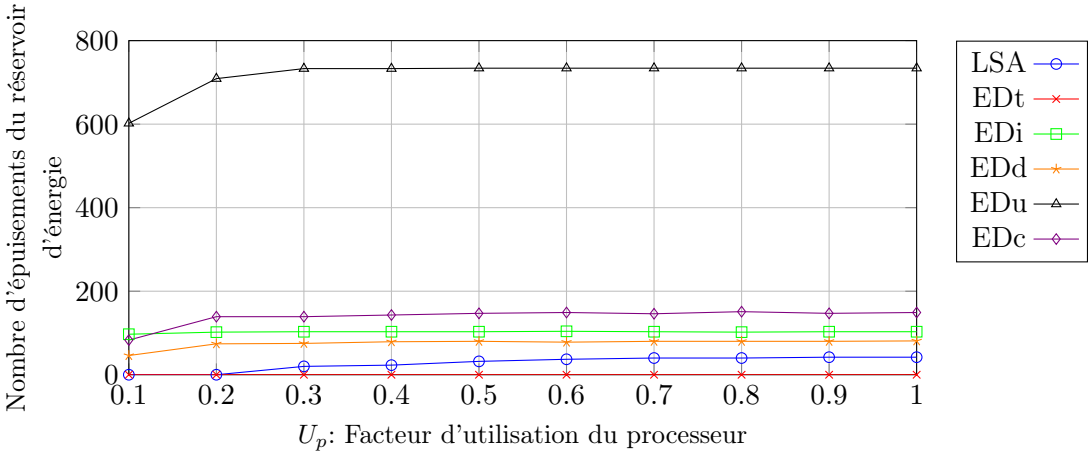


Figure 6.24: – Evaluation du nombre d'épuisements du réservoir d'énergie

Nous observons sur la figure 6.24, que pendant la durée de la simulation, le nombre d'épuisements du réservoir d'énergie de tous les ordonnanceurs augmente en suivant une pente douce lorsque $U_p \leq 0.2$. A partir de $U_p = 0.2$, toutes les courbes, sauf EDu, sont quasi stables, et donnent des valeurs inférieures à 200. En contraste, la valeur d'EDu est tellement grand, qu'il atteint environ 750 c'est-à-dire quatre fois plus.

Cela signifie que le niveau d'énergie contenue dans le réservoir tend à toujours rester très bas avec EDu, c'est-à-dire qu'il ne reste toujours qu'un fond d'énergie en réserve, affectant beaucoup la vie du réservoir d'énergie.

Sur cet aspect, le comportement de LSA est meilleur, le nombre maximal d'épuisements atteint uniquement 60 fois.

6.5.2 Etude avec puissance de source et facteur d'utilisation fixés

Nous effectuons dans ce paragraphe une étude avec les caractéristiques suivantes :

- La puissance maximale de source d'énergie aléatoire P_{rmax} atteint à 8.
- Le facteur d'utilisation du processeur U_p est successivement fixé à 0.2, 0.5 et 0.8. $U_p = 0.2$ correspond au système faiblement chargé, $U_p = 0.5$ correspond au système moyennement chargé et $U_p = 0.8$ correspond au système lourdement chargé.

– Le réservoir d'énergie est plein initialement, $E_c(0) = C$.
– Nous faisons varier la capacité du réservoir pour évaluer son impact sur la performance d'un système.

- Evaluation du taux d'échéances respectées

Le choix de la capacité du réservoir d'énergie est crucial dans la conception d'un système embarqué. Une plus grande capacité du réservoir signifie une plus grande taille ce qui n'est pas acceptable sur les systèmes embarqués. Dans ce paragraphe, nous allons déterminer le dimensionnement optimal du réservoir d'énergie.

La figure 6.25 (a) nous illustre l'évaluation du taux d'échéances respectées pour un système faiblement chargé ($U_p = 0.2$). Globalement, les taux d'échéances respectées augementent avec la croissance du réservoir d'énergie. Les ordonnanceurs EDt et EDc évoluent en suivant une pente douce, les autres évoluent en suivant une pente rapide, surtout LSA.

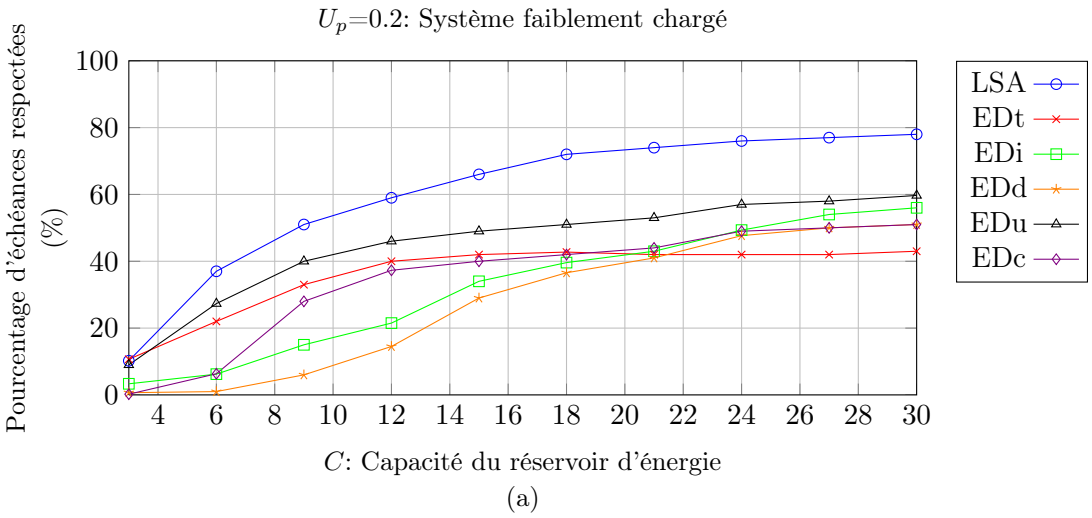
Quand $C = 3$, la différence entre les ordonnanceurs n'est pas très grande, mais avec l'augmentation de C, les écarts sont de plus en plus grands. Quand nous laissons $C = 30$, nous trouvons que LSA peut satisfaire 78% des échéances de tâches. Mais les autres sont capables de respecter 43% - 60% des échéances. Alors, pour satisfaire le même taux des échéances respectées donné, l'ordonnanceur LSA peut minimiser la taille du réservoir. Par exemple, pour réaliser 50% des échéances, l'ordonnanceur LSA a besoin uniquement d'une capacité de 9 au lieu de 21 pour l'ordonnanceur EDu, dons une taille de batterie réduite de 60%.

La figure 6.25 (b) nous montre l'évaluation du taux d'échéances respectées pour un système moyennement chargé ($U_p = 0.5$). Avec cette plus grande valeur de U_p, les performances des ordonnanceurs se dégradent naturellement. Les performances de LSA et EDu sont optimales. Avec l'augmentation de U_p, les écarts de performance entre les ordonnanceurs sont de plus en plus minces ; il apparaît que les comportements des ordonnanceurs ne sont pas très sensibles à la taille du réservoir d'énergie. Afin de garantir 40% des échéances, ils exigent un réservoir d'énergie d'une capacité minimum de 48. Parce que la source d'énergie est très faible, donc afin de respecter parfaitement les échéances pour tous les ordonnanceurs, la capacité du réservoir doit être énorme.

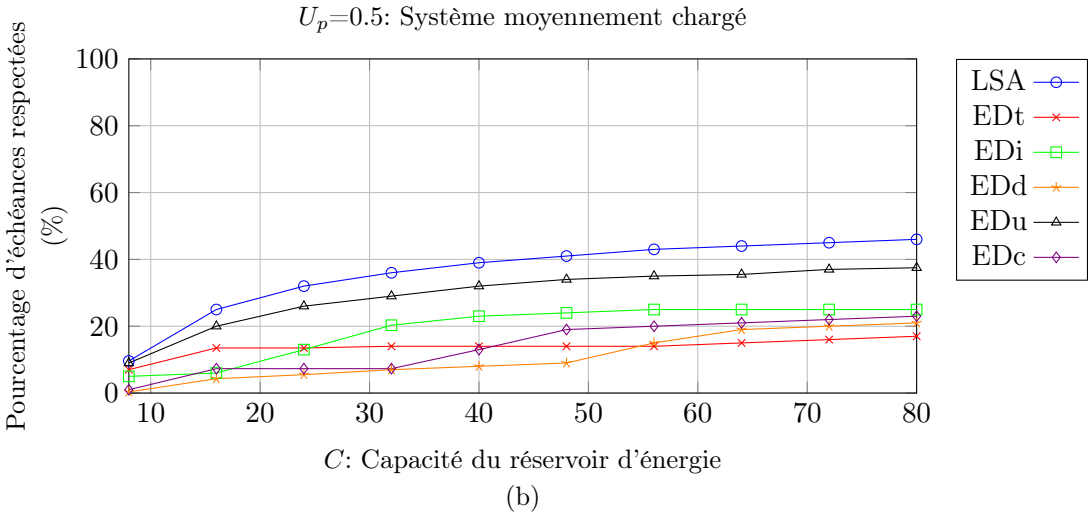
La figure 6.25 (c) nous présente l'évaluation du taux d'échéances respectées pour un système lourdement chargé ($U_p = 0.8$). En comparant avec les courbes précédentes, il apparaît clairement que les comportements des ordonnanceurs ne sont pas sensibles de plus en plus à la taille du réservoir d'énergie : une plus grande capacité du réservoir est nécessaire pour assurer le même niveau de performance. Par conséquent, l'utilisation d'une plus grande capacité du réservoir pour améliorer la performance n'est pas une solution.

- Evaluation du taux d'énergie gaspillée

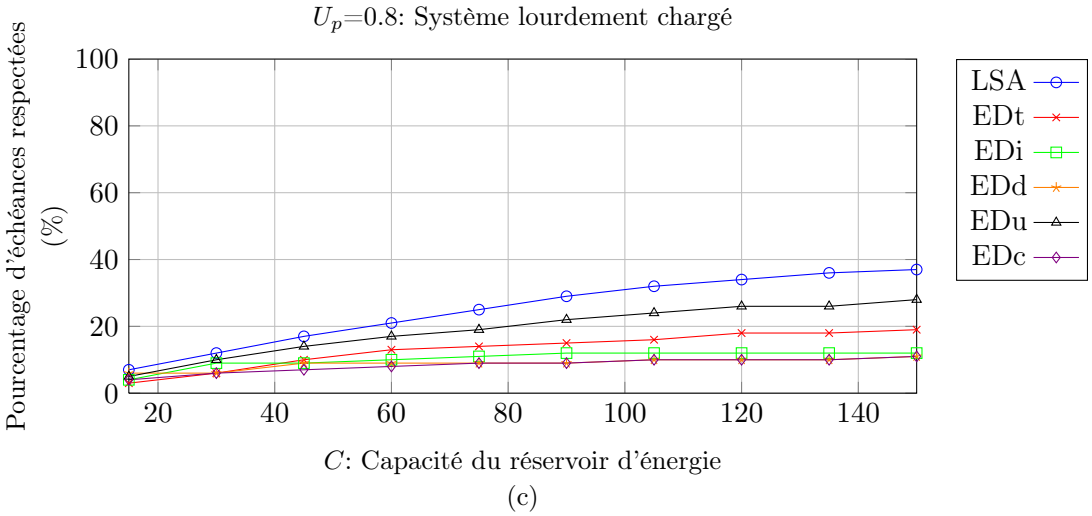
La figure 6.26 nous illustre les taux d'énergie gaspillée pour cause de réservoir d'énergie plein. Nous pouvons conclure que les courbes convergent uniformément vers une valeur définie avec l'augementation de C, sauf les ordonnanceurs EDt et EDu, par exemple, 35% pour $U_p = 0.2$, 5% pour $U_p = 0.5$ et 0% pour $U_p = 0.8$. En particulier, l'ordonnanceur EDt, ses valeurs définies



(a)

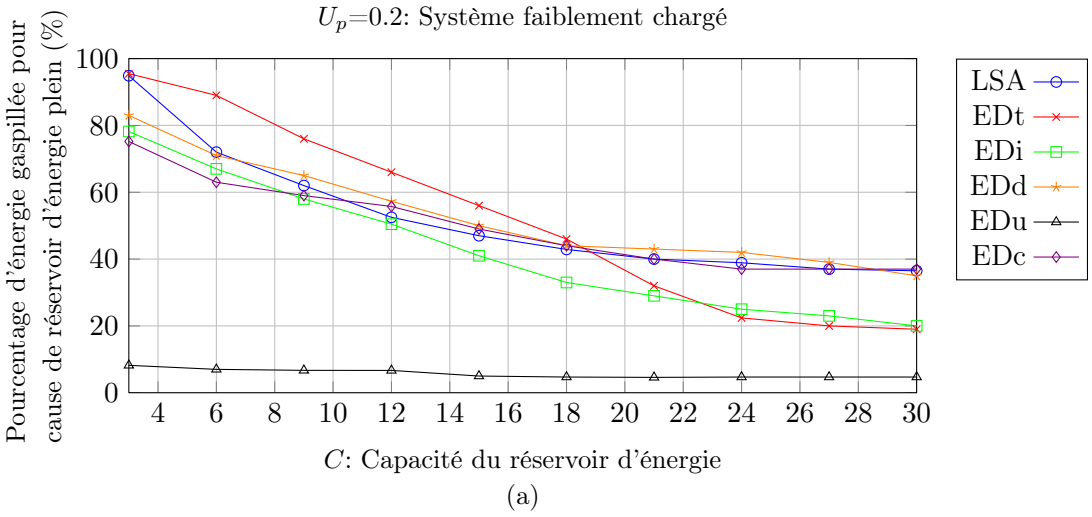


(b)

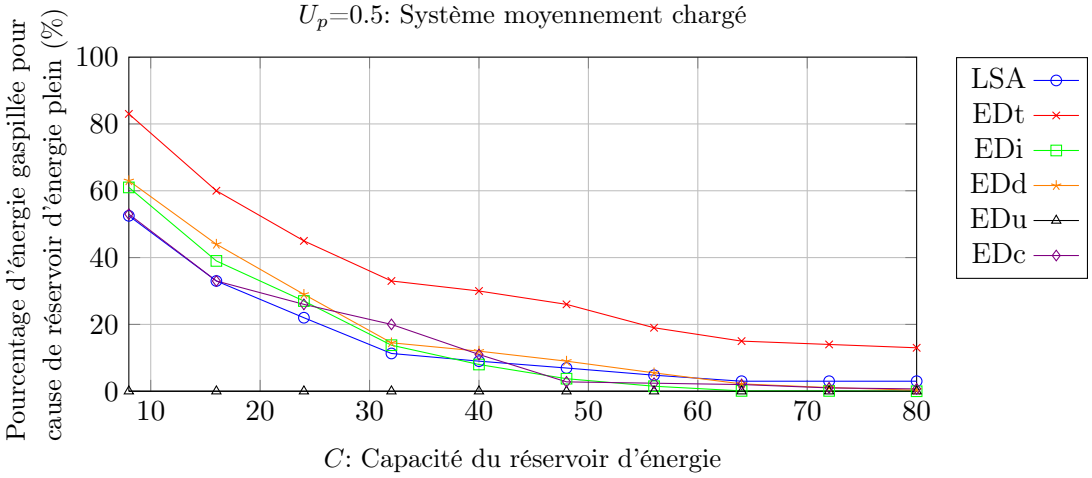


(c)

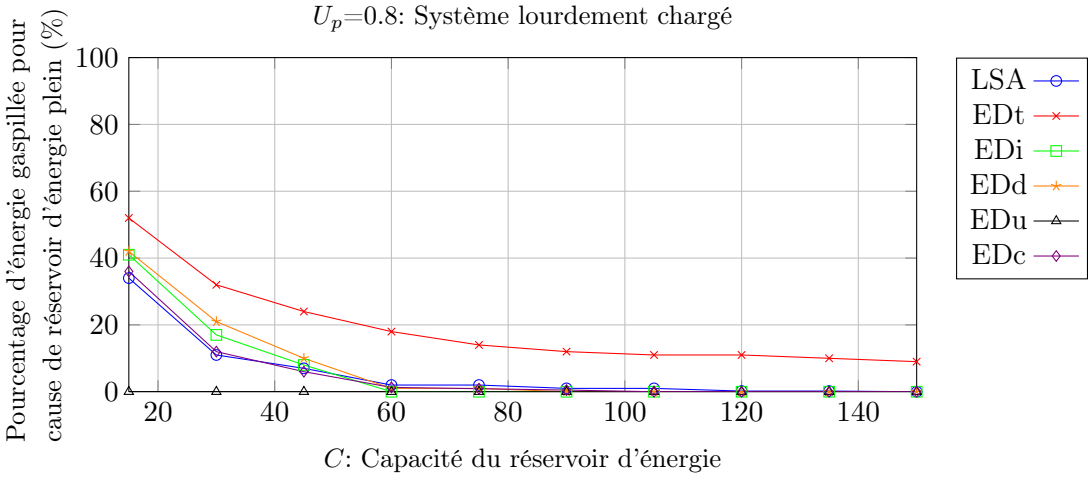
Figure 6.25: – Evaluation du taux d'échéances respectées



(a)

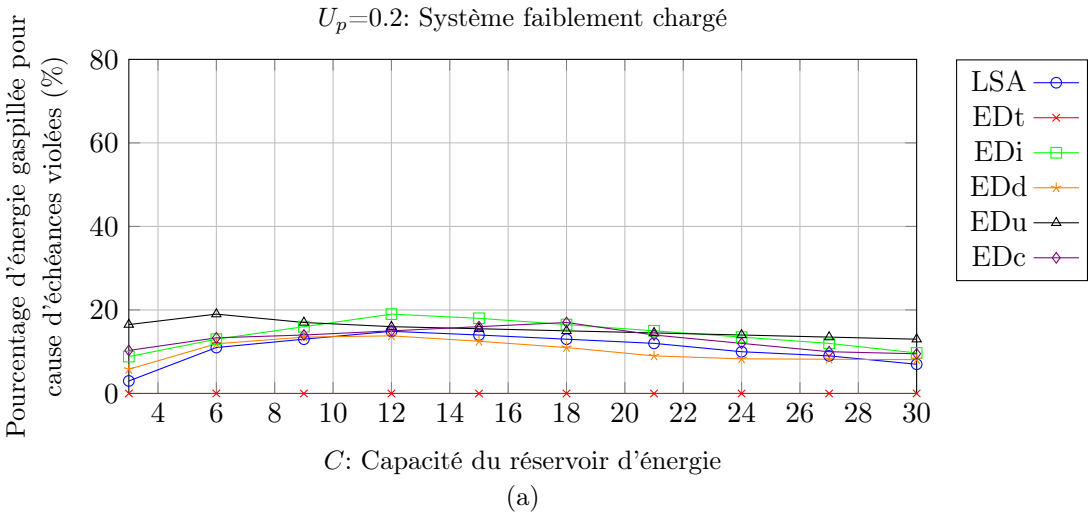


(b)

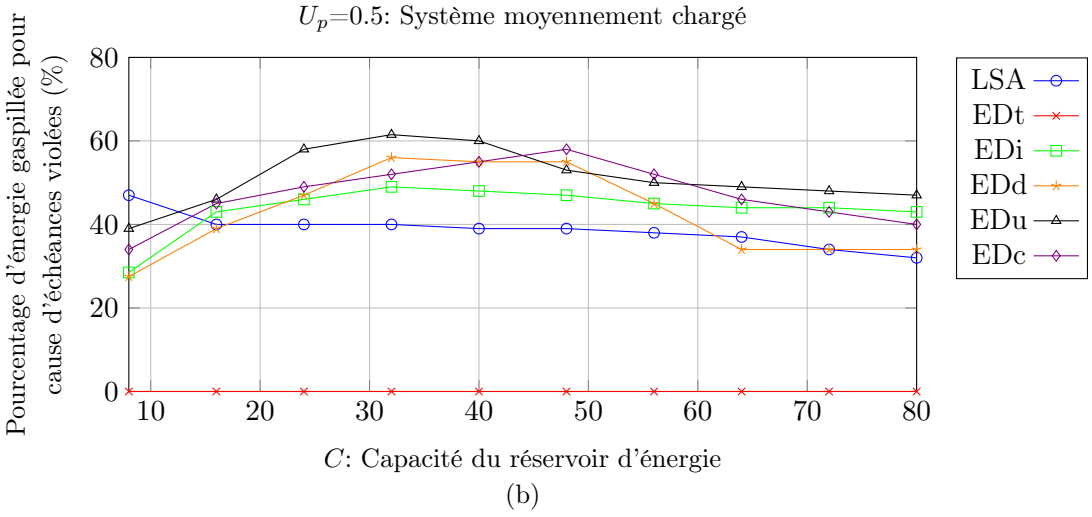


(c)

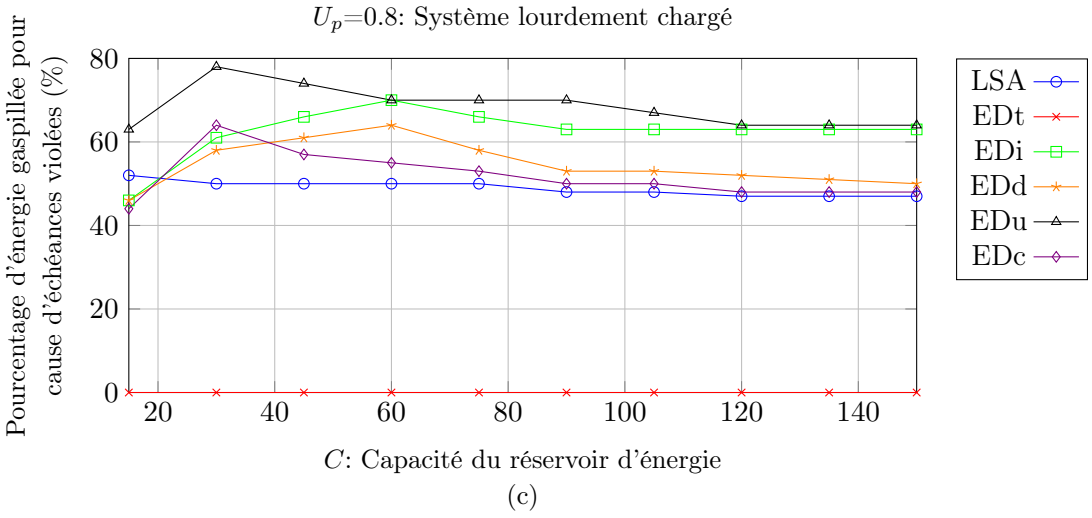
Figure 6.26: – Evaluation du taux d'énergie gaspillée pour cause de réservoir d'énergie plein



(a)



(b)



(c)

Figure 6.27: – Evaluation du taux d'énergie gaspillée pour cause d'échéances violées

ne sont pas très différentes dans n'importe quel cas, par exemple, 20% pour $U_p = 0.2$, 15% pour $U_p = 0.5$ et 10% pour $U_p = 0.8$. Sous cet aspect, le taux de gaspillage d'énergie d'EDt est le plus grand, tandis que celui d'EDu est le plus petit.

Sur la figure 6.26 (a), quand $C = 9$, les comportements des ordonnanceurs sont quasi-identiques, sauf EDt et EDu. Sur la figure 6.26 (b), quand $C = 24$, les comportements des ordonnanceurs sont quasi-identiques, sauf EDt et EDu. Sur la figure 6.26 (c), quand $C = 45$, les comportements des ordonnanceurs sont quasi-identiques, sauf EDt et EDu.

En conclusion, nous voyons aussi que l'impact de C est plus fort dans cette étude sur le taux d'énergie gaspillée pour cause de réservoir d'énergie plein.

La figure 6.27 nous présente les résultats sur l'évaluation du taux d'énergie gaspillée pour cause d'échéances violées. Avec l'augmentation de U_p, les écarts des ordonnanceurs deviennent graduellement larges, et les taux d'énergie gaspillée augmentent aussi. Dans ce type d'évaluation, ce n'est pas une solution efficace que d'augmenter indéfiniment la capacité du réservoir d'énergie pour diminuer le taux d'énergie gaspillée. D'ailleurs, nous constatons que la performance d'EDt est la meilleure.

- Evaluation du nombre d'épuisements du réservoir d'énergie

La figure 6.28 nous présente les résultats sur l'évaluation du nombre d'épuisements du réservoir d'énergie. A cause de la spécificité de l'ordonnanceur EDu, le nombre d'épuisements du réservoir reste toujours le plus grand, et il est vraiment indépendant de la capacité du réservoir d'énergie C et le facteur d'utilisation du processeur U_p. En contraste, celui de l'ordonnanceur EDt est le plus petit.

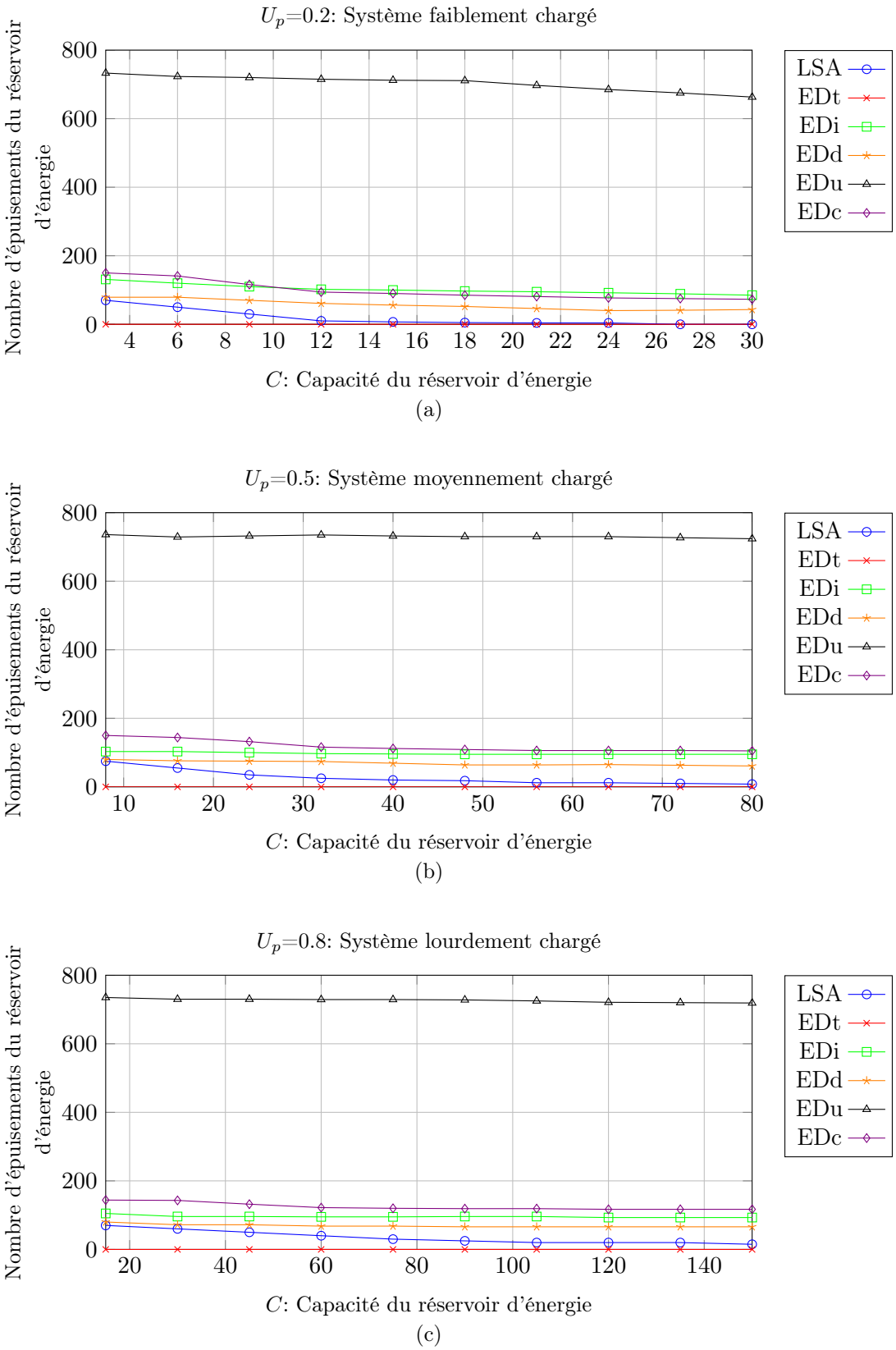


Figure 6.28: – Evaluation du nombre d'épuisements du réservoir d'énergie

6.5.3 Evaluation du surcoût d'exécution (overhead)

Nous allons comparer les surcoûts d'exécution pour les systèmes alimentés par une sourece d'énergie aléatoire. Comme dans l'analyse précédente, parce que les heuristiques sont des ordonnanceurs non clairvoyants, ils n'ont pas besoin d'un grand nombre de calculs en-ligne. Cependant, en tant qu'ordonnanceur clairvoyant du point de vue énergétique, LSA va effectuer des calculs complexes pour déterminer la date optimale d'exécution de toute tâche arrivante en fonction du profil énergétique à venir.

Ici, nous testons un système moyennement chargé ($U_p = 0.5$) avec un réservoir d'énergie donné ($C = 10$) et une source d'énergie aléatoire qui suit une loi normale (cf. figure 6.20). Les résultats sur le surcoûts d'exécution sont illustrés sur les figures 6.29. Evidemment, nous constatons que le surcoût d'exécution de l'ordonnanceur LSA est beaucoup plus grand que celui des heuristiques.

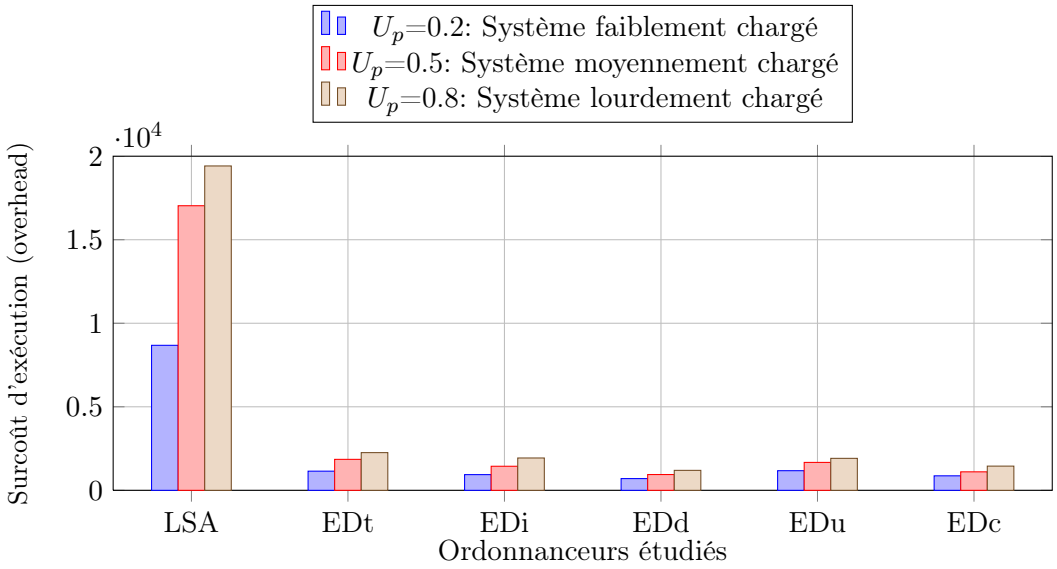


Figure 6.29: – Evaluation du surcoût d'exécution en fonction de la charge de système

6.6 Synthèse des résultats de simulation

Le but de cette étude de simulation a été d'évaluer les performances des heuristiques proposées de façon comparative avec l'ordonnanceur démontré optimal par la théorie et ce, avec une source d'énergie variable. Nous voulions particulièrement nous intéresser aux sources d'énergie quasi-périodiques de type impulsionnel. A travers les résultats expérimentaux obtenus dans ce chapitre, au vu d'une part des descriptions algorithmiques, et d'autre part des résultats de simulation, nous pouvons dresser le tableau récapitulatif suivant (cf. tableau 6.2) résumant les performances des différents ordonnanceurs en termes de performances et de complexité d'implémentation.

	EDt	EDi	EDd	EDu	EDc	LSA
Qualité de service (taux d'échéances respectées)	☹	☹	☹	😐	☹	☺
Sourcoût d'exécution	😐	😐	😐	😐	😐	☹
Taille de réservoir d'énergie	☹	☹	☹	😐	☹	☺
Nombre de décharges complètes	😐	😐	😐	☹	😐	☺

Tableau 6.2: – Synthèse des performance des ordonnanceurs avec profil énergétique variable

6.7 Conclusion

Dans ce chapitre, nous avons effectué des simulations pour comparer l'algorithme clairvoyant LSA et les heuristiques non-clairvoyantes avec une source d'énergie environnementale variable.

Par cette étude, nous avons obtenu des résultats similaires à ceux obtenus dans le chapitre précèdent dans le sens où la hierarchie des ordonnanceurs reste la même quelque soit la forme du profil énergétique de la source, constante ou variable.

- L'ordonnanceur LSA garantit la meilleure qualité de service, c'est-à-dire permet d'obtenir le plus grand taux d'échéances respectées pour une même configuration de tâches. De plus, pour une qualité de service donnée, la taille du réservoir d'énergie de LSA requis est la plus petite.
- L'ordonnanceur EDd fournit la plus mauvaise qualité de service.
- La durée de vie du réservoir d'énergie est la plus courte pour un système utilisant l'ordonnanceur EDu. Le nombre de décharges complètes étant important, cela conduit à un vieillissement plus rapide de la batterie.

Nou avons voulu calculer le surcoût d'exécution de LSA. Compte tenu de la complexité de calcul des dates de démarrage des tâches, ce surcoût temporel devient considérable par rapport à l'ensemble des heuristiques simulées puisque multiplié par dix environ dans notre étude, qu'il s'agisse d'une source énergétique de type impulsionnel ou suivant une loi normale. D'autre part, il ne faut pas non plus négliger l'espace mémoire requis pour le stockage du profil énergétique. Tous ces constats nous amènet à conclure que l'ordonnancement non clairvoyant est certainement moins performant mais a pour grand mérite d'une implémentation aisée.

Chapitre 7

Une proposition : LSA approximé

7.1 Introduction

A travers les travaux précédents, nous concluons que quand la loi de variation de la source d'énergie est complexe, le grand surcoût d'exécution de LSA remet en question son optimalité réelle.

En effet, quand la loi de variation de la source d'énergie est constante ou suit une fonction affine, nous pouvons toujours trouver une méthode simple pour diminuer le surcoût, cf.4.6.1 et 4.6.2.1. Mais si le profil de la source d'énergie est plus complexe, nous devons effectuer D_i fois plus de calculs pour déterminer la date de début d'exécution, cf. section 4.6.2.2.

Créer un profil approximé de la source d'énergie permettrait ainsi de limiter la complexité des calculs. Si la détermination de la date de début d'exécution des tâches devient plus facile, alors le surcoût diminue. Dans ce chapitre, nous proposons de limiter les overheads par une version sous-optimale de LSA qui consiste à effectuer un calcul approximatif de la date de réveil de chaque tâche, en veillant toutefois à garantir une qualité de service acceptable.

7.2 Exemples illustratifs

Exemple 7.1. *Soit une tâche apériodique $\tau_1 = (0, 32, 16)$. La source d'énergie complexe suit une loi parabolique $P_r(t) = 0.03 \cdot t^2 + 0.32$, cf. figure 7.1. Supposons que $P_{max} = 8$, $C = 12$ et $E_c(0) = 0$.*

Ici pour une telle source d'énergie complexe, nous utilisons d'abord la méthode présentée dans 4.6.2.2 pour déterminer la date "s_i".

$$\begin{aligned} s_1^* &= d_1 - \frac{E_c(a_1) + \int_{a_1}^{d_1} P_r(t) \cdot dt}{P_{max}} = d_1 - \frac{E_c(a_1) + \sum_{t=a_1}^{d_1} P_r(t) \cdot dt}{P_{max}} \\ &= 16 - \frac{0 + \sum_{t=0}^{16} P_r(t)}{8} = 16 - \frac{0 + 42.32}{8} = 10.71 \end{aligned} \tag{7.1}$$

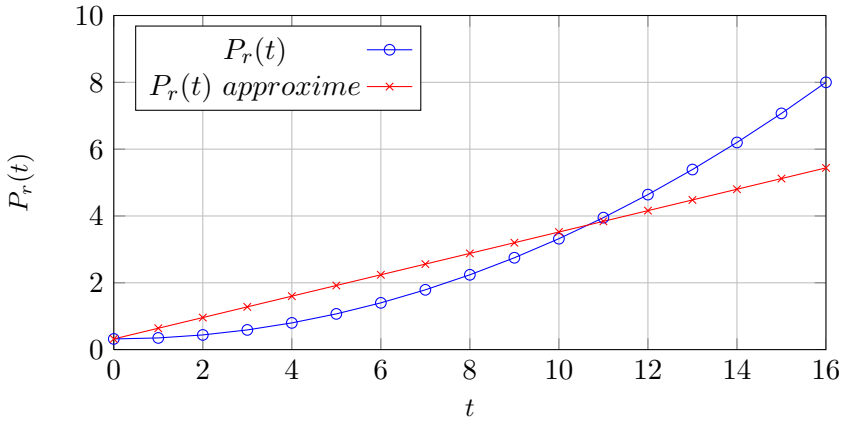


Figure 7.1: – Profil d'une source d'énergie non linéaire

Puis nous calculons $s_1^{'}$. Nous vérifions la valeur de la formule 4.34 à chaque instant entre [0, 16]. Si la valeur de $X(t)$ est nulle à un instant, alors celui-ci est choisi comme la date de début d'exécution s_1. Sinon nous choisissons le temps où la valeur $X(t)$ est la plus proche de zéro pour s_1, cf. tableau 7.1. Parmi ces valeurs, nous ne trouvons pas l'instant t où $X(t) = 0$, mais nous pouvons déterminer que $s_1^{'}$ se trouve dans [11, 12]. Donc nous déterminons $s_1^{'} \approx 11$. Enfin, $s_1 = max(s_1^*, s_1^{'}) = max(10.71, 11) = 11$. La valeur optimale théorique de s_1 est 11.

Dans cet exemple, nous trouvons que l'algorithme effectue 16 (D_i=16) fois des calculs pour déterminer la date de début d'exécution. Nous pouvons approximer cette source d'énergie complexe par une source d'énergie qui suit une loi affine. Nous utilisons alors la méthode présentée dans 4.6.2.1.

Ici, nous avons créé une source d'énergie approximée qui suit $P_r(t) = 0.32 \cdot t + 0.32$ pour remplacer la source originale $P_r(t) = 0.03 \cdot t^2 + 0.32$, dont $\int_0^{16} P_{r\ approxime}(t) \cdot dt = \int_0^{16} P_r(t) \cdot dt$. Dans la figure 7.1, la courbe rouge présente cette source d'énergie approximé.

Nous utilisons les équations 4.32 et 4.33 pour déduire "s_i" :

$$
\begin{aligned}
s_1^* &= d_1 - \frac{E_c(a_1) + \int_{a_1}^{d_1} P_{r\ approxime}(t) \cdot dt}{P_{max}} = 16 - \frac{0 + 42.32}{8} = 10.71 \\
s_1^{'} &= P_{max} - b \pm \sqrt{b^2 + P_{max}^2 - 4 \cdot b \cdot P_{max} + 2 \cdot C + d_i^2 + 2 \cdot b \cdot d_i - 2 \cdot P_{max} \cdot d_i} = 12 \\
s_1 &= max(s_1^*, s_1^{'}) = max(10.71, 12) = 12
\end{aligned}
$$

Dans la partie suivante, nous allons faire des simulations pour évaluer cette proposition c'est-à-dire déterminer l'impact de cette approximation sur la détermination de la date de démarrage et par voie de conséquence sur la performance qui en résulte.

t	$X(t) = P_{max} \cdot d_i - C - \int_t^{d_i} P_r(t) \cdot dt - P_{max} \cdot t$
0	$X(0) = 8 \cdot 16 - 12 - 42.32 - 8 \cdot 0 = 73.68$
1	$X(1) = 8 \cdot 16 - 12 - 42 - 8 \cdot 1 = 66$
2	$X(2) = 8 \cdot 16 - 12 - 41.65 - 8 \cdot 2 = 58.35$
3	$X(3) = 8 \cdot 16 - 12 - 41.21 - 8 \cdot 3 = 50.79$
4	$X(4) = 8 \cdot 16 - 12 - 40.62 - 8 \cdot 4 = 43.38$
5	$X(5) = 8 \cdot 16 - 12 - 39.82 - 8 \cdot 5 = 36.18$
6	$X(6) = 8 \cdot 16 - 12 - 38.75 - 8 \cdot 6 = 29.25$
7	$X(7) = 8 \cdot 16 - 12 - 37.35 - 8 \cdot 7 = 22.65$
8	$X(8) = 8 \cdot 16 - 12 - 35.56 - 8 \cdot 8 = 16.44$
9	$X(9) = 8 \cdot 16 - 12 - 33.32 - 8 \cdot 9 = 10.68$
10	$X(10) = 8 \cdot 16 - 12 - 30.57 - 8 \cdot 10 = 5.43$
11	$X(11) = 8 \cdot 16 - 12 - 27.25 - 8 \cdot 11 = 0.75$
12	$X(12) = 8 \cdot 16 - 12 - 23.3 - 8 \cdot 12 = -3.3$
13	$X(13) = 8 \cdot 16 - 12 - 18.66 - 8 \cdot 13 = -6.66$
14	$X(14) = 8 \cdot 16 - 12 - 13.27 - 8 \cdot 14 = -9.27$
15	$X(15) = 8 \cdot 16 - 12 - 7.07 - 8 \cdot 15 = -11.07$
16	$X(16) = 8 \cdot 16 - 12 - 0 - 8 \cdot 16 = -12$

Tableau 7.1: – Calcul de s_i

7.3 Résultats expérimentaux

Nous effectuons ici une étude avec les caractéristiques suivantes :

- La source d'énergie varie en suivant une fonction sinus. La puissance maximale de source d'énergie P_{rmax} est fixée à 7, cf. figure 7.2.
- La capacité du réservoir d'énergie C est fixé à 10.
- Le réservoir d'énergie est plein initialement, $E_c(0) = C = 10$.
- Nous générons des configurations de tâches ayant un taux d'utilisation du processeur U_p compris entre 0 et 1.

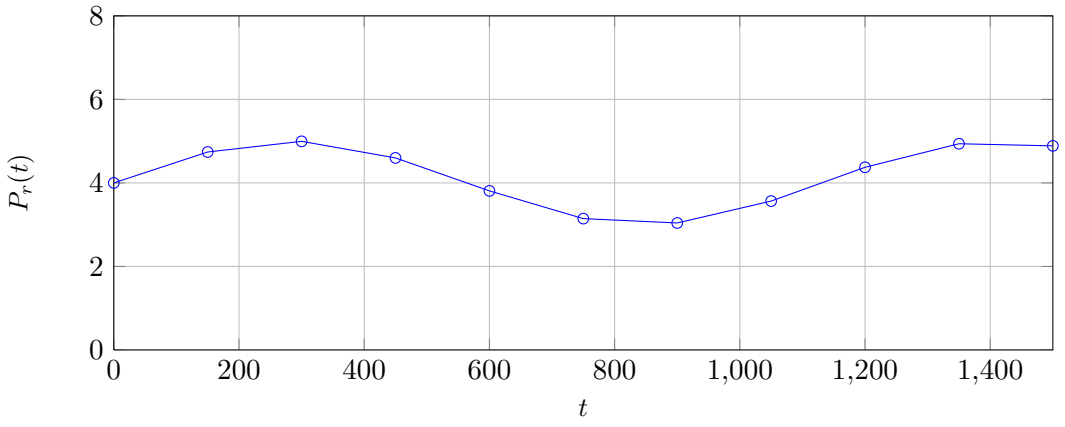


Figure 7.2: – Source d'énergie étudiée

La comparaison sur le taux d'échéances respectées est présenté sur la figure 7.3. Avec l'augmentation de U_p, l'écart entre LSA et LSA approximé devient de plus en plus grand. Quand le système est lourdement chargé ($U_p \geq 0.7$), cet écart arrive à une valeur maximale, qui reste inférieure à 7%.

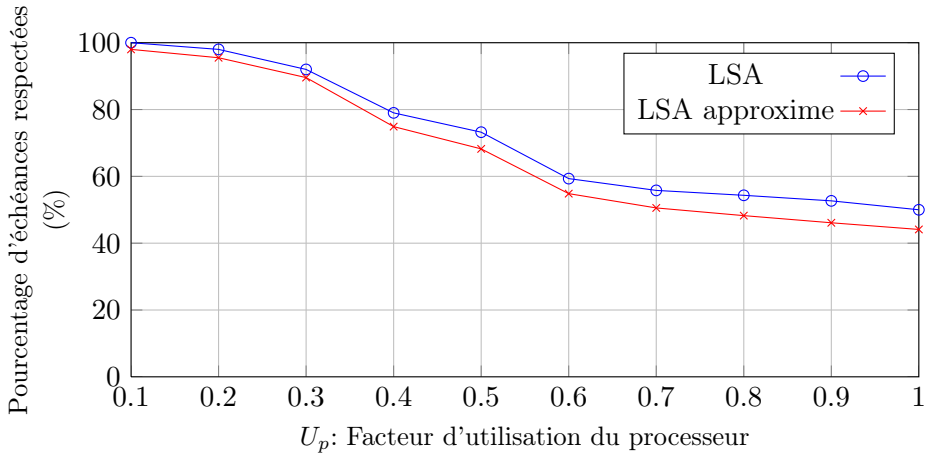


Figure 7.3: – Evaluation du taux d'échéances respectées

Enfin, nous avons mesuré le gain obtenu en terme de surcoût. La plus grande contribution de l'ordonnanceur LSA approximé est de diminuer largement le surcoût d'exécution puisque celui-ci est seulement 25% de celui de LSA, cf. figure 7.4.

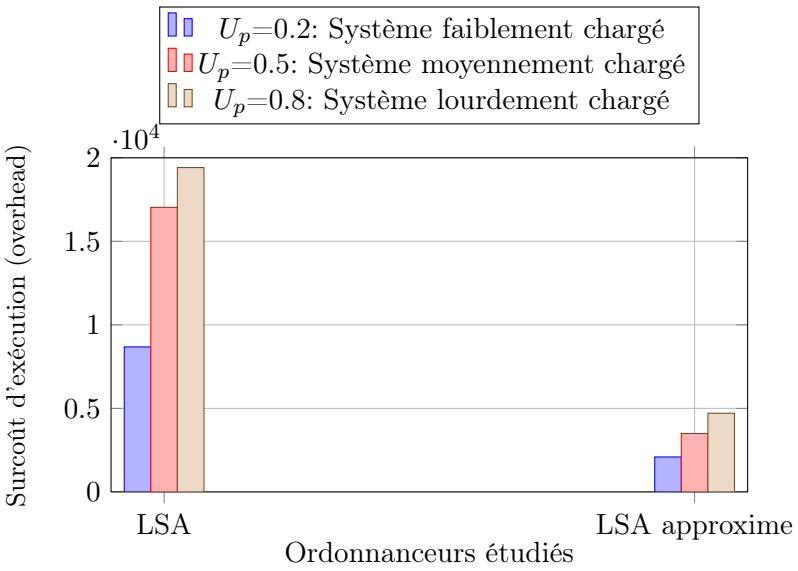


Figure 7.4: – Evaluation du surcoût d'exécution en fonction de la charge de système

7.4 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle méthode pour calculer plus aisément la date de début d'exécution de tâches, "s_i", en approximant le profil énergétique. La performance de LSA est légèrement dégradée mais sa mise en oeuvre se fait en quatre fois moins de temps.

Cependant, l'ordonnanceur LSA approximé ne convient pas à toutes les sources d'énergie. Il s'applique seulement aux sources d'énergie qui varient avec une faible dynamique.

Chapitre 8

Présentation d'un outil de test

8.1 Introduction

Dans ce chapitre, nous allons présenter un environnement logiciel configuratif de création d'applications de test temps réel. L'objectif d'un tel logiciel est d'aider les ingénieurs à tester et à paraméter des systèmes temps réel alimentés par de l'énergie renouvelable et utilisant un réservoir d'énergie, comme ceux décrits dans ce qui précède. Cet outil a pour but de rendre l'intégration d'un système plus rapide et ainsi de mieux faire face aux exigences présentes portant sur les coûts de mise au point.

Afin d'évaluer, comparer et améliorer les performances des heuristiques d'ordonnancement, variantes d'EDF et LSA, nous les avons soumis à une série de tests de performance : test d'ordonnançabilité, test de qualité de service, test de capacité du réservoir d'énergie et test de robustesse. L'outil de test a été implémenté en langage C. La structure globale de cet outil est illustrée sur la figure 8.1.

Les fonctionalités de l'outil permettent de :

- créer les configurations de tâches périodiques ;
- émuler différentes formes d'énergies renouvelables (périodiques, non périodiques, aléatoires, etc.) ;
- simuler différents ordonnanceurs (LSA, EDt, EDi, EDd, EDu et EDc) ;
- gérer toutes les données de simulation ;
- illustrer les résultats de façon graphique (séquences produites d'ordonnancement, variation de l'énergie contenue du réservoir d'énergie).

8.2 Description de l'outil de test

8.2.1 Objectif de l'outil de test

L'objectif de cet outil de test est essentiellement de permettre aux ingénieurs d'effectuer des tests de performance. Les critères d'évaluation sont ceux proposés dans la section 5.3 du chapitre 5. Cet outil de test a été conçu spécifiquement pour des tâches périodiques, lesquelles sont liées à des contraintes énergétiques. A travers cet outil de test, nous pouvons obtenir les détails d'ordonnancement d'une configuration de tâches. Selon des résultats de sortie, nous pouvons

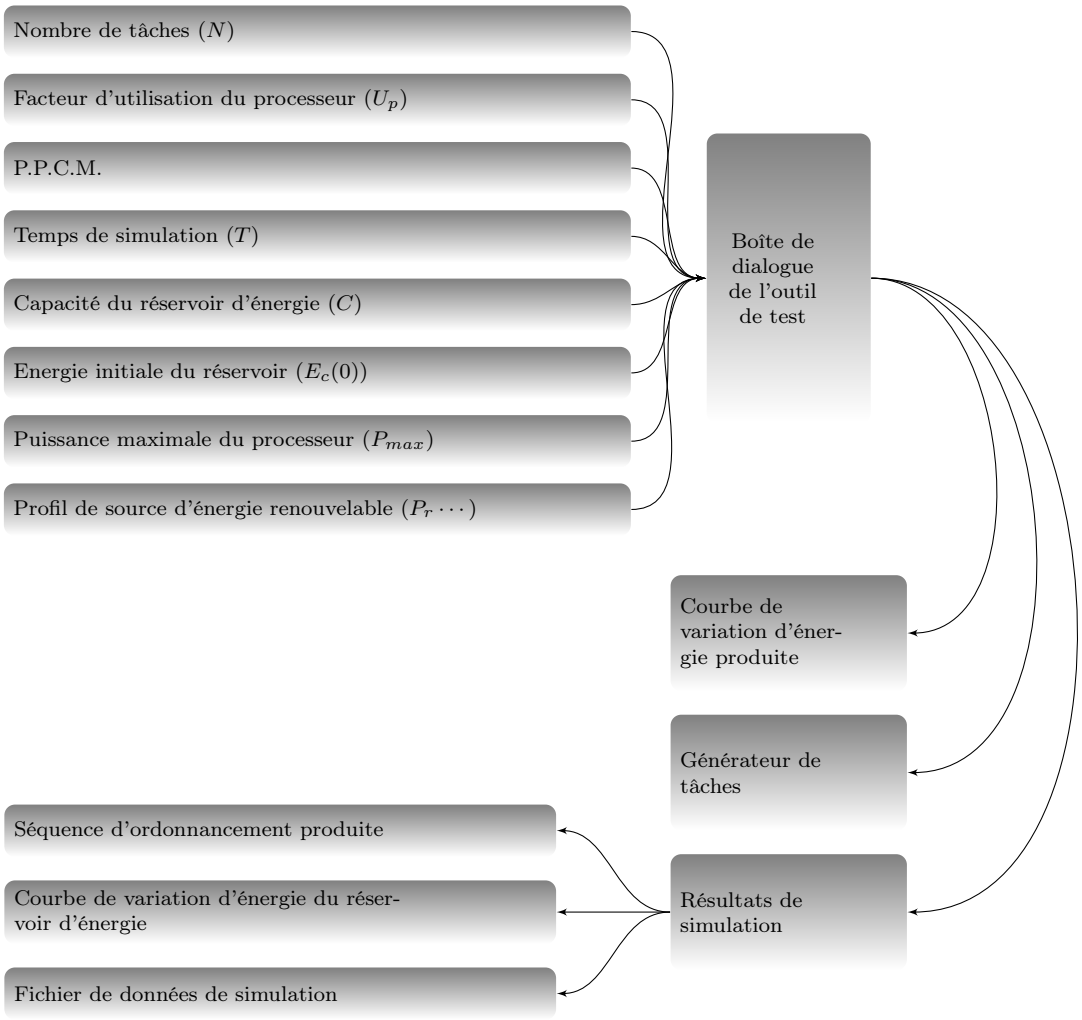


Figure 8.1: – Structure de l'outil de test

facilement évaluer les performances d'un système temps réel et vérifier l'ordonnançabilité d'une configuration de tâches. Par exemple, nous pouvons évaluer le taux d'échéance respectées d'une configuration de tâches et l'efficacité de l'énergie récupérée, etc.

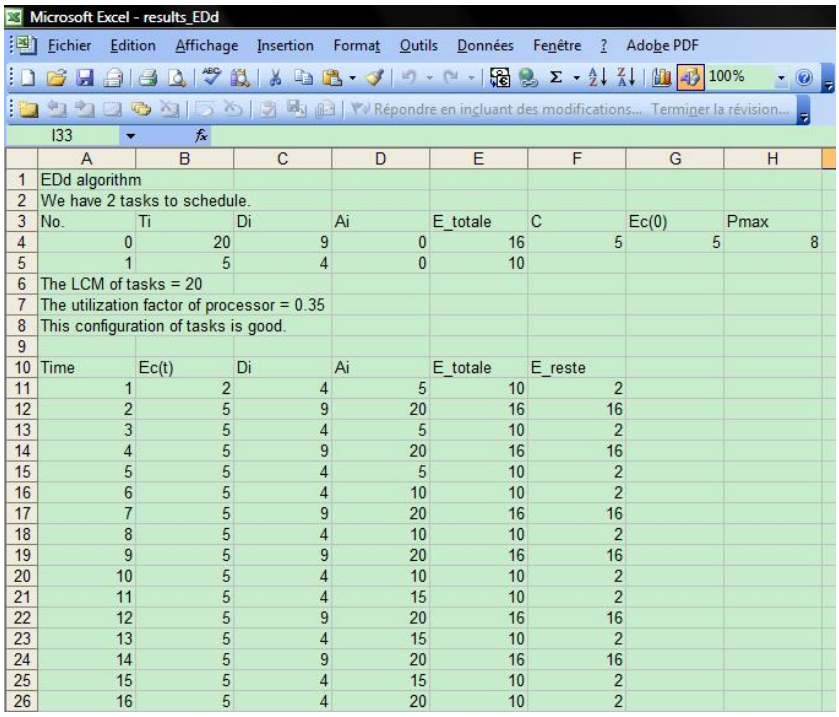
Pendant la simulation, les données de simulation sont enregistrées dans un fichier *Excel*, appelé aussi *feuille de calculs*, comme l'illustre la figure 8.2. Parmi les résultats de test, la séquence produite d'exécution des tâches peut être visuellement affichée sur l'écran (cf. figure 8.11). En outre, la courbe de variation de l'énergie contenue dans le réservoir d'énergie est aussi engendrée et illustrée sur l'écran (cf. figure 8.10).

Par ailleurs, si nous voulons changer les conditions de simulation ou les valeurs initiales de simulation, cela peut aisément se réaliser à travers une interface conviviale (cf. figure 8.3).

8.2.2 Composants de l'outil de test

Cet outil de simulation se compose de quatre modules principaux :

– un module de dialogue interactif avec le client,



Microsoft Excel - results_EDd

Fichier Edition Affichage Insertion Format Outils Données Fenêtre ? Adobe PDF

I33

	A	B	C	D	E	F	G	H
1	EDd algorithm							
2	We have 2 tasks to schedule.							
3	No.	Ti	Di	Ai	E_totale	C	Ec(0)	Pmax
4	0	20	9	0	16	5	5	8
5	1	5	4	0	10			
6	The LCM of tasks = 20							
7	The utilization factor of processor = 0.35							
8	This configuration of tasks is good.							
9								
10	Time	Ec(t)	Di	Ai	E_totale	E_reste		
11	1	2	4	5	10	2		
12	2	5	9	20	16	16		
13	3	5	4	5	10	2		
14	4	5	9	20	16	16		
15	5	5	4	5	10	2		
16	6	5	4	10	10	2		
17	7	5	9	20	16	16		
18	8	5	4	10	10	2		
19	9	5	9	20	16	16		
20	10	5	4	10	10	2		
21	11	5	4	15	10	2		
22	12	5	9	20	16	16		
23	13	5	4	15	10	2		
24	14	5	9	20	16	16		
25	15	5	4	15	10	2		
26	16	5	4	20	10	2		

Figure 8.2: – Feuille de calculs

- un module pour engendrer la courbe E.V.C.C. pour simuler les différentes sources d'énergie renouvelable,
- un module de générateur de tâches périodiques,
- et un module d'affichage des résultats de simulation.

Dans les paragraphes suivants, nous décrirons en detail la fonction de chacun de ces modules.

8.2.2.1 Module de dialogue interactif

Ce logiciel est un outil idéal pour la simulation des comportements d'un prototype physique. Dans ce logiciel, nous avons mis beaucoup de modèles numériques pour simuler un environnement réel.

A travers ce dialogue, nous configurons d'abord les paramètres N, U_p et $PPCM$ pour le générateur de tâches. Puis nous saisissons la durée de simulation et la puissance maximale du processeur de la même façon. La durée de simulation doit être un multiple du PPCM des périodes.

Quant au réservoir d'énergie, nous le caractérisons par C (la capacité) et $E_c(0)$ (l'énergie initiale dans le réservoir). Il est en général fait l'hypothèse dans les applications que $E_c(0) = C$, c'est-à-dire, le réservoir est plein en début de simulation.

De par les types des sources d'énergie renouvelables les plus souvent rencontrées, nous avons considéré quatre types possibles (puissance constante, à impulsions périodiques, à distribution uniforme et à distribution normale). Nous pouvons obtenir l'énergie protéiforme en changeant des valeurs du paramètre clef.

Avec l'interface interactive (cf. la figure 8.3), l'ingénieur peut facilement configurer les para-

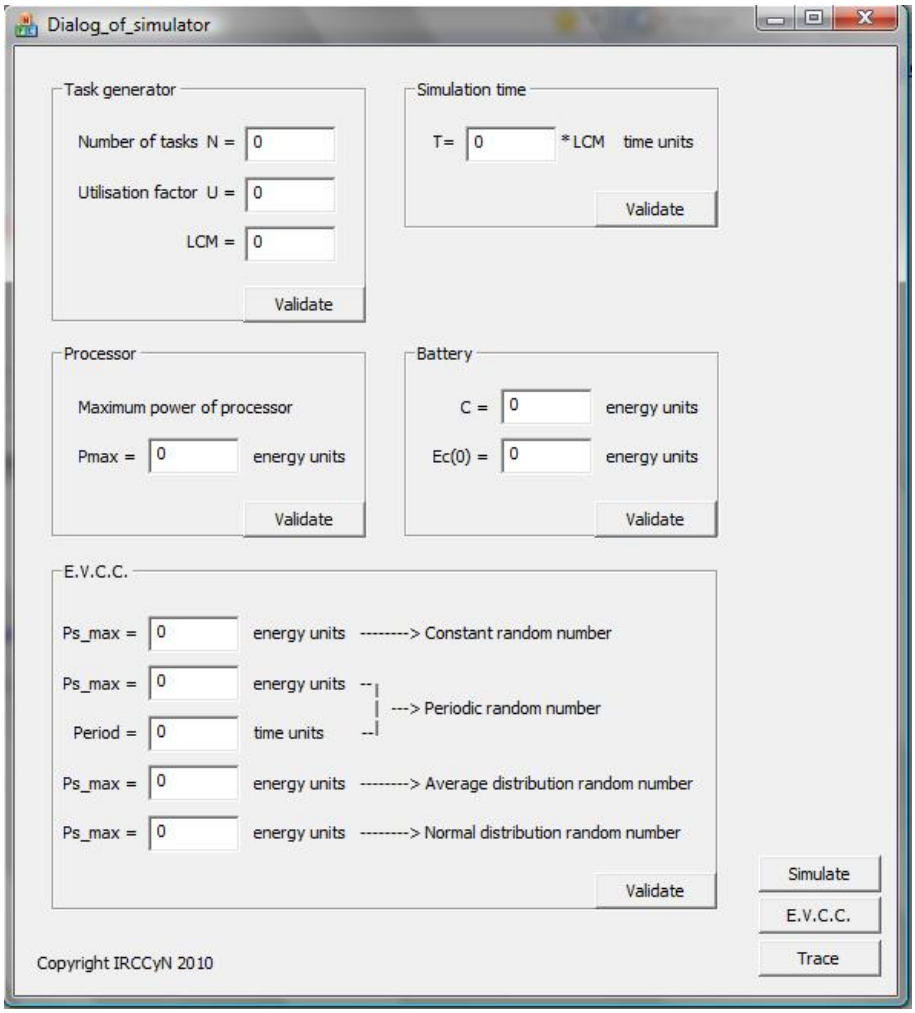


Figure 8.3: – Dialogue de l'outil de test

mètres nécessaires de simulation pour construire une plate-forme expérimentale.

8.2.2.2 Module générateur de tâches

Au niveau du générateur, chaque tâche sera caractérisée par le quadruplet (a_i, E_i, D_i, T_i). Les principaux critères de choix de tâches à générer à un instant donné sont mis en évidence par ce quadruplet. Par ailleurs, les tâches doivent avoir la même probabilité d'être générées, ce qui nécessite un algorithme efficace et particulier faisant recours aux méthodes statistiques de tirages aléatoires. L'algorithme de génération de tâches suit la démarche suivante :

- Calcul des périodes des tâches sur la base du paramètre PPCM fourni par l'utilisateur. Les périodes sont des diviseurs entiers du paramètre PPCM.
- Calcul des durées d'exécution des tâches.
- Affinement du calcul des durées d'exécution pour obtenir une charge totale effective la plus proche possible de celle désirée.

Parmi les tâches générées par ce générateur de tâches, il n'existe pas d'équilibre de charge, pas de lien direct entre durée d'exécution et période. La durée d'exécution est aléatoire. Par exemple, nous lançons ce générateur avec les paramètres suivants :

- Le nombre de tâches : $N = 4$;
- Le facteur d'utilisation du processeur : $U_p = 0.8$;
- Un PPCM des périodes : $PPCM = 100$.

Alors le générateur retourne la configuration de tâches donnée sur la figure 8.4.

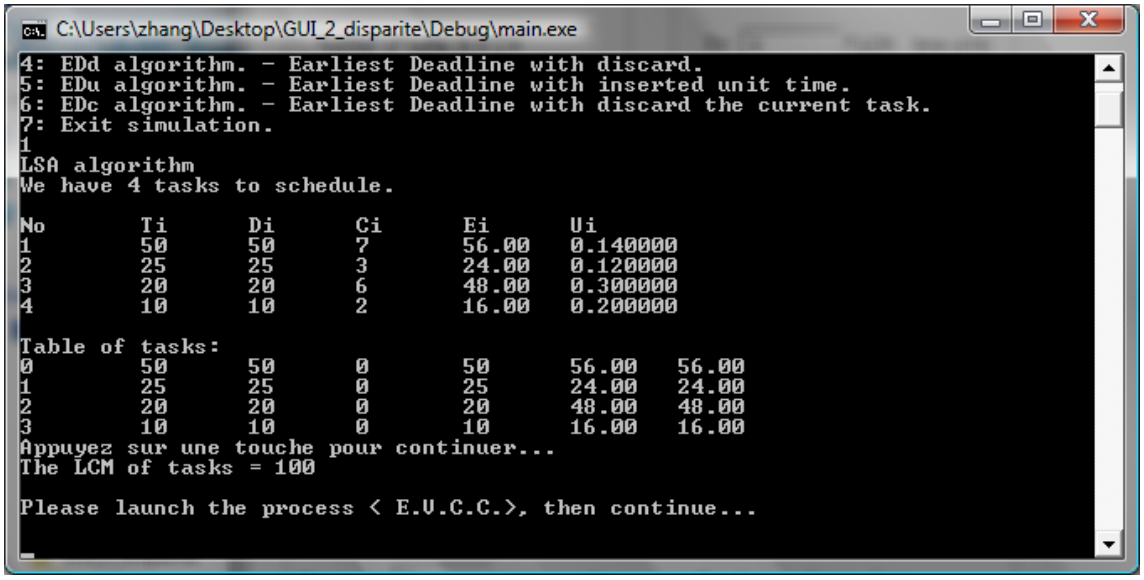


Figure 8.4: – Génération de tâches périodiques

8.2.2.3 Module source d'énergie

Dans ce module de source d'énergie, nous allons lancer un processus pour produire les différents E.V.C.C. (en anglais Energy Variability Characterization Curves). Chaque courbe représente un type de source d'énergie. Comme nous le voyons sur la figure 8.5, nous pouvons choisir l'une des quatre différentes sources d'énergie grâce aux boutons.

Il existe la source d'énergie constante (pour modéliser des sources photovoltaïques indoor,par exemple, cf. figure 8.5), la source d'énergie périodique (pour modéliser des sources piézo-électriques, etc., cf. figure 8.6 et figure 8.7), la source d'énergie en distribution uniforme (pour modéliser des sources électromagnétiques, des sources de vibrations mécaniques, etc., cf. figure 8.8), et la source d'énergie en distribution normale (pour modéliser la source solaire, etc., cf. figure 8.9).

8.2.2.4 Module résultats de simulation

Les résultats de simulation, liés à la performance du système, ne sont pas seulement affichés sur l'écran sous forme graphique (cf. figures 8.10 et 8.11). Toutes les données du processus sont

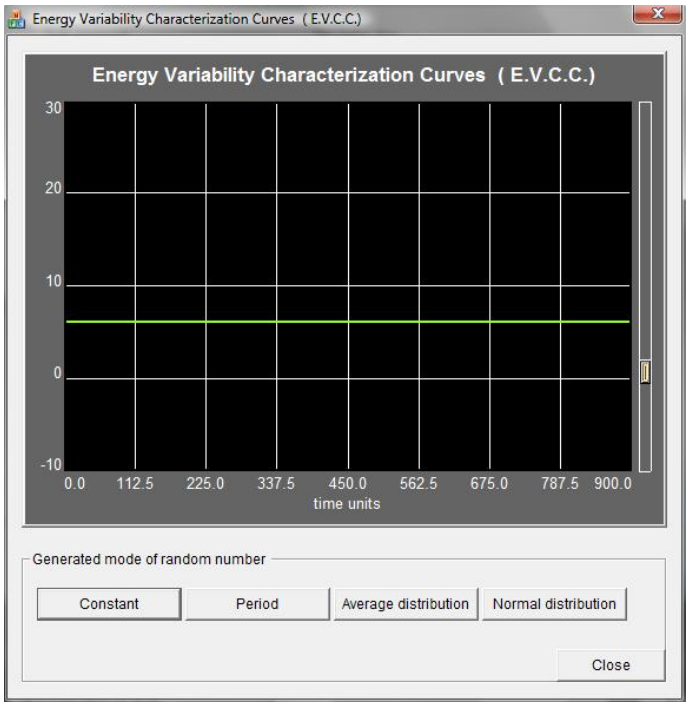


Figure 8.5: – Une source d'énergie constante

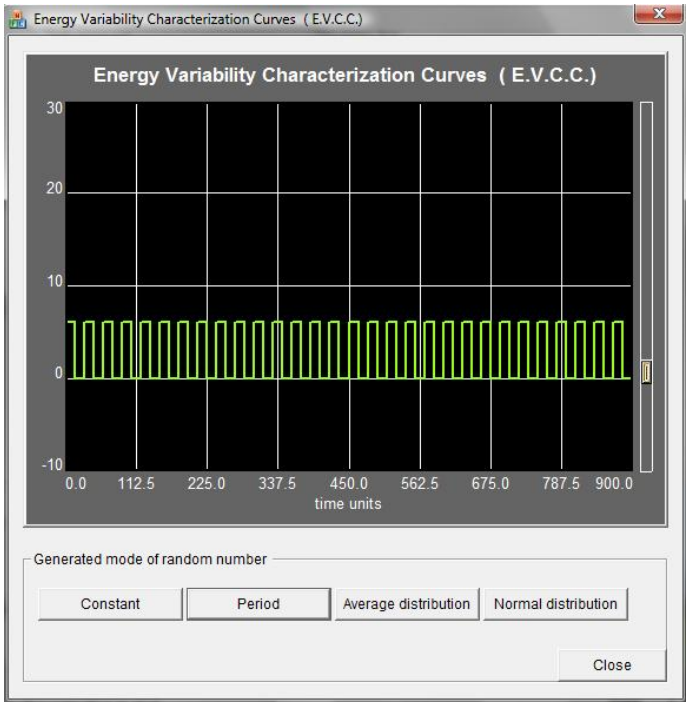


Figure 8.6: – Une source d'énergie périodique à petite période

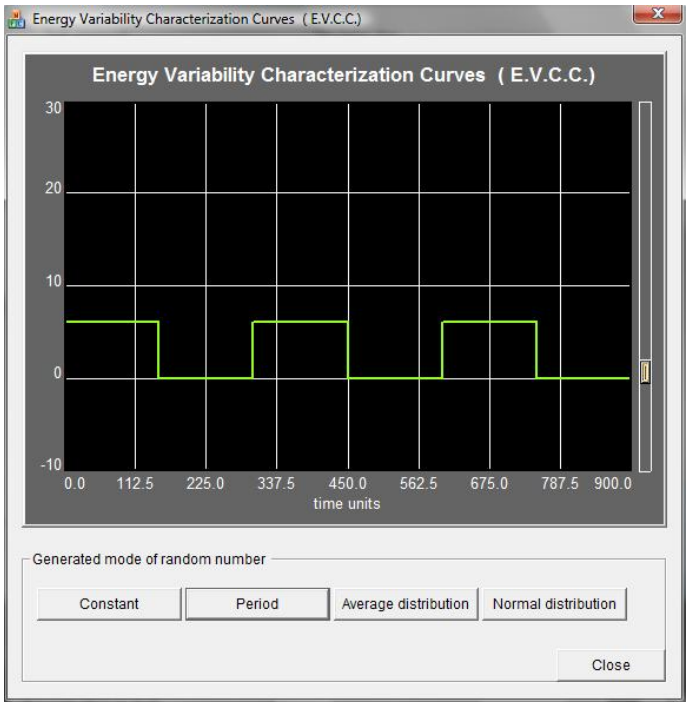


Figure 8.7: – Une source d'énergie périodique à grande période

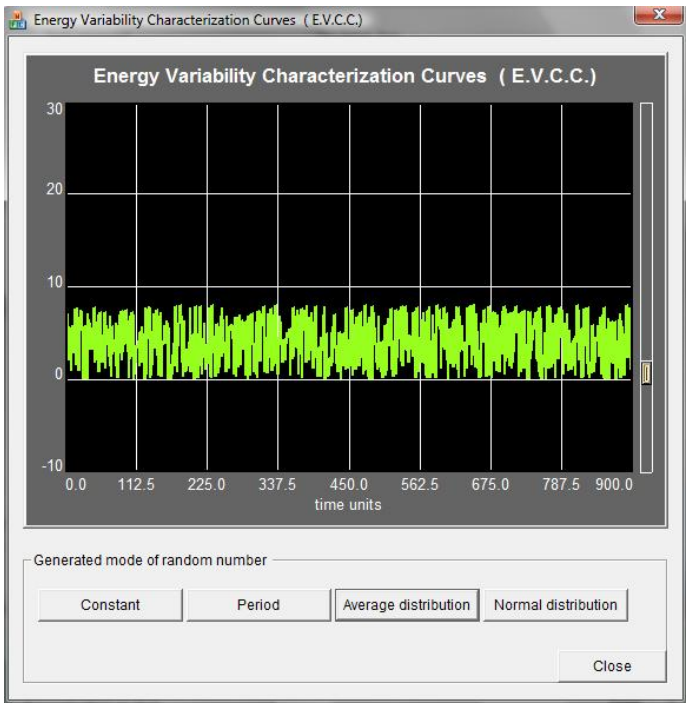


Figure 8.8: – Une source d'énergie en distribution uniforme

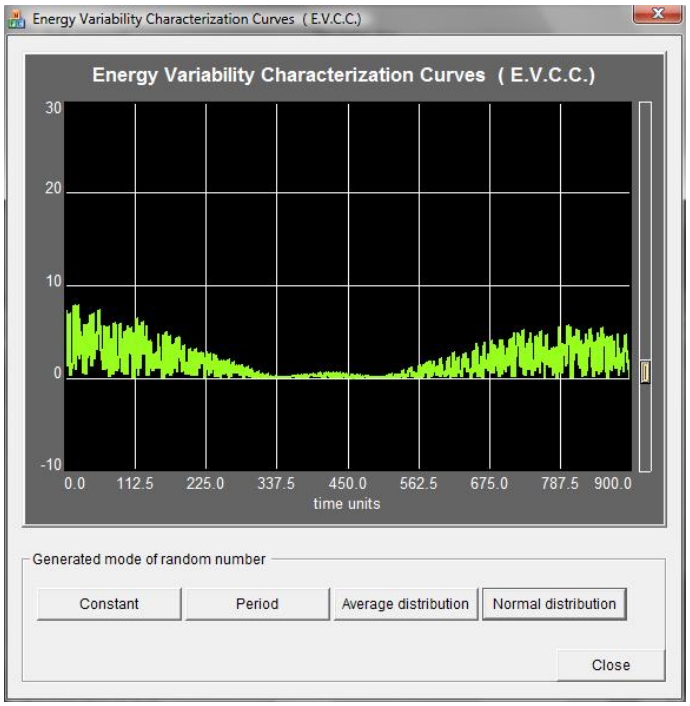


Figure 8.9: – Une source d'énergie en distribution normale

enregistrées dans un fichier *Excel*. Les paramètres suivants sont calculés ou surveillés dans le processus de simulation :

- Le taux d'échéances respectées ;
- L'énergie gaspillée pour cause de réservoir d'énergie plein ;
- L'énergie gaspillée pour cause d'échéances violées ;
- Le nombre d'épuisements du réservoir d'énergie ;
- Le surcoût d'exécution ;
- La date de début d'exécution de tâche s_i ;
- Le niveau d'énergie contenue dans le réservoir à un instant donné t $(E_c(t))$;
- La séquence produite d'exécution des tâches.

8.2.3 Instruction de l'outil de test

Pour utiliser cet outil, nous devons d'abord ouvrir le dialogue comme décrit sur la figure 8.3, et saisir les valeurs indispensables selon l'application à tester. Ensuite, en appuyant sur le bouton *Simulate*, une interface s'affiche sur l'écran comme indiqué sur la figure 8.4 où nous pouvons confirmer l'ordonnanceur sélectionné et les tâches générées automatiquement. Puis nous lançons le processus de génération E.V.C.C..

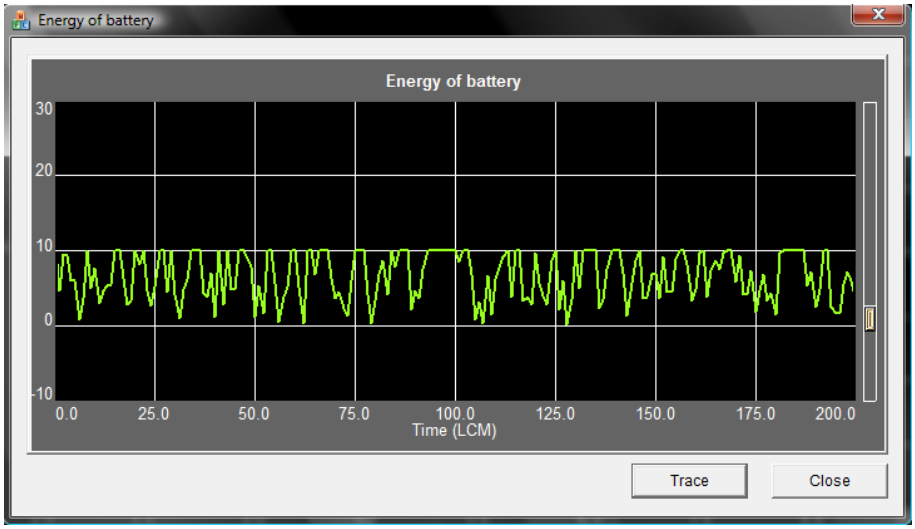


Figure 8.10: – Variation de l'énergie contenue du réservoir d'énergie

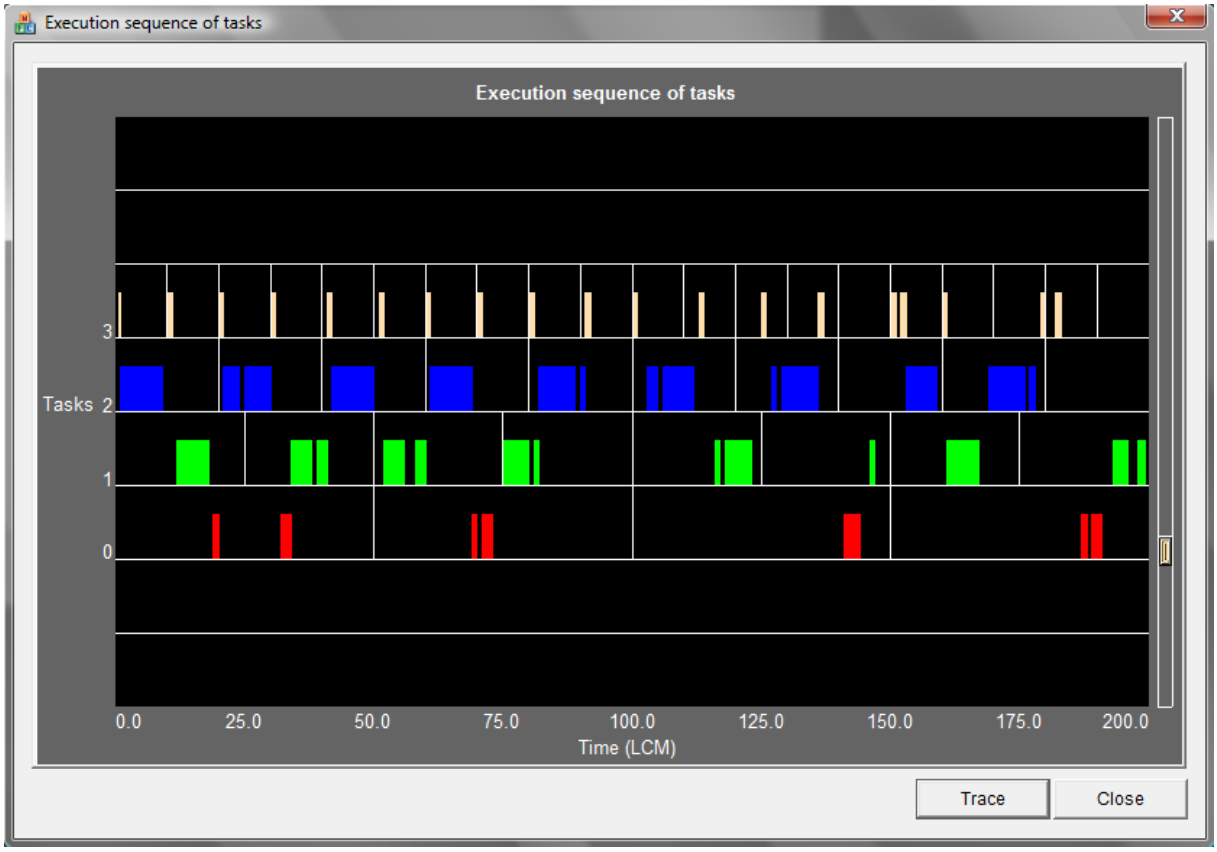


Figure 8.11: – Séquence produite d'exécution des tâches

8.3 Conclusion

Le test logiciel est une activité obligatoire, qui se fait normalement avant de commencer à implémenter l'ordonnanceur. Dans ce chapitre, nous avons présenté un outil de test pour vérifier l'ordonnansabilité d'une application temps réel et évaluer sa performance. Cet outil de test s'adresse principalement aux ingénieurs de projets, chercheurs scientifiques et éventuellement aux développeurs.

Actuellement, de nombreux outils commerciaux et universitaires existent et permettent de simuler une application temps réel soit sur une architecture monoprocesseur soit sur une architecture multiprocesseur. Ces outils offrent effectivement la possibilité de simuler des séquences d'ordonnancement sur des configurations de tâches.

La particularité de l'outil que nous avons conçu dans cette thèse est qu'il s'adresse à des systèmes temps réel bien plus sophistiqués car caractérisés non seulement par des valeurs temporelles mais aussi et surtout par des valeurs énergétiques. Celles ci portent respectivement sur le profil de la source d'énergie environnementale, la capacité du réservoir d'énergie ainsi que sur la consommation énergétique des tâches de l'application.

Conclusion générale et perspectives

Contexte

La technologie dite en anglais «Energy Harvesting» consiste à générer de l'énergie électrique en toute autonomie à partir de l'environnement. Cette technique ultramoderne puisque née durant cette dernière décennie est de plus en plus utilisée dans le domaine de la microélectronique et permet maintenant de concevoir des systèmes sans fil autonomes sur plusieurs années voire dizaines d'années. L'alimentation de ces derniers grâce à de l'énergie ambiante gratuite se fait par le biais de dispositifs qui exploitent des principes physiques connus (**objet du chapitre 2**). Cette technologie va devenir un atout incontournable pour le développement de demain des systèmes autonomes communicants (réseaux de capteurs sans fil), en ce qui concerne aussi bien les applications civiles (médecine, protection de l'environnement, etc.) que les applications de défense militaire (surveillance de zones ennemies, matériel embarqué sur le fantassin, etc.).

Ces nouveaux systèmes présentent par nature des contraintes temps réel car ils traitent et communiquent des informations, en particulier des données physiques issues de capteurs, ayant des impératifs temporels (échéances, périodes). L'ordonnancement est la problématique centrale des systèmes temps réel car il consiste à plannifier l'activité du processeur qui traite ces données avec pour objectif, le respect des contraintes temporelles (**objet du chapitre 1**). Jusqu'à maintenant, la recherche a principalement apporté des solutions aux applications traditionnelles qui ne possèdent pas ce type de contraintes énergétiques. La question majeure qui se pose dans ce nouveau contexte peut donc se résumer ainsi : Comment ordonnancer les tâches temps réel de façon à éviter une famine énergétique et que le plus grand nombre de tâches puissent respecter leur échéance, et ce avec un ordonnanceur de complexité raisonnable.

Contributions

L'objectif principal de la thèse a donc été dans une première étape d'introduire cette nouvelle problématique liée à la présence conjointe de contraintes d'ordre temporel et énergétique (**objet du chapitre 3**). Cela nous a amené à compléter la terminologie usuelle de l'ordonnancement temps réel par de nouvelles notions telles que *famine énergétique*, *surcharge énergétique*, etc. et des qualificatifs tels que *temporellement faisable*, etc. Nous avons repris un modèle de système introduit en 2006 composé d'un monoprocesseur, d'un réservoir d'énergie de capacité finie et d'une source d'énergie qui alimente ce réservoir en continu. Nous avons proposé d'une part une

condition nécessaire d'ordonnançabilité, et d'autre part une condition suffisante d'ordonnançabilité pour une configuration de tâches périodiques par un algorithme d'ordonnancement donné. Ce résultat est certainement le plus significatif : il montre que la séquence produite par un ordonnanceur sur une configuration de tâches périodiques possède certaines propriétés et qu'elle est en particulier périodique de période égale au PPCM des périodes et ce sous l'hypothèse où l'énergie est drainée de l'environnement avec une puissance identique au cours du temps.

Nous nous sommes ensuite focalisés sur l'ordonnanceur optimal LSA (**objet du chapitre 4**), introduit en 2006, qui consiste pour toute tâche réveillée à calculer sa date de démarrage optimale. Nous avons voulu faire une étude détaillée de cet ordonnanceur. Dans une première étape, nous avons mis en évidence sa complexité d'implémentation ainsi que la complexité de mise en oeuvre du test d'ordonnançabilité hors-ligne associé pour différents profils énergétiques d'applications. Nous pouvons résumer en établissant que tant que l'énergie est émise avec la même puissance au cours du temps, la complexité de LSA reste polynomiale et donc acceptable. Alors que si le profil énergétique est variable, la complexité de LSA dépend directement du nombre de valeurs utilisées pour mémoriser et caractériser ce profil. Nous avons aussi mis en évidence les hypothèses que nous pouvons considérer comme irréalistes sur lesquelles s'appuie la preuve d'optimalité de LSA. Ces hypothèses sont les suivantes : vitesse du processeur continuellement ajustable, puissance consommée nulle en mode veille, profil energétique connu avec précision, énergie consommée proportionnelle à la durée d'exécution, puissance consommée par le processeur à tout instant toujours supérieure à la puissance de la source renouvelable.

Les constats du chapitre 4 nous ont naturellement conduits à entreprendre une étude de simulation afin de comparer l'ordonnanceur théoriquement optimal à d'autres ordonnanceurs plus aisés d'implémentation. C'est pourquoi, nous avons proposé plusieurs heuristiques, toutes des variantes de EDF qui se différencient les unes des autres par la façon dont elles gèrent les situations de pénurie d'énergie c'est-à-dire lorsque le réservoir d'énergie devient vide et que le processeur est amené à s'arrêter. La question posée est alors : pendant combien de temps le processeur doit il rester en mode veille afin que le réservoir d'énergie se remplisse ?

Nous avons entrepris cette étude d'abord en supposant la puissance d'énergie environnementale constante (**objet du chapitre 5**). Nous avons reproduit l'étude de simulation précédente en supposant que l'énergie environnementale arrive de la source avec une puissance variable. Nous avons considéré successivement des fonctions puissance de nature périodique et obéissant à une loi normale (**objet du chapitre 6**). De ces études, ressort que l'heuristique non clairvoyante EDu qui consiste à recharger le réservoir pendant une unité de temps lorsque celui-ci devient vide, offre un bon compromis performance/surcoût.

Ayant noté l'overhead prohibitif lié à la mise en oeuvre de l'ordonnanceur LSA lorsque celui-ci opère avec une source d'énergie de puissance variable, nous avons proposé une version sous-optimale de LSA, appelé LSA approximé (**objet du chapitre 7**). Celle-ci va consister à approxi-

mer le profil énergétique par une fonction de type affine par exemple. La réduction de l'overhead se traduit naturellement par un calcul approximatif des dates de démarrage des tâches. Notre étude de simulation montre toutefois que cette approche nous offre un bon compromis performance/complexité.

Enfin, nous avons développé un outil logiciel qui permet de configurer une application en sélectionnant un profil énergétique, les caractéristiques du logiciel d'application et du réservoir d'énergie. L'outil simule les différents ordonnanceurs puis construit la séquence ainsi produite en représentant également l'évolution temporelle du niveau d'énergie dans le réservoir (**objet du chapitre 8**).

Travaux futurs

Extension à un modèle plus fidèle

Comme nous l'avons indiqué précèdemment, le modèle sur lequel s'est appuyé ce travail concerne le modèle utilisé dans la preuve de l'optimalité de l'ordonnanceur LSA. Supprimer une seule de ces hypothèses rend cet ordonnanceur sous-optimal. Il convient donc de poursuivre les recherches en étendant l'étude de simulation que nous avons reportée ici à un modèle moins restrictif et donc plus réaliste. Une des hypothèses qui nous a semblé la plus irréaliste dans le modèle étudié concerne le fait qu'à tout instant, lorsqu'une tâche s'exécute, elle consomme une puissance instantanée supérieure à la puissance de production environnementale. En d'autres termes, dés lors que le processeur est en activité, le niveau d'énergie dans le réservoir ne peut que déccroitre. Il faudrait donc compléter l'étude de simulation en supprimant cette hypothèse et évaluer l'impact sur la performance résultante de LSA. Il est à prévoir que la performance de l'heuristique EDu deviendrait tout à fait comparable à celle de LSA, celui-ci perdant alors son optimalité.

Applications distribuées

Une extension potentielle de notre travail serait d'étendre la problématique restreinte ici à une architecture monoprocesseur, aux applications distribuées. En effet, la technologie du energy harvesting intéresse particulièrement les réseaux de capteurs sans fil et les applications distribuées utilisant des processeurs reliés en réseau. Dans un tel réseau, chaque processeur dispose de son propre réservoir et de sa propre source d'énergie. Ainsi, par exemple dans une automobile, certains processeurs peuvent être alimentés par l'énergie récupérée gràce aux vibrations du moteur, au freinage, à des panneaux photovoltaiques, à des convertisseurs thermo-électriques, etc. Nous pouvons donc admettre qu'à certains moments, des processeurs aient un surplus d'énergie alors que d'autres soient en manque d'énergie. De façon à équilibrer la charge de traitement dans ces situations, il faudrait donc concevoir des méthodes qui dynamiquement, en fonction de l'énergie disponible réaffecte transitoirement l'exécution des tâches sur les processeurs.

Modèle intégrant la notion d'importance

La technologie du Energy Harvesting ne concerne pas les applications critiques où un manquement à une seule échéance pourrait engendrer une défaillance matérielle voire pire, une perte en vie humaine. Cette technologie concerne les applications temps réel fermes où nous souhaitons optimiser la qualité de service qui se mesure par le pourcentage d'échéances satifaites relatif au nombre total d'échéances à respecter. Notre étude a été réalisée dans ce contexte. Toutefois, cette étude ne prend pas en considération l'importance des tâches vis à vis de critères applicatifs. Dans une automobile, les tâches relatives au contrôle du moteur ou à la sécurité (par exemple, l'ABS) sont plus importantes que les tâches relatives au confort des passagers (par exemple, la gestion du chauffage de l'habitacle). En effet, il convient non seulement de minimiser le taux d'échéances violées mais en plus, de tenir compte du critère d'importance pour gérer les situations de surcharge énergétique et ainsi pouvoir choisir les tâches que nous pouvons supprimer temporairement et celles qu'il faut impérativement exécuter.

Liste des Publications effectuées au cours de la thèse

- Chetto, M. ; Hui Zhang ; **Issues in Real-Time Scheduling for Energy Harvesting Sensor Nodes** First International Conference on Wireless Information Networks & Information System (WINBIS 09), 27-28 Febuary, 2009
- Chetto, M. ; Hui Zhang ; **Performance Evaluation of Real-Time Scheduling Heuristics for Energy Harvesting Systems** 2010 IEEE/ACM Int'l Conference on Green Computing and Communications (GreenCom), pp 398 - 403, 18-20 Dec. 2010. (IEEE XPLORE)
- Chetto, M. ; Hui Zhang ; **A Simulation Tool for Real-time Systems using Environmental Energy Harvesting** 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010), Oct. 2010. (HAL INRIA)

Bibliographie

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks : a survey. *Computer networks*, 38(4) :393–422, 2002.

[2] A. Allavena and D. Mossé. Scheduling of frame-based embedded systems with rechargeable batteries. In *Workshop on power management for real-time and embedded systems (RTAS 2001)*, 2001.

[3] Y. Ammar. *Conception de systèmes de gestion d'énergie pour microsystèmes autonomes*. PhD thesis, Université Joseph Fourier, 2006.

[4] F. B. Types of batteries, and differences in energy and power, 2007.

[5] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, (2) :301–324, 1990.

[6] A. Burns. Scheduling hard real-time systems : a review. *Software Engineering Journal*, 6(3) :116–128, 1991.

[7] G. Buttazzo. *Hard real-time computing systems : predictable scheduling algorithms and applications*. Springer-Verlag New York Inc, 2005.

[8] Y. Chee, M. Koplow, M. Mark, N. Pletcher, M. Seeman, F. Burghardt, D. Steingart, J. Rabaey, P. Wright, and S. Sanders. PicoCube : a 1 cm3 sensor node powered by harvested energy. In *Proceedings of the 45th annual Design Automation Conference*, pages 114–119. ACM, 2008.

[9] H. Chetto and M. Chetto. Scheduling periodic and sporadic tasks in a real-time system. *Information Processing Letters*, 30(4) :177–184, 1989.

[10] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *Software Engineering, IEEE Transactions on*, 15(10) :1261–1269, 1989.

[11] M. Chetto and C. Plot. ASTORE : Autonomie des Systèmes embarqués par la Technologie de l'Ordonnancement temps réel et la Récupération de l'Énergie renouvelable. In *actes des 3èmes Journées Démonstrateurs 2010*, Angers, France, Nov. 2010.

[12] C. Corporation. Permanent power for wireless sensors, 2008. Doc WP-72-01 rev1.

[13] M. Dertouzos. Control robotics : the procedural control of physical processors. *Proceedings of the IFIP Congress*, 74 :807–813, 1974.

[14] J. Elloy. Système temps réel et informatique embarqué, 2009.

[15] L. George, P. Muhlethaler, and N. Rivierre. Optimality and non-preemptive real-time scheduling revisited. Technical Report RR-2516, INRIA, 1995. Le Chesnay Cedex, France.

[16] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Technical Report RR-2966, INRIA, 1996. Projet REFLECS.

[17] P. Glynne-Jones, M. Tudor, S. Beeby, and N. White. An electromagnetic, vibration-powered generator for intelligent sensor systems. *Sensors and Actuators A : Physical*, 110(1-3) :344–349, 2004.

[18] J. Jackson. Scheduling a production line to minimize maximum tardiness. Research Report 43. *Management Science Research Project, University of California, Los Angeles*, 1955.

[19] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 463–468. IEEE, 2005.

[20] J. Kymissis, C. Kendall, J. Paradiso, and N. Gershenfeld. Parasitic power harvesting in shoes. In *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*, pages 132–139. IEEE, 1998.

[21] K. Lahiri, A. Raghunathan, S. Dey, and D. Panigrahi. Battery-driven system design : A new frontier in low power design. In *Design Automation Conference, 2002. Proceedings of ASP-DAC 2002. 7th Asia and South Pacific and the 15th International Conference on VLSI Design. Proceedings.*, pages 261–267. IEEE, 2002.

[22] J. Lee, S. Yuen, W. Li, and P. Leong. Development of an AA size energy transducer with micro resonators. In *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on*, volume 4, pages 876–879. IEEE, 2003.

[23] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm : Exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171. IEEE, 2002.

[24] J. Leung and M. Merril. A note on preemprive scheduling of periodic real-time tasks. *Information Processing Letters*, 11(3) :115–118, 1980.

[25] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4) :237–250, 1982.

[26] K. Lin, J. Yu, J. Hsu, S. Zahedi, D. Lee, J. Friedman, A. Kansal, V. Raghunathan, and M. Srivastava. Heliomote : enabling long-lived sensor networks through solar energy harvesting. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 309–309. ACM, 2005.

[27] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1) :46–61, 1973.

[28] S. Liu, Q. Qiu, and Q. Wu. Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting. In *Proceedings of the conference on Design, automation and test in Europe*, pages 236–241. ACM, 2008.

[29] A. Marchand. *Ordonnancement temps-réel avec contraintes de qualité de service.* PhD thesis, Université de Nantes, 2006.

[30] P. Martineau. *Ordonnancement en-ligne dans les systémes informatiques temps-réel.* PhD thesis, Université de Nantes, 1994.

[31] Micropelt. Datasheet : Thin film thermogenerators and sensing devices. Micropelt GmbH, Germany.

[32] C. Moser, D. Brunelli, L. Thiele, and L. Benini. Lazy scheduling for energy harvesting sensor nodes. *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, pages 125–134, 2006.

[33] C. Moser, D. Brunelli, L. Thiele, and L. Benini. Real-time scheduling with regenerative energy. In *Real-Time Systems, 2006. 18th Euromicro Conference on.* IEEE, 2006.

[34] C. Moser, D. Brunelli, L. Thiele, and L. Benini. Real-time scheduling for energy harvesting sensor nodes. *Real-Time Systems*, 37(3) :233–260, 2007.

[35] T. O'Donnel, C. Saha, S. Beeby, and J. Tudor. Scaling effects for electromagnetic vibrational power generators. *Microsystem Technologies*, 13(11) :1647–1653, 2007.

[36] G. Ottman, H. Hofmann, A. Bhatt, and G. Lesieutre. Adaptive piezoelectric energy harvesting circuit for wireless remote power supply. *Power Electronics, IEEE Transactions on*, 17(5) :669–676, 2002.

[37] J. Paradiso and T. Starner. Energy scavenging for mobile and wireless electronics. *Pervasive Computing, IEEE*, 4(1) :18–27, 2005.

[38] V. Pop, R. de Francisco, H. Pflug, J. Santana, H. Visser, R. Vullers, H. de Groot, and B. Gyselinckx. Human++ : Wireless autonomous sensor technology for body area networks . In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, pages 561–566. IEEE, 2011.

[39] J. Rabaey, F. Burghardt, D. Steingart, M. Seeman, and P. Wright. Energy Harvesting - A Systems Perspective. In *Electron Devices Meeting, 2007. IEDM 2007. IEEE International.*, pages 363–366. IEEE, December 2007.

[40] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 457–462. IEEE, 2005.

[41] J. Randall, N. Bharatula, N. Perera, T. v. Büren, S. Ossevoort, and G. Tröster. Indoor Tracking Using solar cell powered system : Interpolation of Irradiance. In *The 6th Internaional Conference on Ubiquitous Computing*, pages 7–10. ACM, 2004.

[42] F. Ren, H. Huang, and C. Lin. Wireless sensor networks. *Journal of Software*, 14(7) :1282–1291, 2003.

[43] F. Ridouard, P. Richard, and F. Cottet. Ordonnancement de tâches indépendantes avec suspension. *13th INTERNATIONAL CONFERENCE ON REAL-TIME SYSTEMS (RTS)*, 2005.

[44] S. Roundy, P. Wright, and J. Rabaey. A study of low level vibrations as a power source for wireless sensor nodes. *Computer Communications*, 26(11) :1131–1144, 2003.

[45] D. Sadoway and A. Mayes. Portable power : Advanced rechargeable lithium batteries. *MRS bulletin*, 27(8) :590–596, 2002.

[46] N. Shenck. A demonstration of useful electric energy generation from piezoceramics in a shoe. Technical report, Massachusetts Institute of Technology, 1999.

[47] A. Sinha and A. Chandrakasan. Dynamic power management in wireless sensor networks. *IEEE Design & Test of Computers*, 18(2) :62–74, 2001.

[48] P. Sorenson. *A methodology for real-time system developement.* PhD thesis, University of Toronto, Canada, 1974.

[49] J. Stankovic and K. Ramamritham. *Tutorial : hard real-time systems.* IEEE Computer Society Press Los Alamitos, CA, USA, 1989.

[50] I. Stark. Thermal energy harvesting with Thermo Life. In *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*, pages 19–22. IEEE, 2006.

[51] T. Starner. Human-powered wearable computing. *IBM SYSTEMS JOURNAL*, 35(3) :618–629, 1996.

[52] T. Starner and J. Paradiso. Human generated power for mobile electronics. *Low-Power Electronics Design*, 1990 :1–35, 2004.

[53] G. Waltisperger. *Architectures intégrées de gestion de l'énergie pour les microsystémes autonomes.* PhD thesis, Université de Grenoble, 2011.

[54] N. White, P. Glynne-Jones, and S. Beeby. A novel thick-film piezoelectric micro-generator. *Smart Materials and Structures*, 10 :850–852, 2001.

Annexe A

Pseudo-codes des heuristiques d'ordonnancement

A.1 Pseudo-code de l'algorithme EDt

Le pseudo-code de l'algorithme d'ordonnancement EDt est fourni ci-dessous dans le tableau A.1. Il représente l'ensemble des opérations mises en oeuvre pour construire en-ligne la séquence d'ordonnancement dans chaque cycle.

Données : une liste à jour des tâches prêtes à s'exécuter, notée $\mathscr{L}$.
Résultats : une séquence d'ordonnancement.

Tant que (la liste $\mathscr{L}$ non vide) **faire**
  $t \leftarrow$ instant courant ;
  $d_j \leftarrow min\{d_i : \tau_i \in \mathscr{L}\}$;
  Si $(E_c(t) + E_r(t, d_j) \geq E_j)$ **Alors**
    exécuter la tâche τ_j avec la puissance P_{max} ;
  Sinon
    mettre le processeur en mode veille, $E_c(t) \leftarrow E_c(t) + P_r(t)$;
  Fin Si
  Si $(t = a_k)$ **Alors**
    ajouter la tâche τ_k dans la liste $\mathscr{L}$;
  Fin Si
  Si $(t = f_j)$ **Alors**
    supprimer la tâche τ_j de la liste $\mathscr{L}$;
  Fin Si
  $t \leftarrow t + 1$;
Fait

Tableau A.1: – Pseudo-code de l'algorithme EDt

A.2 Pseudo-code de l'algorithme EDi

Le pseudo-code de l'algorithme d'ordonnancement EDi est fourni ci-dessous dans le tableau A.2.

Données : une liste à jour des tâches prêtes à s'exécuter, notée $\mathscr{L}$.
Résultats : une séquence d'ordonnancement.

Tant que (la liste $\mathscr{L}$ non vide) **faire**
  $t \leftarrow$ instant courant ;
  $d_j \leftarrow min\{d_i : \tau_i \in \mathscr{L}\}$;
  Si $(E_c(t) > 0)$ **Alors**
    exécuter la tâche τ_j avec la puissance P_{max} ;
    Si $(t = a_k)$ **Alors**
      ajouter la tâche τ_k dans la liste $\mathscr{L}$;
    Fin Si
    Si $(t = f_j)$ **Alors**
      supprimer la tâche τ_j de la liste $\mathscr{L}$;
    Fin Si
    $t \leftarrow t + 1$;
  Sinon
    $a_{j+1} \leftarrow min\{a_i : \tau_i \in \mathscr{L}\ et\ a_i > t\}$;
    Tant que (t $< a_{j+1}$) **faire**
      mettre le processeur en mode veille, $E_c(t) \leftarrow E_c(t) + P_r(t)$;
      Si $(t = a_k)$ **Alors**
        ajouter la tâche τ_k dans la liste $\mathscr{L}$;
      Fin Si
      Si $(t = f_j)$ **Alors**
        supprimer la tâche τ_j de la liste $\mathscr{L}$;
      Fin Si
      $t \leftarrow t + 1$;
      $a_{j+1} \leftarrow min\{a_i : \tau_i \in \mathscr{L}\ et\ a_i > t\}$;
    Fait
  Fin Si
Fait

Tableau A.2: – Pseudo-code de l'algorithme EDi

A.3 Pseudo-code de l'algorithme EDd

Le pseudo-code de l'algorithme d'ordonnancement EDd est fourni ci-dessous dans le tableau A.3.

Données : une liste à jour des tâches prêtes à s'exécuter, notée $\mathscr{L}$.
Résultats : une séquence d'ordonnancement.

Tant que (la liste $\mathscr{L}$ non vide) **faire**
  $t \leftarrow$ instant courant ;
  $d_j \leftarrow min\{d_i : \tau_i \in \mathscr{L}\}$;
  Si $(E_c(t) > 0)$ **Alors**
    exécuter la tâche τ_j avec la puissance P_{max} ;
    Si $(t = a_k)$ **Alors**
      ajouter la tâche τ_k dans la liste $\mathscr{L}$;
    Fin Si
    Si $(t = f_j)$ **Alors**
      supprimer la tâche τ_j de la liste $\mathscr{L}$;
    Fin Si
    $t \leftarrow t + 1$;
  Sinon
    supprimer toutes les tâches réveillées de la liste $\mathscr{L}$;
    $a_{j+1} \leftarrow min\{a_i : \tau_i \in \mathscr{L}\ et\ a_i > t\}$;
    Tant que (t $< a_{j+1}$) **faire**
      mettre le processeur en mode veille, $E_c(t) \leftarrow E_c(t) + P_r(t)$;
      Si $(t = a_k)$ **Alors**
        ajouter la tâche τ_k dans la liste $\mathscr{L}$;
      Fin Si
      Si $(t = f_j)$ **Alors**
        supprimer la tâche τ_j de la liste $\mathscr{L}$;
      Fin Si
      $t \leftarrow t + 1$;
      $a_{j+1} \leftarrow min\{a_i : \tau_i \in \mathscr{L}\ et\ a_i > t\}$;
    Fait
  Fin Si
Fait

Tableau A.3: – Pseudo-code de l'algorithme EDd

A.4 Pseudo-code de l'algorithme EDu

Le pseudo-code de l'algorithme d'ordonnancement EDu est fourni ci-dessous dans le tableau A.4.

Données : une liste à jour des tâches prêtes à s'exécuter, notée $\mathscr{L}$.
Résultats : une séquence d'ordonnancement.

Tant que (la liste $\mathscr{L}$ non vide) **faire**
  $t \leftarrow$ instant courant ;
  $d_j \leftarrow min\{d_i : \tau_i \in \mathscr{L}\}$;
  Si $(E_c(t) > 0)$ **Alors**
    exécuter la tâche τ_j avec la puissance P_{max} ;
  Sinon
    mettre le processeur en mode veille, $E_c(t) \leftarrow E_c(t) + P_r(t)$;
  Fin Si
  Si $(t = a_k)$ **Alors**
    ajouter la tâche τ_k dans la liste $\mathscr{L}$;
  Fin Si
  Si $(t = f_j)$ **Alors**
    supprimer la tâche τ_j de la liste $\mathscr{L}$;
  Fin Si
  $t \leftarrow t + 1$;
Fait

Tableau A.4: – Pseudo-code de l'algorithme EDu

A.5 Pseudo-code de l'algorithme EDc

Le pseudo-code de l'algorithme d'ordonnancement EDc est fourni ci-dessous dans le tableau A.5.

Données : une liste à jour des tâches prêtes à s'exécuter, notée $\mathscr{L}$.
Résultats : une séquence d'ordonnancement.

Tant que (la liste $\mathscr{L}$ non vide) **faire**
  $t \leftarrow$ instant courant ;
  $d_j \leftarrow min\{d_i : \tau_i \in \mathscr{L}\}$;
  Si $(E_c(t) > 0)$ **Alors**
    exécuter la tâche τ_j avec la puissance P_{max} ;
    Si $(t = a_k)$ **Alors**
      ajouter la tâche τ_k dans la liste $\mathscr{L}$;
    Fin Si
    Si $(t = f_j)$ **Alors**
      supprimer la tâche τ_j de la liste $\mathscr{L}$;
    Fin Si
    $t \leftarrow t + 1$;
  Sinon
    supprimer la tâche courante de la liste $\mathscr{L}$;
    $a_{j+1} \leftarrow min\{a_i : \tau_i \in \mathscr{L}\ et\ a_i > t\}$;
    Tant que (t $< a_{j+1}$) **faire**
      mettre le processeur en mode veille, $E_c(t) \leftarrow E_c(t) + P_r(t)$;
      Si $(t = a_k)$ **Alors**
        ajouter la tâche τ_k dans la liste $\mathscr{L}$;
      Fin Si
      Si $(t = f_j)$ **Alors**
        supprimer la tâche τ_j de la liste $\mathscr{L}$;
      Fin Si
      $t \leftarrow t + 1$;
      $a_{j+1} \leftarrow min\{a_i : \tau_i \in \mathscr{L}\ et\ a_i > t\}$;
    Fait
  Fin Si
Fait

Tableau A.5: – Pseudo-code de l'algorithme EDc