Ordonnancement temps réel pour l'optimisation de la Qualité de Service dans les systèmes autonomes en énergie

Maissa Abdallah

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Maïssa ABDALLAH

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le label de l'Université de Nantes Angers Le Mans

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Automatique et Informatique Appliquée, section CNU 61
Spécialité : Informatique
Unité de recherche : Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN)

Soutenue le 18 Juin 2014

Ordonnancement temps réel pour l'optimisation de la Qualité de Service dans les systèmes autonomes en énergie

JURY

Rapporteurs :	**M^me^ Annie GENIET**, Professeur des Universités, Université de Poitiers
	M. Zoubir MAMMERI, Professeur des Universités, Université de Toulouse III
Examinateurs :	**M^me^ Samia BOUZEFRANE**, Maitre de Conférences HDR, CNAM Paris
	M. Daniel SIMON, Chargé de Recherches HDR, INRIA, Montpellier
	M^me^ Audrey QUEUDET, Maître de Conférences, Université de Nantes
	M. Rafic HAGE, Professeur, Université Libanaise
Directrice de thèse :	**M^me^ Maryline CHETTO**, Professeur des Universités, Université de Nantes

Table des matières

Table des figures

Liste des tableaux

Introduction générale

De nos jours, les applications temps réel embarquées prennent de plus en plus d'ampleur dans notre vie quotidienne. Elles sont de plus en plus variées et apparaissent dans des secteurs extrêmement divers tels que le transport (avionique, automobile, bus,..), le multimédia, les téléphones portables, les consoles de jeux, etc.. Depuis les années soixante-dix, les enjeux économiques et les intérêts scientifiques font que l'on s'intéresse de plus en plus aux algorithmes, aux langages, aux protocoles de communication, etc., pour le temps réel et l'embarqué. Une grande partie des systèmes embarqués ont des besoins d'autonomie et des limitations d'ordre spatial (encombrement réduit) et énergétique (consommation restreinte). De ce fait, le défi technologique et scientifique majeur est de construire des systèmes de confiance du point de vue des fonctionnalités assurées et de la qualité de service rendue. Il s'agit de plus de concevoir ces systèmes à coût acceptable.

Ayant largement puisé dans les réserves d'énergies fossiles (pétrole, charbon,..), l'utilisation des énergies renouvelables (solaire, éolienne,..) se révèle être une alternative de choix pour alimenter bon nombre de systèmes. Cependant l'utilisation de cette énergie pour alimenter un système embarqué induit un certain nombre de problèmes liés aux caractéristiques de l'informatique et de l'électronique embarquées. En effet la singularité d'un système embarqué tient dans son fonctionnement en temps réel : celui-ci est soumis à des contraintes temporelles attachées à la réalisation des différentes activités qu'il doit mettre en oeuvre et qui consomment de l'énergie. Dans des applications dites critiques, non seulement le non respect de ces contraintes temporelles peut influer sur la qualité de service rendue mais il peut dans certaines applications critiques être inacceptable car engendrant l'arrêt définitif du système avec en outre une dégradation du matériel voire la perte de vies humaines. Lorsqu'un système embarqué n'a aucune contrainte énergétique, le problème majeur à résoudre lors de sa conception consistera à vérifier la faisabilité de l'application au regard des spécifications temporelles, des capacités de traitement de l'architecture matérielle utilisée et des besoins en temps de traitement des logiciels d'application. Cette problématique est bien connue, étudiée depuis une quarantaine d'années. Le problème sous-jacent est un problème d'ordonnancement classique où seule la dimension "temps" intervient et se ramène à optimiser une grandeur appelée Qualité de Service (QoS). Celle-ci se définit classiquement par le ratio de tâches temps réel qui terminent leur exécution dans le respect de leur échéance. Jusqu'à très récemment, on supposait que l'énergie ne constituait pas une contrainte. Celle-ci était supposée disponible en quantité suffisante pour assurer le fonctionnement du système sur toute la durée de vie de l'application. Les études précédentes ne peuvent plus convenir pour un système embarqué alimenté par une source d'énergie renouvelable et utilisant une batterie de taille réduite dont le niveau fluctue au cours du temps selon la quantité d'énergie récupérée.

Dans le cadre de cette thèse, nous nous intéressons donc au problème de l'ordonnancement temps réel sous contraintes de Qualité de Service (QoS) et d'énergie. Il s'agit de considérer des tâches temps réel qui ont des besoins qui s'expriment d'une part en termes de temps de traitement sur le processeur et d'autre part en termes d'énergie consommée. Une configuration de tâches peut être en surcharge énergétique (c'est-à-dire que la quantité d'énergie consommée est supérieure à la quantité d'énergie disponible) et/ou en surcharge de traitement (i.e. la quantité de traitement demandée est supérieure à la capacité de traitement disponible). Un système temps réel surchargé sera donc typiquement dans l'impossibilité de satisfaire ses échéances. Une question majeure à laquelle il faut pouvoir répondre est donc : *comment ordonnancer les tâches temps réel en cas de surcharge de telle sorte que l'on optimise la qualité de service, que cette surcharge soit d'origine temporelle ou énergétique ?*.

Dans cette thèse, nous nous basons sur le modèle connu sous le nom Skip-Over pour préciser sous quelles conditions les tâches sont autorisées à violer leur échéance. Chaque tâche est alors munie d'un facteur de pertes. Le travail de thèse a pour objectif de concevoir et valider par la simulation des stratégies d'ordonnancement de

tâches périodiques et de tâches apériodiques capables de gérer des surcharges de traitement et des surcharges énergétiques avec comme unique critère de performance la Qualité de Service.

Le chapitre 1 permet d'introduire les concepts et algorithmes concernant l'ordonnancement dans des systèmes temps réel fermes dans un premier temps. Puis dans un deuxième temps, il traite de l'ordonnancement temps réel dans le cas de surcharge temporelle.

Le deuxième chapitre traite de l'ordonnancement temps réel sous contraintes énergétiques. Dans un premier temps, une description des systèmes embarqués puis des réseaux de capteurs sans fil est faite. Puis un état de l'art de la récupération et du stockage d'énergie est donné. Enfin, nous citons quelques techniques existantes d'ordonnancement de tâches périodiques temps réel sous contraintes temporelles et énergétiques.

Le chapitre 3 décrit le modèle de système étudié dans le cadre de cette thèse, les algorithmes RTO et BWP du modèle Skip-Over ainsi que l'heuristique EDeg qui constituent des éléments sur lesquels nous nous appuyons tout au long de nos travaux. Ensuite, les algorithmes proposés dans le cadre de cette thèse, *Green-RTO* et *Green-BWP* ainsi que les conditions d'ordonnançabilité sont décrits. Enfin une étude comparative est faite par le biais de simulations dans le but d'évaluer les performances de Green-RTO, Green-BWP et EDeg.

Dans le chapitre 4, nous nous intéressons à la robustesse et la stabilité de Green-BWP dans le cas de surcharge temporelle et/ou énergétique. Nous proposons deux nouveaux algorithmes d'ordonnancement pour Green-BWP, qui améliorent la robustesse et la stabilité du système.

Le chapitre 5 traite du problème de l'ordonnancement des tâches apériodiques non critiques conjointement à des tâches périodiques temps réel strictes. L'objectif consiste à minimiser le temps de réponse des tâches apériodiques. Dans un premier temps nous proposons de nouveaux serveurs de tâches apériodiques appelés BG-EDeg, TBS-EDeg et TBS_{op}-EDeg qui se basent sur les serveurs bien connus BG et TBS. Ensuite, une étude comparative est faite pour évaluer les performances des différents algorithmes proposés.

Dans le chapitre 6, nous considérons des tâches périodiques temps réel fermes. L'objectif étant d'exploiter les pertes autorisées au niveau des tâches périodiques dans le but de minimiser le temps de réponse des tâches apériodiques non critiques. Dans un premier temps, nous proposons de nouveaux serveurs de tâches apériodiques qui se basent sur les algorithmes Green-RTO et Green-BWP et les serveurs bien connus BG et TBS. Dans un deuxième temps, nous rapportons les résultats d'une étude comparative des performances des différents algorithmes proposés.

Enfin dans la conclusion, nous résumons les principales contributions et présentons les axes de développements futurs.

Première partie

Rappels et Etat de l'art

Chapitre 1

Introduction à l'ordonnancement temps réel

Sommaire

Ce premier chapitre introduit les systèmes qualifiés de "temps réel". Dans un premier temps, nous rappelons les principaux concepts et spécificités des systèmes temps réel. Puis, nous nous focalisons sur les algorithmes d'ordonnancement de tâches périodiques et apériodiques soumis à des contraintes temporelles. Ensuite, nous exposons le problème de surcharge temporelle et nous présentons des approches adaptées à la résolution de cette problématique.

1 Les systèmes temps réel

1.1 Notion de temps réel

La notion de temps réel a été définie à de nombreuses reprises dans la littérature scientifique. La définition la plus reprise a été définie en 1988 par le chercheur américain Stankovic [Sta88] : *"En informatique temps réel, le comportement correct d'un système dépend non seulement des résultats logiques des traitements, mais aussi du temps auquel les résultats sont produits"*. En d'autres termes, les contraintes temporelles à respecter font partie de la spécification du système considéré et elles sont relatives à un temps physique mesurable. De plus,

le respect des contraintes temporelles est aussi pertinent que l'exactitude logique des résultats fournis par le système.

Ainsi, un système temps réel consiste en une association logiciel/matériel qui se différencie des autres systèmes par la prise en considération des délais de traitement des informations. Un système d'exploitation qualifié de temps réel et en particulier son noyau intégrant l'ordonnanceur, doit permettre entre autres, une gestion adéquate des ressources matérielles de façon à exécuter certaines tâches ou fonctions dans des limites temporelles définies par les spécifications applicatives.

La grande majorité des systèmes temps réel sont physiquement intégrés à l'environnement qu'ils contrôlent. C'est pourquoi les qualificatifs "embarqué" et "temps réel" sont complémentaires.

1.2 Qualité de service et temps réel

Un système temps réel est donc un système réactif soumis à des contraintes temporelles : il réagit de façon continue avec l'environnement ou le procédé depuis lequel il reçoit des informations et sur lequel il exécute ses commandes. L'information à traiter doit rester pertinente après son acquisition et son traitement, sans quoi le système est considéré comme défaillant. C'est pourquoi, la Qualité de Service (QoS) a une signification très spécifique en informatique temps réel.

La notion de qualité de service est employée dans de nombreux domaines. Une définition générale de la QoS est proposée par l'ISO [ISO95] :"*La qualité de service désigne un ensemble d'exigences de qualité sur le comportement collectif d'un ou de plusieurs objets*". En informatique, la notion de qualité de service est largement utilisée notamment dans les réseaux et le multimédia.

Concernant les systèmes temps réel, le terme QoS a trait au respect des contraintes temporelles. Ainsi la QoS pourrait par exemple être mesurée par le taux de requêtes ayant respecté leur échéance, le taux de requêtes ayant manqué leur échéance, le temps de réponse d'une requête, etc. Par requête, on entend ici un job de tâche périodique ou une tâche apériodique. Dans le cadre de nos travaux, la QoS sera définie comme suit :

Définition 1.1 *La qualité de service d'une application temps réel correspond au taux de requêtes ayant été exécutées dans le respect de leur échéance.*

Suivant le niveau de QoS minimum exigé, un système temps réel est considéré comme temps réel dur (*hard real-time* en anglais), temps réel souple (*soft real-time* en anglais) ou temps réel ferme (*firm real-time* en anglais) :

- Un système temps réel dur [Liu00] est celui dans lequel les traitements doivent absolument respecter toutes les contraintes temporelles. Dans ce type de système, rien qu'un seul manquement d'échéance peut engendrer des conséquences graves sur l'environnement contrôlé voire aussi des pertes économiques et des pertes de vie humaines considérées alors comme inacceptables. Les applications concernées sont notamment celles du secteur nucléaire, de la supervision médicale, de l'aérospatial, etc. La Qualité de service fournie doit donc être de 100% et égale à la QoS spécifiée au moment de la conception.

- Un système temps réel souple [Liu00] est celui dans lequel le non respect des contraintes temporelles n'aura aucun effet catastrophique sur l'environnement contrôlé. Cependant, le fait de manquer des échéances va engendrer une dégradation du service rendu sans remettre en péril la poursuite de l'application. Cela concerne les systèmes de projections vidéo dans lesquels des dérives de synchronisation du son et des images ne feront que dégrader momentanément la qualité de la vidéo restituée.

- Un système temps réel ferme [BBL01] représente un intermédiaire vis-à-vis des deux systèmes cités précédemment. Ce type de système temps réel tolère un nombre limité de violations d'échéances au-delà duquel le système est considéré comme défaillant. En effet, on autorise qu'un nombre précis de traitements ne respectent pas leur contrainte temporelle. Néanmoins, ceci a pour effet une dégradation de la QoS. On retrouve ce genre de systèmes dans les applications de contrôle/commande où une instabilité temporaire du système contrôlé est acceptée. On tolère donc que certaines commandes soient émises avec retard sur les actionneurs.

1.3 Les tâches temps réel

1.3.1 Caractérisation d'une tâche temps réel

Une application temps réel se définit par l'exécution de traitements spécifiques qui requièrent des besoins en ressources logiques ou matérielles. En outre elle peut être soumise à des contraintes (autres que les contraintes temporelles) liées aux spécifications du système. Parmi celles-ci, nous pouvons citer [SC93] :

- Les contraintes de ressources : le système d'exploitation a pour rôle d'arbitrer entre les demandes des tâches de l'application et les ressources effectivement disponibles dans le système. Les tâches partagent des ressources critiques en exclusion mutuelle.
- Les contraintes de synchronisation : des tâches peuvent avoir entre elles des relations de précédence. Ainsi s'il existe une relation de précédence qui impose un ordre dans lequel les tâches doivent s'exécuter, on dit que les tâches sont dépendantes. Sinon elles sont dites indépendantes.
- Les contraintes d'exécution : Deux modes d'exécution existent : le mode préemptif et le mode non-préemptif. Dans le mode préemptif, une tâche (dite préemptable) peut être interrompue à un instant quelconque et être reprise ultérieurement. Contrairement à ce comportement, dans le cas non-préemptif, une tâche s'exécute complètement sans interruption jusqu'à sa terminaison.
- Les contraintes de localité : Les tâches doivent s'exécuter sur un seul processeur sans possibilité de migrer.
- Les contraintes d'anti-localité : Les tâches doivent s'exécuter sur des processeurs différents.

Une application est donc structurée en tâches. Une tâche temps réel est ainsi l'entité logicielle de base qui donne lieu à l'exécution d'une ou plusieurs requêtes. Elle représente une succession de traitements ou d'opérations à exécuter sur le processeur de façon répétitive ou une seule fois selon que cette tâche est périodique ou non. Un système temps réel est un système multitâche puisqu'à tout instant plusieurs tâches peuvent demander à s'exécuter. Elles sont donc concurrentes pour l'accès au processeur, la gestion de cette concurrence étant assurée par l'ordonnanceur. A tout instant, une tâche se voit attacher un état selon qu'elle a terminé ou non son exécution, selon qu'elle a reçu ou non son signal de réveil l'autorisant à commencer son exécution. Il existe trois états principaux pour une tâche : *prêt* (ready en anglais), *actif* (running en anglais), *en attente* (waiting en anglais) [AP91]. Ces états ainsi que les transitions sont illustrés sur la Figure 1.1.

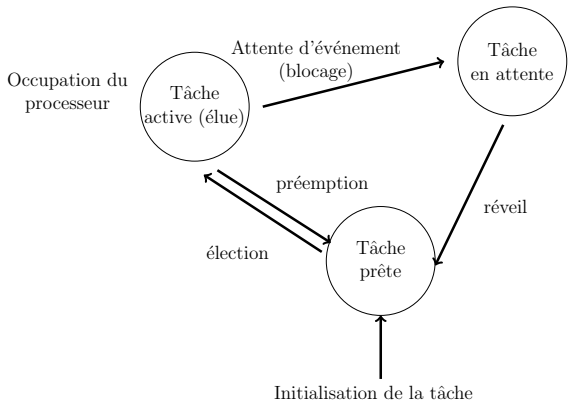


FIGURE 1.1 – Diagrammes des états d'une tâche

On dit que la tâche est :

- Active (*running*) : si la tâche qui est en train de s'exécuter.
- Prête (*ready*) : si la tâche est en attente de pouvoir s'exécuter. La priorité des tâches prêtes est inférieure ou égale à la priorité de la tâche active.
- En attente (*waiting*) : c'est le cas des tâches qui sont en attente d'événements qui provoqueront leur réveil (interruption, réception de message ou fin d'un décompte de timer).

Chaque occurence d'une tâche est appelé *job*. Chaque job (aussi appelé instance ou requête) est exécuté sur le processeur à un instant donné et sur une certaine durée. Les jobs d'une tâche peuvent être executés par le processeur à intervalles de temps réguliers ou non. Trois modèles de tâches sont couramment rencontrés dans la littérature. Selon le type de récurrence des jobs d'une tâche, celle-ci sera une tâche périodique, sporadique ou apériodique.

1.3.2 Modèle de tâches périodiques

Les tâches temps réel périodiques représentent les tâches dont les jobs sont récurrents avec un intervalle de récurrence constant appelé période. Plusieurs modèles de tâches périodiques ont été décrits et étudiés dans la littérature. Le plus simple et le plus étudié est celui qu'on appelle communément modèle de Liu et Layland [LL73]. Ce modèle consiste à caractériser une tâche par deux paramètres : C sa durée d'exécution maximale appelée aussi durée d'exécution du pire cas, et T sa période d'activation. D'autres modèles sont proposés dans lesquels on ne fait que rajouter des paramètres à ce modèle de base.
Un modèle de tâches périodiques un peu plus sophistiqué que le précèdent est proposé par [CDKZ00].
Une tâche temps réel périodique τ_i y est définie par les paramètres suivants :

- $r_{i,k}$ la date de réveil du job $\tau_{i,k}$, le $k^{ième}$ job de τ_i
- C_i la durée d'activation pire-cas de τ_i
- D_i l'échéance relative (ou délai critique) de τ_i
- $d_{i,k} = r_{i,k} + D_i$ l'échéance absolue ou la date critique du job $\tau_{i,k}$
- T_i la période d'activation de la tâche τ_i

La date de réveil $r_{i,k}$ représente la date à partir de laquelle le job $\tau_{i,k}$ est prêt pour démarrer son exécution. La durée d'exécution au pire cas C_i correspond à l'intervalle de temps maximal dont a besoin un job de τ_i pour s'exécuter sur le processeur. Le délai critique D_i ou échéance relative représente l'intervalle de temps séparant le réveil de τ_i de son échéance. La date critique ou échéance absolue $d_{i,k}$ est la date délimitant la fin d'exécution au plus tard du job $\tau_{i,k}$ mesurée par rapport à sa date de réveil.

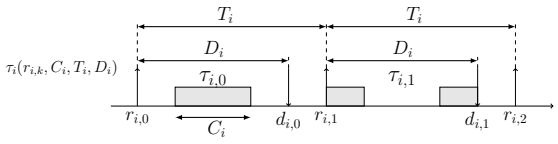


FIGURE 1.2 – Caractéristiques d'une tâche périodique τ_i

Les tâches périodiques sont dites :

- à *échéances contraintes* : si l'échéance relative est inférieure ou égale à la période ($D_i \leq T_i$).
- à *échéances implicites* (ou à échéance sur requêtes) si l'échéance relative et la période sont égales ($D_i = T_i$) ; c'est donc un cas particulier de tâches périodiques à échéances contraintes.
- à *échéances arbitraires* : s'il n'existe pas de contrainte entre l'échéance et la période.
- *concrètes* : si leurs dates de réveil sont connues a priori.
- *synchrones* : si elles ont la même date de réveil initial ; dans le cas contraire, on dit qu'elles sont asynchrones.
- *critiques* : si les échéances doivent être impérativement respectées.
- *semi-critiques ou fermes* : si le non-respect de quelques échéances est toléré.
- *non critiques* : si elles n'ont pas d'échéances strictes à respecter.

Quelques caractéristiques supplémentaires sur les tâches peuvent être considérées :

1. La *laxité* L_i de la tâche τ_i est la durée maximale durant laquelle τ_i peut être retardée de sorte que son échéance demeure respectée.
2. Le *facteur de charge du processeur associé à la tâche* τ_i est calculé ainsi : $ch_i = \frac{C_i}{D_i}$.
3. Le *facteur d'utilisation du processeur associé à la tâche* τ_i représente le pourcentage de l'activité du processeur dédié à τ_i, $u_i = \frac{C_i}{T_i}$.
4. Le *facteur d'utilisation du processeur* pour une configuration de *n* tâches périodiques est défini comme étant la somme des facteurs d'utilisation des *n* tâches s'exécutant sur le processeur, $U_p = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{C_i}{T_i}$.
5. la *gigue de démarrage* $G_{i,k}$ est l'écart entre la date de réveil du job $\tau_{i,k}$ et la date du début de son exécution.
6. Le *temps de réponse* $TR_{i,k}$ du job $\tau_{i,k}$ est la différence de temps entre l'instant de fin d'exécution et la date de réveil du job $\tau_{i,k}$.

Introduit par Mok [Mok83], le modèle de tâches sporadiques est caractérisé par le tuple suivant $\tau_i^s = (C_i^s, D_i^s, T_i^s)$. Les tâches sporadiques diffèrent des tâches périodiques sur le fait que T_i^s ne correspond plus à la période exacte de la tâche τ_i^s. En effet, la période T_i^s représente la durée *minimale* séparant l'arrivée de deux jobs successifs de τ_i^s. Par conséquent, une tâche périodique est un cas particulier de tâche sporadique.

1.3.3 Modèle de tâches apériodiques

Le modèle de tâches apériodiques reprend celui proposé pour les tâches périodiques en considérant une période égale à l'infini car cette tâche ne s'exécute qu'une seule fois. En effet une tâche apériodique est produite lors d'un évenement non récurrent, une alarme déclenchée par exemple. De ce fait, la date de réveil d'une tâche apériodique n'est pas connue préalablement. Les tâches apériodiques peuvent être critiques ou non critiques. Une tâche apériodique non critique n'est pas contrainte temporellement. Elle peut être caractérisée par deux paramètres : $\varphi = (r^a, C^a)$ où r^a est sa date de réveil et C^a sa durée d'exécution pire-cas.

Contrairement à une tâche apériodique non-critique, une tâche apériodique critique est munie d'une date d'échéance et son exécution doit s'effectuer avant cette échéance.

1.3.4 Durée d'exécution pire-cas

La durée d'exécution au pire cas WCET (*Worst Case Execution Time*) d'une tâche représente une majoration de son temps d'exécution. En général, il faut pouvoir garantir le bon comportement d'un système sur une durée infinie et quelles que soient les conditions environnementales de fontionnement. Dès la conception du sytème, il faut pouvoir prévoir la pire des situations pouvant se réaliser sur le système. La difficulté consiste à déterminer quelle sera la plus grande charge de travail dont aura besoin chaque tâche s'exécutant sur le processeur. Ainsi une fois que le plus juste WCET de chaque tâche est calculé, un test d'ordonnançabilité hors-ligne pourra être mis en oeuvre permettant de s'assurer de la faisabilité et du bon fonctionnement de l'application dans le pire-cas. Toutefois la détermination du meilleur WCET reste une problèmatique de très grande actualité. En l'état actuel des choses, il est difficile de prévoir l'impact d'événements extérieurs pour une tâche (par exemple : partage de ressources critiques avec d'autres tâches, conflits sur le bus pendant des transferts entre mémoire centrale et périphériques, etc.). Actuellement, le calcul du paramètre WCET demeure un problème non trivial qui donne encore lieu à de nombreux travaux en vue d'améliorer sa détermination. [SPH+05,WEE+08,HPP09].

2 Formulation du problème d'ordonnancement temps réel

Un système d'exploitation multitâche temps réel permet l'exécution de plusieurs programmes à la fois. Lorsque plusieurs tâches demandent à s'exécuter simultanément sur un même processeur, plusieurs problèmes se posent

dont l'accès au processeur, l'accès concurrent à la mémoire, l'accès aux périphériques, etc. Par conséquent, il se doit de choisir, à chaque instant, la tâche la plus appropriée à s'exécuter sur le processeur. Pour permettre ce choix, il faut prévoir un ordonnanceur (pour la mise en oeuvre d'un algorithme d'ordonnancement) permettant le bon fonctionnement du système c'est-à-dire le respect des échéances des tâches. L'ordonnanceur est alors le composant principal d'un système d'exploitation choisissant l'ordre d'exécution des tâches sur le (ou les) processeur(s). Il existe deux grandes classes d'ordonnanceurs :

- Les ordonnanceurs en temps partagé : Ils permettent d'attribuer les ressources de la manière la plus équitable possible entre les différents traitements dans le système.
- Les ordonnanceurs temps réel : Ils permettent de garantir le respect des échéances pour chaque traitement.

Dans le cadre de la thèse, nous nous intéressons aux ordonnanceurs temps réel.

2.1 Catégorisation des algorithmes d'ordonnancement

En géneral, les ordonnanceurs sont classifiés selon des caractéristiques du système sur lequel ils sont implantés. On indique ci-dessous quelques caractéristiques propres à un ordonnanceur :

Monoprocesseur ou multiprocesseur

L'ordonnancement est de type monoprocesseur si toutes les tâches ne peuvent s'exécuter que sur un seul et même processeur. Si plusieurs processeurs sont disponibles dans le système, l'ordonnancement est de type multiprocesseur.

Hors-ligne ou en-ligne

Les algorithmes d'ordonnancement peuvent être classés en deux catégories, à savoir les algorithmes d'ordonnancement hors-ligne et en-ligne :
Un ordonnancement hors-ligne est établi, avant le lancement de l'application, pour déterminer une séquence fixe d'exécution des tâches à partir de toutes les différentes caractéristiques et contraintes de celles-ci. Ensuite cette séquence est rangée dans une table et exécutée en ligne par le processeur.
Un ordonnanceur en-ligne consiste à construire une séquence dynamiquement en fonction des événements qui surviennent. Cependant il s'appuit sur des données collectées suite à une analyse préalable du système effectuée hors-ligne afin d'assurer le respect des contraintes temporelles des tâches.

Conduit par la priorité

Une grande majorité des algorithmes d'ordonnancement en-ligne rangent les tâches prêtes en leur associant une valeur appelée *priorité*. On parle ainsi d'algorithmes conduits par la priorité. Plus haute est la priorité d'une tâche à un instant donné, plus courte sera sa durée d'attente du processeur. On distingue les ordonnanceurs à priorités fixes de ceux à priorités dynamiques selon que cette priorité est constante ou variable dans le temps.

Préemptif ou non-préemptif

Un ordonnanceur est préemptif si l'exécution de toute tâche peut être interrompue pour réquisitionner le processeur au profit d'une autre tâche jugée plus urgente ou plus prioritaire. A contrario, si une fois lancée la tâche en cours d'exécution ne peut pas être interrompue avant la fin de son exécution en dépit du réveil d'une tâche plus prioritaire, l'ordonnancement est dit non préemptif. Dans le cas d'ordonnanceurs monoprocesseur non-préemptifs, le problème de synchronisation à l'accès des ressources n'existe plus puisque, en l'absence de préemption, il ne peut plus y avoir de concurrence sur l'accès aux ressources. Mais malheureusement, ce type d'ordonnancement s'avère moins performant au regard de la qualité de service car conduisant à manquer davantage d'échéances que son homologue préemptif.

Oisif ou non-oisif

Un ordonnanceur non oisif fonctionne sans insertion de temps creux (*non-idling* ou *work-conservative* en anglais). Par conséquent, dans le cas d'un ordonnanceur non-oisif, s'il existe au moins une tâche prête et si le processeur est libre, alors l'ordonnanceur élit la tâche la plus prioritaire et l'exécute immédiatement.

L'ordonnanceur oisif fonctionne par insertion de temps creux (*with inserted idle times* en anglais). Même si le processeur est libre, il se peut qu'une tâche prête attende un certain temps avant d'être exécutée sur décision de l'ordonnanceur. On montrera dans la suite de ce travail que l'oisiveté d'un ordonnanceur est une propriété nécessaire pour pouvoir obtenir la meilleure qualité de service dans un contexte contraint par l'énergie.

Centralisé ou réparti

Coulouris et al. proposent la définition suivante [CDK94] : *"Un système réparti est un ensemble de machines autonomes connectées par un réseau, et équipées d'un logiciel dédié à la coordination des activités du système ainsi qu'au partage de ses ressources."*

Un système est centralisé lorsque l'algorithme d'ordonnancement pour tout le système, distribué ou non, se produit sur un site privilégié de l'architecture distribuée qui contient l'ensemble des paramètres des tâches.

Ce travail de thèse se focalise sur les systèmes centralisés restreints à une architecture monoprocesseur.

2.2 Propriétés des algorithmes d'ordonnancement

Le choix d'un algorithme d'ordonnancement dépend essentiellement du contexte défini par les différentes caractéristiques du système temps réel sur lequel il sera mis en oeuvre. Nous citons dans cette partie, quelques propriétés et définitions utilisées dans l'analyse d'ordonnancabilité et de faisabilité d'une application temps réel.

Définition 2.1 ***Validité***
Une séquence d'ordonnancement d'une configuration de tâches est valide si et seulement si toutes les échéances des tâches sont respectées.

Définition 2.2 ***Faisabilité***
Une configuration de tâches est dite faisable s'il existe au moins une séquence d'ordonnancement dans laquelle toutes les tâches respectent bien leur échéance.

Définition 2.3 ***Ordonnançabilité***
Une configuration de tâches est dite ordonnançable si et seulement si, il existe une séquence d'ordonnancement valide de longueur infinie.

Définition 2.4 ***Optimalité***
L'algorithme d'ordonnancement A est dit optimal s'il est capable d'ordonnancer toute configuration de tâches faisable.

Autrement dit, si un ordonnanceur optimal ne parvient pas à construire une séquence valide pour une configuration de tâches donnée, alors aucun autre ordonnanceur ne pourra trouver une séquence valide pour cette même configuration.

Définition 2.5 ***Test d'ordonnançabilité***
Un test d'ordonnançabilité permet de déterminer, avant exécution, si un ordonnanceur donné pourra fournir une séquence d'ordonnancement valide pour une configuration de tâches donnée.

Concevoir une application de type temps réel dur impose donc avant son démarrage de vérifier que l'application est réalisable. Cela s'effectue donc dans la phase de mise au point par l'exécution d'un test d'ordonnançabilité après avoir sélectionné l'ordonnanceur qui sera implanté dans le système d'exploitation.

Définition 2.6 *Un algorithme d'ordonnancement est dit clairvoyant lorsqu'il connaît toutes les caractéristiques des tâches à exécuter et dont le réveil n'a pas encore eu lieu. Sinon, il est dit non clairvoyant.*

Définition 2.7 *Un algorithme est déterministe s'il ne fait intervenir aucune composante aléatoire dans la prise de décisions d'ordonnancement. Par conséquent, une configuration de tâches périodiques ordonnancée plusieurs fois par un même algorithme déterministe présentera la même planification de tâches.*

Définition 2.8 *[LM80] Considérons une configuration de n tâches périodiques. La séquence d'ordonnancement pour cette configuration de tâches, produite par tout algorithme d'ordonnancement préemptif est toujours périodique et de période égale à H. H, appelé hyperpériode, représente le plus petit commun multiple des périodes des tâches de la configuration.*

2.3 Complexité

Un algorithme d'ordonnancement a pour objectif de fournir une séquence d'ordonnancement qui assure le respect des contraintes temporelles. Cependant l'évaluation de l'efficacité d'un algorithme dépend non seulement du résultat qu'il fournit en sortie mais aussi du temps de calcul requis pour son exécution et de la place mémoire nécessaire à son exécution. Ceci implique de déterminer la complexité de l'algorithme.

Dans l'étude de la complexité, il faut distinguer deux catégories de problèmes : problèmes de décision et problèmes d'optimisation.

Un problème de décision est un problème dont la réponse est soit *oui* soit *non*. Les travaux théoriques dans ce domaine ont permis d'identifier différentes classes de problèmes de décision en fonction de la complexité de leur résolution [Pap94] :

- Un problème de décision appartient à la **classe P**, s'il peut être résolu par un algorithme polynomial en n.
- Un problème de décision appartient à la **classe NP** (Non-déterministe Polynomial) s'il peut être résolu par un algorithme non déterministe polynomial capable de vérifier la validité d'une solution donnée.
- Les problèmes les plus difficiles de classe NP appartiennent à la classe des problèmes **NP-complets** dans le sens où l'on ne trouve pas d'algorithme polynomial pour les résoudre avec une machine déterministe.

Tester la faisabilité d'une application temps réel ou tester l'ordonnançabilité d'une configuration de tâches par un algorithme d'ordonnancement sont des problèmes de décision.

Un problème d'optimisation est un problème pour lequel on doit chercher à déterminer une solution en vue d'optimiser un critère. Chercher à minimiser le nombre d'échéances violées est donc un problème d'optimisation. Chercher à minimiser le plus grand retard par rapport à l'échéance en est un autre. A chaque problème d'optimisation, un problème de décision peut être associé. Plus précisément, lorsque le problème de décision est NP-complet, le problème d'optimisation est dit NP-difficile.

Soit *f(n)* la fonction de complexité temporelle d'un algorithme dans le pire cas. *f(n)* représente la borne supérieure sur le nombre d'opérations élémentaires effectuées par un algorithme A pour résoudre un problème X lorsque la taille de l'entrée est *n*.

Définition 2.9 *Les algorithmes sont dit de complexité pseudo-linéaire, lorsque la complexité est en O(log n).*

Définition 2.10 *Un algorithme est de complexité polynomiale lorsque* $f(n) \in O(p(n))$ *sachant que p est un polynôme en n.*

Définition 2.11 *Un algorithme est dit de complexité linéaire lorsque* $f(n) \in O(n)$ *ou* $f(n) \in O(nlogn))$.

Définition 2.12 *Un algorithme est dit de complexité exponentielle lorsque* $f(n) \in O(n!)$ *ou* $f(n) \in O(k^n)$, $k > 1$.

Il est évident que plus la complexité d'un algorithme est élevée, plus son temps de calcul et son overhead d'implémentation le sont aussi. Par définition, on appelle *overhead* le surcoût de traitement lié à différents mécanismes, tels que les changements de contexte, le nombre de calcul de paramètres pertinents lors de la création de la séquence d'ordonnancement.

Les algorithmes polynomiaux sont évidemment les algorithmes les plus efficaces en termes de temps d'exécution. Par contre, les algorithmes non polynomiaux (les algorithmes exponentiels par exemple) sont certainement beaucoup plus lents et sont considérés comme étant impraticables dès que la taille des données est supérieure à quelques dizaines d'unités. Cependant, les algorithmes polynomiaux ayant une complexité en $O(n^k)$ (k étant un entier positif) peuvent s'avérer inefficaces à partir d'une certaine valeur de k.

3 Ordonnancement de tâches périodiques

Suivant les spécificités de l'application, le concepteur doit définir une méthode d'ordonnancement afin de respecter les contraintes temporelles des tâches. Dans cette section, nous détaillons les principaux algorithmes classiques pour l'ordonnancement de tâches périodiques. Les algorithmes présentés sont basés sur la considération du pire cas, c'est-à-dire qu'ils évaluent l'ordonnançabilité d'une configuration de tâches périodiques à un instant critique. L'instant critique d'une tâche [LL73] est l'instant où la demande d'exécution de cette tâche arrive simultanément avec la demande d'exécution de toutes les tâches de priorités supérieures. Par conséquent, cet instant correspond au plus long temps de réponse de l'ordonnanceur suite à la demande d'exécution de toutes les tâches.

3.1 Ordonnancement à priorités fixes

3.1.1 Algorithme RM (Rate Monotonic)

L'algorithme RM a été introduit par Liu et Layland en vue d'ordonnancer une configuration de tâches à échéances implicites [LL73]. Selon RM, une tâche dispose d'une priorité fixe inversement proportionnelle à sa période. Par conséquent, RM donne à tout instant la priorité au job appartenant à la tâche de plus petite période d'activation.

Illustration :
Considérons une configuration de tâches périodiques $\mathcal{T} = \{\tau_i(r_i,\ C_i,\ D_i,\ T_i)\} = \{\ \tau_1(0,\ 1,\ 3,\ 3), \tau_2(0, 3, 6, 6)\}$. La Figure 1.3 illustre l'ordonnancement de la configuration $\mathcal{T}$ avec l'algorithme RM.
On peut remarquer qu'à *t=0*, la tâche τ_1 est la plus prioritaire. Son job s'exécute pendant une unité de temps

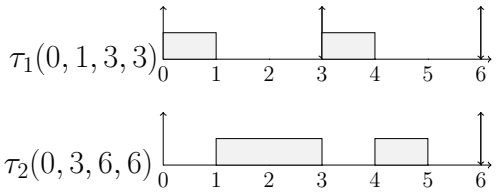


FIGURE 1.3 – Ordonnancement selon RM

et finit son exécution à l'instant *t=1*. A l'instant *t=1*, le job de la tâche τ_2 est prêt et commence son exécution. Cependant, étant donné que le job de τ_1 est le plus prioritaire à l'instant *t=3* ($T_1 < T_2$), celui-ci va interrompre le job de τ_2 pour s'exécuter.

Résultat d'optimalité :
RM est optimal dans la classe des algorithmes préemptifs à priorités fixes pour l'ordonnancement des tâches périodiques, indépendantes, synchrones et à échéances sur requêtes [LL73]. Si l'une de ces conditions n'est pas respectée, alors RM n'est plus optimal.

Condition d'ordonnançabilité :
Une condition suffisante d'ordonnançabilité a été proposée par Liu et Layland [LL73]. Soit $U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$ le facteur d'utilisation d'une configuration de n tâches périodiques à échéances sur requêtes. Celle-ci est ordonnançable par l'algorithme RM si U_p vérifie la condition suivante :

$$U_p \leq n\left(2^{\frac{1}{n}} - 1\right) \tag{1.1}$$

En interprétant ce résultat, nous constatons que pour un nombre élevé de tâches, la limite supérieure de la charge processeur qui garantit un ordonnancement faisable avec RM est d'environ $ln(2) \cong 0.69$. Par contre, si les tâches sont harmoniques (toutes les périodes sont multiples ou sous-multiples des autres), cette limite tend vers 1. En outre dans son étude expérimentale sur des tâches avec des périodes et des coûts d'exécution aléatoires, Lehoczky et al. [LSD89] montrent que l'algorithme RM est capable d'ordonnancer des configurations de tâches avec une limite du facteur d'utilisation du processeur de près de 0.88.

3.1.2 Algorithme DM (Deadline Monotonic)

L'algorithme DM a été introduit par Leung et Whitehead [LW82]. DM classe les tâches par ordre croissant des *échéances relatives*. DM et RM se confondent pour des tâches à échéances sur requêtes. En d'autre termes, plus son échéance relative D_i est petite, plus une tâche τ_i est prioritaire. DM est meilleur que RM pour les tâches périodiques dont les échéances sont très inférieures à leurs périodes. En effet, RM pénalise les tâches rares mais urgentes (i.e. les tâches ayant de très grandes échéances).

Illustration :
La Figure 1.4 illustre l'ordonnancement d'une configuration de tâches périodiques par DM : $\mathcal{T} = \{\tau_i(r_i,\ C_i,$

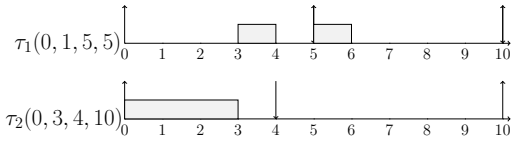


FIGURE 1.4 – Ordonnancement selon DM

$D_i,\ T_i)\} = \{\tau_1(0,\ 1,\ 5{,}5), \tau_2(0,\ 3,\ 4,\ 10)\}$. Dans cet exemple, les jobs des tâches τ_1 et τ_2 sont réveillés initialement à l'instant *t=0*. Le job de la tâche τ_2 est le plus prioritaire ($D_2 < D_1$) ; il s'exécute pendant trois unités de temps. A l'instant *t=3*, le job de la tâche τ_1 est le plus prioritaire et commence son exécution. A l'instant *t=4*, aucun job n'est prêt et le processeur est inactif. A l'instant *t=5*, le job de la tâche τ_1 étant le seul prêt s'exécute pendant une unité de temps.

Résultat d'optimalité : [LW82]
L'algorithme DM est optimal dans la classe des algorithmes préemptifs, à priorités fixes pour l'ordonnancement des tâches périodiques, indépendantes, synchrones et à échéances contraintes.

3.2 Ordonnancement à priorités dynamiques

3.2.1 L'algorithme EDF (Earliest Deadline First)

Earliest Deadline First (EDF) est un ordonnanceur à priorités fixes au niveau des jobs et à priorités dynamiques au niveau des tâches. Il fût présenté par Jackson en 1955 [Jac55]. Avec l'algorithme *EDF*, la plus haute priorité à l'instant *t* est accordée à la tâche dont l'échéance est la plus proche de cet instant. *EDF* s'utilise majoritairement avec préemptions : si un job très urgent se réveille, le job en cours d'exécution est préempté au profit du plus urgent. Cependant, *EDF* peut aussi être utilisé sans préemption ; Cette version ne sera pas abordée dans cette thèse connaissant les mauvaises performances des ordonnanceurs non préemptifs.

Illustration :
Considérons la configuration de tâches périodiques $\mathcal{T}$=$\{\tau_i(C_i,\ D_i,\ T_i)\}$ = $\{\tau_1(2, 5, 5), \tau_2(2, 9, 10), \tau_3(6, 20, 20)\}$. Les tâches sont synchrones et réveillées à l'instant *t=0*. L'ordonnancement de cette configuration selon EDF est illustré par la Figure 1.5.

Résultats d'optimalité : *EDF* est optimal pour des configurations de tâches indépendantes et à échéances contraintes ($D_i \leq T_i$) [Der74, Lab74].

Tests d'ordonnançabilité
Un test d'ordonnançabilité par *EDF* pour des tâches périodiques ou sporadiques à échéances implicites, est donnée par le théorème suivant :

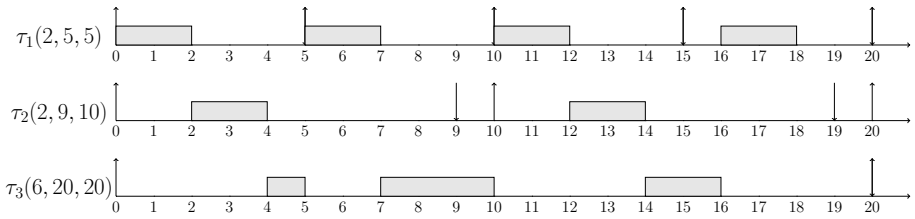


FIGURE 1.5 – Ordonnancement selon EDF

Théorème 3.1 *[LL73, Der74] Une configuration de n tâches périodiques synchrones à échéances sur requêtes est ordonnançable avec EDF si et seulement si son facteur d'utilisation* $U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$ *vérifie :*

$$U_p \leq 1 \tag{1.2}$$

La condition précédente a le mérite d'être très simple à tester. C'est aussi une condition à la fois nécessaire et suffisante. Elle nous permet de constater que l'on peut garantir la faisabilité d'une application qui utilise jusqu'à 100% de la capacité de traitement du processeur. Cependant, cette condition suppose que toutes les tâches aient une échéance relative égale à la période. C'est pourquoi d'autres études un peu plus récentes dont celle présentée par Baruah [BMR90] ont abouti à proposer un autre test d'ordonnançabilité par EDF pour des tâches périodiques synchrones à échéances arbitraires :

Théorème 3.2 *Une configuration de n tâches périodiques est ordonnançable avec EDF si et seulement si :*

$$\forall L > 0, \sum_{i=1}^{n} \left(\left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor \right) C_i \leq L \tag{1.3}$$

où L correspond à toutes les échéances absolues des jobs sur l'hyperpériode.

Bien que fournissant une condition nécessaire et suffisante d'ordonnançabilité, ce test basé sur l'approche "demande processeur" induit une plus grande complexité que le test basé sur l'approche "facteur d'utilisation".

3.2.2 L'algorithme LLF (Least Laxity First)

L'algorithme LLF [Mok83] attribue à tout instant la plus haute priorité à la tâche ayant la plus faible laxité dynamique. A chaque instant *t*, le job le plus prioritaire sera celui dont la laxité sera la plus petite. La laxité dynamique $L_i(t)$, illustrée sur la Figure 1.6, représente le temps maximum pendant lequel l'exécution du job de la tâche τ_i peut être retardée sans que celui-ci manque son échéance. Elle se définit par :
$L_i(t) = d_i - (t + C_i(t))$ où d_i est l'échéance absolue, t l'instant courant et $C_i(t)$ le temps d'exécution restant du job de la tâche τ_i à l'instant t. C'est donc un algorithme à priorités dynamiques au niveau des jobs car la priorité de chaque job non exécuté augmente avec le temps.

Illustration :
Considérons la configuration de tâches périodiques $\mathcal{T} = \{\tau_i(C_i,\ D_i,\ T_i)\} = \{\tau_1(2,\ 5,\ 5),\ \tau_2(2,\ 9,\ 10),\ \tau_3(6,\ 20,\ 20)\}$. Les tâches sont synchrones et donc réveillées simultanément à l'instant *t=0*. L'ordonnancement de cette configuration selon LLF est illustré par la Figure 1.6.

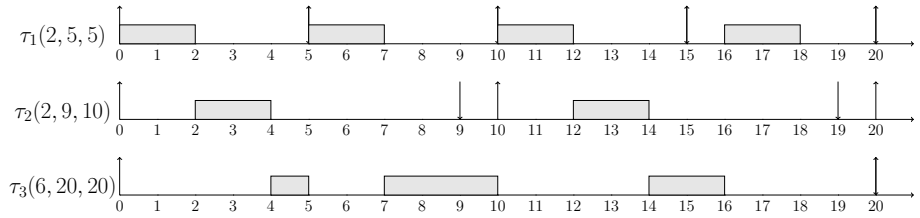


FIGURE 1.6 – Ordonnancement selon LLF

Résultats d'optimalité :
LLF est optimal [Mok83, DM89] pour des configurations de tâches périodiques indépendantes à échéances contraintes.

Contrairement à un algorithme à priorités fixes au niveau des jobs tel que EDF, LLF nécessite de mettre à jour continuellement les priorités des jobs. Pour EDF cette mise à jour n'est faite qu'au réveil du job. Puisqu'on calcule la laxité à chaque réveil ou terminaison de job, LLF entraîne plus de préemptions et donc plus de changements de contexte que EDF. On peut en déduire que EDF est plus efficace et plus facilement implémentable que LLF.

4 Ordonnancement de tâches apériodiques

Certains systèmes doivent prendre en compte l'arrivée de tâches apériodiques. Ils donc doivent pouvoir les ordonnancer conjointement avec les tâches périodiques et ce, en respectant les contraintes temporelles de celles-ci. Plusieurs méthodes ont été proposées pour l'ordonnancement des tâches apériodiques pour lesquelles on cherche à minimiser leur temps de réponse. Ainsi nous pouvons citer par ordre chronologique croissant, le serveur à scrutation (*Polling server*) [SSL89], le serveur ajournable (*Deferrable Server*) [LSS87, SLS95], le serveur sporadique (*Sporadic Server*) [SSL89, SGR88, SB94], le serveur *Slack Stealer* [LRT92], le serveur à échange de priorité (*Exchange Server*) [SBLS88,LSS87,SB94]). Nous décrirons dans la suite le serveur BG, des serveurs à priorités fixes et des serveurs à priorités dynamiques. Nous nous intéresserons plus particulièrement au comportement des serveurs : BG [LSS87], EDL [CC89], TBS [SB94, SB96] que nous allons implémenter et adapter dans le cadre de la thèse pour faire face à des contraintes énergétiques.

4.1 Approche basée sur un traitement en arrière-plan

4.1.1 L'algorithme BG (Background Server)

La méthode d'ordonnancement dite d'arrière-plan [LSS87] est une méthode simple à réaliser et peu coûteuse (en termes d'overhead). Elle consiste à traiter les tâches apériodiques avec une priorité toujours inférieure à celles des tâches périodiques. Les tâches apériodiques sont ainsi traitées pendant les périodes d'inactivité du processeur. S'il existe plusieurs tâches apériodiques en attente, celles-ci sont servies selon la technique *FIFO* (First-In First-Out) appelée aussi *FCFS* (First-Come First-Serve). Cet algorithme permet de garantir l'exécution des tâches périodiques dans le respect de leurs contraintes temporelles, le serveur n'ayant aucun impact sur la séquence. Cependant, le temps de réponse des tâches apériodiques peuvent s'avérer très grands surtout quand la charge des tâches périodiques devient importante et s'approche de 1.

Illustration :
La Figure 1.7 illustre le fonctionnement du serveur BG. Nous considérons une configuration de tâches périodiques $\mathcal{T} = \{\tau_i(C_i, D_i, T_i)\} = \{\tau_1(3, 10, 10), \tau_2(4, 15, 15)\}$. Les tâches sont synchrones et donc réveillées simultanément à l'instant *t=0*. Durant l'hyperpériode *H=10*, une tâche apériodique $\varphi_1(3, 5)$ arrive à l'instant *t=3* et a une durée d'exécution C_1^a de 5 unités de temps. Les tâches périodiques sont ordonnancées selon l'algorithme EDF. Ainsi à l'instant *t=0*, le job $\tau_{1,1}$ est le plus prioritaire et commence son exécution. Ensuite à l'instant *t=3*, $\tau_{1,1}$ finit son exécution et $\tau_{2,1}$ est le suivant à être exécuté. Il finit son exécution à l'instant *t=7* ; Comme il n'y a aucun job prêt, la tâche apériodique φ_1 commence son exécution.

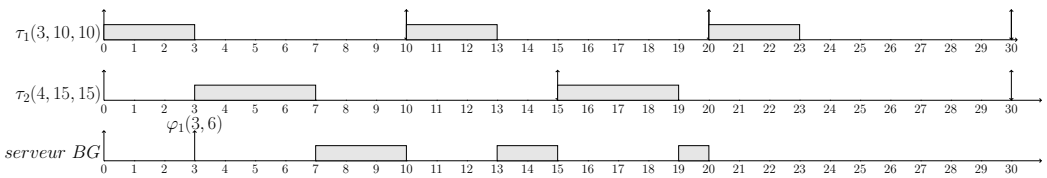


FIGURE 1.7 – Ordonnancement avec le serveur BG

Afin de minimiser le temps de réponse des tâches apériodiques dans le contexte d'un ordonnancement conduit par priorités, l'idée consiste à les faire prendre en charge par une tâche particulière qui possède une réelle priorité (statique ou dynamique) sans toutefois compromettre l'exécution des tâches périodiques : on parle alors de tâche serveur.

4.2 Approches basées sur un serveur en priorité fixe

4.2.1 L'algorithme PS (Polling Server)

L'algorithme PS (*Polling Server*) [LSS87] permet de servir périodiquement les requêtes apériodiques par le biais d'une tâche serveur. Cette tâche possède une période T_S et une capacité C_S qui est consommée par les tâches apériodiques lors de leur exécution. Le PS consiste à servir périodiquement les requêtes apériodiques en attente selon la technique FIFO tant que la capacité C_S n'est pas nulle. Dès que celle-ci est épuisée, il s'arrête. À chaque période le serveur récupère sa capacité maximale si et seulement s'il y a au moins une tâche apériodique en attente. S'il n'y a pas de tâches apériodiques en attente, sa capacité devient nulle. De plus, si le temps de traitement associé à une tâche apériodique dépasse la capacité, alors l'exécution de celle-ci est interrompue jusqu'à la période d'activation suivante du serveur. Ainsi toutes les tâches périodiques y compris la tâche serveur sont ordonnancées selon l'algorithme RM. Le PS permet de minimiser le temps de réponse des tâches apériodiques en les assimilant à une tâche périodique sans compromettre l'exécution des tâches périodiques.

Illustration :
Le fonctionnement du serveur PS est illustré par la Figure 1.8 pendant une hyperpériode *H*. Nous considérons une configuration de tâches périodiques $\mathcal{T} = \{\tau_i(C_i,\ D_i,\ T_i)\} = \{\tau_1(3,\ 10,\ 10), \tau_2(4,\ 15,\ 15)\}$ et un serveur PS de capacité $C_S = 2$ et de période $T_S = 5$. Les tâches sont synchrones et sont donc réveillées simultanément à l'instant *t=0*. Durant l'hyperpériode *H=10*, une tâche apériodique $\varphi_1(3, 5)$ arrive à l'instant *t=3* et a une durée d'exécution C_1^a de 5 unités de temps. Les tâches périodiques et la tâche serveur sont ordonnancées selon l'algorithme *RM*. A l'instant *t=0*, la tâche serveur est la plus prioritaire cependant il n'y a aucune tâche apériodique en attente, donc le serveur est suspendu et perd sa capacité. Par conséquent, la tâche apériodique arrivant à *t=3* ne pourra être servie qu'à partir de l'instant *t=5* et c'est le job périodique $\tau_{1,1}$ qui s'exécute à l'instant *t=0*. On peut noter que le temps de réponse de la tâche apériodique φ_1 est de 14 unités de temps alors qu'il était de 17 unités de temps avec BG. Donc le serveur PS s'avère plus efficace que BG dans ce cas. Cependant son efficacité en termes de temps de réponse dépend des paramètres attribués au serveur PS. Ainsi plus sa période T_S et sa capacité C_S sont grands, plus le temps de réponse des tâches apériodiques est court.

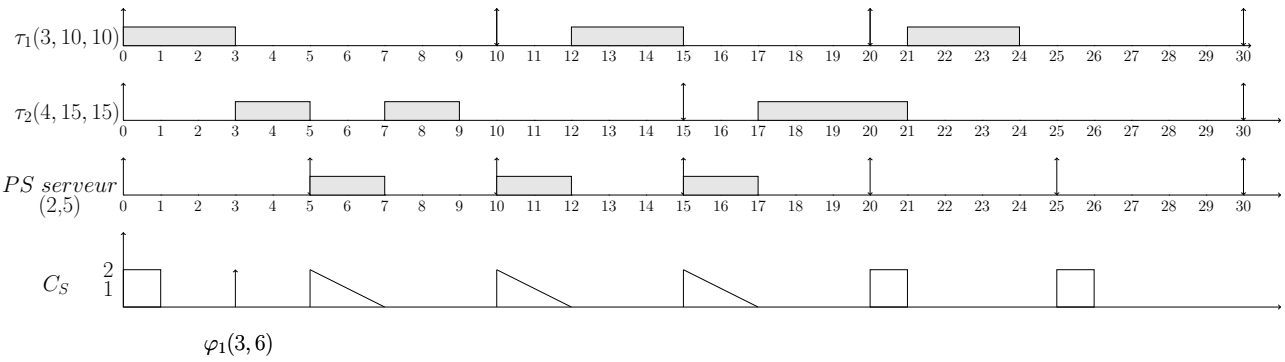


FIGURE 1.8 – Ordonnancement avec l'ordonnanceur PS

4.2.2 L'algorithme DS (Deferrable Server)

Tout comme le *Polling Server*, la méthode du *Serveur Différé* (Deferrable Server) [SLS95] repose sur l'exécution d'une tâche serveur possédant une période T_S et une capacité C_S. Le serveur *DS* a le même comportement que le serveur *PS*. Toutefois, sa capacité est conservée jusqu'à la fin de sa période même s'il n'y a pas de tâches apériodiques à traiter en début de période. La capacité de traitement de la tâche serveur, C_S, est utilisée pour traiter les tâches apériodiques et est renouvelée à chaque activation du serveur. La tâche serveur se comporte alors comme une tâche sporadique. Ainsi, dès l'arrivée d'une requête apériodique, elle débute son exécution si sa capacité le lui permet.

Illustration :
Le fonctionnement du serveur DS est illustré par la Figure 1.9 pendant une hyperpériode *H*. Nous considérons la configuration de tâches périodiques et apériodiques illustrée dans l'exemple précédent et un serveur DS de

capacité $C_S = 2$ et de période $T_S = 5$. Les tâches sont synchrones et donc réveillées simultanément à l'instant *t=0*. On note que le comportement du serveur DS conserve sa capacité durant toute sa période T_S. Ainsi à l'instant *t=3*, la tâche serveur étant la plus prioritaire selon *RM*, φ_1 commence son exécution et s'exécute pendant deux unités de temps. On peut remarquer que le temps de réponse est nettement amélioré. Avec le serveur DS, φ_1 a un temps de réponse de 10 unités de temps. Ainsi le serveur DS est plus performant que les serveurs BG et PS en termes de temps de réponse. Cependant, il est à noter qu'il peut compromettre l'exécution des tâches périodiques. De plus, les complexités de calcul et d'implémentation restent faibles.

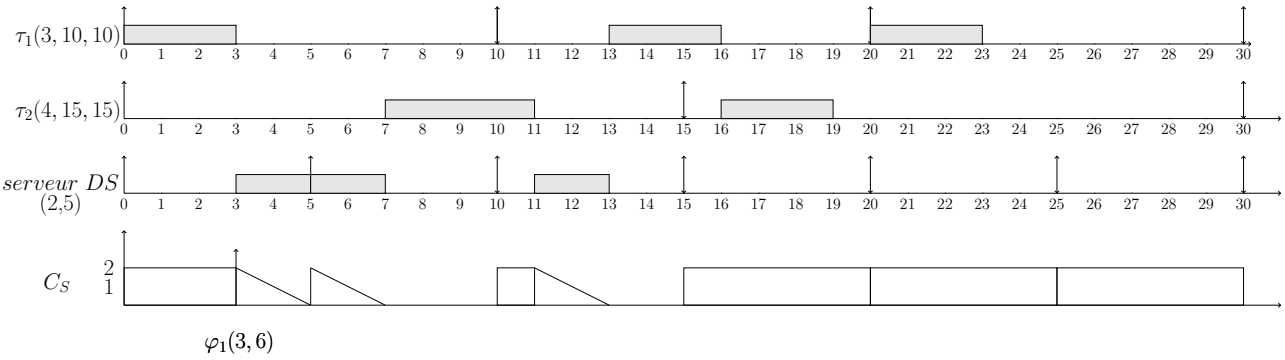


FIGURE 1.9 – Ordonnancement avec le serveur DS

4.3 Approches basées sur un serveur en priorité dynamique

4.3.1 L'algorithme EDL (Earliest Deadline as Late as Possible)

Le serveur *Earliest Deadline as Late as possible* (EDL) [CC89] est un algorithme d'ordonnancement dynamique, capable d'assurer la gestion de tâches apériodiques au sein d'une application temps réel. La technique revient à exécuter les tâches apériodiques au plut tôt en reportant (autant que faire se peut) les tâches périodiques au plus tard. Par ailleurs, Lehoczky et Ramos-Thuel [LRT92] ont proposé le serveur *Slack Stealer* qui repose sur le même concept sauf qu'il est dédié aux systèmes à priorités fixes.

Le serveur EDL calcule les temps de disponibilité appelés temps creux du processeur qui pourraient être rendus disponibles pour exécuter les tâches apériodiques, et ce, à chaque fois qu'une requête apériodique survient. Ainsi toutes les tâches périodiques sont ordonnancées au plus tard, dans le respect de l'ensemble des échéances. Lorsqu'une tâche apériodique se réveille, le serveur calcule la séquence EDL. Donc il récupère la localisation dans le temps et la durée des temps creux pour l'exécution des tâches apériodiques au plus tôt. Si aucune tâche apériodique n'est présente dans le système, ce serveur ordonnance les tâches périodiques selon l'algorithme EDF classique c'est-à-dire au plus tôt.

Dans [CC89], Chetto et Chetto proposent une méthode simple pour déterminer la localisation et la durée des temps creux dans n'importe quelle fenêtre d'une séquence produite par les deux différentes implémentations de EDF, à savoir EDS (Earliest Deadline as Soon as possible) et EDL. La terminologie utilisée par les auteurs est la suivante : f_Y^X représente la fonction de disponibilité définie pour une configuration de tâches Y et un algorithme d'ordonnancement X,

$$f_Y^X = \begin{cases} 1 & \text{si} \quad le\ processeur\ est\ inactif\ à\ t \\ 0 & \text{sinon} \end{cases} \tag{1.4}$$

Ainsi, l'intégrale $\int_{t_1}^{t_2} f_Y^X(t)dt$ fournit le nombre total d'unités de temps creux disponibles durant l'intervalle de temps $[t1, t2]$.

4.3.1.1 Calcul des temps creux statiques :

Le calcul de f^{EDL} hors-ligne repose sur deux vecteurs :
- Le vecteur des échéances qui représente l'ensemble des instants auxquels débute un temps creux sur une hyperpériode H correspondant à l'intervalle [0,H[.
- Le vecteur des temps creux qui représente la durée de temps creux associés aux différents instants dans le

vecteur des échéances.

Vecteur statique des échéances :
Soit une configuration de n tâches périodiques à échéances sur requêtes $\mathcal{T}$=$\{\tau_j(\ C_j,\ D_j,\ T_j),\ j\ =\ 1\ à\ n\}$, C_j est le temps d'exécution pire-cas, D_j est l'échéance relative et T_j est la période. Le vecteur statique des échéances est noté $\mathcal{K} = (k_0, .., k_i, .., k_q)$. Il représente l'ensemble des instants précédant un temps creux sur une hyperpériode. Ce vecteur se construit à partir des échéances des jobs périodiques [Sil99]. Les instants de départ des intervalles de temps creux sont définis par le théorème suivant [CC89] :

Théorème 4.1 *Les instants k_i du vecteur $\mathcal{K} = (k_0, ..., k_i, ..., k_q)$ sont définis comme suit :*

$$k_i = x.D_j \tag{1.5}$$

sachant que $x = 1, ..., \frac{H}{T_j}$, $k_i < k_{i+1}$, $k_0 = 0$ et $k_q = H - min(T_j; 1 \leq j \leq n)$

Vecteur statique des échéances
Le vecteur statique des échéances s'écrit comme suit, $\mathcal{D}^* = (\Delta_0^*, ..., \Delta_i^*, \Delta_{i+1}^*, ..., \Delta_q^*)$. Δ_i^* représente la longueur du temps creux débutant à l'instant k_i. La formule permettant de calculer le vecteur $\mathcal{D}^*$ est fournie par le théorème suivant [CC89] :

Théorème 4.2 *Les durées des temps creux du vecteur $\mathcal{D}^* = (\Delta_0^*, ..., \Delta_i^*, \Delta_{i+1}^*, ..., \Delta_q^*)$ présents sur [0,H] sont calculées comme suit :*

$$\Delta_q^* = min\{T_j; 1 \leq j \leq n\} \tag{1.6}$$

$$\Delta_i^* = sup(0, F_i) \tag{1.7}$$

avec

$$F_i = (H - k_i) - \sum_{j=1}^{n} \left\lceil \frac{H - k_i}{T - j} \right\rceil C_j - \sum_{l=i+1}^{q} \Delta_l^* \tag{1.8}$$

Illustration du calcul du vecteur statique des temps creux :
Nous considérons la configuration de tâches périodiques $\mathcal{T}\ =\ \{\tau_i(C_i,\ D_i,\ T_i)\}$=$\{\tau_1(3,\ 10,\ 10),\ \tau_2(4,\ 15,\ 15)\}$. Les tâches sont synchrones et réveillées simultanément à l'instant *t=0*. Nous appliquons les équations précédentes pour calculer hors-ligne, le vecteur statique des échéances et le vecteur statique des temps creux. Ainsi $\mathcal{K} = \{0,\ 9,\ 12,\ 18,\ 24,\ 27\}$ et $\mathcal{D}^* = \{5,\ 0,\ 3,\ 2,\ 0,\ 2\}$. La séquence EDL calculée est représentée par la Figure 1.10.

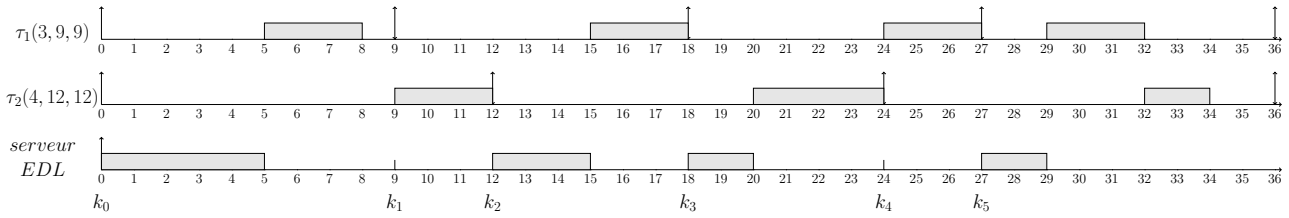


FIGURE 1.10 – Calcul des temps creux statiques avec l'algorithme EDL

4.3.1.2 Calcul des temps creux dynamiques Le calcul de f^{EDL} en-ligne repose sur deux vecteurs, le vecteur dynamique des échéances et le vecteur dynamique des temps creux .

Vecteur dynamique des échéances :
Le vecteur dynamique des échéances à un instant *t*, $\mathcal{K}(t)$ représente les instants supérieurs ou égaux à *t* qui précèdent un temps creux. Il se construit à partir des échéances distinctes des tâches périodiques supérieures ou égales à l'instant *t*. Ainsi $\mathcal{K}(t) = (t, k_{h+1}, ..., k_i, ..., k_q)$ sachant que la valeur de *t* vient remplacer l'index *h* qui est le plus grand index présent dans le vecteur statique des échéances tel que $k_h = sup(d, d < t)$.

Vecteur dynamique des temps creux
Le vecteur dynamique des temps creux $\mathcal{D}^*(t) = (\Delta_h^*(t), \Delta_{h+1}^*(t), ..., \Delta_i^*(t), ..., \Delta_q^*(t))$ représente les durées

des temps creux associés aux instants du vecteur $\mathcal{K}(t)$. Soit *M* la plus grande échéance parmi celles de tous les jobs périodiques actifs à l'instant *t*. Par conséquent l'indice f fait référence au plus petit index dans le vecteur des temps creux tel que $k_f = min\{k_i; k_i > M\}$. On calcule les éléments de $\mathcal{D}^*(t)$ en utilisant le théorème suivant [CC89] :

Théorème 4.3 *Les durées de temps creux du vecteur $\mathcal{D}^*(t)$ sont définies comme suit :*

$$\Delta_i^*(t) = \Delta_i^*, pour\ i = q\ à\ f \tag{1.9}$$

$$\Delta_i^*(t) = sup(0, F_i(t)), pour\ i = f-1\ à\ h+1 \tag{1.10}$$

avec

$$F_i(t) = (H - k_i) - \sum_{j=1}^{n} \left\lceil \frac{H-k_i}{T_j} \right\rceil C_j + \sum_{j=1 d_j > k_i}^{n} A_j(t) - \sum_{k=i+1}^{q} \Delta_k^*(t) \tag{1.11}$$

$$\Delta_h(t) = (H - t) - \sum_{j=1}^{n} (\left\lceil \frac{H-t}{T_j} \right\rceil C_j - A_j(t)) - \sum_{k=i+1}^{q} \Delta_k^*(t) \tag{1.12}$$

$A_j(t)$ représente la quantité de temps processeur déjà allouée à son job courant jusqu'à l'instant *t* et d_j est l'échéance absolue de la tâche périodique τ_j.

Illustration du calcul du vecteur statique des temps creux :
Nous considérons toujours la même configuration de tâches périodiques $\mathcal{T}$=$\{\tau_i(C_i, D_i, T_i)\}$ = $\{\tau_1(3, 10, 10), \tau_2(4, 15, 15)\}$. Une tâche apériodique $\varphi_1 = (4, 5)$ arrive à l'instant *t=4*. Nous appliquons les équations précédentes pour calculer en-ligne le vecteur dynamique des échéances et le vecteur dynamique des temps creux. Ainsi $\mathcal{K}(4) = \{4, 9, 12, 18, 24, 27\}$ et $\mathcal{D}^*(4) = \{2, 2, 4, 2, 0, 2\}$. La séquence EDL calculée est représentée par la Figure 1.11. On peut remarquer que la Figure 1.11 montre les temps creux disponibles pouvant être utilisés pour exécuter la tâche apériodique φ_1, ceci en retardant l'exécution des jobs périodiques au plus tard.

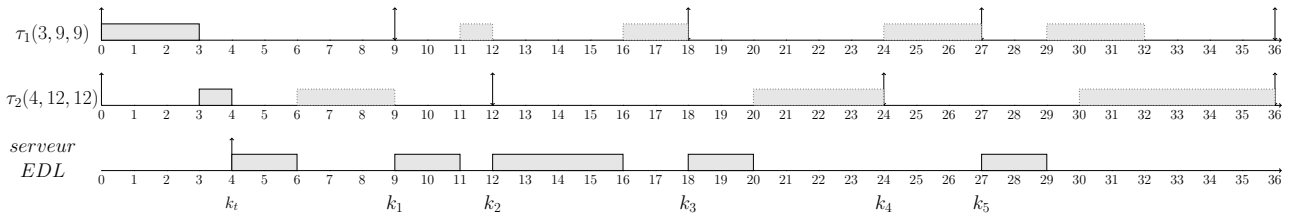


FIGURE 1.11 – Calcul des temps creux dynamiques selon EDL

La Figure 1.12 illustre quant à elle l'exécution de la tâche apériodique φ_1 conjointement aux tâches périodiques. Dans ce cas, φ_1 est servie immédiatement et affiche un temps de réponse de 9 unités de temps. Une fois son exécution finie, les jobs périodiques reprennent leur exécution normale, au plus tôt.

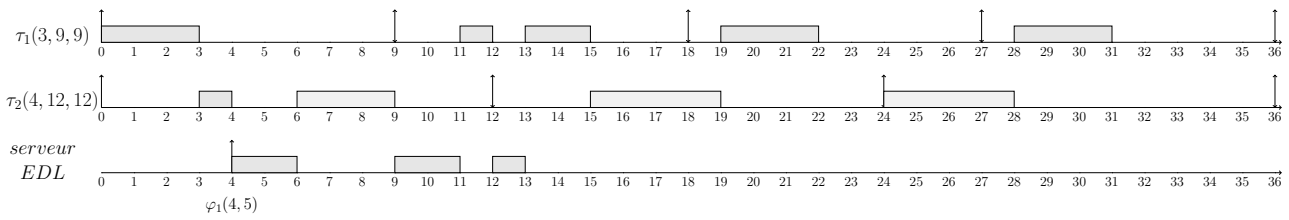


FIGURE 1.12 – Ordonnancement suivant le serveur EDL

4.3.2 L'algorithme TBS (Total Bandwith Server)

Description

Le serveur Total Bandwith Server (TBS) [SB94,SB96] est un mécanisme simple permettant de servir des tâches apériodiques conjointement aux tâches périodiques dans un environnement à priorités dynamiques utilisant EDF. A chaque fois qu'une requête apériodique entre dans le système, le serveur TBS lui assigne une échéance suivant sa largeur de bande temporelle (i.e. capacité CPU disponible). Lorsque la $k^{ième}$ requête apériodique arrive à l'instant $t = r_k$, une échéance fictive d_k lui sera assignée comme suit :

$$d_k = max(r_k, d_{k-1}) + \frac{C_k^a}{U_s} \tag{1.13}$$

où C_k^a est la durée d'exécution de la tâche apériodique qui arrive et U_s correspond au facteur d'utilisation du serveur (i.e. sa largeur de bande temporelle).
Par définition, $d_0 = 0$. De plus, quand une nouvelle échéance d_k est assignée, la largeur de bande déjà allouée aux requêtes précédentes est prise en compte par la valeur d_{k-1}.

Une fois que cette échéance fictive est assignée à la requête apériodique, celle-ci est ordonnancée conjointement avec les tâches périodiques suivant l'algorithme *EDF*.

Conditions d'ordonnançabilité

Le calcul des échéances fictives doit permettre que la proportion de temps processeur allouée aux requêtes apériodiques dans chaque intervalle de temps n'excède jamais le facteur d'utilisation du serveur U_s. Par conséquent, le test d'ordonnançabilité d'une configuration de tâches périodiques en présence d'un serveur TBS est défini par la condition nécessaire et suffisante formellement prouvée par Spuri et Buttazzo dans [SB96] :

Théorème 4.4 *Etant données une configuration de n tâches périodiques à échéances sur requêtes avec un facteur d'utilisation U_p, et des tâches apériodiques servies par un serveur TBS avec un facteur d'utilisation U_s, la configuration globale de tâches est ordonnançable si et seulement si :*

$$U_s + U_p \leq 1 \tag{1.14}$$

Illustration du calcul de l'échéance fictive avec TBS :

Nous considérons toujours la même configuration de tâches périodiques $\mathcal{T}$=$\{\tau_i(C_i,\ D_i,\ T_i)\} = \{\tau_1(3,\ 9,\ 9),\ \tau_2(4,\ 12,\ 12)\}$. La Figure 1.13 illustre le fonctionnement du serveur *TBS*. Une tâche apériodique $\varphi_1 = (4, 5)$ arrive à l'instant *t=4*. $U_p = \frac{3}{9} + \frac{4}{12} = \frac{2}{3}$ donc $U_s = 1 - U_p = \frac{1}{3}$. L'échéance fictive $d_1 = 19$ est calculée à $t = 4$. Ainsi la tâche apériodique φ_1 est exécutée conjointement avec les tâches périodiques selon l'algorithme *EDF*. On note un temps de réponse pour φ_1 égal à 11 unités de temps. On constate donc que EDL est plus performant dans la minimisation du temps de réponse. En effet, EDL retarde les tâches périodiques au plus tard pour exécuter au plus tôt les tâches apériodiques.

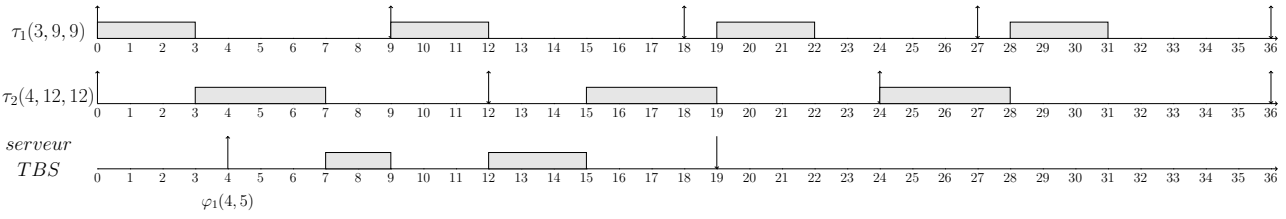


FIGURE 1.13 – Ordonnancement suivant le serveur TBS

4.3.3 L'algorithme TB* (Optimal Total Bandwith Server)

L'algorithme TB* [BS99] est une optimisation de l'algorithme TBS dans le sens où il permet d'assigner à chaque requête apériodique une échéance fictive plus petite que celle fournie par TBS. Ainsi, à chaque fois qu'une tâche apériodique arrive, le serveur TB* lui assigne en premier lieu une échéance selon l'algorithme *TBS*. Celui-ci va ensuite essayer de raccourcir au maximum cette échéance de manière à améliorer le temps

de réponse des requêtes apériodiques sans compromettre l'exécution des tâches périodiques. Si d_0^k correspond à la première échéance assignée à la requête apériodique J_k selon TBS, le processus de raccourcissement de cette échéance est appliqué de façon itérative jusqu'à ce qu'aucune amélioration ne soit plus possible, tout en conservant l'ordonnançabilité de la configuration des tâches périodiques. Par conséquent, si d_s^k est l'échéance assignée à la requête apériodique J_k au pas s, l'ordonnançabilité est conservée en assignant à J_k une nouvelle échéance donnée par :

$$d_k^{s+1} = t + C_k^a + I_p(t, d_k^s) \tag{1.15}$$

où t est le temps courant (correspondant à la date de réveil r_k de la requête J_k ou bien à la date de terminaison de la requête précédente), C_k^a est la durée d'exécution au pire cas requise par J_k, et $I_p(t, d_k^s)$ est l'interférence due aux jobs périodiques dans l'intervalle [t, d_k).
L'interférence périodique $I_p(t, d_k^s)$ est la somme de deux termes, $I_a(t, d_k^s)$ et $I_f(t, d_k^s)$. $I_a(t, d_k^s)$ correspond à l'interférence due aux jobs périodiques actifs au temps courant ayant des échéances strictement inférieures à d_k^s. $I_f(t, d_k^s)$ correspond à l'interférence future due aux jobs périodiques ayant leur date d'arrivée supérieure au temps courant *t* et ayant leur échéance inférieure à d_k^s. Leurs formules sont données ci-après,

$$I_a(t, d_k^s) = \sum_{\tau_i active; d_i < d_k^s} c_i(t) \tag{1.16}$$

et

$$I_f(t, d_k^s) = \sum_{i=1}^{n} max(0, \left\lceil \frac{d_k^s - next\_r_i}{T_i} \right\rceil) C_i \tag{1.17}$$

où $next\_r_i$ est le prochain réveil de la tâche τ_i supérieur ou égal à *t*.

Illustration du calcul de l'échéance fictive avec TB*
Nous considérons toujours la même configuration de tâches périodiques $\mathcal{T} = \{\tau_i(C_i, D_i, T_i)\} = \{\tau_1(3, 9, 9), \tau_2(4, 12, 12)\}$. La Figure 1.14 illustre le fonctionnement du serveur *TB**. Une tâche apériodique $\varphi_1 = (4, 5)$ arrive à l'instant *t=4*. $U_p = \frac{3}{9} + \frac{4}{12} = \frac{2}{3}$ donc $U_s = \frac{1}{3}$. A l'instant *t=4*, la première échéance fictive est calculée avec TBS en utilisant la formule (1.13), $d_1^0 = 19$. Une deuxième étape consiste à recalculer l'échéance en utilisant les formules (1.15) (1.16) (1.17). On obtient $d_1^1 = t + C_1^a + I_a(t, 19) + I_f(t, 19) = 4 + 5 + 3 + 3 = 15$ et $d_1^2 = t + C_1^a + I_a(t, 15) + I_f(t, 15) = 4 + 5 + 3 + 0 = 12$. Au troisième pas, on obtient $d_1^3 = 12$ donc on arrête le calcul. Par conséquent, la tâche apériodique φ_1 a l'échéance fictive 12 et est exécutée conjointement aux tâches périodiques selon l'algorithme *EDF*. On note un temps de réponse pour φ_1 de 8 unités de temps. On constate donc que TB* est plus performant que TBS dans la minimisation du temps de réponse.

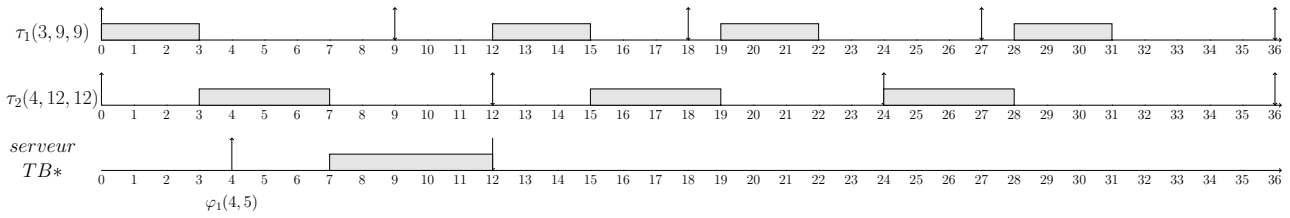


FIGURE 1.14 – Ordonnancement selon le serveur TB*

5 Ordonnancement et gestion de surcharge temporelle

5.1 Description du problème

Le phénomène de surcharge peut être le résultat d'un comportement anormal de l'environnement ou d'une mauvaise conception du système (l'activation de tâches sporadiques sans respect du délai minimal d'inter-arrivée ou l'arrivée d'un flux élevé de tâches apériodiques). Ce phénomène peut aussi être rattaché à une décision prise au préalable comme la considération des temps d'exécution moyens plutôt que des temps pire-cas dans l'ordonnancement ou la non prise en compte de durées d'exécution réelles qui s'avèrent supérieures aux temps pire-cas spécifiés.

En général, un système est en surcharge lorsque le besoin en ressources nécessaires pour exécuter les tâches du système est plus important que la disponibilité des ressources (en particulier les ressources processeur). Par ailleurs, si aucune précaution permettant d'éviter la surcharge n'est prise, une conséquence évidente sera la violation d'échéances de certaines tâches. En effet, les tâches manquant leur échéance vont retarder d'autres tâches qui à leur tour vont manquer leur échéance et ainsi de suite. On appelle ce phénomène, l'*effet domino* [SSD+94]. Un ordonnancement de type EDF est particulièrement sensible à cet effet, puisque celui-ci prend ses décisions d'ordonnancement suivant l'échéance de la tâche, donnant ainsi la plus grande priorité à une tâche qui est en train de manquer son échéance.

Par conséquent, l'algorithme d'ordonnancement doit pouvoir être conscient d'une situation de surcharge lorsque celle ci se produit et réagir à une telle situation. Un système peut ainsi être soumis à une surcharge de traitement, comme définie ci-après [Dar03] :

Définition 5.1 *Un système temps réel est en surcharge pendant une période donnée si sur cette période, l'utilisation du processeur dépasse la borne autorisée par l'algorithme d'ordonnancement mis en oeuvre.*

Une surcharge de traitement implique donc :

$$U_p = 1 + \epsilon > 1 \tag{1.18}$$

5.2 Solution à la surcharge de traitement

Ainsi une fois la surcharge détectée ou anticipée, il faut chercher à trouver une méthode appropriée pour diminuer les besoins en ressource processeur. Une méthode consiste à abandonner volontairement certaines tâches (décisions microscopiques). Une autre consiste à modifier la structure globale du système, i.e. le système est reconfiguré (décisions macroscopiques). Ainsi, on parle de deux genres d'ordonnancement : l'ordonnancement sans reconfiguration et l'ordonnancement avec reconfiguration. Les ordonnanceurs avec reconfiguration sont capables de passer d'une configuration à une autre plus adaptée à la charge du processeur suivant l'état courant du système. Ainsi, un système est composé de plusieurs services. Chaque service donnant en résultat une qualité de service plus ou moins bonne en fonction des différentes contraintes considérées par le système.
Les ordonnanceurs sans reconfiguration consistent à découper le système en deux parties : une partie critique et une partie non-critique. Ainsi la partie critique doit être exécutée et ne peut pas être remise en cause. Par contre, la partie non-critique représente la partie concernant la surcharge. Pour ordonnancer cette partie non-critique, il existe différentes approches : l'approche basée sur la valeur des tâches et l'approche utilisant des tâches bipartites.

5.3 Approches basées sur la valeur

L'approche basée sur la valeur consiste à ajouter un critère d'importance à une tâche. Par conséquent, lors d'une surcharge, une tâche moins importante qu'une autre se verra retardée par une tâche plus importante. Ainsi même s'il y a surcharge, la qualité de service exigée pourra être assurée par le respect des échéances de tâches jugées plus importantes que les autres. Par conséquent, il s'agit de déterminer l'importance d'une tâche vis-à-vis de l'application.

5.3.1 Critère d'importance

Les notions d'importance des tâches sont généralement représentées numériquement par des fonctions de valeur [JNC+89] dégagées par le système. En outre, une fonction de valeur v_i d'une tâche τ_i permet de définir l'importance de celle-ci par rapport à l'application ; elle peut être constante ($v_i = V_i$), être liée au temps d'exécution ($v_i = k.C_i$), etc. Ainsi lors d'une surcharge, les tâches ayant les fonctions de valeur les plus élevées seront privilégiées par rapport à d'autres. La fonction de valeur globale dégagée par le système permet de définir la qualité du service effectivement fournie par ce système. Elle représente la somme des valeurs dégagées par chacune des tâches et peut être modélisée par une fonction plus complexe [MC96, BPB+00].

Il existe trois classes d'ordonnanceurs basés sur les valeurs d'importance des tâches : les ordonnanceurs au mieux (*Best Effort* en anglais), les ordonnanceurs à garantie et les ordonnanceurs robustes.

5.3.2 Ordonnanceurs Best Effort

Avec ce type d'ordonnanceur, au réveil d'une tâche, celle-ci est systématiquement acceptée et insérée dans une file d'attente de tâches prêtes. Ainsi, les tâches sont ordonnancées, selon les priorités qui leurs sont affectées, jusqu'à la fin d'exécution ou la violation d'échéances. Cependant un inconvénient réside dans le fait qu'aucune prédiction de cas de surcharge n'est faite. Par conséquent ce type d'ordonnanceur peut s'avérer sensible à l'effet Domino.

5.3.3 Ordonnanceurs à garantie

En général, les ordonnanceurs à garantie acceptent une tâche sur la base du passage d'un test d'acceptation : si celui-ci est validé, la tâche est définitivement intégrée dans le système. C'est ainsi que fonctionne l'algorithme GED (*Guaranteed Earliest Deadline*) [BS93], qui est une variante de EDF avec test d'acceptation. Avec ce type d'ordonnanceur, toute tâche pouvant entraîner une surcharge est rejetée grâce au test et ainsi la surcharge est évitée. On pourrait aussi imaginer des ordonnanceurs qui refusent une tâche alors que le test d'acceptation réussit, par anticipation de l'activation prochaine d'une tâche (sporadique par exemple) plus importante.

5.3.4 Ordonnanceurs robustes

Les ordonnanceurs robustes sont souvent des ordonnanceurs avec test d'acceptation dans lesquels la politique de rejet n'est activée qu'en cas de refus d'une tâche. Quand le test d'acceptation n'est pas validé, l'ordonnanceur tente d'abandonner au besoin (en cas de surcharge temporelle par exemple) les tâches de moindre importance. C'est le fonctionnement de RED (pour Robust Earliest Deadline) [BS93]. D'autres ordonnanceurs robustes ne reposent pas sur un test d'acceptation, mais sur un test de démarrage, comme D^{over} [KS93] qui est issu de EDF. Les ordonnanceurs robustes s'avèrent généralement plus performants en cas de surcharge temporaire de traitements, que les deux catégories d'ordonnanceurs précédents [BS93].

5.4 Approches à tâches bipartites

Aujourd'hui, la qualité de service requise pour le bon fonctionnement du système est souvent définie par l'exécution d'une proportion de jobs respectant strictement leurs échéances temporelles au dépit de quelques jobs abandonnés par manque de temps en raison d'une surcharge de traitement. Ainsi de nombreux modèles ont été proposés pour prévenir le bon fonctionnement d'un système en pareille situation. Ces modèles tolèrent le fait qu'un nombre de jobs soient abandonnés. Nous citons par la suite quelques modèles prenant en compte la surcharge de traitement. Nous nous intéressons plus spécialement au modèle Skip-Over qui sera modifié dans le cadre de nos travaux pour prendre l'aspect énergétique en considération.

5.4.1 Mécanisme à échéance

Cette méthode consiste à implanter chaque tâche en deux versions [CHB79] : une version *primaire* et une version *secondaire*. La version *primaire* fournit un résultat de bonne qualité de service ; cependant, son temps d'exécution est inconnu. La version *secondaire* fournit un résultat juste acceptable dans un temps borné. Ce résultat fourni par le secondaire est donc de moindre qualité/précision que celui de la version *primaire*. L'implémentation du mécanisme permet d'élaborer une méthode d'ordonnancement de façon à exécuter les tâches périodiques dans le respect de leurs contraintes temporelles. Il existe deux techniques d'ordonnancement [Sil86] :
- La technique de la première chance, qui consiste à garantir d'abord une exécution fiable à toutes les versions secondaires avant de tenter l'exécution de leurs versions primaires respectives.
- La technique de la dernière chance, qui revient à déterminer la date de début d'exécution au plus tard de chaque version secondaire garantissant une exécution fiable. Ceci libère du temps au plus tôt mis à profit pour tenter d'exécuter les versions primaires. La méthode du mécanisme à échéance est donc particulièrement bien adaptée lorsque l'on utilise dans les tâches des algorithmes dont le temps d'exécution est extrêmement variable et qu'il est impossible d'utiliser des tests d'ordonnançabilité basés sur le calcul du WCET. Pour ces systèmes, on se contentera de vérifier la faisabilité de l'application en ne considérant que les versions secondaires de durées d'exécution très faibles garantissant la faisabilité de l'application dans le pire cas de surcharge.

5.4.2 Méthode du calcul imprécis

Le Calcul imprécis a été développé dans le cadre du projet CONCORD à l'université de l'Illinois par Liu, Lin et Natarajan. [LNL87,LLN87]. Cette technique est particulièrement adaptée aux tâches itératives dans le cas de surcharge passagère. Elle consiste à utiliser le temps où le processeur est oisif pour améliorer les résultats déjà obtenus et donc améliorer la qualité de service du système. Ainsi, les tâches périodiques gérées par ce modèle sont divisées en deux parties [LLS+91] :
- une partie mandataire (*mandatory part* en anglais) qui doit être impérativement exécutée pour fournir un résultat valable.
- une partie optionnelle (*optional part* en anglais) qui permet d'affiner le résultat de la partie mandataire à laquelle elle est associée. Cette partie peut ainsi être abandonnée à tout moment. Or, lorsqu'une partie optionnelle est abandonnée, il est parfois possible de conserver une partie du résultat qu'elle a fourni. Par conséquent, il existe deux classes de parties optionnelles :
- Les parties optionnelles *irrévocables* pour lesquels l'exécution doit être complète pour que les résultats qu'elles fournissent soient valables.
- Les parties optionnelles *révocables* qui affinent peu à peu le résultat de sorte qu'il soit toujours valable, et de plus en plus précis au cours de leur exécution. Elles peuvent ainsi être abandonnées à tout moment.

Si la partie mandataire d'une tâche est complètement exécutée dans le respect de son échéance, le résultat fourni est acceptable. Néanmoins, la qualité du résultat produit est d'autant meilleure que le processeur passe plus de temps à exécuter la partie optionnelle. Mais le temps d'oisiveté étant limité, il peut s'avérer intéressant d'arrêter le traitement d'une tâche une fois que son résultat est jugé acceptable dans l'objectif de permettre à d'autres tâches d'être exécutées.

Les parties optionnelles sont dotées d'une échéance à laquelle elles sont abandonnées. L'erreur d'exécution est d'autant plus grande quand la partie optionnelle est abandonnée tôt. L'objectif de ce modèle vise à minimiser l'erreur totale [LLS+91, LLB+94, SL95], ou l'erreur maximale [SL95] avec des algorithmes dérivés de l'algorithme *RM* ou *EDF*.

5.4.3 Modèle (m,k)-Firm

Hamdaoui et Ramanathan [HR95, Ram99] introduisent la notion d'échéance (m,k)-firm que l'on peut considérer comme une généralisation des échéances *firm* et *soft*. Selon leur définition, un système est soumis à une contrainte (m,k)-firm si au moins *m* jobs consécutifs sur *k* doivent être exécutés dans le respect de leur échéance. Dans le cas contraire, si dans une fenêtre de *k* jobs consécutifs, moins de *m* jobs respectent leur échéance alors le système est dit en échec dynamique. Une échéance (1,2)-firm signifie que la tâche ne doit pas manquer 2 jobs consécutifs. On pourrait assimiler une tâche à contraintes strictes comme celle ayant une échéance (1,1)-firm.

5.4.4 Modèle (m,k)-Hard

Bernat et Burns [BB97] propose la notion d'échéance (m,k)-Hard. Une contrainte (m,k)-Hard représente une contrainte temporelle stricte. Par conséquent, une tâche est considérée en échec si moins de *m* échéances sur *k* activations successives de la tâche sont respectées. L'approche consiste à utiliser un schéma d'exécution à double priorité (Dual priority [DW95]) qui repose sur un algorithme à réservation de bande de faible overhead, visant à ordonnancer de façon conjointe des tâches critiques (*hard*) et des tâches non critiques (*soft*). Toutefois, cette approche nécessite que les contraintes temporelles des tâches soient garanties par des tests d'ordonnançabilité hors-ligne.

5.4.5 Modèle Skip-Over

Dans ce modèle dit à pertes sporadiques, on considère une configuration de tâches périodiques temps réel fermes c'est-à-dire autorisant sporadiquement des violations d'échéances appelées pertes : $\mathcal{T}^* = \{\tau_i(C_i, T_i, s_i)\}$. Chaque tâche périodique τ_i est alors caractérisée par une durée d'exécution au pire cas C_i, une période T_i, une échéance relative égale à sa période, et un paramètre de pertes s_i. Celui-ci fixe le niveau de tolérance au non-respect des échéances des jobs des tâches.

Selon la terminologie introduite par Koren et Shasha dans [KS95], les jobs de chaque tâche peuvent être soit rouges (*red*) soit bleus (*blue*). Un job rouge doit s'exécuter avant son échéance alors qu'un job bleu peut être abandonné à tout moment.
Une configuration de tâches est dite *profondément rouge* (*deeply red* en anglais) si les jobs sont synchrones et rouges à l'instant *t=0*.

Le modèle Skip-Over répond à un certain nombre de règles élémentaires que nous énonçons ci-après :

- les $s_i - 1$ premiers jobs de chaque tâche doivent être rouges, $2 \leq s_i \leq \infty$.
- la distance entre deux pertes consécutives doit être au minimum de s_i périodes.
- si un job bleu est abandonné, les $s_i - 1$ prochains jobs de la tâche doivent être rouges.
- si un job bleu s'exécute complètement avant son échéance, le prochain job de la tâche reste bleu.

5.4.5.1 Conditions de faisabilité Koren et Shasha [KS95] ont prouvé que le problème de la faisabilité d'une configuration de tâches périodiques temps réel autorisant des pertes au sens Skip-Over est NP-difficile. Cela signifie que la seule possibilité de trouver une séquence valide est l'énumération de toutes les séquences d'ordonnancement possibles, ce qui n'est pas pratiquement réalisable.

Les auteurs ont énoncé la condition nécessaire de faisabilité suivante :

Théorème 5.1 *Une configuration de n tâches périodiques temps réel fermes* $\mathcal{T}^* = \{\tau_i(C_i, T_i, s_i)\}$ *est ordonnançable si :*

$$\sum_{i=1}^{n} \frac{C_i(s_i - 1)}{T_i.s_i} \leq 1 \tag{1.19}$$

Plus tard, Caccamo et Buttazzo [CB97] ont introduit le facteur d'utilisation équivalent U_p^*. Celui-ci représente la charge imposée au processeur dans le cas où seuls les jobs rouges sont exécutés.

Théorème 5.2 *Etant donnée une configuration de n tâches périodiques temps réel fermes* $\mathcal{T}^* = \{\tau_i(C_i, T_i, s_i)\}$*, le facteur d'utilisation équivalent du processeur est défini par :*

$$U_p^* = \sum_{i=1}^{n} \left(\left\lfloor \frac{L}{T_i} \right\rfloor - \left\lfloor \frac{L}{T_i.s_i} \right\rfloor \right) C_i \tag{1.20}$$

où L correspond aux échéances des jobs rouges sur l'hyperpériode $[0, H)$.

De plus, les même auteurs proposent une condition nécessaire et suffisante de faisabilité comme suit :

Théorème 5.3 *Une configuration de n tâches périodiques temps réel fermes profondément rouges* $\mathcal{T}^* = \{\tau_i(C_i, T_i, s_i)\}$ *est ordonnançable si et seulement si* $U_p^* \leq 1$.

On pourrait considérer que le modèle Skip-Over est un cas particulier du modèle (m,k)-firm et ce pour $m = k - 1$. Si une tâche a son paramètre s_i qui vaut 2 ceci implique qu'un job sur deux doit complètement être exécuté. De plus, si un job bleu de la tâche a manqué son échéance alors la prochaine requête de la même tâche doit être rouge. De plus, si une tâche a un paramètre s_i infini, ceci implique qu'aucune perte n'est autorisée et donc que la tâche est à contraintes strictes. Plus le paramètre de perte est grand, moins la tâche tolère le non-respect des échéances de ses jobs. Ainsi s_i peut être vu comme une métrique de QoS.

Dans le cadre de cette thèse, nous nous sommes basés sur deux algorithmes proposés par Koren et Shasha dans [KS95] utilisant la règle EDF que nous traiterons en détails dans le chapitre 3. Le premier algorithme s'appelle RTO (*Red Task Only*) et le second s'appelle BWP (*Blue When Possible*). Comme son nom l'indique RTO permet d'ordonnancer les jobs rouges et rejette systématiquement les jobs bleus. Ainsi, la QoS est constante et dépend directement des paramètres de pertes des tâches. Par contre, BWP ordonnance les jobs bleus prêts quand il n'y a aucun job rouge en attente du processeur. Par conséquent, BWP permettra d'améliorer la qualité de service résultante par rapport à RTO.

5.4.6 Amélioration du modèle Skip-Over

Un inconvénient majeur de BWP réside dans le fait que les jobs bleus ayant une priorité inférieure aux jobs rouges peuvent être conduits à abandonner leur exécution, partiellement ou quasi-totalement, au réveil d'un job rouge. On constate un gaspillage du temps processeur due à l'exécution des job bleus abandonnés au profit des jobs rouges. De ce fait l'algorithme *Red tasks as Late as Possible* (RLP) a été proposé [Mar06] supprimant cet inconvénient. Cet algorithme consiste à retarder au maximum l'exécution des jobs rouges en utilisant l'algorithme EDL [CC89], de manière à minimiser le nombre de jobs bleus abandonnés. Les simulations montrent que cet algorithme offre une meilleure QoS que l'algorithme BWP. Toutefois, un inconvénient réside toujours dans le fait que du temps processeur peut être gaspillé par un job bleu qui n'a pas pu s'exécuter complètement avant son échéance. De ce fait, l'algorithme RLP/T (*Red tasks as Late as Possible with blue acceptance Test*) est proposé [Mar06, MSC05] en rajoutant un test d'acceptation sur les jobs bleus. Ainsi un job bleu ne sera exécuté que s'il n'y a pas de job rouge à l'état prêt et si le temps processeur disponible est suffisant pour que le job s'exécute complètement avant son échéance.

6 Conclusion

Ce chapitre avait pour objectif de présenter quelques notions de base nécessaires à la compréhension du contexte de cette thèse. Nous avons dans un premier temps, rappelé les caractéristiques générales des systèmes temps réel. Puis nous avons décrit la terminologie et des concepts relatifs au domaine du temps réel et plus précisément concernant l'ordonnancement. Dans un deuxième temps, nous avons présenté les principaux algorithmes d'ordonnancement qui permettent d'ordonnancer des tâches périodiques et apériodiques. Ensuite nous avons traité du problème de surcharge temporelle qui se produit lorsque la charge de traitement infligée au système s'avère supérieure à la capacité de traitement de ce système. Nous avons décrit les approches d'ordonnancement pour traiter ces cas de surcharge plus précisément avec le modèle Skip-Over que nous avons l'intention d'étendre aux systèmes temps réel présentant à la fois des surcharges de traitement et des surcharges dites énergétiques. Dans le chapitre suivant, nous introduirons les système temps réel dits autonomes car alimentés par l'énergie ambiante. Et nous présenterons les algorithmes d'ordonnancement spécialement adaptés aux configurations de tâches périodiques dans ces systèmes.

Chapitre 2

Ordonnancement temps réel sous contraintes énergétiques

Sommaire

Dans ce chapitre nous nous intéressons au problème d'ordonnancement de tâches caractérisées par une durée d'exécution contrainte par le temps mais aussi par des besoins en énergie pour réaliser cette exécution. Dans un premier temps, nous décrivons les systèmes embarqués puis plus précisément les réseaux de capteurs sans fil. A l'heure actuelle, ceux-ci constituent la majorité des applications embarquées bâties autour de systèmes autonomes du point de vue énergétique. Ensuite, nous nous focalisons sur le stockage et la récupération de l'énergie renouvelable pour alimenter ces systèmes autonomes. Nous décrivons un état de l'art sur les technologies associées à la récupération d'énergie. Ensuite, nous décrivons le problème de l'autonomie énergétique dans un système souvent qualifié d'énergétiquement neutre. Enfin, nous décrivons différentes techniques d'ordonnancement d'abord à visée de minimisation de la consommation énergétique puis à visée d'autonomie énergétique.

1 Les systèmes embarqués

1.1 Définition

Wayne Wolf [Wol00] définit un système embarqué comme tout dispositif programmable, sans pour autant être un ordinateur même si les ordinateurs sont très souvent utilisés pour construire des systèmes informatiques embarqués.

Un système embarqué (appelé aussi système enfoui) est un système électronique/informatique autonome. Par autonome, on entend qu'il n'est pas attaché de façon filaire au réseau électrique. Il est généralement composé d'un ou plusieurs microprocesseurs destinés à exécuter un ensemble de programmes qui sont définis préalablement lors de sa conception et stockés dans des mémoires. Un système embarqué se caractérise par sa très forte connexion avec l'environnement dans lequel il est installé et avec lequel il interagit. Cette interaction

se décrit comme suit :
- Les informations en entrée du système embarqué proviennent de capteurs.
- Le système embarqué effectue des calculs dont les résultats sont envoyés vers des actionneurs ou des périphériques de sortie tels que des écrans d'affichage (cf. la Figure 2.1).

Quelles que soient sa nature et sa complexité, une application embarquée intègre un système de contrôle et un système contrôlé. Le système contrôlé est le procédé ou l'environnement qui interagit avec le système de contrôle. Le système de contrôle est l'ensemble des éléments logiciels et matériels (microprocesseurs...), le logiciel ayant une fonctionnalité à délivrer spécifique à l'application.

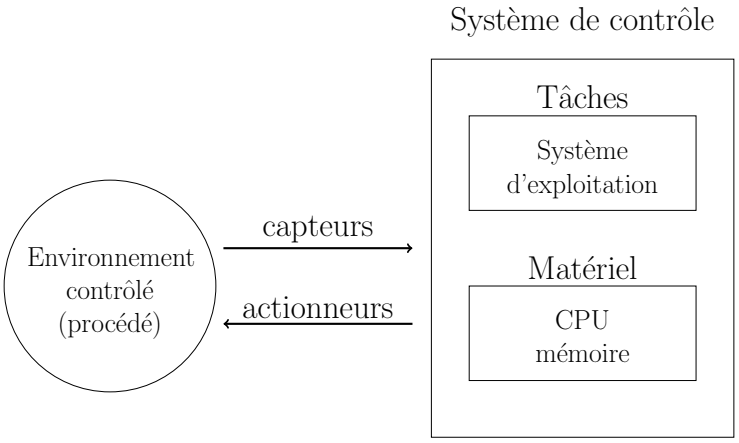


FIGURE 2.1 – Description schématique d'un système embarqué

1.2 Exemple : les réseaux de capteurs sans fil

Les réseaux de capteurs sans fil (RCSF) (*Wireless Sensor Network*) constituent une thématique de recherche en pleine expansion. Ils sont omniprésents dans de nombreux domaines en particulier celui de la détection d'intrusions, du monitoring environnemental, du monitoring médical, etc. Un RCSF est ainsi constitué de noeuds qui communiquent entre eux et avec une station de base. Un noeud de capteur constitue un exemple significatif de système embarqué temps réel, celui-ci recevant des données de son environnement via un ou plusieurs capteurs physiques qui lui sont attachés. Par capteur physique, on entend un instrument qui mesure une grandeur physique et la transforme en une grandeur numérique.
La Figure 2.2 montre le schéma d'un système de capteur sans fil. Un système de capteur sans fil est consti-

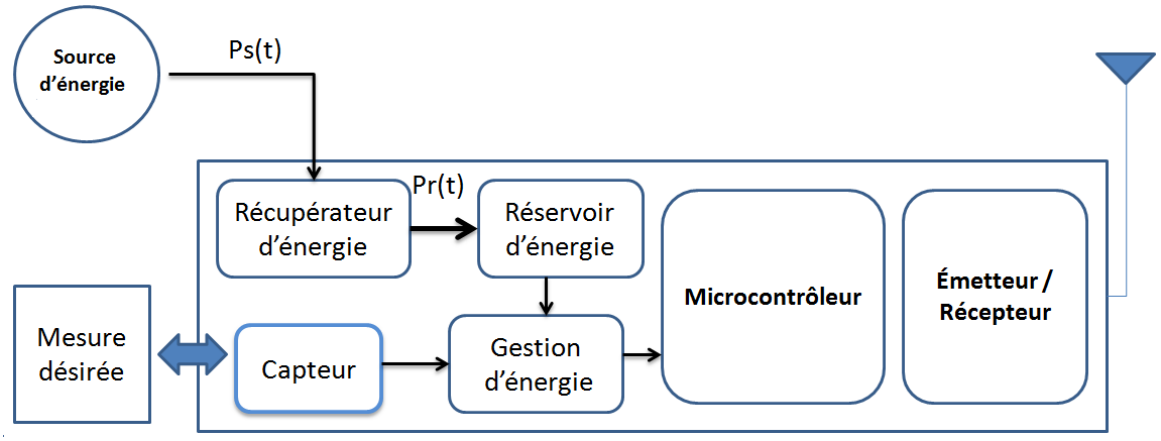


FIGURE 2.2 – Exemple d'un système capteur sans fil

tué de trois éléments principaux : le récupérateur d'énergie, le réservoir d'énergie et le microprocesseur. Le récupérateur d'énergie (par exemple une cellule photovoltaïque) a pour rôle de récupérer l'énergie délivrée par une source (par exemple le soleil) pour ensuite la convertir en énergie électrique. Cette énergie électrique va

recharger un réservoir d'énergie. Ce réservoir peut être une batterie ou un super-condensateur et est caractérisé par sa capacité, notée C. Nous supposons qu'il est idéal dans le sens qu'il peut toujours être chargé à sa capacité maximale. A tout instant *t*, l'énergie stockée dans le réservoir est notée *E(t)* et sa valeur est bornée par deux constantes telles que $0 \leq E(t) \leq C$.

L'énergie reçue par le réservoir pendant un intervalle $[t_1, t_2]$ est notée $E_r(t_1, t_2)$ et se calcule comme suit :

$$E_r(t_1, t_2) = \int_{t_1}^{t_2} P_r(x).dx \tag{2.1}$$

où $P_r(x)$ est la puissance instantanée reçue à l'instant *x* incluant toutes les pertes d'énergie occasionnées par la mise en oeuvre matérielle du processus de récupération de l'énergie.

L'alimentation d'un noeud de capteur (appelé simplement capteur) constitue un point primordial. Son choix dépend non seulement des caractéristiques matérielles du capteur mais aussi des différents traitements qu'il aura à effectuer. La plupart des RCSFs sont utilisés dans des secteurs hostiles où l'humain ne peut y accéder facilement (satellite dans l'espace, fonds marins, environnements nucléaires, etc..). Par conséquent, recharger ou remplacer une batterie se révèle pour la plupart des applications une tâche sinon impossible tout du moins extrêmement onéreuse et périlleuse. D'où l'importance de bien estimer l'énergie nécessaire pour permettre le bon fonctionnement du système et d'assurer son autonomie sur une très longue durée sans intervention extérieure. Ainsi, on pourra se satisfaire de six mois à un an comme durée d'autonomie pour certaines classes de systèmes tels que des capteurs médicaux non intrusifs. Par contre, une telle durée sera jugée insuffisante pour d'autres applications compte tenu des coûts et des risques associés au remplacement des batteries. C'est la raison pour laquelle depuis quelques années, on s'oriente vers un procédé qui consiste à ne jamais changer de batterie sachant que celle-ci peut se recharger en continu à partir de l'environnement.

La récupération de l'énergie ambiante (*Energy Harvesting* en anglais) est de plus en plus répandue dans les RCSFs, l'environnement offrant de multiples sources d'énergie. Cette technique consiste à associer au système consommateur d'énergie, un réservoir d'énergie assurant le stockage temporaire de l'énergie récoltée par l'environnement. La mise place d'une telle technologie nous amène à décrire un système embarqué sous forme de trois composants qui sont :
- Le récupérateur d'énergie (*energy harvester*, en anglais),
- Le réservoir d'énergie permettant le stockage d'énergie (batterie et/ou super-condensateur),
- Le consommateur d'énergie (le système informatique).

Le dimensionnement est une question clé dans la conception de tout système embarqué. Il va donc concerner ici la taille minimale de la batterie (volume et poids) qui permettra un fonctionnement perpétuel du système compte tenu d'une part de l'énergie consommée et d'autre part de l'énergie produite par la source environnementale.

1.3 Stockage et récupération d'énergie

1.3.1 Eléments de stockage d'énergie

L'énergie récupérée est stockée dans un réservoir qui peut être une batterie et/ou un supercondensateur.

1.3.1.1 Batteries Une batterie est un dispositif électrochimique qui convertit l'énergie chimique en énergie électrique grâce à une réaction chimique d'oxydo-réduction. L'énergie électrique fournie par ces réactions électrochimiques est exprimée en watt heure (Wh). Ces réactions sont activées au sein d'une cellule élémentaire, entre deux électrodes baignant dans un électrolyte, lorsqu'une charge électrique est branchée à ses bornes (un moteur électrique par exemple). Le courant circulant à partir des bornes de la batterie constitue le "circuit externe". Une batterie est caractérisée par :
- sa tension, exprimée en volts (V), qui représente le potentiel d'oxydo-réduction entre les deux électrodes de la batterie.
- sa charge électrique en ampère-heure (Ah), qui correspond à la quantité d'électrons que peut contenir la batterie.

- sa capacité de charge électrique qui représente la charge maximale fournie par la batterie, correspondant à un cycle complet de décharge (entre le moment où elle est chargée à sa pleine capacité et le moment où elle est complètement déchargée).
- sa cyclabilité, exprimée en nombre de cycles, caractérise la durée de vie de la batterie, c'est-à-dire le nombre de fois où celle-ci peut restituer un niveau d'énergie supérieur à 80 % de son énergie nominale.
- sa densité d'énergie massique (ou volumique), exprimée en watt heure par kilogramme, (Wh/kg) (ou en watt heure par litre, Wh/l), qui définit l'autonomie de la batterie et représente la quantité d'énergie stockée par unité de masse (ou de volume) de la batterie.
- sa densité de puissance massique, en watt par kilogramme (W/kg), représente la puissance (énergie électrique fournie par unité de temps) que la batterie peut délivrer.

L'utilisation de batteries rechargeables au lieu des batteries classiques, va ainsi permetre de rallonger la durée de vie des systèmes embarquées dans lesquels elles sont intégrées. Quand une batterie classique est épuisée, le système dépourvu à jamais d'énergie, n'est plus fonctionnel et on dit d'ailleurs que le noeud de capteur est mort [CC05].

Pour allonger sa durée de vie, il faudrait alors augmenter la capacité de la batterie classique. Cela supposerait d'augmenter ses dimensions, décision inenvisageable pour les microsystèmes très contraints en termes de poids et de volume. A contrario, dans le cas d'une batterie rechargeable, celle-ci se recharge par une source d'énergie naturelle et inépuisable comme par exemple celle fournie par un panneau photovoltaïque. Ainsi l'application n'utilisera l'énergie disponible dans la batterie que quand le système est en surcharge énergétique (i.e. la puissance consommée du système est supérieure à la puissance transmise par la source). C'est ainsi que le surplus d'énergie photovoltaïque produit pendant la journée servira à alimenter le système pendant la nuit lorsqu'aucune énergie ne peut être drainée de l'environnement.

La Figure 2.1 présente une comparaison entre les différents types de technologies utilisées pour le stockage d'énergie.

Type de batterie	**Tension nominale**	**Densité d'énergie massique (Wh/l)**	**Densité d'énergie Volumique (Wh/l)**	**Durée de vie**	**Coût**
Ni-Cd	1,2	40	100	Longue	+
Ni-MH	1,2	90	245	Moyenne	++
Li-ion	3,6	125	440	Longue	+++
Li-polymer	3,1	300	800	Longue	++++

TABLE 2.1 – Comparaison entre les différents types de batteries rechargeables

Les microbatteries de type Li-polymer sont très performantes pour alimenter un microsystème autonome. Elles ont un coût élevé, mais leurs densités d'énergie remarquables, la simplicité de leur recharge et leur utilisation sous forme intégrée (technologie en couches minces) [DlJ03] et [WWWR02] font qu'elles conviennent à l'alimentation des petits systèmes électroniques autonomes.

En outre, les chercheurs se sont intéressés aux batteries à base de silicium, plus performantes que celles utilisant le traditionnel couple lithium-ion. Ce type de batterie est obtenu en remplaçant le graphite par du silicium. Cette solution consisterait à conduire des électrons avec un blindage en silicium qui auraient pour principal avantage d'éviter de détériorer la batterie. Cela évite aussi que celle-ci ne perde sa capacité au fil du temps contrairement au graphite qui est un matériau qui s'use et perd sa capacité au fur et à mesure des charges/décharges de la batterie. Récemment, la société PROLLION, start-up créée à l'initiative du CEA-Liten et du groupe Alcen, a mis sur le marché la batterie EnerSi 250. Cette batterie première du genre au monde fabriquée en série à partir d'un accumulateur de plus de 250 Wh/kg. L'énergie massique est obtenue grâce à une électrochimie à base de silicium offrant une autonomie, une puissance et une sécurité remarquables

1.3.1.2 Supercondensateurs Les supercondensateurs se révèlent être des réservoirs d'énergie compétitifs par rapport aux batteries dans le monde des petits objets autonomes. Un super-condensateur (cf. Figure 2.3) est un condensateur électrochimique qui a une capacité de stockage d'énergie exceptionnelle en comparaison des

condensateurs traditionnels.

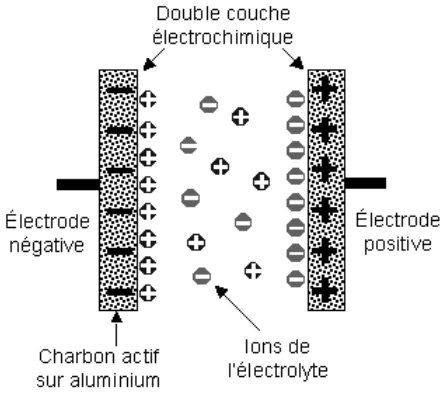


FIGURE 2.3 – Structure d'un supercondensateur

Par rapport à une batterie rechargeable, les super-condensateurs sont caractérisés par [Sch06] :

- une puissance très élevée pendant le cycle de charge/décharge
- une cyclabilité importante (milliers de cycles de charge/décharge)
- une tolérance aux basses températures pouvant atteindre les -40°C sachant que les batteries ne fonctionnent plus correctement à partir d'une température inférieure à -10°C.
- la rapidité de leur recharge. Une batterie peut être endommagée suite à une charge trop rapide.

1.3.2 Eléments de récupération d'énergie

L'activité humaine a provoqué d'énormes bouleversements à l'échelle continentale et planétaire, en particulier durant le siècle dernier par l'usage intensif de combustibles fossiles. La plupart des combustibles fossiles ou minéraux polluent l'atmosphère et sont responsables de l'effet de serre à l'origine du réchauffement climatique qui menace notre planète et les générations futures. Ces combustibles sont aussi limités dans leur stock. Ils ne constituent pas des sources d'énergie renouvelables dans le sens où l'humain les consomme plus rapidement que ces sources ne se régénèrent. Cela a principalement motivé les nouvelles techniques visant à exploiter les sources d'énergies environnementales dites renouvelables. Celles-ci sont des sources propres dont la consommation ne diminue pas la ressource à l'échelle humaine. C'est pourquoi, on les sollicite de plus en plus dans de multiples domaines d'applications de la plus petite à la plus grande échelle et en particulier dans le domaine de l'embarqué.

La récupération d'énergie est un domaine en plein essor. En effet, si les systèmes de récupération d'énergie existent depuis très longtemps comme les dynamos pour s'éclairer à vélo, de nouveaux systèmes ingénieux sont aujourd'hui mis au point pour récupérer l'énergie depuis de multiples sources. Diverses techniques peuvent être employées : exploiter les mouvements d'un corps, la chaleur, les vibrations ou encore les ondes électromagnétiques. Chaque jour, de nouvelles découvertes scientifiques permettent d'exploiter plus facilement ces sources d'énergies.

En particulier, les avancées dans le domaine des nanotechnologies permettent de créer des systèmes miniatures et très sensibles pour récupérer de faibles sources d'énergies qui toutefois s'avèrent suffisantes pour faire fonctionner des capteurs sans fil.

Certaines technologies peuvent s'implanter dans une chaussure pour récupérer une partie de l'énergie dépensée lors de la marche. On peut également créer de l'électricité en exploitant la chaleur dégagée par le corps humain pour alimenter des capteurs bio-médicaux. Dans la suite, nous décrirons brièvement les sources d'énergie renouvelables les plus couramment utilisées à l'heure actuelle.

1.3.2.1 Sources d'énergie renouvelables

L'énergie solaire L'énergie solaire est l'énergie fournie par les rayons de soleil. Le soleil est la source d'énergie la plus puissante. Certaines technologies dites actives transforment l'énergie solaire en une forme d'énergie électrique ou thermique. Par exemple les cellules photovoltaïques transforment le rayonnement solaire en énergie électrique. Elles sont réalisées à l'aide de matériaux semi-conducteurs, en particulier le silicium. En disposant une cellule photovoltaïque face au soleil, une tension électrique apparaît. Toutefois pour obtenir le maximum de rayonnement sur la surface de cellules solaires, il faut trouver l'angle d'inclinaison et l'orientation des cellules photovoltaïques optimaux [Rei02]. Généralement, le rayonnement solaire est mesuré suivant la valeur de l'irradiance exprimée en $W.m^{-2}$ ou en $\mu W.cm^{-2}$.

Randall et al. ont proposé un modèle permettant d'estimer l'énergie disponible à l'intérieur (énergie lumineuse dite *in-door*) en prenant en compte toutes les sources lumineuses présentes [RJ02, RJ03, RBP+04]. La Figure 2.2 affiche la quantité d'énergie disponible selon la localisation de la cellule solaire. Nous remarquons que l'énergie mesurée diffère selon qu'il s'agit d'un rayonnement solaire ou d'un éclairage artificiel.

Energie	**Intérieur**			**Extérieur**	
	éclairage tamisé	éclairage de travail	éclairage de couloir	temps couvert	temps ensoleillé
E(lux)	80	150	740	3030	100000
$I(\mu W.cm^{-2})$	67	125	616.7	2525	100000

TABLE 2.2 – Energie photovoltaïque suivant l'éclairage

L'énergie électromagnétique 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. 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.

Un des premiers travaux dans ce domaine concerne la réalisation d'un système RF autonome nommé "RF badge". Ce système utilise l'énergie délivrée pendant la marche [KKPG98]. Ainsi, le générateur est monté sur une chaussure et permet de fournir une puissance moyenne de 0,2W. Un autre exemple d'application est la montre à secousses. En 1770, l'ancêtre de la montre automatique, la "montre à secousses" a été créée par A. L. Perrelet. Plus récemment, Seiko a fabriqué une montre alimentée par le mouvement du bras. Le générateur donne $5\mu W$ en moyenne et $1mW$ quand la montre est fortement secouée [PS05].

La piézoélectricité La piézoélectricité se définit par la propriété que possèdent certains corps à se polariser électriquement sous l'action d'une force mécanique et, réciproquement, à se déformer lorsque nous leur appliquons un champ électrique. Les matériaux piézoélectriques sont très nombreux : le quartz, les piézocéramiques comme le plomb zirconium de titane (PZT), les films fins de nitrure d'aluminium (AlN) ou l'oxyde de zinc, etc.. Le quartz est un des matériaux le plus connu utilisé dans les montres pour créer des impulsions d'horloge. Aujourd'hui, les PZT, des céramiques synthétiques, sont les plus utilisées dans l'industrie.
Un exemple d'application décrit dans [SP01] consiste enla génération d'énergie pendant la marche.

L'énergie thermique Le générateur thermoélectrique est associé à l'effet Seebeck découvert par un physicien allemand Thomas Johann Seebeck en 1821. Cet effet se produit lors du passage d'un courant électrique dans un matériau soumis à un flux de chaleur et à un gradient thermique. Ainsi sur ce principe, les thermogénérateurs constitués d'un assemblage de jonctions, utilisent l'effet Seebeck pour convertir la différence de température entre deux milieux, en électricité. Cette différence de potentiel se décrit comme suit :

$$V = S_{AB}(T_0.T_1) \tag{2.2}$$

où S_{AB} est le coefficient de Seebeck, T_0 et T_1 sont les températures des deux milieux (mesurées en degrés Kelvin). Le corps humain peut être un exemple de source d'énergie thermique [SP04]. En général, la température du corps humain est régulée aux environs de 37°C. Cependant il faut garder cette température intérieure constante

quelle que soit la température de l'air ambiant. Par conséquent, quand la température extérieure est plus faible ou plus élevée que celle du corps, un flux de chaleur thermique se crée, d'où une dissipation d'énergie.

Douseki présente un exemple de système autonome basé sur l'effet thermoélectrique [TDUIH03]. Le générateur proposé fournit une énergie de 1.6mW pour alimenter le transmetteur de ce système. Ce générateur accepte des sources chaudes (tension positive) ou froides (tension négative).

Un inconvénient réside dans le fait que sur une échelle microscopique, on ne peut pas garantir un gradient de température suffisant pour faire fonctionner un générateur thermoélectrique. Les microsystèmes autonomes de très faible surface ne peuvent donc bénéficier de cette technologie.

Source d'énergie	**Dimensions**	**Densité de puissance**	**Excitation**	**Rendement**
Thermoélectrique : Thermo LifeTM [LIF10]	$0.2cm^2$	$60\mu W.cm$	$-25C$	12%
Piézo-électrique : [MAB07]	$800\mu m * 1200\mu m$, 1 mm3, poutre MEMS	$40\mu W.cm^{-2}$	$39ms^{-2}$, 1.3 kHz	NC
Electrostatique : [DCJ+07]	104 g	$333\mu W.cm^{-2}$	50 Hz	60%
Electromagnétique : [BTM+07]	0.15 cm3	$307\mu W.cm^{-2}$	$0.59ms^{-2}$	30%
Energie solaire : (Photovoltaïque)				
Extérieur	très ensoleillé	10-20 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 $\mu W.cm^{-2}$	Sans soleil	10%

TABLE 2.3 – Comparaison des principaux systèmes de récupération d'énergie [Wal11]

1.3.2.2 Comparaison des sources d'énergie Le tableau 2.3 permet de faire une comparaison entre quelques sources de récupération d'énergie pour des microsystèmes autonomes. Les sources d'énergie ne délivrent pas une puissance linéaire selon la variation de leurs dimensions [Wal11]. Les systèmes de récupération vibratoire (piézo-électrique, électrostatique, électromagnétique) montrent les meilleurs rendements. Bien qu'ils soient les plus performants, ils sont aussi les plus sélectifs par rapport aux caractéristiques de leurs stimuli : une vibration ayant une fréquence et une amplitude évoluant sur une large plage sera très difficilement convertie en énergie électrique. De plus, 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 [Wal11]. Actuellement, les meilleures cellules photovoltaïques ont un rendement d'environ 15%. Cela implique donc que 85% de l'énergie arrivant sur la surface de la photopile n'est pas transformée en électricité. Cela représente une perte considérable. Cependant l'énergie récupérée à partir de la lumière reste toutefois accessible à un coût raisonnable.

2 Problématique de l'autonomie des systèmes embarqués

Dans ce paragraphe, nous introduisons une nouvelle terminologie.

2.1 Nécessité d'une nouvelle terminologie

Ordonnancement avec clairvoyance énergétique : Un ordonnanceur est clairvoyant énergétiquement lorsqu'il a connaissance a priori de la quantité d'énergie produite par la source qui alimente le système. Dans le cas

contraire, l'ordonnanceur est dit non clairvoyant du point de vue énergétique.

Ordonnancement avec clairvoyance de traitement : Un ordonnanceur est clairvoyant temporellement lorsqu'il connait a priori toutes les caractéristiques des tâches futures à exécuter.

Ordonnancement totalement clairvoyant : Un ordonnanceur est totalement clairvoyant s'il 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 valide temporellement de manière à satisfaire toutes les contraintes temporelles de cette configuration sans prendre en compte ses contraintes énergétiques.

Configuration de tâches ordonnançable : Une configuration de tâches est ordonnançable s'il existe au moins un ordonnanceur capable de créer une séquence d'ordonnancement satisfaisant toutes les contraintes temporelles et énergétiques de cette configuration. A contrario, elle est dite non ordonnançable quand il y a non-respect soit de ses contraintes temporelles, soit de ses contraintes énergétiques.

2.2 Nécessité d'un modèle de tâches adapté aux contraintes énergétiques

Considérons une configuration de tâches périodiques temps réel strict (i.e. sans pertes) $\mathcal{T}$. Chaque tâche τ_i est caractérisée par :

- C_i : sa durée d'exécution au pire cas (WCET, *Worst Case Execution Time*),
- D_i : son échéance relative,
- T_i : sa période,
- E_i : sa demande énergétique au pire cas (WCEC, *Worst Case Energy Consumption*).

Les tâches périodiques sont supposées préemptables, indépendantes et initialement synchrones.

2.3 Nécessité de politiques d'ordonnancement spécifiques

Considérons une configuration $\mathcal{T}$ constituée de trois tâches périodiques indépendantes préemptables et à échéances requêtes, $\mathcal{T} = \tau_i(C_i, D_i, T_i, E_i)$; $\tau_1(3, 6, 6, 8)$, τ_2(3,10,10,8) et $\tau_3(1, 15, 15, 3)$. Les tâches sont réveillées à l'instant *t=0* et sont ordonnancées selon EDF sur un processeur alimenté par une batterie de capacité $C = 6$. Cette batterie est rechargée par une source environnementale. Elle reçoit une puissance constante $P_r(t) = P_r = 2$. La Figure 2.4 illustre l'ordonnancement de cette configuration selon EDF. Nous remarquons

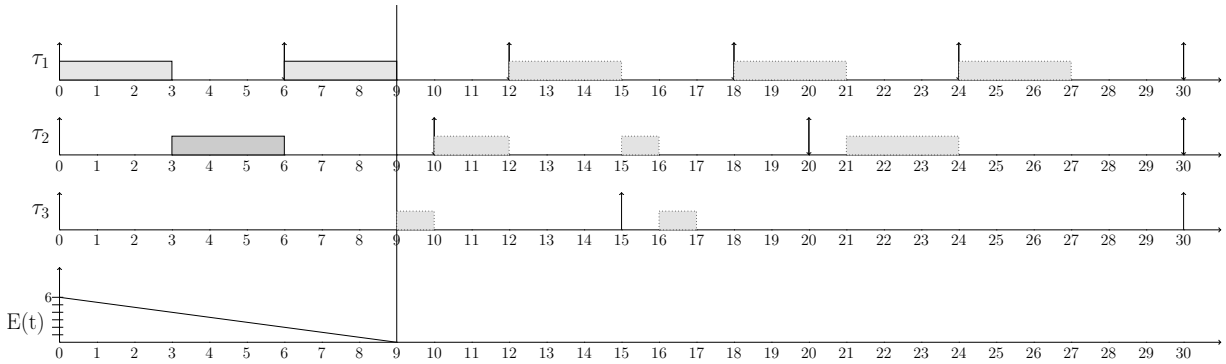


FIGURE 2.4 – Ordonnancement selon EDF

qu'à l'instant *t=9*, l'énergie dans la batterie est épuisée. Les tâches prêtes sont alors interrompues par manque d'énergie. Il faut donc prévoir une durée pour recharger la batterie et pouvoir ensuite continuer l'exécution des tâches. Cependant cette durée de temps perdu pendant la recharge de la batterie doit être calculée finement de

manière à prévenir la violation des échéances des tâches et l'écroulement du système. En résumé, EDF n'est pas un algorithme clairvoyant énergétiquement et n'est pas adapté à ce type de système temps réel.

La contrainte énergétique rajoute une nouvelle dimension aux problèmes d'ordonnancement. Nous devons trouver un algorithme capable d'exécuter toutes les tâches dans les délais exigés et sans manquer d'énergie. Ainsi, dans le cadre de cette thèse, nous devons considérer conjointement deux contraintes : la contrainte temporelle et la contrainte énergétique. Les tâches peuvent être exécutées s'il y a assez d'énergie dans le réservoir (celui-ci est rechargé par une source d'énergie) sachant qu'il faut garantir leurs exécutions dans le respect de leurs contraintes temporelles. De nouveaux types d'ordonnanceurs sont proposés dans la littérature pour les systèmes temps réel monoprocesseur soumis à des contraintes temporelles et énergétiques. Ces techniques d'ordonnancement doivent créer une séquence valide de la configuration de tâches constituant l'application. Quand l'on parle de validité de la séquence d'ordonnancement, ceci est valable tant du point de vue temporel que du point de vue énergétique. Nous allons dans un premier temps explorer différentes techniques existantes visant à ordonnancer une configuration de tâches tout en satisfaisant leurs contraintes temporelles et énergétiques.

3 Politiques d'ordonnancement existantes

3.1 Approches à visée de minimisation de la consommation énergétique

Pour réduire la consommation énergétique d'un système informatique, deux types de méthodes existent.

- Les méthodes connues sous le nom de DPM (*Dynamic Power Management*) permettent de gérer dynamiquement l'activité du système en faisant des commutations d'un état de veille à un état actif et vice versa [GN06]. En effet, les méthodes DPM permettent de réduire la consommation de puissance du système sans dégrader considérablement les performances en basculant en mode veille lorsque qu'il n'y a aucune tâche à exécuter et de rebasculer en mode actif quand le processeur est sollicité. Ces méthodes utilisent des processeurs qui disposent d'une fonction de mise en veille. Par conséquent, le processeur est éteint temporairement lorsque cela s'avère intéressant ou nécessaire. Ainsi, il ne consommera que peu ou pas d'énergie (appelée énergie statique) dans cet état de veille. On peut citer par exemple le processeur Intel 80200 qui possède trois modes de fonctionnement, dont deux modes à faible consommation énergétique qui diffèrent par le nombre de composants mis en veille [NG06].

- Les méthodes connues sous le nom de DVFS (*Dynamic Voltage and Frequency Scaling*) permettent de modifier la fréquence du processeur quand cela est nécessaire. Par conséquent, ces méthodes utilisent des processeurs conçus pour réduire l'énergie utilisée en variant la tension d'alimentation et donc la fréquence de fonctionnement [NG06]. Ainsi, si la fréquence du processeur est réduite, le job en exécution mettra plus de temps à s'exécuter. Par exemple, si la fréquence est réduite de moitié, le job mettra deux fois plus de temps pour s'exécuter. Comme la consommation énergétique est une fonction quadratique de la fréquence, le fait de diminuer celle-ci impacte significativement la consommation énergétique. Dans ce contexte, économiser l'énergie se fait au prix d'un étirement des durées d'exécution qu'il s'agit de contrôler de sorte que l'on continue de respecter les contraintes temporelles. Par conséquent, en utilisant cette catégorie de processeurs, il y aura possibilité de ralentir la fréquence de fonctionnement et ainsi ralentir la tâche courante de manière à réduire sa consommation énergétique tout en satisfaisant ses contraintes temporelles.
On peut citer par exemple, le processeur Transmeta Crusoe, le processeur lpARM (UC Berkeley) et le processeur Intel Pentium 4M [Gru02, SFR03]. Ces processeurs doivent non seulement établir un ordre d'exécution entre les traitements afin que chacun puisse être entièrement exécutés avant leur échéance mais aussi déterminer à tout instant la fréquence de fonctionnement admissible du processeur de façon à minimiser leur consommation en énergie.

Des auteurs se sont intéressés à la réduction de la consommation énergétique du système. Des techniques pour la réduction de la consommation énergétique en utilisant la méthode DPM sont étudiées par Benini et al. dans [BAM00]. De plus, en utilisant la technique DVFS, Yao et al. [YDS95] proposent un algorithme qui calcule les fréquences de fonctionnement optimales du point de vue énergétique pour un ensemble de tâches périodiques. La politique d'ordonnancement sous-jacente étant EDF, cette stratégie est également optimale

d'un point de vue ordonnançabilité. Puis, Aydin et al. [AMMMA01] ont proposé une approche basée sur la technique DVFS et la condition nécessaire et suffisante proposée dans [LL73]. Ils supposent que la puissance reçue par la source est constante. Leur algorithme permet de minimiser la consommation énergétique des tâches périodiques en garantissant leur exécution avant leur échéance. Des techniques ont également été développées pour les tâches périodiques et apériodiques permettant de réduire la consommation énergétique du système [SK04, MaWX+07].

3.2 Approche à visée d'autonomie énergétique

Dans cette partie nous citons des algorithmes existant dans la littérature qui permettent d'ordonnancer une configuration de tâches périodiques sur un processeur à vitesse constante de façon à faire un bon usage de l'énergie disponible dans le système tout en satisfaisant leurs contraintes temporelles et énergétiques. Les méthodes que nous décrivons ici sont donc adaptées au modèle décrit dans le paragraphe 2.2 avec des hypothèses restrictives.

3.2.1 Algorithme d'Allavena et Mossé

L'algorithme d'Allavena et Mossé s'applique au modèle qualifié en anglais de "frame based" signifiant ainsi que toutes les tâches ont une échéance commune égale à leur période. On considère une configuration de *n* tâches qui ont une échéance relative commune D_i et sont réveillées à l'instant *t=0*. Chaque tâche τ_i s'exécute pendant $t_i = C_i.S_i$ où C_i est le temps pire-cas d'exécution et S_i est la vitesse d'exécution de τ_i. On distingue les tâches par le paramètre $p'_i = p_i - r$ où *r* est la puissance de recharge de la batterie supposé constante et p_i est la puissance de consommation instantanée de τ_i. Selon la valeur de p'_i, on range les tâches dans deux listes : une liste de tâches de type rechargeantes *R* ($p'_i \geq 0$) et une liste de tâches de type déchargeantes *D* ($p'_i \leq 0$).

L'idée est d'exécuter à la suite une liste de tâches rechargeantes et une liste de tâches déchargeantes de sorte à maintenir le niveau d'énergie dans la batterie entre deux bornes E_{min} et E_{max}. Ainsi une préemption se produit à chaque fois que le niveau d'énergie atteint l'une des deux bornes pour commuter entre le mode de *recharge* et le mode de *décharge*.
Dans un premier temps, tant que la batterie n'est pas vide, l'ordonnanceur sélectionne les tâches déchargeantes de sorte que le niveau d'énergie disponible dans la batterie baisse le plus possible jusqu'à ce qu'il atteigne le niveau minimum E_{min}. Ensuite, il sélectionne les tâches rechargeantes jusqu'à ce que le niveau d'énergie augmente le plus possible pour atteindre le niveau maximum E_{max}.

Quand $|R| < |D|$ alors les tâches déchargeantes ont un besoin énergétique plus important que l'apport énergétique délivré par les tâches rechargeantes. Afin d'assurer l'équilibre entre la production d'énergie et la consommation, l'ordonnanceur doit insérer un intervalle de temps d'inactivité du processeur. Le rajout de cet intervalle de temps creux se fait dès lors que la condition $|R| < |D|$ est vérifiée. Ainsi, il n'y a ni consommation ni production d'énergie liée à l'exécution d'une tâche. Cet intervalle d'inactivité est calculé comme suit :

$$t_{idle} = max(\frac{|D| - |R|}{r}, 0) \tag{2.3}$$

Cet algorithme a été ensuite modifié, par les même auteurs [AM01], de manière à s'adapter à un processeur à vitesse variable.

Condition de faisabilité :
La condition $|R| < |D|$ est vérifiée initialement hors-ligne. Si cette condition est initialement vérifiée et si la somme des durées d'exécution des tâches, rajoutées à ce temps d'inactivité, dépasse l'échéance commune *D*, $\sum_{i=1}^{n} t_i + t_{idle} > D$, la configuration n'est pas ordonnançable. Par conséquent, la condition $\sum_{i=1}^{n} t_i + t_{idle} > D$ est une condition nécessaire et suffisante d'ordonnançabilité.

Cet algorithme est optimal pour des tâches périodiques indépendantes. De plus, il est clairvoyant énergétiquement puisqu'il suppose la production d'énergie parfaitement connue initialement. Il a l'avantage d'être simple à mettre en oeuvre et d'avoir une complexité linéaire comme les listes de tâches rechargeantes et déchargeantes ne nécessitent pas de tri. Néanmoins, les hypothèses demeurent très restrictives voire impraticables dans des applications réelles. En effet, les auteurs supposent une période et une échéance commune pour toutes

les tâches de l'application. De plus, ils considèrent une puissance reçue par la source d'énergie environnementale constante au cours du temps.

Après les travaux d'Allavena et al. présentés en 2001, l'algorithme LSA sur la thématique de l'ordonnancement temps réel sous contraintes énergétiques est présenté par Moser et al. en 2006 [MBT+06].

3.2.2 Ordonnancement optimal LSA

L'algorithme LSA (*Lazy Scheduling Algorithm*) [MBT+06] est un algorithme d'ordonnancement en-ligne. Il permet d'ordonnancer toute configuration de tâches périodiques ou apériodiques critiques selon EDF (i.e. tâches non récurrentes ayant des échéances strictes). Ces tâches sont exécutées par un monoprocesseur alimenté par un réservoir d'énergie rechargeable par une source d'énergie renouvelable. Ce réservoir d'énergie reçoit une puissance instantanée $P_r(t)$ pouvant varier au cours du temps (contrairement à l'hypothèse des travaux décrits ci-dessus).

A chaque réveil d'une tâche, l'ordonnanceur calcule une date de démarrage propre à cette tâche, notée s_i. Cette date représente l'instant à partir duquel la tâche peut commencer à s'exécuter sur le processeur en utilisant la puissance de consommation maximum P_{max} durant toute son exécution. Entre la date d'arrivée et la date de démarrage, le processeur est volontairement laissé en veille pour permettre au réservoir de se recharger. Cet intervalle de temps de recharge est calculé en faisant en sorte que le processeur dispose de suffisamment d'énergie pour exécuter la tâche dans le respect de son échéance. LSA est un algorithme clairvoyant du point de vue énergétique ; donc il suppose avoir une connaissance, au moins pour un futur proche, de la quantité d'énergie récupérée. Cependant comme il ne nécessite pas de connaître a priori les dates d'arrivée futures des tâches, il n'est pas clairvoyant de point de vue temporel.

Résultat d'optimalité :
Considérons une architecture matérielle de type monoprocesseur caractérisé par une puissance de consommation P_{max}, en charge d'exécuter une configuration de tâches. Une tâche $\tau_i = \{r_i, C_i, D_i, E_i\}$ est caractérisée par sa durée d'exécution C_i, son échéance D_i et sa demande énergétique E_i. Si LSA ne peut pas ordonnancer de manière fiable cette configuration de tâches, alors aucun autre algorithme ne peut le faire, même s'il est totalement clairvoyant énergétiquement.
Même si *LSA* est optimal, il présente quelques inconvénients. En effet, les hypothèses faites sur la configuration de tâches sont très restrictives. L'énergie consommée par une tâche est supposée proportionnelle à sa durée d'exécution ($E_i = k.C_i$). Or, l'énergie totale que consomme une tâche au cours de son exécution n'est pas proportionnelle à la durée de son exécution. Cette énergie dépend en particulier des différents circuits électroniques dont aura besoin la tâche.

3.2.3 Heuristique d'ordonnancement EDeg

L'heuristique d'ordonnancement *EDeg* (Earliest Deadline with energy guaranted) [CG09] est comme LSA une variante de l'algorithme *EDF* qui gère les intervalles de passivité du processeur selon la disponibilité d'énergie. Les auteurs considèrent un système composé d'un monoprocesseur alimenté par un réservoir d'énergie conforme au modèle décrit dans le paragraphe 2.2. Ce réservoir d'énergie est rechargé par une source d'énergie renouvelable. Le processeur doit être capable d'exécuter une configuration de tâches périodiques selon EDF tout en considérant leurs demandes énergétiques, leurs échéances et l'énergie disponible dans le réservoir.

Ainsi à chaque instant, une tâche prête n'est exécutée que si le réservoir n'est pas vide et qu'il y a assez d'énergie dans le système de sorte que l'exécution de celle-ci ne compromette pas l'exécution de tâches futures. Dans le cas contraire, s'il n'y a pas assez d'énergie dans le système ou si le réservoir est vide, le processeur doit être mis en mode veille pendant une durée t_{idle} pour permettre de recharger la batterie.

Comme nous nous basons sur cet algorithme dans nos travaux de thèse, une description plus détaillée de cet algorithme sera effectuée dans le chapitre suivant.

4 Conclusion

Dans ce chapitre nous nous sommes focalisés dans un premier temps sur les réseaux de capteurs qui constituent désormais les systèmes temps réel embarqués les plus répandus. Les RCSFs sont autonomes, et leurs besoins énergétiques doivent être satisfaits sachant qu'ils sont généralement utilisés dans des secteurs hostiles où l'accès pour maintenance peut s'avérer difficile. C'est pourquoi une solution dans ce cas consiste à utiliser l'énergie environnementale pour les alimenter par le biais d'un réservoir d'énergie. Nous avons donc par la suite cité les principales sources d'énergies renouvelables et quelques réservoirs de stockage d'énergie. Pour permettre le bon fonctionnement du système, il faut s'assurer que l'énergie électrique disponible soit suffisante pour exécuter toutes les tâches dans le respect de leurs contraintes temporelles. Les techniques DPM et DVFS permettent de réduire la consommation énergétique d'un système mais ne peuvent assurer à elles seules l'autonomie de celui-ci. Elles doivent nécessairement s'accompagner de techniques d'ordonnancement très spécifiques. Nous avons présenté trois de ces techniques, montrant ainsi l'évolution des modèles étudiés depuis 2001, date de démarrage de la recherche dans ce domaine.

Le chapitre suivant présente la première partie de ma contribution. Celle-ci va concerner la proposition de stratégies d'ordonnancement monoprocesseur pour un système temps réel alimenté par une source d'énergie renouvelable.

Deuxième partie

Contributions à l'ordonnancement de tâches périodiques

Chapitre 3

Gestion de la surcharge dans les systèmes temps réel autonomes

Sommaire

Ce chapitre concerne l'ordonnancement temps réel sous contraintes énergétiques. Dans un premier temps, nous décrivons les algorithmes EDeg, RTO et BWP. Dans un deuxième temps, nous présentons Green-RTO et Green-BWP deux algorithmes proposés dans le cadre de cette thèse qui permettent d'ordonnancer une configuration de tâches périodiques temps réel fermes sous contraintes énergétiques. Ensuite, nous présentons une étude comparative des performances obtenues avec Green-RTO, Green-BWP et EDeg.

1 Heuristique EDeg

1.1 Principe de EDeg

L'algorithme *EDeg* [CG09] (*Earliest Deadline with energy guarantee*) est une variante de l'algorithme *EDF* et de l'algorithme *EDL* permettant d'ordonnancer des tâches périodiques conformes au modèle décrit dans le chapitre précédent. Les auteurs dans [CG09] considèrent un système composé d'un monoprocesseur qui est alimenté par un réservoir d'énergie. Ce réservoir d'énergie est rechargé par une source d'énergie renouvelable.

Le processeur doit être capable d'exécuter une configuration de tâches périodiques temps réel strictes tout en considérant leurs demandes énergétiques, leurs échéances et l'énergie disponible dans le réservoir.

Ainsi à chaque instant, la tâche ayant la plus proche échéance parmi les tâches prêtes est exécutée uniquement si d'une part le réservoir n'est pas vide et si d'autre part il y a suffisamment d'énergie dans le système de sorte que l'exécution de celle-ci ne compromette pas l'exécution de tâches périodiques futures. Dans le cas contraire, c'est-à-dire s'il n'y a pas assez d'énergie dans le système ou que le réservoir est vide, le processeur doit être mis en mode *idle* (ou veille) pendant une durée t_{idle} afin de recharger la batterie. t_{idle} est calculé en utilisant l'algorithme *EDL* [CC89].

Test de faisabilité :
Considérons une configuration de *n* tâches périodiques temps réel strictes où chaque tâche τ_i est caractérisée par son temps d'exécution pire-cas C_i, son échéance relative D_i, sa période T_i et sa demande énergétique E_i. Le réservoir d'énergie a une capacité C et la puissance instantanée reçue par le réservoir est donnée par $P_r(t)$ qui prend en compte toutes les pertes dues à la conversion. Cette configuration est ordonnançable avec EDeg seulement si :

$$U_p = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{3.1}$$

et

$$U_e = \sum_{i=1}^{n} \frac{E_i}{T_i} \leq \overline{P_r} \tag{3.2}$$

U_p est le facteur d'utilisation du processeur, U_e est défini dans [CG09] comme étant l'utilisation énergétique et $\overline{P_r}$ est la moyenne de la puissance reçue sur toute la durée de vie de l'application.

1.2 Description de l'algorithme

Algorithm 1 Earliest Deadline with energy guarantee (EDeg)

```
while (1) do
    while PENDING=true do
        while (E(t) > E_min and SlackEnergy(t) > 0) do
            execute()
        end while
        while (E(t) < E_max and SlackTime(t) > 0) do
            wait()
        end while
    end while
    while PENDING=false do
        wait()
    end while
end while
```

Eléments de l'algorithme

1. $SlackEnergy(t)$ est appelée la laxité énergétique du système calculée à l'instant courant *t*. Cette variable représente la quantité maximale d'énergie qui peut être consommée à partir de l'instant *t* tout en satisfaisant toutes les contraintes temporelles des tâches. Si une tâche τ_i est prête et élue à un instant *t*, si la batterie est non vide et si la *laxité énergétique* du système est positive, la tâche pourra être exécutée. La *laxité énergétique* du système à l'instant *t* correspond à la laxité énergétique minimale parmi celles des tâches ayant leur date de réveil supérieure à l'intant *t* et leurs échéances inférieures à d (i.e. d est l'échéance absolue de la tâche la plus prioritaire selon EDF).

La laxité énergétique de $\tau_{i,k}$ ($k^{ème}$ job de la tâche τ_i) à l'instant *t* est calculée comme suit :

$$SlackEnergy(\tau_{i,k}, t) = E(t) + \int_t^d P_r(x)dx - A_{i,k} \tag{3.3}$$

où $A_{i,k}$ est la somme des demandes d'énergie requises par les jobs réveillés aprés $r_{i,k}$ et d'échéance inférieure ou égale à $d_{i,k}$ telle que $A_{i,k} = \sum_{r_j \geq t, d_j \leq d} E_j$

2. $SlackTime(t)$ est la laxité temporelle c'est-à-dire le temps creux maximal disponible à partir de l'instant *t*. Il est calculé avec l'algorithme EDL qui permet de retarder au plus tard les jobs périodiques tout en garantissant le respect de leurs échéances.

3. $PENDING$ est une variable booléenne qui a la valeur **vrai** lorsqu'il y a au moins une tâche prête en attente et **faux** sinon.

1.3 Exemple illustratif

Considérons une configuration de 2 tâches périodiques temps réel dur $\tau_i(C_i, D_i, T_i, E_i)$ (i.e. ne tolérant pas de pertes) : $\mathcal{T}$ ={$\tau_1(3, 9, 9, 7)$; $\tau_2(4, 12, 12, 12)$}. Les tâches sont réveillées à l'instant *t=0* et sont exécutées selon EDeg sur un monoprocesseur alimenté par une batterie initialement pleine et de capacité $C = 4$, $E(0) = 4$. La puissance reçue par la source est considérée constante sur l'hyperpériode et intègre toutes les pertes : $P_r(t) = 2$.

Les jobs périodiques sont exécutés selon EDF tant qu'il y a assez d'énergie dans la batterie et tant que la laxité énergétique du système calculée à chaque instant est positive. A l'instant $t = 0$, le job $\tau_{1,1}$ est le plus prioritaire. La laxité énergétique du système à l'instant $t = 0$ représente la laxité énergétique minimale des jobs ayant leur instant de réveil supérieur ou égal à $t = 0$ et leur échéance inférieure ou égale à $d_{1,1} = 9$. Comme il n'y a que le job $\tau_{1,1}$ qui répond à ces conditions, la laxité énergétique du système à l'instant t=0 correspond à la laxité énergétique du job $\tau_{1,1}$. $SlackEnergy(\tau_{1,1}, 0) = E(0) + \int_0^9 P_r(x)dx - E_1 = 4 + 18 - 7 = 15$. Comme la laxité énergétique du système est positive, le job $\tau_{1,1}$ est exécuté. A l'instant $t = 3$, la laxité énergétique du système correspond à la laxité énergétique du job $\tau_{2,1}$ et vaut $SlackEnergy(\tau_{2,1}, 3) = E(3) + \int_3^{12} P_r(x)dx - E_2 = 3 + 18 - 12 = 9$. Comme la laxité énergétique est positive le job $\tau_{2,1}$ est exécuté. A l'instant *t=6*, la batterie est vide et le job $\tau_{2,1}$ est interrompu. Par conséquent, le calcul de la laxité temporelle à l'instant *t=6* avec l'algorithme *EDL*, est requis pour calculer la durée du temps creux disponible à *t=6* pour recharger la batterie tout en retardant au maximum les tâches périodiques. La laxité temporelle calculée à l'instant *t=6* vaut 5 unités de temps. Ainsi, le processeur est inactif et la batterie se recharge. A l'instant *t=8*, la batterie est à nouveau pleine, le processeur est remis à l'état actif et le job prêt ayant la plus petite échéance peut être exécuté. C'est donc le job $\tau_{2,1}$ qui continue son exécution.

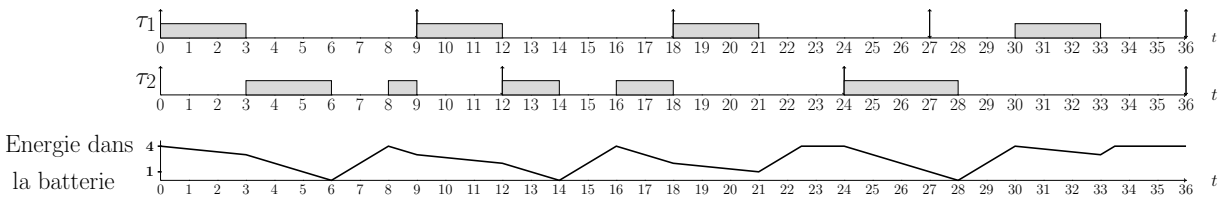


FIGURE 3.1 – Ordonnancement selon EDeg

1.4 Performance de l'ordonnancement EDeg

L'ordonnanceur EDeg a fait l'objet d'une étude de performance exhaustive rapportée dans la thèse de El Ghor [Gho12] et publiée dans [GCC11] et [GCC13]. Cette étude a permis de mettre en évidence la supériorité de EDeg sur l'ordonnanceur conventionnel EDF. Notamment, il est montré que l'heuristique EDeg permet l'ordonnancement d'une configuration de tâches temps réel dur compte tenu d'une source d'énergie propre avec une production d'énergie variable et une unité de stockage d'énergie de capacité limitée. Par le biais de simulations dans [Gho12], EDeg montre de réelles performances par rapport à des variantes d'EDF en termes du nombre de configurations de tâches ordonnançables suivant la taille de la batterie, du taux de gaspillage

d'énergie, du profil de consommation d'énergie et du temps moyen d'inactivité du processeur. Par ailleurs, EDeg permet une réduction de la capacité de la batterie de 1.8 à 3 fois celle requise avec EDF suivant que le système est faiblement, moyennement ou fortement chargé temporellement. D'où l'intérêt de nous baser sur EDeg dans nos travaux de thèse.

Dans des applications réelles, il se peut qu'à un moment donné, la puissance moyenne reçue par la source d'énergie soit inférieure à $U_e = \sum_{i=1}^{n} \frac{E_i}{T_i}$. Si l'on prend le cas d'une batterie alimentée par une source d'énergie photovoltaïque, pendant la nuit la puissance reçue est négligeable. Par conséquent, il faut prévoir une batterie de capacité suffisante pour pouvoir assurer les besoins énergétiques de la configuration de tâches sans quoi des violations d'échéances en grand nombre pourront engendrer la panne du système. C'est pourquoi, notre objectif est de pouvoir assurer la meilleure QoS possible en fonction du profil de la source énergétique et de la taille de batterie de façon à garantir une autonomie perpétuelle avec un niveau de performance toujours acceptable. En effet, nous ne considérons plus une configuration de tâches périodiques temps réel strictes mais une configuration de tâches périodiques temps réel fermes qui tolère un seuil de pertes en-deçà duquel les performances du système sont certes dégradées mais le résultat est toujours jugé acceptable.

2 Gestion de surcharge

Une surcharge causée soit par une *surcharge de traitement* soit par une *surcharge énergétique* peut être détectée dans un système à un instant *t*. En effet, une configuration de tâches est en surcharge de traitement si elle a un taux d'utilisation du processeur trop élevé pour pouvoir respecter ses échéances indépendamment de ses besoins énergétiques. Par ailleurs, elle est en surcharge énergétique lorsqu'elle a des besoins en énergie supérieurs à l'énergie disponible, entraînant également des violations d'échéances.

Quelques techniques ont été proposées dans la littérature [[WSP03, AA05, NQ06, NQ07, Niu10]] utilisant la technique *DVFS* permettant d'abandonner des instructions pour réduire la consommation énergétique dans des systèmes temps réel fermes tout en assurant une QoS imposée par le système. En outre certains systèmes (par exemple les réseaux de capteurs) sont principalement limités en quantité d'énergie disponible même si la demande d'énergie des tâches reste connue et fixe. Dans ce genre de système, au lieu de proposer des techniques permettant de réduire la consommation énergétique, il pourrait être intéressant de proposer des algorithmes qui font le meilleur usage de l'énergie disponible et demande un réservoir d'énergie de taille restreinte.

Dans le cadre de cette thèse, nous essayons de répondre à la question suivante : *Comment planifier les tâches afin de garantir leur exécution dans le respect de leurs contraintes de temps et ce, perpétuellement, en exploitant convenablement à la fois le temps processeur et l'énergie disponible dans le système ?*

La réponse revient à proposer des algorithmes permettant la bonne utilisation de l'énergie disponible dans le système de façon à garantir au moins une QoS minimale au système c'est-à-dire exécuter toutes les tâches importantes dans le respect de leurs contraintes temporelles. Dans cette optique, nous considérons ici un système autonome muni d'un monoprocesseur alimenté par un réservoir d'énergie rechargé par une source d'énergie renouvelable. Nous nous intéressons aux applications temps réel fermes. Ainsi nous adaptons l'ordonnancement de tâches suivant leurs contraintes énergétiques et temporelles et nous veillons à garantir une QoS minimale sans quoi le résultat délivré par le système ne serait pas jugé acceptable. Nos travaux nous ont amenés à proposer deux stratégies d'ordonnancement nommées *Green-RTO* et *Green-BWP*. Ces algorithmes se basent sur le modèle *Skip-Over* et *EDeg* permettant d'abandonner des jobs dans le cas de surcharge temporelle et/ou énergétique.

2.1 Modèle Skip-Over [KS95]

Dans cette section, nous considérons une configuration de tâches périodiques temps réel fermes (i.e. tolérant des pertes) $\mathcal{T}^*$=$\{\tau_i(C_i,\ D_i,\ T_i,\ s_i)\}$ où chaque tâche τ_i est caractérisée par sa durée d'exécution C_i, son échéance relative D_i, sa période T_i et son paramètre de pertes s_i au sens Skip-Over. Les tâches sont indépendantes, préemptables et à échéances sur requêtes. Un job noté $\tau_{i,j}$ désigne la $j^{ème}$ occurrence de la tâche τ_i. Soit H^* = PPCM($s_1.T_1$, ..., $s_n.T_n$), l'hyperpériode équivalente (i.e. prenant en compte les pertes) définie sur une configuration de tâches $\mathcal{T}^*$.

2.1.1 RTO (Red Task Only) [KS95]

Comme expliqué dans le chapitre 1, chaque tâche est divisée en jobs rouges et bleus. Sur un intervalle de durée $s_i.T_i$, chaque tâche τ_i aura $s_i - 1$ jobs rouges et un job bleu. L'algorithme RTO consiste à ordonnancer les jobs rouges des tâches au plus tôt selon l'algorithme EDF et à rejeter systématiquement les jobs bleus. Les conflits d'échéances sont résolus en faveur du job ayant la date de réveil la plus ancienne par rapport au temps courant.

2.1.1.1 Description de l'algorithme L'algorithme d'ordonnancement RTO est chargé de gérer deux listes : une liste de jobs en attente et une liste de jobs prêts. A chaque instant *t*, les jobs dans la liste de tâches périodiques en attente ayant une date de réveil inférieure ou égale à *t* passent à l'état prêt. Chaque job à l'état prêt est inséré dans la liste de job prêts si son paramètre de pertes dynamique s_{i_dyn} est inférieur à s_i. Par contre, si $s_{i\_dyn} = s_i$, le job de la tâche τ_i est bleu et est systématiquement rejeté. Par la suite les paramètres de τ_i sont mis à jour. Par conséquent, son paramètre de pertes dynamique est réinitialisé et prend la valeur 1 ($s_{i\_dyn} = 1$). Notons qu'à l'initialisation, le paramètre de pertes dynamique pour toutes les tâches vaut 1 et à chaque fois qu'un job rouge d'une tâche τ_i est exécuté, celui-ci est incrémenté de 1 : $s_{i\_dyn} = s_{i\_dyn} + 1$. La description algorithmique de RTO est fournie dans l'Algorithme 2.

Algorithm 2 Red Task Only (RTO) [Mar06]

t : temps courant
/*Recherche des réveils de tâches associés au temps courant*/
while liste des tâches en attente=not(ϕ) **do**
  if tâche$\rightarrow r_i > t$ **then**
    break
  end if
  if tâche$\rightarrow s_{i_dyn} < \tau_i \rightarrow s_i$ **then**
    Retirer la tâche de la liste des tâches en attente
    Insérer la tâche dans la liste des jobs rouges prêtes
  else
    tâche$\rightarrow r_i$=tâche $\rightarrow r_i$+ tâche $\rightarrow T_i$
    tâche$\rightarrow s_{i_dyn} = 0$
  end if
  tâche$\rightarrow s_{i_dyn}$ =tâche$\rightarrow s_{i_dyn} + 1$
end while

2.1.1.2 Exemple illustratif Soit une configuration de 2 tâches périodiques temps réel fermes indépendantes et préemptables. $\mathcal{T}^*$ = $\{\tau_i(C_i,\ D_i,\ T_i,\ s_i)$; $\tau_1(3, 6, 6, 2)$, $\tau_2(4, 8, 8, 2)\}$. Les tâches sont réveillées à l'instant *t=0*, et comme la configuration de tâches est profondément rouge (*deeply red* en anglais), tous les jobs sont initialement rouges. La Figure 3.2 illustre l'ordonnancement de cette configuration selon l'algorithme *RTO*. Comme le paramètre de pertes des tâches $s_1 = s_2 = 2$, ceci implique que pour chaque tâche, un job sur deux est rouge et est par conséquent exécuté sur un intervalle de deux périodes d'activation.
Le job $\tau_{1,1}$ est le job rouge le plus prioritaire à l'instant *t=0*. Il s'exécute sur le processeur pendant 3 unités de temps. Ensuite c'est le job rouge $\tau_{2,1}$ qui s'exécute pendant 4 unités de temps sans interruption sachant qu'à l'instant *t=6*, le job bleu $\tau_{1,2}$ est réveillé et est systématiquement rejeté. A l'instant *t=12*, le job rouge de la tâche τ_1 est prêt et exécuté et ainsi de suite.

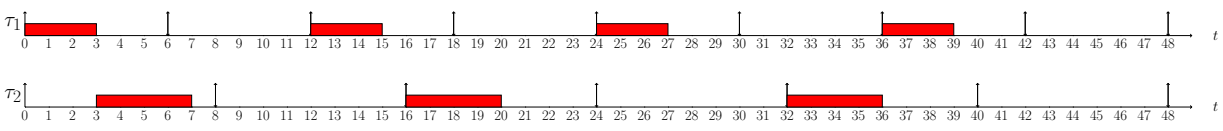


FIGURE 3.2 – Ordonnancement selon RTO

2.1.2 BWP (Blue When Possible) [KS95]

Le second algorithme étudié dans [KS95] est l'algorithme Blue When Possible (BWP). Cet algorithme est une amélioration de l'algorithme RTO dans le sens où BWP permet d'ordonnancer les job bleus prêts lorsqu'il n'y a plus de jobs rouges à l'état prêt. Les jobs rouges sont toujours plus prioritaires que les jobs bleus. Les jobs bleus et rouges sont ordonnancés selon EDF et les conflits d'échéances sont toujours résolus en faveur du job ayant la plus ancienne date de réveil.

2.1.2.1 Description de l'algorithme L'algorithme d'ordonnancement BWP consiste à gérer 3 listes :
- une liste de tâches bloquées,
- une liste de jobs rouges prêts,
- une liste de jobs bleus prêts.

L'algorithme *BWP* opère ainsi(cf. Algorithme 3) :
- Les jobs dans la liste des jobs bleus prêts ayant leurs échéances supérieures ou égales au temps courant sont supprimés de la liste.
- La liste des tâches bloquées est parcourue de manière à réveiller les jobs ayant une date de réveil inférieure ou égale au temps courant. Une fois réveillés, suivant la valeur de leurs paramètres de pertes dynamiques, les jobs prêts sont insérés, soit dans la liste des jobs bleus (si $s_{i\_dyn} = s_i$), soit dans la liste des jobs rouges (si $s_{i\_dyn} < s_i$).

Algorithm 3 Blue When Possible (BWP) [Mar06]

t : temps courant
/*Recherche des abandons des jobs bleus associés au temps courant*/
while liste des jobs bleus prêts=not(ϕ) **do**
  if $job \rightarrow r_i + job \rightarrow D_i < t$ **then**
    break
  end if
  Retirer la job de la liste des job bleus prêts
  $job \rightarrow r_i = job \rightarrow r_i + job \rightarrow T_i$
  $job \rightarrow s_{i\_dyn} = 0$
  Insérer le job dans la liste des tâches en attente
end while
/*Recherche des réveils de tâches associés au temps courant*/
while liste des tâches en attente=not(ϕ) **do**
  if tâche $\rightarrow r_i > t$ **then**
    break
  end if
  Retirer la tâche de la liste des tâches en attente
  if tâche $\rightarrow s_{i\_dyn} <\rightarrow s_i$ **then**
    Insérer la tâche dans la liste des jobs rouges prêts
  else
    Insérer la tâche dans la liste des jobs bleus prêts
  end if
  tâche $\rightarrow s_{i\_dyn}$ = tâche $\rightarrow s_{i\_dyn} + 1$
end while

2.1.2.2 Exemple illustratif On considère la même configuration que celle de l'exemple de RTO. Les tâches sont réveillées à l'instant *t=0*, et tous les jobs sont initialement rouges. La Figure 3.3 illustre l'ordonnancement de cette configuration selon l'algorithme *BWP* sur une hyperpériode H^* = PPCM(12,16)=48. Le job $\tau_{1,1}$ est le job rouge le plus prioritaire à l'instant *t=0*. Il s'exécute sur le processeur pendant 3 unités de temps. Ensuite c'est le job rouge $\tau_{2,1}$ qui s'exécute pendant 4 unités de temps sans interruption. En effet, à l'instant *t=6*, le job bleu $\tau_{1,2}$ est réveillé mais est moins prioritaire que le job rouge. A l'instant $t = 7$, comme il n'y a aucun job rouge à l'état prêt, le job bleu $\tau_{1,2}$ débute son exécution. On remarque qu'à partir de l'instant $t = 7$, les jobs de

τ_1 et τ_2 sont bleus vu qu'aucune violation d'échéance n'a été observée auparavant. Ils sont exécutés selon EDF donc le job ayant la plus proche échéance est le plus prioritaire parmi tous les jobs bleus prêts.

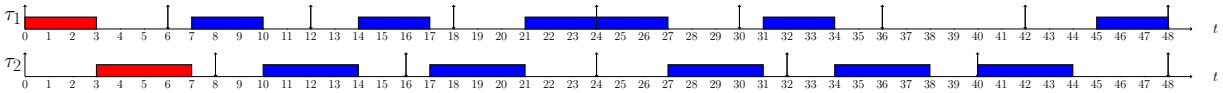


FIGURE 3.3 – Ordonnancement selon BWP

2.2 Adaptation de l'algorithme EDL au modèle Skip-Over

Dans cette partie, nous décrivons une adaptation des calculs EDL [Mar06] pour les algorithmes RTO et BWP que nous utiliserons dans nos travaux.

2.2.1 Application de l'algorithme EDL au modèle RTO

Le calcul des temps creux se réduit à l'intervalle $[0, H^*]$ car les jobs rouges des tâches périodiques fermes sont synchrones toutes les H^* unités de temps. Une séquence RTO se répète toutes les H^* unités de temps. Dans un premier temps nous présentons les calculs faits hors-ligne.

2.2.1.1 Calcul du vecteur statique des échéances sur l'intervalle $[0, H^*)$. Le vecteur statique des échéances, noté $\mathcal{K}$, est l'ensemble des instants considérés sur l'hyperpériode $[0, H^*)$, précédant un temps creux. Il se construit à partir des échéances distinctes des jobs périodiques rouges. La distance entre deux pertes consécutives est exactement de s_i périodes dans lesquelles on observe *m* jobs rouges suivies d'un job bleu avec *m*=(s_i - 1). Ces *m* jobs rouges, sont périodiques de période $s_i T_i$. Toutes les tâches ont leur échéance égale à leur période d'activation (c'est-à-dire que chaque job doit se terminer avant la prochaine activation de la tâche). Ainsi, chaque tâche τ_i aura $\frac{m.H^*}{s_i T_i}$ jobs rouges sur l'intervalle $[0, H^*)$. Les instants k_i précédant un temps creux sous RTO sont définis par le théorème suivant :

Théorème 2.1 *[Mar06]*
Les instants k_i du vecteur $\mathcal{K} = (k_0, ..., k_i, ..., k_q)$ sont calculés sur l'intervalle $[0, H^*)$ comme suit :

$$k_i = (x.s_j + k).T_j \tag{3.4}$$

avec $x = 0, ..., (\frac{H^*}{s_j T_j} - 1)$, $k = 1, ..., m$, $k_i < k_{i+1}$
$k_0 = min\{T_j; 1 \leq j \leq n\}$
$k_q = H^* - min\{T_j, 1 \leq j \leq n\}$

2.2.1.2 Calcul du vecteur statique des temps creux sur $[0, H^*)$. Le vecteur statique des temps creux $\mathcal{D}^* = (\Delta_0^*, ..., \Delta_i^*, ..., \Delta_q^*)$ représente les durées des temps creux correspondant aux différents instants du vecteur des échéances $\mathcal{K}$. Le théorème suivant fournit la formule de récurrence pour le calcul du vecteur $\mathcal{D}^*$ sous RTO, tenant compte des pertes s_j autorisées par chacune des tâches T_j :

Théorème 2.2 *[Mar06] Les durées des temps creux du vecteur $\mathcal{D}^* = (\Delta_0^*, ..., \Delta_i^*, ..., \Delta_q^*)$ pour des tâches RTO sont calculées sur $[0, H^*)$ comme suit :*

$$\Delta_q^* = min\{T_i; 1 \text{ à } n\} \tag{3.5}$$

$$\Delta_i^* = sup(0, F_i), pour\ i\ = q - 1 \text{ à } 0 \tag{3.6}$$

$$F_i = (H^* - k_i) + \sum_{j=1}^{n} \left(\left\lceil \frac{(H^* - k_i)}{T_j} \right\rceil - \left\lceil \frac{(H^* - k_i)}{T_j.s_j} \right\rceil \right) C_j - \sum_{i+1}^{q} \Delta_i^* \tag{3.7}$$

Exemple illustratif
Considérons une configuration de tâches périodiques temps réel fermes $\mathcal{T}^* = \{\tau_i(C_i,\ D_i,\ T_i, s_i)\,;\tau_1(3,\ 6,\ 6,\ 2),\ \tau_2(4,\ 8,\ 8,\ 2)\}$. La Figure 3.4 représente la durée des temps creux calculés hors-ligne sur une hyperpériode. Le vecteur statique des échéances sur l'intervalle [0,48) est formé par toutes les échéances des jobs rouges sur cet intervalle, $\mathcal{K} = \{0,6,8,18,24,30,40,42\}$. Le vecteur statique des temps creux est calculé selon les formules 3.5 et 2.2 : $\mathcal{D}^* = \{1,0,7,2,3,5,0,6\}$.

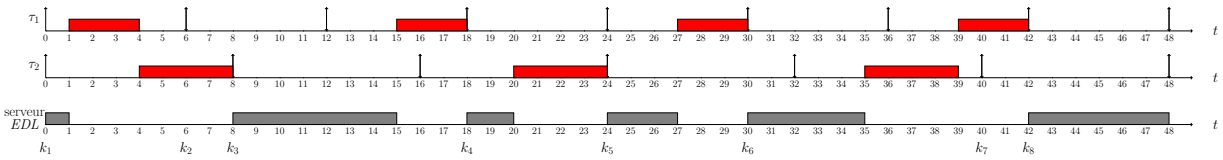


FIGURE 3.4 – Séquence EDL sur les jobs rouges

L'établissement de la séquence EDL statique pour un modèle de tâches ordonnancées selon l'algorithme RTO est obtenue en $O(nN)$ opérations dans le pire-cas, où *N* représente le nombre total de jobs pour l'ensemble des tâches périodiques sur l'intervalle $[0, H^*)$. Comme ce calcul concerne les jobs rouges, cette complexité dépend non seulement des périodes des tâches périodiques mais aussi des paramètres de pertes associés aux tâches. Elle peut donc s'avérer élevée dans le cas où les paramètres de pertes des tâches sont élevés (i.e. les pertes autorisées sont faibles) et/ou lorsque les périodes des tâches ne sont pas multiples entre elles.

Ensuite, nous allons présenter les calculs faits en-ligne à un instant $t = \chi$ sachant que χ représente le temps courant et kH^* correspond à la date de fin de l'hyperpériode courante contenant χ avec $k = (\lfloor \frac{\chi}{H^*} + 1 \rfloor)$.

2.2.1.3 Calcul du vecteur dynamique des échéances sur l'intervalle $[\chi, kH^*)$. Le vecteur dynamique des échéances calculé à l'instant courant *t*=χ, $\mathcal{K}(\chi) = (\chi, k_{h+1}, ..., k_i, ..., k_q)$ représente les instants supérieurs ou égaux à *t* précédant un temps creux. Soit *h* l'index tel que k_h = sup d ; $d \in \mathcal{K}$ et $d < \chi$. Comme dans le cas statique, tous les instants k_i représentent les échéances distinctes des jobs rouges [Mar06].

2.2.1.4 Calcul du vecteur dynamique des temps creux sur l'intervalle $[\chi, kH^*)$. Le vecteur dynamique des temps creux $\mathcal{D}^*(\chi) = (\Delta^*_h(\chi),\ \Delta^*_{h+1}(\chi),\ ...,\Delta^*_i(\chi),\ ...,\Delta^*_q(\chi))$ représente les durées des temps creux associées aux instants consignés dans le vecteur $\mathcal{K}(\chi)$. $\Delta^*_i(\chi)$ est la durée du temps creux débutant au temps k_i. Par ailleurs, on note $A_i(\chi)$ la quantité de processeur déjà allouée au job courant de la tâche périodique τ_i à l'instant $t = \chi$. Ce job possède également une échéance dynamique notée d_i. On note *M*, la plus grande échéance parmi tous les jobs rouges actifs et l'on figure dans le vecteur des temps creux l'indice f tel que k_f =mink_i ; $k_i > M$. $\mathcal{D}^*(\chi)$ est défini par la relation de récurrence suivante décrite dans le théorème qui suit :

Théorème 2.3 *[Mar06] Les durées des temps creux du vecteur $\mathcal{D}^*(\chi) = (\Delta^*_h(\chi), \Delta^*_{(h+1)}(\chi), ..., \Delta^*_i(\chi), ..., \Delta^*_q(\chi))$ pour des tâches ordonnancées par l'algorithme RTO sont calculées sur $[0, kH^*[$ comme suit :*

$$\Delta^*_i(\chi) = \Delta^*_i,\ pour\ i = q\ à\ f \tag{3.8}$$

$$\Delta^*_i(\chi) = sup(0, F_i(\chi)), pour\ i\ = f-1\ à\ h+1 \tag{3.9}$$

avec

$$F_i(\chi) = (H^* - k_i) - \sum_{j=1}^{n}(\left\lceil \frac{(H^*-k_i)}{T_j} \right\rceil - \left\lceil \frac{(H^*-k_i)}{T_j.s_j} \right\rceil)C_j + \sum_{j=1,dj>ki}^{n} A_j(\chi) - \sum_{x=i+1}^{q} \Delta^*_x(\chi) \tag{3.10}$$

$$\Delta^*_h(\chi) = (H^* - \chi) - \sum_{j=1}^{n}((\left\lceil \frac{(H^*-\chi)}{T_j} \right\rceil - \left\lceil \frac{(H^*-\chi)}{T_j.s_j} \right\rceil)C_j - A_j(\chi)) - \sum_{x=h+1}^{q} \Delta^*_x(\chi) \tag{3.11}$$

Exemple illustratif
Nous reprenons la même configuration de tâches périodiques temps réel fermes que celle considérée précédemment pour le calcul hors-ligne. La Figure 3.5 représente la durée des temps creux calculés à l'instant *t=20*.

Le vecteur dynamique des échéances sur l'intervalle [0,48) est formé de toutes les échéances des jobs rouges sur cet intervalle supérieur à *t=20*, $\mathcal{K}(20) = \{20, 24, 30, 40, 42\}$. Le vecteur dynamique des temps creux est calculé selon les formules 3.8 et 3.9 : $\mathcal{D}^*(20) = \{4, 3, 5, 0, 6\}$.

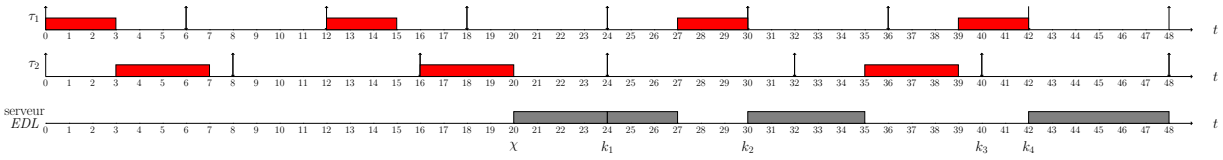


FIGURE 3.5 – Séquence EDL en ligne sur les jobs rouges

En ce qui concerne la complexité algorithmique du calcul du vecteur dynamique des temps creux, on observe que la détermination des temps creux avec un modèle de tâches RTO est la même que celle observée avec un modèle de tâches classiques ; Elle est alors en $O(\left\lceil \frac{R}{p} \right\rceil n)$ où *n* représente le nombre de tâches périodiques, *R* la plus grande échéance parmi les jobs rouges prêts et partiellement exécutés, et *p* la plus petite période. Etant donné que le vecteur statique des temps creux est mémorisé, la reconstruction de la séquence n'est refaite qu'entre l'instant *t* et la plus grande échéance du job rouge parmi les job prêts à l'instant *t*, d'où le facteur $\left\lceil \frac{R}{p} \right\rceil$. La complexité spatiale, est quant à elle déterminée par la taille du vecteur statique des temps creux. Dans le pire-cas, ce dernier est en $O(N)$ où N représente le nombre de jobs rouges sur l'hyperpériode H^*.

2.2.2 Application de l'algorithme EDL au modèle BWP

La différence ici par rapport à l'algorithme *RTO*, réside dans le fait qu'à l'instant *t*, l'ensemble des jobs réveillés n'est pas uniformément rouge. L'algorithme BWP tente d'exécuter des jobs bleus, lorsqu'il n'y a aucun job rouge à exécuter. Ainsi, le calcul des temps creux effectué en-ligne diffère de celui pour RTO, dans la mesure où il doit prendre en compte l'exécution de jobs bleus. En effet, si un job bleu est complètement exécuté, le prochain job de la tâche est encore bleu, ceci entraîne un décalage par rapport à la séquence RTO d'origine. Ce décalage dans le temps (appelé *move forward*) à l'instant *t* est modélisé par un paramètre $m_i(t)$ associé à la tâche τ_i. Ainsi à chaque fois qu'un job bleu est complètement exécuté dans le respect de son échéance, le paramètre $m_i(t)$ est incrémenté modulo s_i.

2.2.2.1 Calcul du vecteur dynamique des échéances sur l'intervalle $[\chi, kH[$.

Le calcul du vecteur dynamique des échéances se fait sur la sous-hyperpériode contenant l'instant $t = \chi : k = \lfloor \frac{\chi}{H} \rfloor + 1$

Définition 2.1 *Une sous-hyperpériode pour une configuration de tâches périodiques temps réel revient à considérer une hyperpériode de cette même configuration sans tolérance de pertes. Ainsi* $H = ppcm(T_1, T_2, ..., T_n)$.

Le vecteur dynamique des échéances calculé à l'instant *t*=χ, $\mathcal{K}(t)$ = (t,$k_{(h+1)}$, ...,k_i, ..., k_q), représente les instants supérieurs ou égaux à l'instant $t = \chi$ précédant un temps creux. Soit *h* l'index tel que $k_h = sup\{d; d \in \mathcal{K}$ et $d < \chi\}$.
Comme dans le cas de RTO, tous les instants k_i représentent tous les échéances distinctes des jobs rouges sur l'intervalle considéré. Les instants k_i sous BWP sont définis par le théorème suivant :

Théorème 2.4 *[Mar06] Les instants* k_i *du vecteur* $\mathcal{K}(\chi)$ = (χ, $k_{(h+1)}$, ..., k_i, ..., k_q) *pour des tâches ordonnancées selon BWP sont calculés sur* $[\chi, H)$ *comme suit :*

$$k_i = (x.s_j + (k + m_j(\chi) + pos_j(\chi))\%s_j).T_j \tag{3.12}$$

avec x = 0, ..., ($\frac{H}{s_j.T_j} - 1$*) ; j = 1, ..., n ; k = 1, ...,*$(s_j - 1)$.

$pos_j(\chi)$ est le paramètre de positionnement défini dans [Mar06] comme suit :

Définition 2.2 *Le paramètre de positionnement* $pos_i(\chi)$ *correspond à l'avance de chaque tâche* τ_i *à l'instant t=*χ*, vis-à-vis d'une séquence dans laquelle le dernier job de la sous-hyperpériode courante contenant l'instant*

$t = \chi$ est bleu. Il est donné par la formule suivante :

$$pos_i(\chi) = (\left\lceil \frac{kH}{s_i.T_i} \right\rceil) s_i - \left\lfloor \frac{kH}{T_i} \right\rfloor \tag{3.13}$$

avec $k = (\lfloor \frac{\chi}{H} \rfloor + 1)$.

2.2.2.2 Calcul du vecteur dynamique des temps creux sur $[t, kH[$**.** Le vecteur dynamique des temps creux $\mathcal{D}^*(\chi) = (\Delta^*_h(\chi), \Delta^*_{(h+1)}(\chi), ..., \Delta^*_i(\chi), ..., \Delta^*_q(\chi))$ représente les durées des temps creux associés aux instants du vecteur $\mathcal{K}(t)$.

Ainsi $\Delta^*_i(\chi)$ représente la longueur du temps creux débutant à l'instant k_i. De plus, chaque tâche périodique τ_i est caractérisée par la quantité de processeur $A_i(\chi)$ déjà allouée à son job rouge courant à l'instant *t*=χ. Ce job possède également une échéance dynamique notée d_i qui est l'échéance absolue de la tâche τ_i. Le calcul de $\mathcal{D}^*(\chi)$ est alors présenté par le théorème qui suit [Mar06] :

Théorème 2.5 *Les durées de temps creux calculés sur [*χ*, kH) pour des tâches ordonnancées selon BWP sont définies par* $\mathcal{D}^*(\chi) = (\Delta^*_h(\chi), \Delta^*_{(h+1)}(\chi), ..., \Delta^*_i(\chi), ..., \Delta^*_q(\chi))$ *et sont calculées comme suit :*

$$\Delta^*_i(\chi) = sup(0, F_i(\chi)), pour\ q \leq i \leq \chi \tag{3.14}$$

avec $F_i(\chi) = (H - k_i) - \sum_{j=1}^{n} (\left\lceil \frac{H-k_i}{T_j} \right\rceil - \left\lceil \frac{H-k_i+shift_j(\chi).T_j}{s_j.T_j} \right\rceil + \left\lceil \frac{shift_j(t).T_j}{s_j.T_j} \right\rceil) C_j + \sum_{j=1;d_j>k_i}^{n} A_j(\chi) - \sum_{x=i+1}^{q} \Delta^*_x(\chi)$
et
$shift_j(\chi) = (m_j(\chi) + pos_j(\chi))\%s_j$

Exemple illustratif
Nous reprenons la même configuration de tâches que celle considérée précédemment pour le modèle RTO : $\mathcal{T}^* = \tau_i(C_i, D_i, T_i, s_i)$; $\tau_1(3, 6, 6, 2)$, $\tau_2(4, 8, 8, 2)$. La Figure 3.6 illustre le calcul des temps creux sur une sous-hyperpériode (H=24) calculé à l'instant t=20. Le vecteur dynamique des échéances sur l'intervalle [0, 24) est $\mathcal{K}(20) = \{20\}$. Le vecteur dynamique des temps creux calculé selon la formule 3.14 est $\mathcal{D}^*(20) = \{4\}$

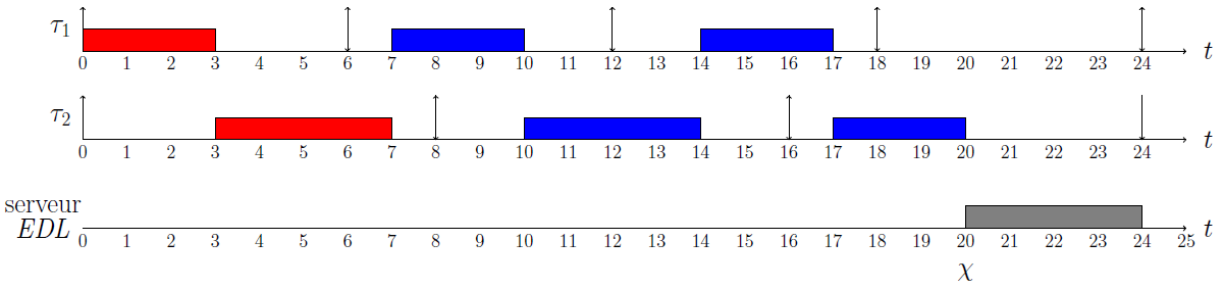


FIGURE 3.6 – Calcul EDL en-ligne pour l'algorithme BWP

La complexité algorithmique du calcul du vecteur dynamique des temps creux est en $O(N)$ où N représente le nombre de jobs périodiques rouges sur l'intervalle $[t, kH)$. Quant à la complexité spatiale, celle-ci est déterminée par la taille du vecteur statique des temps creux. Dans le pire-cas, ce dernier est en $O(N)$.

3 Modèle considéré et Définitions

Dans cette section, notre objectif est d'adapter l'approche précédente à une configuration de tâches avec des besoins énergétiques connus et une disponibilité limitée en énergie.

3.1 Système considéré

Le système que l'on considère est illustré par la Figure 3.7. C'est un système embarqué temps réel composé d'un monoprocesseur alimenté par une batterie qui est rechargée par une énergie renouvelable. Ce processeur exécute

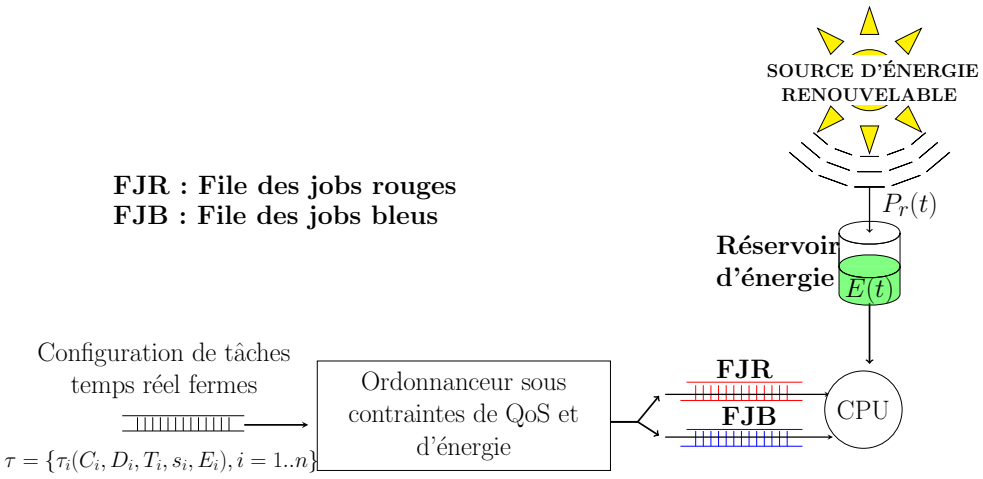


FIGURE 3.7 – Système considéré

des tâches périodiques temps réel fermes (i.e. il y a une tolérance aux pertes : un job peut occasionnellement ne pas être exécuté mais le nombre de pertes ainsi que leur répartition dans le temps est maîtrisé).

Dans nos travaux, nous considérons une configuration de tâches tolérant des pertes $\mathcal{T}^* = \{\tau_i(C_i,\ D_i,\ T_i,\ s_i)\}$. Les tâches sont préemptables, indépendantes, à échéances sur requêtes et *profondément rouges* (i.e. les tâches sont activées de manière synchrone et les $s_i - 1$ premiers jobs de chaque tâche sont rouges). Chaque tâche est caractérisée par :

- C_i : sa durée d'exécution au pire cas (WCET),
- D_i : son échéance relative,
- T_i : sa période,
- s_i : son paramètre de pertes. Plus s_i est grand, moins le système tolère de pertes,
- E_i : sa demande énergétique au pire cas (WCEC).

Pour chaque tâche τ_i, il n'y a aucune relation liant les valeurs de C_i et E_i. Nous considérons que le réservoir d'énergie est une batterie de capacité C. Le niveau d'énergie dans la batterie varie entre deux niveaux : E_{min} et E_{max} avec $C = E_{max} - E_{min}$.
E(t) représente le niveau d'énergie dans la batterie à l'instant *t* et $P_r(t)$ est la puissance reçue à l'instant *t* incluant toutes les pertes d'énergie occasionnées par la mise en oeuvre matérielle de la récupération de l'énergie.

3.2 Definitions et conditions d'ordonnançabilité

3.2.1 Configuration sans tolérance de pertes

Soit une configuration de *m* tâches périodiques temps réel strictes $\mathcal{T}$=$\{\tau_i(C_i, D_i, T_i, E_i), i = 1..m\}$.

Nous définissons :

- L'**utilisation énergétique** U_e de cette configuration comme étant la proportion d'énergie maximale utilisée par les jobs périodiques sur une hyperpériode. Elle se calcule comme suit :

$$U_e = max_{t \geq 0} \left\{ \sum_{i=1}^{n} \frac{\sum g_i(0,t)}{E(0) + E_r(0,t)} \right\}. \tag{3.15}$$

où :
t représente toutes les échéances des jobs périodiques sur l'intervalle [0,H) sachant que H est l'hyperpériode de la configuration de tâches périodiques temps réel strictes $\mathcal{T}$, H = $PPCM(T_1, .., T_i, T_m)$.
$E(0)$ correspond au niveau d'énergie initial dans la batterie.

$E_r(0,t)$ correspond à l'énergie reçue pendant l'intervalle $[0,t]$. Si l'on considère que la puissance reçue est constante $P_r(t) = P_r$, alors $E_r(0,t) = P_r.t$
$g_i(0,t)$ est la demande énergétique de tous les jobs périodiques s'exécutant dans l'intervalle $[0,t]$ et est calculée comme suit :

$$g_i(0,t) = \left\lfloor \frac{t}{T_i} \right\rfloor .E_i \tag{3.16}$$

- La **puissance de consommation énergétique moyenne** $\bar{P_e}$ qui est calculée comme suit :

$$\bar{P_e} = \sum_{i=1}^{m} \frac{E_i}{T_i} \tag{3.17}$$

- Le **ratio de criticité énergétique** R_e qui est égal à la puissance moyenne énergétique consommée par les tâches périodiques ($\bar{P_e}$) divisée par la puissance reçue (P_r).

Nous proposons le théorème suivant :

Théorème 3.1 *Une configuration de n tâches périodiques temps réel strictes, avec un facteur d'utilisation énergétique U_e, est ordonnançable seulement si :*

$$U_e \leq 1 \tag{3.18}$$

Preuve :
Supposons qu'il existe un intervalle $[0,t)$ sur lequel la demande totale énergétique des jobs périodiques excède l'énergie disponible sur cet intervalle : $\sum_{i=1}^{n} g_i(0,t) > E(0) + E_r(0,t)$. Ceci implique qu'à un instant *t1*, un job manque d'énergie et par conséquent ne peut pas s'exécuter complètement avant son échéance. Par conséquent, la configuration n'est pas ordonnançable. □

En fonction des valeurs de R_e, U_e et U_p, le système est en surcharge temporelle et/ou énergétique :
- Si R_e > 1, le système est en surcharge énergétique.
- Si U_p> 1, le système est en surcharge temporelle.
- Si U_e> 1, ceci implique qu'il existe un intervalle $[0,t]$ pendant lequel la demande énergétique des tâches périodiques est supérieure à l'énergie disponible sur cet intervalle.

Une surcharge temporelle et/ou énergétique implique qu'à un instant *t*, une tâche périodique manquera de temps et/ou d'énergie et en conséquence violera son échéance.

3.2.2 Configuration avec tolérance de pertes

Soit $\mathcal{T}^*$ une configuration de *n* tâches périodiques temps réel fermes, indépendantes, préemptables et à échéances sur requêtes. Pour étudier son ordonnançabilité selon notre modèle, il faut prendre en compte les contraintes temporelles et énergétiques.

1. Point de vue temporel :

   Pour assurer l'ordonnancement de cette configuration sous contraintes temporelles, il existe une condition nécessaire et suffisante proposée par Caccamo et Buttazo [CB97] :

   Théorème 3.2 *Une configuration de tâches périodiques temps réel fermes profondément rouges est ordonnançable avec le modèle Skip-Over si et seulement si $U_p^* \leq 1$.*

   U_p^* est le facteur équivalent d'utilisation du processeur :

$$U_p^* = max_{t \geq 0} \left\{ \frac{\sum_i D(i,[0,t])}{t} \right\} \tag{3.19}$$

sachant que $D(i,[0,t])$, introduit par Koren et Shasha [KS95], est la quantité de temps processeur allouée aux jobs rouges pendant l'intervalle *[0,t)*. Il est défini par :

$$D(i,[0,t]) = (\left\lfloor \frac{t}{T_i} \right\rfloor - \left\lfloor \frac{t}{T_i.s_i} \right\rfloor)C_i \tag{3.20}$$

Les auteurs ont montré qu'il est suffisant d'évaluer $D(i,[0,t])$ pour tous les instants t correspondant aux échéances des jobs rouges sur l'hyperpériode H^* = $PPCM(T_1.s_1, ..T_i.s_i, .., T_n.s_n)$.

2. Point de vue énergétique :

Définition 3.1 *L'énergie consommée par la tâche τ_i pendant l'intervalle $[0,t)$ est notée $g_i^*(0,t)$ et est calculée comme suit :*

$$g_i^*(0,t) = (\left\lfloor \frac{t}{T_i} \right\rfloor - \left\lfloor \frac{t}{T_i.s_i} \right\rfloor)E_i \tag{3.21}$$

Ceci nous amène à proposer la définition suivante :

Définition 3.2 *Le facteur équivalent d'utilisation énergétique U_e^* correspondant à la proportion d'énergie consommée par les jobs rouges sur une hyperpériode est calculé comme suit :*

$$U_e^* = max_{t\geq 0}\left\{\sum_{i=1}^{n} \frac{\sum g_i^*(0,t)}{E(0)+E_r(0,t)}\right\}. \tag{3.22}$$

où $E(0)$ est le niveau d'énergie initial de la batterie et $E_r(0,t)$ est l'énergie reçue pendant l'intervalle $[0,t)$. Dans nos travaux nous supposons que la puissance reçue est constante pendant une hyperpériode, ainsi il est suffisant de considérer tout t correspondant à une échéance de job rouge sur l'hyperpériode $H^* = PPCM(T_1 s_1, .., T_i s_i, ..T_n s_n)$. Comme la puissance reçue est supposée constante $P_r(t) = P_r$, $E_r(0,t)$ est calculé selon la formule suivante :

$$E_r(0,t) = P_r.t \tag{3.23}$$

Remarque : Si l'on suppose que la puissance reçue est variable, ceci implique que $P_r(t) \neq P_r(t+1)$. Par conséquent, le calcul de U_e^* sera plus long et plus complexe. En effet, comme la puissance reçue diffère d'un instant à un autre le calcul $\sum_{i=1}^{n} \frac{\sum g*_i(0,t)}{E(0)+E_r(0,t)}$ devra se faire sur tous les instants de l'hyperpériode et *t* représentera donc toutes les valeurs dans l'intervalle $[0,H]$. De plus, l'énergie reçue par l'environnement se calculera de la façon suivante : $E_r(0,t) = \int_t^0 P_r(x).dx$.

Dans nos travaux nous retiendrons l'hypothèse que la puissance reçue est constante.

En s'appuyant sur la définition de $g^*(0,t)$, on peut définir une condition suffisante faisant intervenir le niveau d'énergie initial dans la batterie et la puissance reçue pendant l'intervalle $[0,t]$ considéré :

Théorème 3.3 *Une configuration de tâches périodiques temps réel fermes, profondément rouges, indépendantes, préemptables et à échéances sur requêtes est ordonnançable considérant ses contraintes temporelles et énergétiques seulement si :*

$$\forall L | \sum_{i=1}^{n} g_i^*(0,L) \leq E(0) + E_r(0,L) \tag{3.24}$$

Preuve :
Supposons qu'il existe un intervalle $[0, t)$ sur lequel la demande totale énergétique des jobs rouges excède l'énergie disponible sur cet intervalle : $\sum_{i=1}^{n} g_i^*(0,t) > E(0) + E_r(0,t)$. Ceci implique qu'à un instant *t1*, un job rouge manque d'énergie et par conséquent ne peut pas s'exécuter complètement avant son échéance. Par conséquent, la configuration n'est pas ordonnançable.□

Ceci nous amène à proposer un deuxième théorème :

Théorème 3.4 *Une configuration de tâches périodiques temps réel fermes, profondément rouges, indépendantes, préemptables et à échéances sur requêtes est ordonnançable considérant ses contraintes énergétiques seulement si :*

$$U_e^* \leq 1 \tag{3.25}$$

Preuve :
Si $U_e^* > 1$, ceci implique que sur un intervalle $[0, t_1)$, $\sum_{i=1}^{n} g_i^*(0,t_1) > E(0) + E_r(0,t_1)$ et le théorème 3.3 montre que $\mathcal{T}^*$ est alors non ordonnançable.□

3. Point de vue temporel et énergétique :
D'après les théorèmes 3.2 et 3.3, nous définissons une condition nécessaire de faisabilité :

Théorème 3.5 *Une configuration de tâches périodiques temps réel fermes, profondément rouges, indépendantes, préemptables et à échéances sur requêtes, est ordonnançable sous contraintes énergétiques et temporelles seulement si :*

$$U_p^* \leq 1 \tag{3.26}$$

et

$$U_e^* \leq 1 \tag{3.27}$$

Preuve :
La preuve triviale, découle directement du théorème 5.1 (cf. le chapitre 1 page 31) et du théorème 3.3.□

Dans la suite nous décrivons respectivement deux nouveaux ordonnanceurs, Green-RTO et Green-BWP, et supposons que les configurations de tâches considérées satisfont les conditions nécessaires de faisabilité décrites ci-dessus.

4 Ordonnanceurs proposés

4.1 Algorithme Green-RTO

4.1.1 Principe

Nous nous intéressons au problème de l'ordonnancement monoprocesseur de tâches périodiques temps réel fermes qui autorisent le non-respect de leur échéance de manière contrôlée. Nous proposons l'algorithme Green-RTO basé sur RTO (*Red tasks only*) et EDeg (*Earliest Deadline with energy guarantee*).

Sous *RTO*, les jobs rouges sont exécutés selon la règle *EDF* et doivent impérativement respecter leur échéance. Par contre, les jobs bleus sont systématiquement rejetés.

Dans nos travaux, la contrainte énergétique vient s'ajouter à la contrainte temporelle. Par conséquent, à chaque instant *t*, s'il existe des jobs rouges à l'état prêt, celui ayant la plus proche échéance est choisi pour

être exécuté mais ce job ne sera exécuté sur le processeur que si la *laxité énergétique* du système est positive ET si le niveau d'énergie disponible dans la batterie est non nul. Dans le cas contraire, s'il n'y a pas assez d'énergie dans la batterie OU si la *laxité énergétique* du système est négative, la *laxité temporelle* est calculée (avec l'algorithme *EDL*) permettant de calculer la durée de temps creux disponible à l'instant *t* et ainsi retarder l'exécution des jobs rouges au plus tard. Si la laxité temporelle calculée est non nulle, le processeur est mis en mode veille et la batterie se recharge. Le calcul de la *laxité énergétique* et de la *laxité temporelle* permet de garantir que tous jobs rouges seront exécutés dans le respect de leur échéance.

Green-RTO n'exécute que les jobs rouges dans le respect de leurs contraintes temporelles. Par conséquent la QoS (i.e. nombre de jobs exécutés dans le respect de leur échéance sur le nombre total de jobs) obtenue est constante et directement dépendante des paramètres de pertes associés aux tâches.

4.1.2 Description de l'algorithme

Algorithm 4 Green-RTO

t : temps courant
$\mathcal{T}(t)$: la liste des tâches périodiques bloquées à l'instant *t*
J(t) : la liste des jobs rouges prêts à l'instant *t*
while VRAI **do**
  idle ← VRAI
  if J(t) $= not(\phi)$ **then**
    Soit τ_i le job rouge ayant la plus proche échéance
    if $E(t) > E_{min}$ **then**
      Calculer $SlackEnergy(t)$
      if $SlackEnergy(t) \geq 0$ **then**
        idle← FAUX
      else
        /* $SlackEnergy(t) < 0$ */
        if E(t)=E_{max} **then**
          idle← FAUX
        else
          /* $E_{min} < E(t) < E_{max}$ */
          Calculer $SlackTime(t)$
          if $SlackTime(t) = 0$ **then**
            idle ← FAUX
          end if
        end if
      end if
    end if
    if idle = FAUX and $SlackEnergy(t) \geq 0$ **then**
      Execute τ_i
    end if
  end if
end while

L'algorithme Green-RTO (cf. Algorithme 4) gère deux listes :

- Liste des tâches périodiques bloquées,
- Liste de jobs rouges prêts $J(t)$ à l'instant *t* : contenant les jobs rouges ayant leur date de réveil inférieure ou égale à l'instant courant *t* ou les jobs rouges non complètement exécutés.

Green-RTO est caractérisé par les éléments suivants :

- *idle* : un booléen prenant la valeur *FAUX* quand le processeur est actif (i.e. exécution d'un job) et *VRAI* quand le processeur est en mode veille (i.e. recharge de la batterie).

- *SlackTime(t)* : est une valeur calculée à l'instant *t* en utilisant l'algorithme *EDL* adapté au modèle *RTO*. C'est la laxité temporelle pendant laquelle le processeur peut rester inactif sans pour autant compromettre l'exécution des jobs rouges avant leur échéance.
  Soient *J(t)* la liste des jobs rouges prêts à l'instant *t* et soit d la plus petite échéance parmi les jobs rouges prêts ; la laxité temporelle se calcule selon l'algorithme EDL adapté au modèle RTO (cf. la Section 2.2.1).

- *SlackEnergy(t)* : la laxité énergétique du système qui représente le surplus d'énergie dans le système consommable à l'instant *t* tout en satisfaisant les contraintes temporelles et énergétiques des jobs rouges ayant leur échéance entre *t* et *d* (i.e. échéance du job le plus prioritaire à l'instant *t*). Sa valeur est déterminée en calculant la laxité énergétique minimale parmi celles de tous les jobs rouges prêts ayant leur date de réveil supérieure ou égale à *t* et leur échéance inférieure ou égale *d* :

$$SlackEnergy(t) = min(SlackEnergy(\tau_{i,k}, t)) \tag{3.28}$$

$SlackEnergy(\tau_{i,k}, t)$: est le surplus d'énergie consommable entre l'instant *t* et son échéance $d_{i,k}$ et est donné par la formule suivante :

$$SlackEnergy(\tau_{i,k}, t) = E(t) + \int_{t}^{d_{i,k}} P_r(x)dx - A_{i,k} \tag{3.29}$$

sachant que :
E(t) est l'énergie restante dans la batterie à l'instant *t*.
$P_r(x)$ est la puissance reçue à l'instant *x*.
$A_{i,k}$ est la somme des demandes d'énergie des job rouges ayant leur instant de réveil supérieur ou égal à *t* et leur échéance inférieure ou égale à $d_{i,k}$, telle que $A_{i,k} = \sum_{r_j \geq t, d_j \leq d_{i,k}} E_j$ où E_j est l'énergie requise pour l'exécution du job rouge la tâche τ_j.

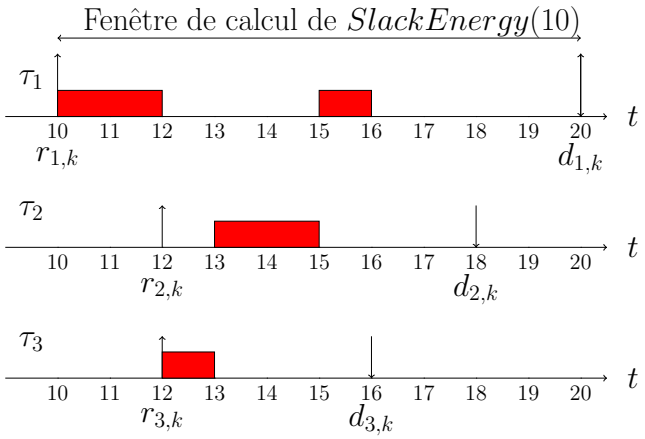


FIGURE 3.8 – Explication du calcul de la laxité énergétique

Remarque :
Une tâche périodique τ_i est exécutée à l'instant *t* si l'énergie disponible dans la batterie est suffisante et si la laxité énergétique du système à l'instant *t* est positive. Comme le montre la Figure 3.8, on suppose qu'un job rouge $\tau_{1,k}$ est réveillé et est le plus prioritaire à l'instant *t=10*. Le calcul de la laxité énergétique du système se fait alors sur l'intervalle [10, $d_{1,k}$] où $d_{1,k} = 20$ est l'échéance absolue de la tâche $\tau_{1,k}$. Il correspond à la laxité énergétique minimale parmi celles calculées pour les jobs $\tau_{1,k}$, $\tau_{2,k}$ et $\tau_{3,k}$ (le job de la tâche $\tau_{2,k}$et le job de la tâche $\tau_{3,k}$ sont réveillés après $r_{1,k}$ à l'instant $t = 12$ et ont leurs dates d'échéances inférieures à $d_{1,k}$ avec respectivement $d_{2,k} = 18$ et $d_{3,k} = 16$.

La formule, permettant de calculer la laxité énergétique du système sur l'intervalle $[t, d_{i,k}]$ prend en compte tous les jobs rouges arrivant entre t et $d_{i,k}$. Par conséquent pour éviter de refaire plusieurs fois le calcul de la

laxité énergétique, avec Green-RTO, si la laxité énergétique est calculée sur l'intervalle $[t, d_{i,k}]$ et est positive, tout job rouge ayant sa date de réveil supérieure ou égale à t et son échéance absolue inférieure à $d_{i,k}$, est exécutée immédiatement une fois élu. On peut alors établir la proposition suivante :

Proposition :
Nous considérons une configuration de n tâches périodiques temps réel fermes $\mathcal{T}^*$ ordonnancée selon Green-RTO. Supposons qu'à l'instant $t = X_1$, la laxité énergétique du système est calculée pour décider de l'exécution d'un job rouge ayant une échéance égale à X_2. Si la laxité énergétique du système obtenue est positive, tout job rouge élu ayant sa date de réveil supérieure ou égale à X_1 et son échéance inférieure ou égale à X_2 débute son exécution sans avoir à faire le calcul de la laxité énergétique.

Preuve :
Avec Green-RTO, seuls les jobs rouges sont exécutés. En effet, Green-RTO assure une sorte de cyclicité en n'exécutant que les $(s_i - 1)$ premiers jobs de chaque tâche sur une hyperpériode équivalente H^* = PPCM($s_1.T_1$, ..., $s_n.T_n$). Supposons qu'un job rouge $\tau_{i,k}$ est élu pour être exécuté à l'instant *t*, on doit s'assurer que tous les jobs devant impérativement s'exécuter entre *t* et $d_{i,k}$ ne manqueront pas d'énergie et ne violeront pas leur échéance.

Par conséquent, nous calculons la laxité énergétique du système à l'instant *t* qui prend en compte tous les jobs ayant leur instant de réveil supérieur ou égal à *t* et leur échéance inférieure ou égale à $d_{i,k}$:
$SlackEnergy(t) = E(t) + \int_t^{d_{i,k}} P_r(x)dx - A_{i,k}$ où $A_{i,k}$ est la somme des demandes d'énergie des jobs rouges ayant leur instant de réveil supérieur ou égal à *t* et leur échéance inférieure ou égale à $d_{i,k}$, telle que $A_{i,k} = \sum_{r_j \geq t, d_j \leq d_{i,k}} E_j$.

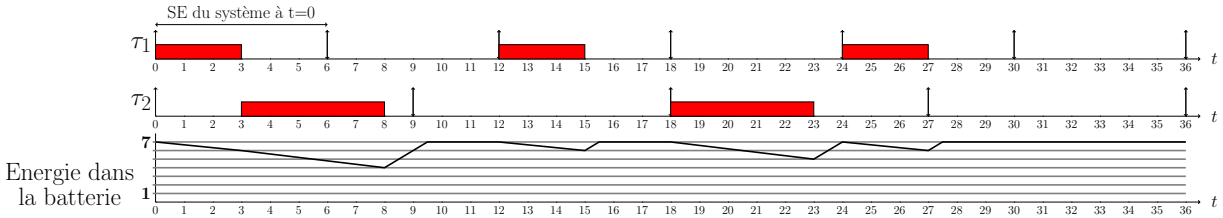
Si la laxité énergétique est positive ceci implique qu'il reste de l'énergie disponible après l'exécution de tous les jobs rouges prêts et ayant leur échéance sur l'intervalle $[t, d_{i,k})$. Et donc $\tau_{i,k}$ peut être exécuté. Cependant il peut être préempté à un instant t_1 par un job ayant son échéance inférieure ou égale à $d_{i,k}$. Comme tous les jobs ayant leur échéance inférieure ou égale à $d_{i,k}$ ont été pris en compte dans le calcul de la laxité énergétique à l'instant t et comme le nombre de jobs rouges est constant et dépend du paramètre de pertes défini pour chaque tâche, il n'y a pas nécessité de refaire la calcul de la laxité énergétique. □

Complexité algorithmique :
Le calcul de $SlackTime(t)$ correspond au calcul du vecteur dynamique des temps creux avec un modèle de tâches ordonnancées selon *RTO*. La complexité de ce calcul est exprimée en $O(\left\lceil \frac{R}{p} \right\rceil n)$ où *n* représente le nombre de tâches périodiques, *R* la plus grande échéance parmi les jobs rouges prêts et *p* la plus petite période. Le calcul de $SlackEnergy(\tau_{i,k}, t)$ correspond au calcul de $E(t) + P_r.(d_i - t) - A_{i,k}$. La complexité de ce calcul dépend alors du calcul de la somme $A_{i,k} = \sum_{r_j \geq r_{i,k}, d_j \leq d_{i,k}} E_j$. Dans le pire des cas, nous supposons que tous les paramètres de pertes sont très grands. Soient *R* la plus grande échéance parmi les jobs rouges et *p* la plus petite période. Le temps de calcul est alors de l'ordre de $O(n \left\lceil \frac{R}{p} \right\rceil)$.

4.1.3 Exemple illustratif

Considérons une configuration de tâches périodiques temps réel fermes $\tau = \{\tau_i(C_i, D_i, T_i, s_i, E_i)\}$ constituée de 2 tâches : $\tau_1(3, 6, 6, 2, 7)$ et $\tau_2(5, 9, 9, 2, 12)$.
Ces tâches sont indépendantes, préemptables, à échéances sur requêtes et profondément rouges. La batterie est pleine à l'instant *t=0* (i.e. *E(0)* $= E_{max} = 7$) et la puissance reçue est considérée constante avec $\bar{P_r} = 2$.
Dans cet exemple, comme $U_e = 110\%$ et $U_p = 105\%$, le système est donc en surcharge temporelle ($U_p > 1$) et en surcharge énergétique ($U_e > 1$). Par ailleurs, $R_e = \frac{\bar{P_e}}{\bar{P_r}} = \frac{2.5}{2} = 1.25$ et donc le système est surchargé énergétiquement.
Comme $U_p^* = 0.88 < 1$, les jobs rouges sont temporellement ordonnançables.
Comme $U_e^* = 0.76 < 1$, les jobs rouges sont énergétiquement ordonnançables.
La Figure 3.9 illustre l'ordonnancement selon Green-RTO de cette configuration sur une hyperpériode (H^*= $PPCM(T_1 s_1, T_2 s_2)$=36) ainsi que la courbe correspondant à la variation du niveau d'énergie dans la batterie en fonction du temps.
A l'instant *t=0*, le job rouge $\tau_{1,1}$ est le plus prioritaire. La laxité énergétique du système est calculée à *t=0*, elle

vaut la laxité énergétique du job $\tau_{1,1}$ comme aucun job prêt à part le job $\tau_{1,1}$ n'a son échéance inférieure à $d_{1,1}$. $SlackEnergy(\tau_{1,1}, 0) = E(0) + \int_0^6 P_r.d(x) - E_1 = 7 + 12 - 7 = 12$. Comme la laxité énergétique du système est positive et que la batterie n'est pas vide, le job $\tau_{1,1}$ commence son exécution à l'instant *t=0*.
A l'instant *t=3*, le job rouge $\tau_{2,1}$ est choisi pour être exécuté. Il n'y a aucun job rouge prêt à part le job $\tau_{2,1}$ ayant son échéance inférieure à $d_{2,1} = 9$, donc la laxité énergétique du système vaut celle du job $\tau_{2,1}$. $SlackEnergy(\tau_{2,1}, 3) = E(3) + \int_3^9 P_r.d(x) - E_2 = 6 + 12 - 12 = 6$. Comme le niveau d'énergie dans la batterie est $E(3) = 6$, et que la laxité énergétique du système calculée est positive, le job $\tau_{2,1}$ commence son exécution et est exécuté pendant 5 unités de temps. A l'instant *t=8*, il n'y a aucun job rouge prêt donc le processeur est inactif et la batterie se recharge et ainsi de suite. On note que sur toute l'hyperpériode H^*, il n'y a pas eu besoin de calculer la laxité temporelle pour recharger la batterie. Ceci est dû aux temps libérés par la non exécution des jobs bleus. De plus, on remarque que 50% des jobs sont exécutés. On en déduit donc que Green-RTO garantit la QoS minimale requise.



SE: laxité énergétique

FIGURE 3.9 – Ordonnancement selon l'algorithme Green-RTO

Dans la suite, nous proposons un nouvel ordonnanceur Green-BWP qui permet d'améliorer la QoS obtenue.

4.2 Algorithme Green-BWP

4.2.1 Principe

L'algorithme Green-BWP est basé sur deux algorithmes : BWP (Blue When Possible) et Green-RTO. Green-BWP est une amélioration de Green-RTO dans le sens où il va non seulement exécuter les jobs rouges mais il va également tenter d'exécuter des jobs bleus quand c'est possible. Les jobs rouges sont exécutés selon EDF et leurs échéances doivent impérativement être respectées. Par contre, les jobs bleus peuvent être abandonnés à tout moment et sont moins prioritaires que les jobs rouges.

Ajoutant la contrainte énergétique, Green-BWP diffère de l'algorithme BWP dans le fait qu'avant d'exécuter un job rouge ou bleu, il s'assure que le niveau d'énergie disponible dans la batterie alimentant le système est suffisant pour exécuter le job courant et les jobs rouges futurs dans le respect de leur échéance. A chaque instant, l'algorithme choisit le job le plus prioritaire parmi les jobs prêts. S'il existe un (ou des) job(s) rouge(s) prêt(s), celui ayant la plus proche échéance est élu pour être exécuté. Sinon, si aucun job rouge n'est prêt et s'il existe un (ou des) job(s) bleu(s) prêt(s), celui ayant la plus proche échéance est élu pour être exécuté. Par conséquent, le processeur est actif à l'instant *t* si l'énergie disponible dans la batterie est non nulle **ET** si la *laxité énergétique* du système calculée à l'instant *t* est positive. Ces conditions permettent d'assurer l'exécution correcte du job tout en satisfaisant les contraintes temporelles des jobs rouges futurs. Dans le cas contraire, en utilisant l'algorithme *EDL*, la laxité temporelle disponible à l'instant *t* est calculée permettant ainsi de mettre le processeur en mode veille le temps de recharger la batterie. Par ailleurs si à l'instant *t* aucun job rouge ou bleu n'est prêt, le processeur se met en mode veille.

4.2.2 Description de l'algorithme

Green-BWP (cf. Algorithme 5) gère trois listes :

- La liste des tâches périodiques bloquées.
- La liste des jobs rouges prêts *J(t)* à l'instant *t*.

Algorithm 5 Green-BWP

t : temps courant
$\mathcal{T}(t)$: la liste des tâches périodiques bloquées à l'instant *t*
J(t) : la liste des jobs rouges prêts à l'instant *t*
$\beta(t)$: la liste des jobs bleus prêts à l'instant *t*
while VRAI **do**
  idle ← VRAI
  if $J(t) = not(\phi)$ **then**
    soit τ_i le job rouge ayant la plus petite échéance
  else if $\beta(t) = not(\phi)$ **then**
    soit τ_i le job bleu ayant la plus petite échéance
  end if
  if $E(t) > E_{min}$ **then**
    Calculer $SlackEnergy(t)$
    if $SlackEnergy(t) \geq 0$ **then**
      idle← FAUX
    else
      /* $SlackEnergy(t) < 0$ */
      if E(t)=E_{max} **then**
        idle← FAUX
      else
        /* $E_{min} < E(t) < E_{max}$ */
        Calculer $SlackTime(t)$
        if $SlackTime(t) = 0$ **then**
          idle ← FAUX
        end if
      end if
    end if
    if idle = FAUX and $SlackEnergy(t) \geq 0$ **then**
      Execute τ_i
    end if
  end if
end while

- La liste des jobs bleus prêts $\beta(t)$ à l'instant *t*.

Green-BWP est caractérisé par les éléments suivants :

- *idle* : un booléen prenant la valeur *FAUX* quand le processeur est actif (i.e. exécution d'un job) et *VRAI* quand il est en mode veille (i.e. recharge de la batterie).

- *SlackTime(t)* : la laxité temporelle calculée à l'instant *t* en utilisant l'algorithme EDL. Elle représente le temps disponible pour recharger la batterie sans pour autant compromettre l'exécution des jobs rouges avant leur échéance. Elle est calculée en utilisant EDL adapté au modèle BWP (cf. la Section 2.2.2)

- *SlackEnergy(t)* : la laxité énergétique qui représente le surplus d'énergie dans le système consommable à l'instant *t* et *d* (i.e. d est l'échéance du job prêt le plus prioritaire) en considérant les contraintes temporelles de tous les jobs rouges prêts ayant leur échéance entre *t* et *d*. Sa valeur est déterminée en calculant la laxité énergétique de tous les jobs rouges prêts ayant leur date de réveil supérieure ou égale à l'instant *t* et leur échéance inférieure ou égale à *d* :

$$SlackEnergy(t) = min(slackEnergy(t, \tau_{i,k}) \tag{3.30}$$

- *SlackEnergy(t,$\tau_{i,k}$) :* le surplus d'énergie qui peut être consommé entre l'instant *t* et l' échéance $d_{i,k}$ du job $\tau_{i,k}$ (c'est le $k^{ième}$ job de la tâche τ_i pouvant être rouge ou bleu).

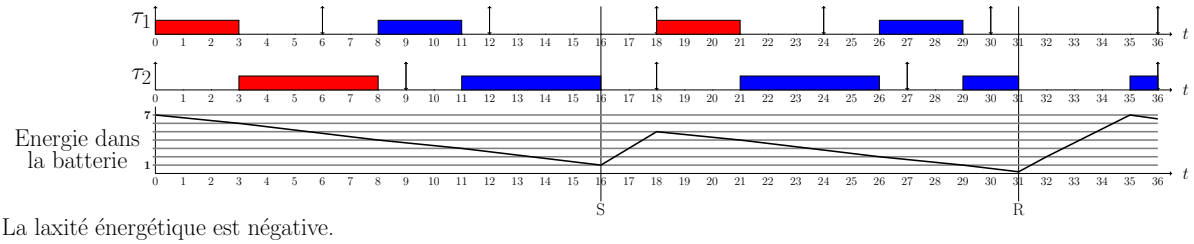
$$SlackEnergy(t, \tau_{i,k}) = E(t) + \int_t^{d_{i,k}} P_r(x)dx \tag{3.31}$$

avec :
$E(t)$: l'énergie restante dans la batterie à l'instant *t*.
$P_r(x)$: la puissance reçue à l'instant *x*.
$A_{i,k}$: l'énergie totale requise par les jobs rouges ayant leur instant de réveil supérieur ou égal à l'instant *t* et ayant une échéance d_j inférieure ou égale à $d_{i,k}$ telle que $A_{i,k} = \sum_{r_j \geq t, d_j \leq d_{i,k}} E_j$.
E_j : l'énergie requise par le job de la tâche τ_j.

Complexité algorithmique :
La complexité de calcul de la laxité temporelle et la laxité énergétique sous Green-BWP est la même que celle observée sous Green-RTO.

4.2.3 Exemple illustratif

Nous reprenons la même configuration que celle considérée pour l'ordonnancement avec Green-RTO. La Figure 3.10 illustre l'ordonnancement de cette configuration selon Green-BWP ainsi que la courbe de variation du niveau d'énergie dans la batterie. A l'instant $t = 0$, la batterie est pleine. Les jobs rouges sont ordonnancés



S: La laxité énergétique est négative.
R: L'énergie dans la batterie n'est pas suffisante.

FIGURE 3.10 – Ordonnancement selon l'algorithme Green-BWP

suivant EDF tant qu'il y a suffisamment d'énergie dans la batterie. À l'instant $t = 16$, la laxité énergétique est négative. Par conséquent, aucun job ne peut être exécuté et la laxité temporelle est calculée pour déterminer le temps maximal disponible permettant de recharger la batterie sans compromettre l'exécution des jobs rouges.

Ainsi, le processeur pourra être inactif pour que la batterie se recharge. A l'instant $t = 18$, le job de la tâche τ_1 est rouge parce que le job bleu de la tâche τ_1 a manqué son échéance. Comme il est le plus prioritaire, il débute son exécution et s'exécute pendant 3 unités de temps. A l'instant $t = 21$, il n'y a aucun job rouge à l'état prêt alors le job bleu de la tâche τ_2 commence son exécution et la finit à l'instant *t=26*.

A l'instant $t = 31$, le job bleu de la tâche τ_2 est interrompu car la batterie est vide. Suite à cela le calcul de la laxité temporelle est fait et le processeur reste inactif pendant 4 unités de temps. A l'instant $t = 35$, la batterie est pleine et le job bleu de la tâche τ_2 continue son exécution.

Comme on peut le remarquer à partir de la Figure 3.10, l'ordonnanceur Green-BWP exécute plus de jobs dans le respect de leurs contraintes temporelles et énergétiques que l'ordonnanceur Green-RTO. Dans l'exemple considéré, 70% des jobs sont exécutés avec Green-BWP alors que 50% le sont avec Green-RTO (cf. Figure 3.9).

5 Evaluation des performances

Cette partie a pour objectif de décrire les résultats de simulations effectuées dans le but d'évaluer comparativement la performance des ordonnanceurs Green-RTO, Green-BWP et EDeg. Rappelons que notre objectif est d'utiliser au mieux l'énergie disponible en vue d'optimiser la QoS, c'est-à-dire le ratio des jobs périodiques qui s'exécutent dans le respect de leur échéance.

5.1 Eléments de simulation

Pour cette étude, nous avons développé un simulateur en langage C. Ce simulateur est composé d'un générateur de tâches périodiques qui prend en entrée les paramètres suivants :

- *n* : le nombre de tâches périodiques constituant la configuration,
- *PPCMmax* : la valeur représentant le ppcm maximal des périodes des tâches constituant la configuration,
- U_p : la charge processeur,
- $\bar{P_e}$: la puissance moyenne de consommation énergétique,
- s_i : la valeur des pertes uniformes.

En sortie, la configuration de tâches temps réel fermes obtenue est $\mathcal{T}*$ = $\{\tau_i(C_i,\ D_i,\ T_i,\ s_i,\ E_i)$, i=1 à $n\}$.

- Les tâches périodiques sont indépendantes, préemptables et à échéances sur requêtes ($D_i = T_i$).
- Les périodes des tâches T_i sont aléatoirement choisies et leur choix dépend du *PPCMmax* donné en entrée.
- Les durées d'exécution des tâches au pire-cas C_i sont générées aléatoirement et le choix dépend de $U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$.
- Les demandes énergétiques des tâches E_i sont générées aléatoirement et le choix dépend de $\bar{P_e} = \sum_{i=1}^{n} \frac{E_i}{T_i}$.
- Aucune corrélation n'existe entre les valeurs de E_i et de C_i.

Les valeurs du facteur équivalent du processeur U_p^* et du facteur équivalent énergétique U_e^* dépendent de la configuration de tâches obtenue et du paramètre de pertes fixé.

5.2 Métriques étudiées

Le simulateur permet d'ordonnancer en-ligne une configuration de tâches périodiques selon Green-RTO, Green-BWP et EDeg pour évaluer différents critères de performances qui sont les suivants :

- **Taux de respect** : le taux de jobs complètement exécutés avant leur échéance.

$$TQoS = \frac{nombre\ d'échéances\ satisfaites}{nombre\ total\ de\ jobs}.100 \tag{3.32}$$

Le taux de respect est la métrique primordiale qui reflète la QoS du système.

- **Taux de batterie pleine** : le pourcentage de temps durant lequel la batterie est pleine.

$$Tbatterie\_pleine = \frac{Temps\ cumulé\ durant\ lequel\ la\ batterie\ est\ remplie}{Temps\ de\ simulation}.100 \tag{3.33}$$

On étudie cette métrique pour évaluer le temps pendant lequel la batterie reste pleine. Elle permet d'avoir une idée vis-à-vis de la taille de la batterie selon le profil énergétique étudié et l'ordonnanceur utilisé. Ainsi on pourra avoir une idée de la dimension de batterie requise pour un des algorithmes étudiés si le nombre de fois durant lequel la batterie est pleine est grand cela signifie que l'on peut utiliser une batterie de capacité plus petite.

- **Taux d'oisiveté** : le pourcentage de temps où il n'y a aucun job à exécuter (le processeur est inactif).

$$Tidle\_processor = \frac{Temps\ processeur\ idle}{Temps\ de\ simulation}.100 \tag{3.34}$$

Cette métrique permet de mesurer combien de temps le processeur est inactif. Ainsi elle permet de donner une idée sur le comportement du système vis-à-vis de la charge de traitement considérée. Par conséquent en fonction du taux d'oisiveté obtenu avec un algorithme, il sera possible par exemple de prendre des décisions pour choisir un processeur plus ou moins performant en fonction des besoins en temps de traitement du système ou bien il sera possible de prévoir le comportement du système vis-à-vis de l'occurrence d'une tâche apériodique.

- **Taux de préemption** : le nombre de préemptions vis-à-vis du nombre total de jobs exécutés.

$$Tpréemption = \frac{nombre\ de\ préemptions}{Nombre\ total\ de\ jobs}.100 \tag{3.35}$$

Cette métrique permet d'évaluer les surcoûts dus aux changements de contexte. En effet, la préemption d'une tâche engendre un changement de contexte qui consiste à sauvegarder l'état de la tâche courante interrompue (état des registres du processeur, pointeur de pile, etc.) puis restaurer celui de la tâche la plus prioritaire. Cette opération peut s'avérer plus ou moins coûteuse vis-à-vis du temps processeur consommé pour les accès mémoires qui dépend de l'architecture matérielle, du système d'exploitation et du type de tâches considérés. Moins il y a de préemptions, moins il y a des changements de contexte et plus l'ordonnanceur est performant.

- **Ratio de gaspillage du temps processeur** : le temps processeur utilisé par des jobs ayant violé leur échéance par rapport à la durée totale pendant laquelle le processeur est actif.

$$Ttemps\_gaspillée = \frac{durée\ totale\ d'exécution\ des\ jobs\ en\ échec}{durée\ totale\ d'activité\ du\ processeur}.100 \tag{3.36}$$

C'est une métrique importante qui permet d'évaluer le gaspillage de temps engendré par les différents algorithmes lorsque ceux-ci lancent l'exécution de jobs qui finalement ne respectent pas leur échéance.

- **Ratio d'énergie gaspillée** : le ratio d'énergie utilisée par des jobs ayant violé leur échéance par rapport à la quantité totale d'énergie consommée.

$$Tenergie\_gaspillée = \frac{énergie\ totale\ utilisée\ par\ des\ jobs\ en\ échec}{énergie\ totale\ utilisée}.100 \tag{3.37}$$

La consommation énergétique d'une configuration de tâches impacte fortement l'autonomie du système. Par conséquent, nous évaluons la quantité totale d'énergie gaspillée engendrée par l'exécution de jobs qui manquent leur échéance. Cette métrique permet de montrer que Green-RTO ne cause pas de gaspillage ce qui n'est pas le cas de Green-BWP et de EDeg. Cependant, rappelons-le, Green-RTO permet d'offrir uniquement une QoS minimale.

- **Taux de l'overhead** : C'est le nombre de calculs de la laxité énergétique rapporté au nombre total de jobs exécutés.

$$Toverhead = \frac{nombre\ de\ calculs\ de\ laxité\ énergétique}{nombre\ total\ de\ jobs}.100 \tag{3.38}$$

Le nombre de calculs de la laxité énergétique peut s'avérer coûteux en termes de temps de calcul et augmente quand le nombre de jobs à exécuter grandit. Par conséquent nous étudions le taux d'overhead engendré par l'exécution d'une configuration de tâches. Plus le taux de l'overhead est important moins l'ordonnanceur est performant en termes de temps de calcul.

Remarque :
Dans le cadre de cette étude, nous avons fait quelques modifications de l'algorithme EDeg de façon à prendre en compte le cas de surcharge temporelle et/ou énergétique. En effet, EDeg ne traite pas le cas de surcharge, et ne tolère pas les violations d'échéances des jobs. Par conséquent, si un job, en phase d'exécution à un instant *t*, est à court d'énergie et si la laxité énergétique du système calculée à cet instant est nulle, nous faisons en sorte que ce job soit supprimé. Ainsi le processeur bascule en mode *idle* pour permettre la recharge de la batterie. Par suite, dans le cas de surcharge temporelle et/ou énergétique, comme il y aura forcément des jobs qui violeront leur échéance, la QoS avec EDeg ne sera pas égale à 100%.

5.3 Variation du ratio de criticité énergétique

Dans cette partie, les simulations présentées ci-après permettent d'évaluer les performances des ordonnanceurs en faisant varier le ratio de criticité énergétique du système R_e et en fixant la charge processeur U_p de façon à considérer différents profils temporels, comme suit :
- un système moyennement chargé temporellement ($U_p = 60\%$),
- un système fortement chargé temporellement ($U_p = 90\%$),
- un système surchargé temporellement ($U_p = 120\%$) qui se traduit par le fait que le processeur n'a pas une capacité de traitement suffisante pour traiter toutes les tâches.

Nous rappelons, que le ratio de criticité R_e représente le rapport de la puissance moyenne consommée par la configuration, $\bar{P_e}$, sur la puissance reçue, P_r, par la source d'énergie. Si ce ratio de criticité est inférieur à 1 ceci veut dire que la consommation moyenne énergétique est inférieure à l'énergie délivrée au système. Si au contraire il est supérieur à 1 ceci implique que l'énergie délivrée au système est inférieure à la demande énergétique de la configuration de tâches et que le système est fortement contraint énergétiquement.

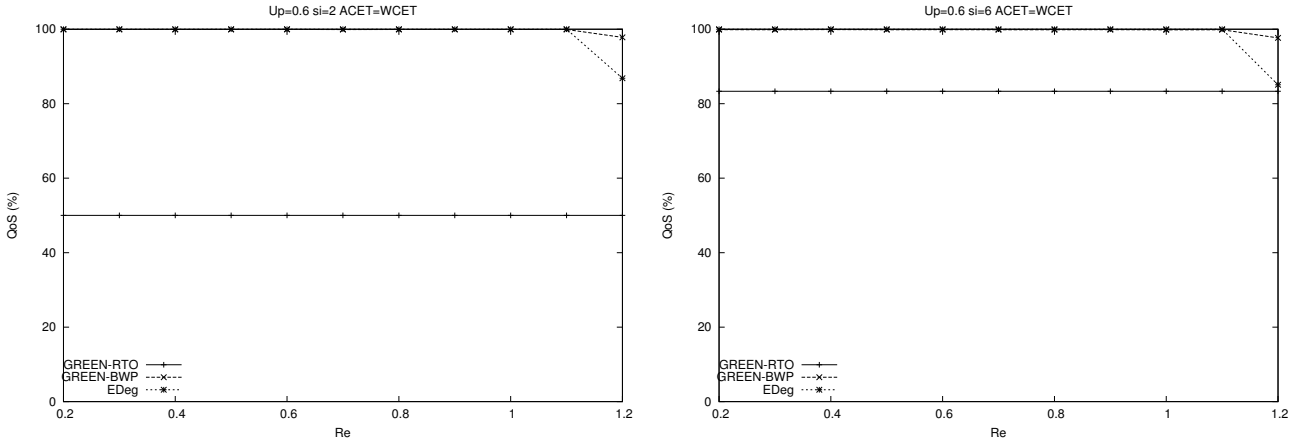
Si l'on considère un processeur alimenté par une batterie rechargée par des cellules photovoltaïques, la puissance reçue dépendra de la surface totale des cellules. Notre objectif est alors de pouvoir évaluer les performances des ordonnanceurs suivant le profil énergétique et temporel du système.

Dans cette partie, les simulations sont effectuées sur 6 hyperpériodes $H^* = PPCM(T_1 s_1, ...,T_n s_n)$. A chaque pas de simulation, 100 configurations de tâches différentes sont générées et chaque configuration est constituée de 10 tâches périodiques. Par ailleurs, nous supposons que la batterie est initialement chargée et a une capacité égale à $C = H^*.P_r$

5.3.1 Etude du taux de respect

Pour chaque profil temporel étudié, nous comparons dans un premier temps les performances des algorithmes en fonction du ratio de criticité énergétique, R_e, en considérant que toutes les tâches s'exécutent selon leur durée d'exécution au pire-cas (WCET). Dans un deuxième temps, nous considérons un cas plus proche de la réalité et nous supposons alors que les tâches s'exécutent selon une durée d'exécution effective inférieure à leur durée d'exécution au pire-cas (ACET) telle que ACET=0.75xWCET.

5.3.1.1 Système moyennement chargé temporellement. Les graphiques sur les Figures 3.11 et 3.12 concernent le cas d'un système moyennement chargé temporellement ($U_p = 60\%$). Sur la Figure 3.11(a) et la Figure 3.11(b), nous considérons que les tâches ont une durée d'exécution pire-cas alors que sur la Figure 3.12(a) et la Figure 3.12(b), elles ont une durée d'exécution effective inférieure au pire-cas telle que ACET=0.75xWCET. De plus, sur les Figures 3.11(a) et 3.12(a), le paramètre de pertes est égal à 2 ($s_i = 2$) et sur les Figures 3.11(b) et 3.12(b) le paramètre de pertes vaut 6 ($s_i = 6$).



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.11 – Taux de respect ($U_p = 0.6$ et ACET=WCET)

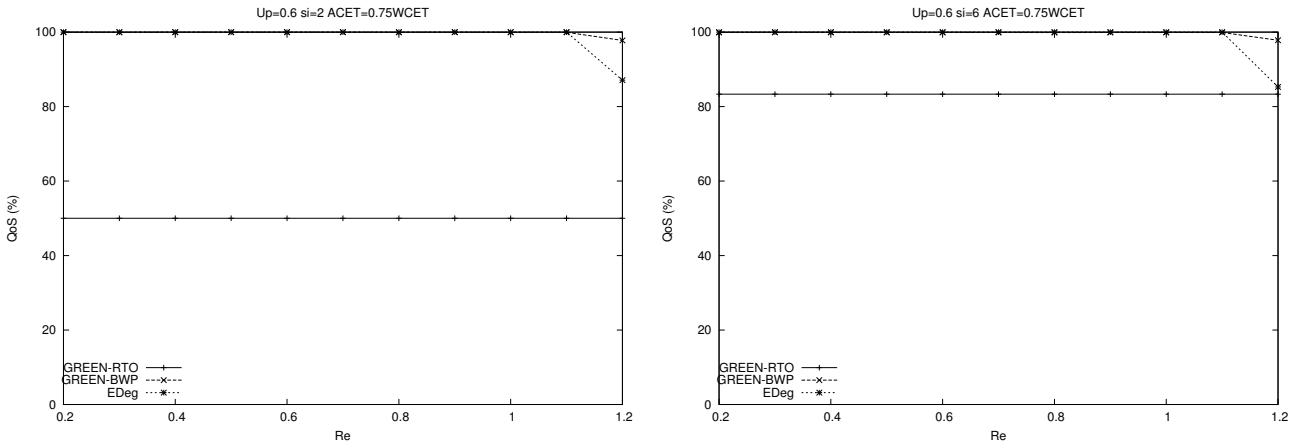
Nous remarquons que :

- Il y a une certaine similarité entre l'aspect des courbes obtenues sur la Figure 3.11(a) et la Figure 3.12(a) et entre l'aspect des courbes obtenues sur la Figure 3.11(b) et la Figure 3.12(b). Par conséquent, dans un système sous-chargé temporellement, le fait que les tâches s'exécutent selon une durée inférieure à leur durée d'exécution au pire-cas n'a pas d'effet significatif sur le taux de respect obtenu.

- Quand le système n'est pas surchargé énergétiquement ($R_e \leq 1$) :
  - Le taux de respect obtenu est de 100% avec EDeg et tend vers 100% avec Green-BWP.
  - Le taux de respect offert avec Green-RTO est toujours constant quelle que soit la valeur du ratio de criticité énergétique. Le taux de respect obtenu est égal à 50% quand le paramètre de pertes s_i vaut 2. La différence entre les taux de respect obtenus avec Green-RTO et ceux obtenus avec EDeg et Green-BWP est dans ce cas proche de 50%.
  - Quand le paramètre de pertes s_i vaut 6, Green-RTO offre un taux de respect constant égal à 83%. La différence entre les taux de respect de Green-RTO et ceux de EDeg et de Green-BWP est alors de 17%.

- Quand le système est en surcharge énergétique :
  - Green-RTO offre toujours un taux de respect constant indépendant du ratio de criticité énergétique qui est de 50% quand le paramètre de pertes vaut 2 et de 83% quand le paramètre de pertes vaut 6.
  - Quand les tâches sont exécutées sur des durées d'exécution pire-cas (cf. Figure 3.11(a) et Figure 3.11(b)), nous remarquons que sur la Figure 3.11(a), pour $s_i = 2$, Green-BWP et EDeg affichent respectivement un taux de respect de 98% et 87% pour $R_e = 1.2$.
  - Quand les tâches sont exécutées avec des durées d'exécution inférieures à celles pire-cas telles que ACET=0.75xWCET, nous remarquons que sur la Figure 3.11(a), pour $s_i = 2$, Green-BWP et EDeg affichent respectivement un taux de respect de 98% et 87% pour $R_e = 1.2$.

5.3.1.2 Système fortement chargé temporellement. Les Figures 3.13 et 3.14 concernent un système fortement chargé temporellement ($U_p = 0.9$). Sur les Figures 3.13(a) et 3.13(b), les tâches s'exécutent sur leurs durées d'exécution pire-cas alors que sur les Figures 3.14(a) et 3.14(b), elles s'exécutent sur leur durée d'exécution effective qui est inférieure au pire-cas telle que ACET=0.75xWCET. De plus, sur la Figure 3.11(a) et la



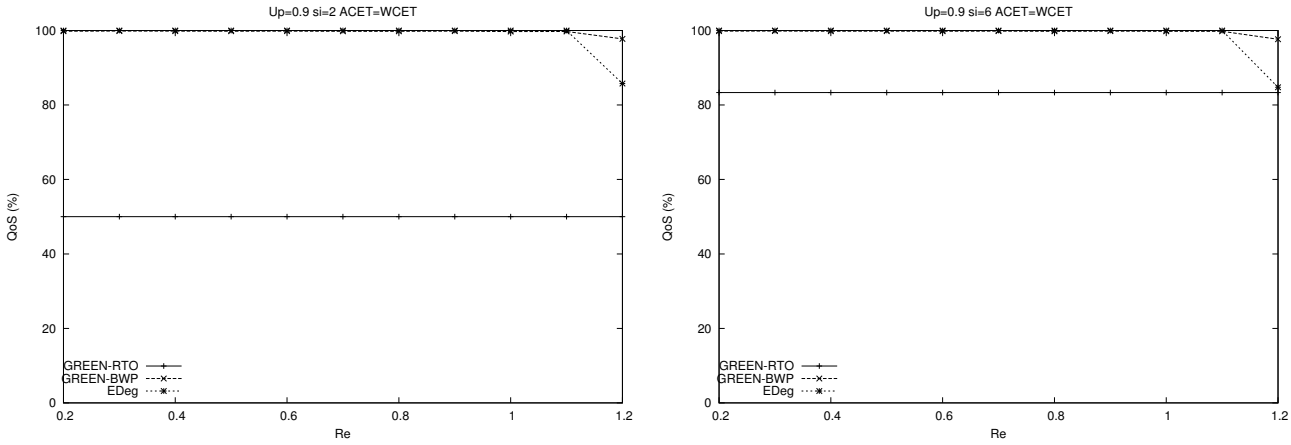
(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.12 – Taux de respect ($U_p = 0.6$ et ACET=0.75xWCET)

Figure 3.12(a), le paramètre de pertes considéré vaut 2 ($s_i = 2$) et sur la Figure 3.13(a) et la Figure 3.14(b) le paramètre de pertes vaut 6 ($s_i = 6$).

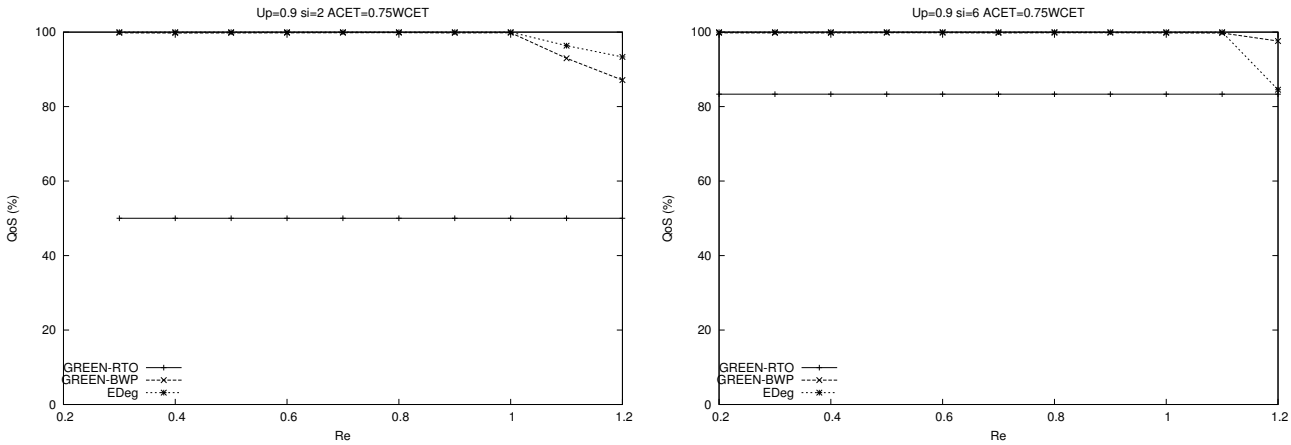
- Quand le système est **sous-chargé énergétiquement** ($R_e \leq 1$), nous observons que le taux de respect est égal à 100% avec EDeg et tend vers 100% avec Green-BWP.

- Quand le système est **surchargé énergétiquement** :
  - Quand les tâches sont exécutées sur des durées d'exécution pire-cas, nous remarquons que sur la Figure 3.13(a), pour $s_i = 2$, Green-BWP et EDeg affichent respectivement un taux de respect de 98% et 87% pour $R_e = 1.2$. De plus, sur la Figure 3.13(b), Green-BWP et EDeg affichent respectivement un taux de respect de 97% et 85% pour $R_e = 1.2$.
  - Quand les tâches sont exécutées avec des durées d'exécution inférieures à celles pire-cas telles que ACET=0.75xWCET, nous remarquons que, pour $s_i = 2$, Green-BWP et EDeg affichent respectivement un taux de respect de 98% et 87% pour $R_e = 1.2$. Sur la Figure 3.14(b), Green-BWP et EDeg affichent respectivement un taux de respect de 98% et 85% pour $R_e = 1.2$.



(a) $s_i = 2$ (b) $s_i = 6$

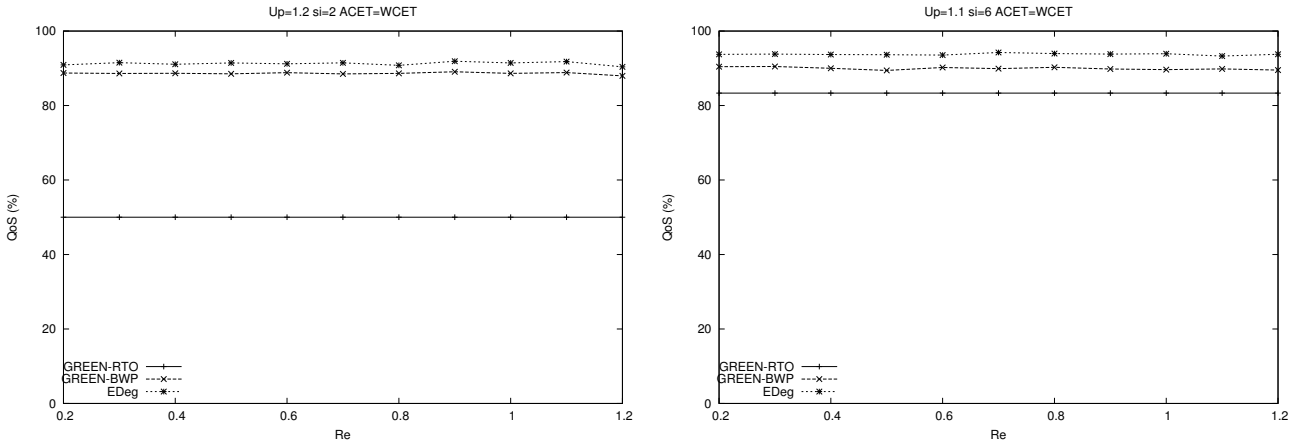
FIGURE 3.13 – Taux de respect ($U_p = 0.9$ et ACET=WCET)



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.14 – Taux de respect ($U_p = 0.9$ et ACET=0.75xWCET)

5.3.1.3 Système surchargé temporellement. Les Figures 3.15 et 3.16 concernent le cas d'un système surchargé temporellement.



(a) $s_i = 2$ (b) $s_i = 6$

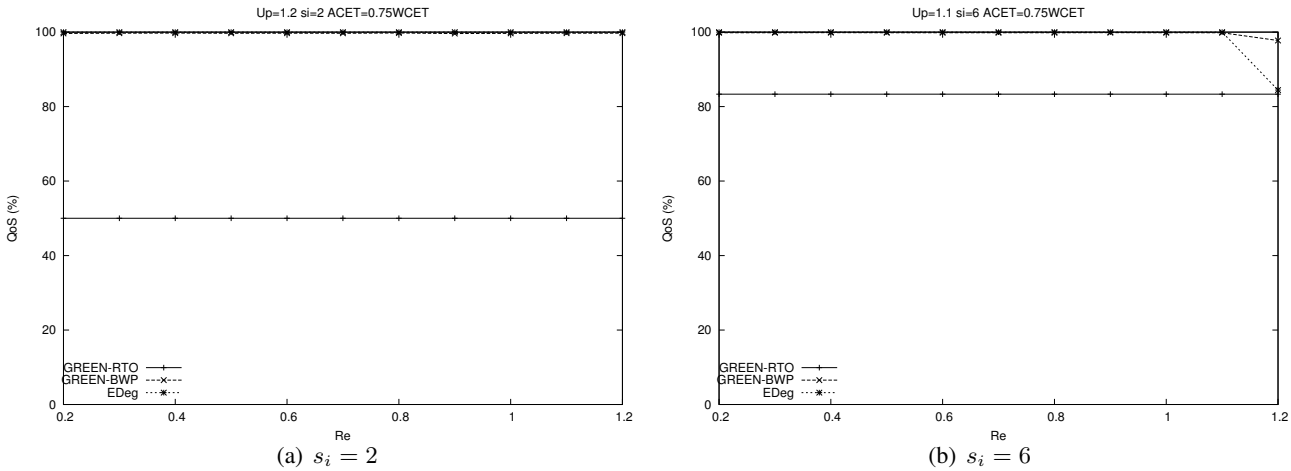
FIGURE 3.15 – Taux de respect ($U_p = 1.2$(a), $U_p = 1.1$(b)) et ACET=WCET

Nous remarquons que :

- Green-RTO offre un taux de respect toujours constant quel que soit le ratio de criticité énergétique. Il affiche toujours un taux de respect de 50% quand $s_i = 2$ et 83% quand $s_i = 6$.

- Quand les tâches sont exécutées sur des durées d'exécution pire-cas (cf. Figure 3.11(a) et Figure 3.11(b)), pour $s_i = 2$ et $U_p = 120\%$, le taux de respect obtenu vaut environ 98% avec Green-BWP et 84% avec EDeg. Et pour $s_i = 6$ et $U_p = 110\%$, Green-BWP et EDeg affichent respectivement un taux de respect égal à 90% et 94%. Ceci met en avant la performance de Green-BWP par rapport à Green-RTO et EDeg en termes de QoS, car il offre une meilleure QoS.

- Quand les tâches sont exécutées avec des durées d'exécution inférieures à celles pire-cas telles que ACET=0.75xWCET (cf. Figure 3.12(a) et Figure 3.12(b)) :
  - si le système est **sous-chargé énergétiquement**, Green-BWP et EDeg affichent des taux de respect égaux à 100%.
  - Quand le système est **surchargé énergétiquement**, Green-BWP et EDeg affichent respectivement un taux de respect environ égal à 97% et 84% pour $R_e = 1.2$ (cf. Figure 3.12(a) et Figure 3.12(b)).



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.16 – Taux de respect ($U_p = 1.2$(a), $U_p = 1.1$(b)) et ACET=0.75xWCET

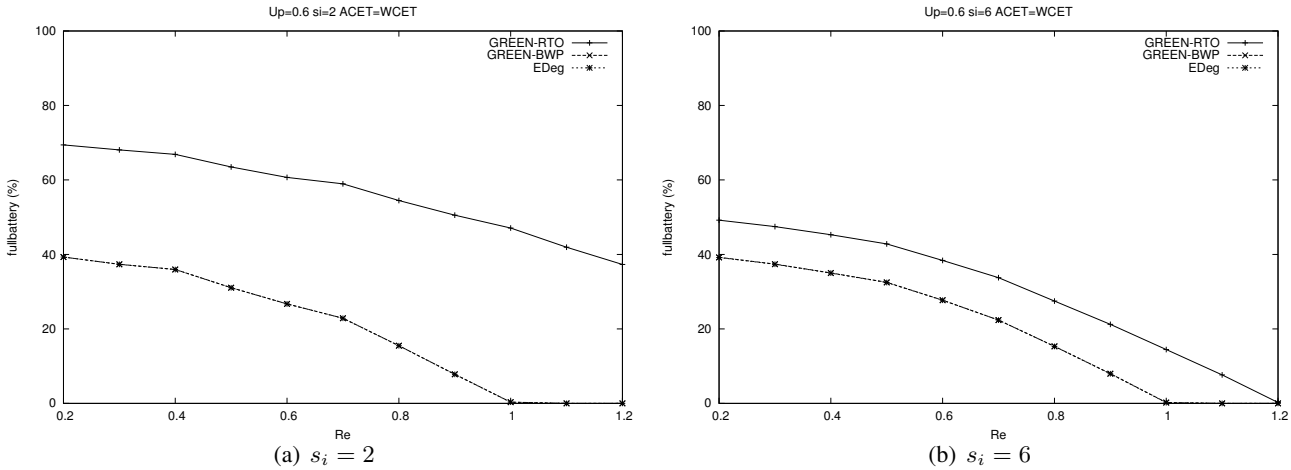
Nous pouvons ainsi dire que dans un système surchargé temporellement et non surchargé énergétiquement, si les tâches s'exécutent sur une durée d'exécution effective inférieure à la durée pire-cas, Green-BWP et EDeg se révèlent plus performants en termes de QoS que Green-RTO car ils offrent une QoS de 100%. Par contre, dans un système surchargé énergétiquement, Green-BWP s'avère le plus performant offrant une QoS supérieure à celle obtenue avec EDeg d'environ 13%.

5.3.1.4 Synthèse sur l'étude du taux de respect

Nous concluons que :
- Green-RTO offre toujours une QoS constante indépendante du profil temporel et énergétique car il n'exécute que les jobs rouges. De plus, la QoS dépend directement du paramètre de pertes fixé et s'améliore quand le paramètre de pertes fixé augmente. Green-RTO peut ainsi servir de référence parce qu'il n'exécute que les jobs rouges et par conséquent, il fournit toujours la QoS minimale acceptable.
- Green-BWP et EDeg offrent toujours de meilleurs taux de respect que Green-RTO.
- Bien que la QoS soit minimale avec Green-RTO, son avantage réside dans la répartition régulière dans le temps des jobs qui respectent leur échéance même si ce taux de réussite est minimum.
- Dans un système **non surchargé temporellement** et **non surchargé énergétiquement** :
- Avec EDeg, le taux de respect vaut 100%.
- Avec Green-BWP, le taux de respect tend vers 100%. En effet, de très faibles pertes peuvent survenir quand quelques jobs bleus manquent leur échéance par manque de temps dû à des jobs rouges plus prioritaires qui peuvent les préempter à tout instant.
- Dans un système **surchargé temporellement** :
- Si les tâches s'exécutent selon leur durée au pire-cas, Green-BWP et EDeg affichent des taux de respect inférieurs à 100% parce que des jobs violent leur échéance par manque de temps. L'avantage de Green-BWP réside dans le fait que les jobs non exécutés sont contrôlés et sont des jobs bleus alors qu'avec EDeg on ne contrôle pas l'identité des jobs en échec. Par conséquent, avec Green-BWP les violations d'échéance de jobs bleus ont pour conséquence la dégradation de la QoS du système alors qu'avec EDeg, les violations d'échéance de jobs peuvent avoir des conséquences significatives et peuvent engendrer la chute du système pour cause d'instabilité.
- Si les tâches s'exécutent selon une durée effective inférieure à leur durée au pire-cas, le taux de respect observé avec Green-BWP et EDeg est meilleur que dans le cas où les tâches s'exécutent selon leur WCET. Ceci est dû au fait que chaque tâche libère une quantité de temps non utilisée WCET-ACET. Ainsi, cette quantité de temps CPU additionnel peut être utilisée pour exécuter un plus grand nombre de jobs dans le respect de leurs échéances.
- Dans un système **surchargé énergétiquement** :
EDeg et Green-BWP affichent des QoS inférieures à 100% et décroissantes suivant que la charge énergétique augmente. Plus la charge énergétique est importante plus l'écart entre Green-BWP et Green-RTO diminue car

Green-BWP aura moins de temps pour exécuter les jobs bleus et n'exécutera donc que les jobs rouges. Green-BWP s'avère être l'ordonnanceur le plus performant en termes de QoS car les jobs abandonnés par manque d'énergie sont les jobs bleus non urgents. Quant à EDeg, il ne réagit pas correctement en situation de surcharge énergétique et il n'y a aucun contrôle possible sur l'identité des jobs abandonnés.

5.3.2 Etude du niveau d'énergie dans la batterie



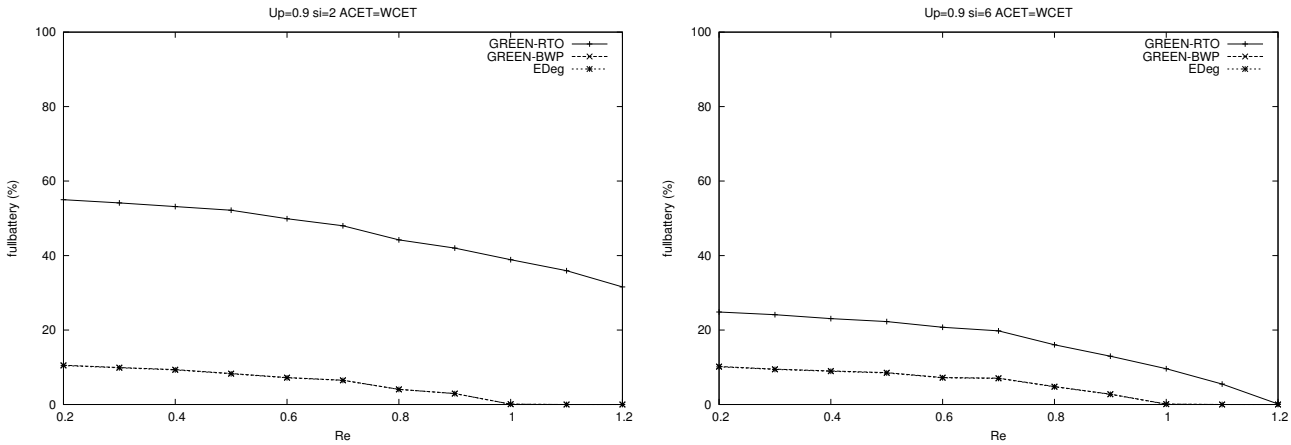
(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.17 – Pourcentage de temps durant lequel la batterie est pleine ($U_p = 0.6$ et ACET=WCET)

5.3.2.1 Système moyennement chargé temporellement. La Figure 3.17(a) et la Figure 3.17(b) montrent le pourcentage de temps durant lequel la batterie est pleine.

Pour $s_i = 2$ (cf. la Figure 3.17(a)), nous remarquons que :
- Avec Green-RTO le pourcentage de temps durant lequel la batterie est pleine est toujours supérieur à ceux obtenus avec Green-BWP et EDeg.
- Green-BWP et EDeg offrent les mêmes résultats au niveau de la batterie.
- Pour $R_e = 0.2$, la batterie est pleine 69% du temps avec Green-RTO, 39% du temps avec Green-BWP et 39% du temps avec EDeg.
- Le pourcentage de temps durant lequel la batterie est pleine décroît au fur et à mesure que la criticité énergétique augmente et ce, quel que soit l'ordonnanceur.
- L'écart maximal affiché entre la courbe de Green-RTO et les courbes de Green-BWP et EDeg est de 30% pour $R_e = 0.2$.
- Pour $R_e = 1$, la batterie est pleine 47% du temps avec Green-RTO, alors qu'avec Green-BWP et EDeg, elle n'est jamais pleine.

Pour $s_i = 6$ (cf. la Figure 3.17(b)), nous remarquons que :
- Pour $R_e = 0.2$, la batterie est pleine 49% du temps avec Green-RTO, 39% du temps avec Green-BWP et 39% du temps avec EDeg.
- Le nombre de fois durant lequel la batterie est pleine avec Green-RTO diminue quand le paramètre de pertes augmente car le nombre de jobs rouges augmente et la batterie est donc plus sollicitée.
- L'écart maximal affiché entre la courbe de Green-RTO et les courbes de Green-BWP et EDeg est de 14% pour $R_e = 1$. Pour $s_i = 6$, le nombre de jobs rouges augmente est par conséquent, Green-RTO présente un pourcentage de temps durant lequel la batterie est pleine inférieur de 30% à celui obtenu pour $s_i = 2$. De plus, Green-RTO présente un pourcentage de temps, durant lequel la batterie est pleine, s'approchant de ceux de Green-BWP et EDeg et devenant nul pour $R_e = 1.2$.
- Pour $R_e = 1$, la batterie est pleine 15% du temps avec Green-RTO alors qu'elle ne l'est jamais avec Green-BWP ou EDeg.



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.18 – Pourcentage de temps durant lequel la batterie est pleine ($U_p = 0.9$ et ACET=WCET)

5.3.2.2 Système fortement chargé temporellement. La Figure 3.18(a) et la Figure 3.18(b) représentent les pourcentages de temps durant lesquels la batterie est pleine.

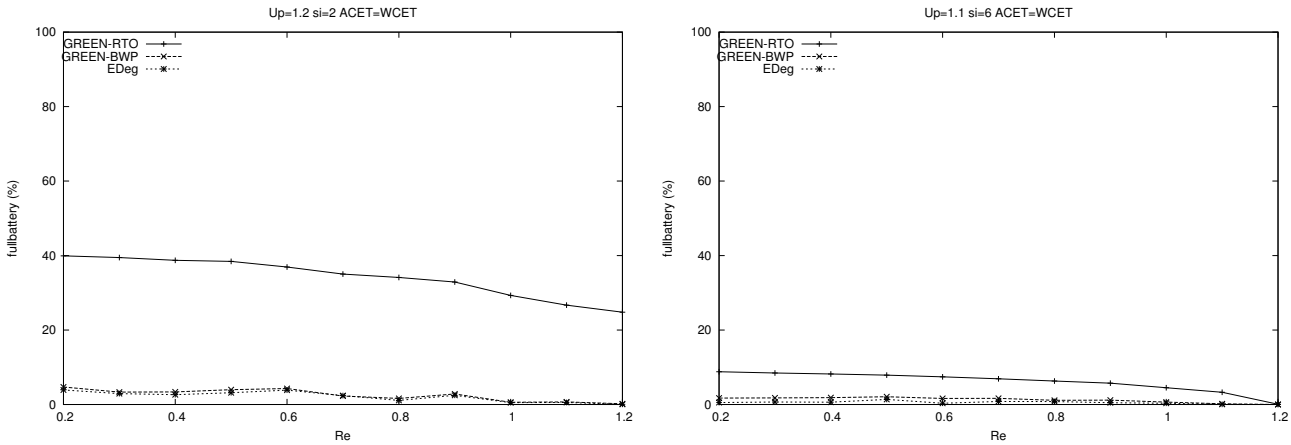
Pour $s_i = 2$ (cf. la Figure 3.18(a)), nous remarquons que :
- Pour $R_e = 0.2$, la batterie est pleine 55% du temps avec Green-RTO, 10% du temps avec Green-BWP et 10% du temps avec EDeg.
- L'écart maximal affiché entre la courbe de Green-RTO et les courbes de Green-BWP et EDeg est d'environ 45% pour $R_e = 0.2$. En effet, le pourcentage de temps durant lequel la batterie est pleine est environ 5 fois plus grand avec Green-RTO qu'avec Green-BWP et EDeg. Ceci signifie que Green-RTO offre plus de laxité énergétique et temporelle parce qu'il n'exécute que les jobs rouges et donc l'énergie dans la batterie est moins utilisée qu'avec les autres ordonnanceurs.
- Pour $R_e = 1$, la batterie est pleine 37% du temps avec Green-RTO, alors qu'avec Green-BWP et EDeg, elle n'est jamais pleine.

Pour $s_i = 6$ (cf. la Figure 3.18(b)), nous remarquons que :
- Pour $R_e = 0.2$, la batterie est pleine 25% du temps avec Green-RTO, 10% du temps avec Green-BWP et 10% du temps avec EDeg.
- Le nombre de fois durant lequel la batterie est pleine avec Green-RTO diminue quand le paramètre de pertes augmente car le nombre de jobs rouges augmente et la batterie est donc plus sollicitée.
- L'écart maximal affiché entre la courbe de Green-RTO et les courbes de Green-BWP et EDeg est de 15 pour $R_e = 0.2$.
- Pour $R_e = 1$, la batterie est pleine 9% du temps avec Green-RTO alors qu'elle ne l'est jamais avec Green-BWP ou EDeg.

5.3.2.3 Système surchargé temporellement Les Figures 3.19(a) et 3.19(b) concernent le cas d'un système fortement chargé. Nous remarquons que Green-RTO affiche le plus grand pourcentage de temps durant lequel la batterie est pleine que ce soit pour $s_i = 2$ ou pour $s_i = 6$.

Pour $s_i = 2$ (cf. la Figure 3.19(a)), nous remarquons que :
- L'écart maximal affiché entre la courbe de Green-RTO et les courbes de Green-BWP et EDeg est de 36% pour $R_e = 0.2$: la batterie est pleine 40% du temps avec Green-RTO, 5% du temps avec Green-BWP et 4% du temps avec EDeg.
- Pour $R_e = 1$, la batterie est pleine 30% du temps avec Green-RTO, 2% du temps avec Green-BWP et 1% du temps avec EDeg.

Pour $s_i = 6$ (cf. la Figure 3.19(b)), nous remarquons que :
- Plus le paramètre de pertes augmente, plus il y a de jobs rouges à exécuter et donc avec Green-RTO le pourcen-



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.19 – Pourcentage de temps durant lequel la batterie est pleine ($U_p = 1.2$(a), $U_p = 1.1$(b)) et ACET=WCET

tage de temps durant lequel la batterie est pleine diminue et s'approche des valeurs obtenues avec Green-BWP et EDeg.
- La différence maximale affichée entre la courbe de Green-RTO et les courbes de Green-BWP et EDeg est de 6 pour $R_e = 0.2$: la batterie est pleine 9% du temps avec Green-RTO, 2% du temps avec Green-BWP et 1% du temps avec EDeg.
- Avec Green-BWP ou EDeg, le pourcentage de temps durant lequel la batterie est pleine dépend uniquement de la charge processeur et de la criticité énergétique et est indépendant du paramètre de pertes.
- Pour $R_e = 1$, la batterie est pleine 5% du temps avec Green-RTO, 1% du temps avec Green-BWP et 0% du temps avec EDeg.

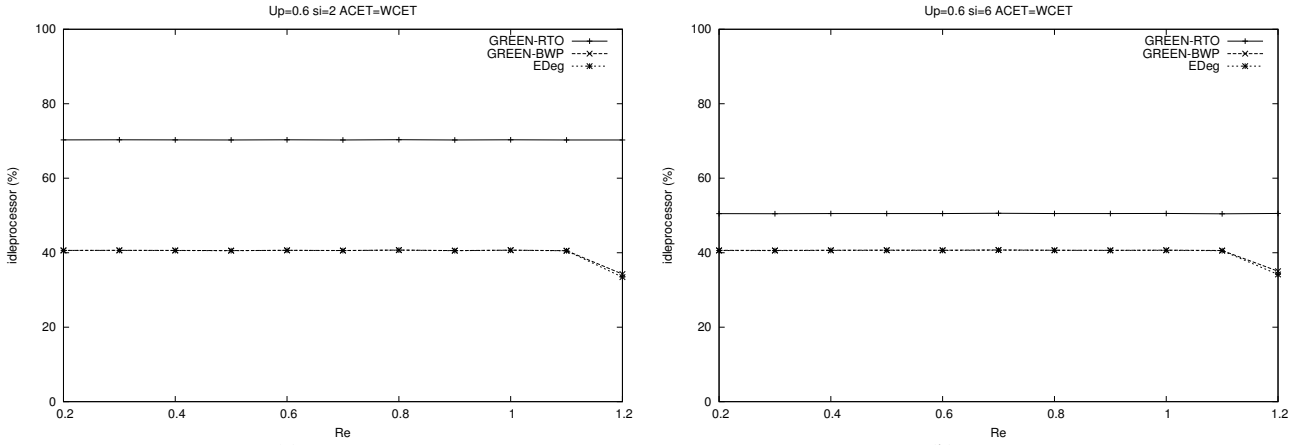
5.3.2.4 Synthèse sur l'étude du pourcentage de temps durant lequel la batterie est pleine

Nous concluons que :
Dans un système **non surchargé ou surchargé temporellement** :
- Le nombre d'instants durant lequel la batterie est pleine avec Green-RTO, Green-BWP et EDeg est décroissant en fonction de la criticité énergétique parce que l'énergie demandée par les tâches augmente. En effet, R_e représente la puissance moyenne consommée par les tâches par rapport à la puissance reçue. Par suite si la puissance moyenne consommée par les tâches augmente, l'énergie dans la batterie sera davantage utilisée.
- La batterie est pleine plus souvent avec Green-RTO qu'avec Green-BWP et EDeg car Green-RTO n'exécute que les jobs rouges.
- Quand le paramètre de pertes augmente, le nombre d'instants durant lequel la batterie est pleine diminue avec Green-RTO et s'approche de ceux obtenus avec Green-BWP et EDeg. En effet, quand les pertes sont moins tolérées, l'énergie disponible est sollicitée par un nombre de jobs rouges plus important.
- Comme Green-BWP et EDeg tentent d'exécuter le même nombre de jobs, le nombre d'instants durant lequel la batterie est pleine est similaire avec EDeg et Green-BWP mais ce ne sont pas forcément les mêmes jobs qui sont exécutés.
- Quand la charge processeur augmente, le nombre d'instants durant lequel la batterie est pleine diminue avec tous les ordonnanceurs.
- Un avantage de Green-RTO réside dans le fait que comme il n'exécute que les jobs rouges, il peut se contenter d'une batterie de capacité plus petite que celle requise par EDeg pour offrir cependant un niveau de QoS minimale.
- Green-BWP s'adapte à l'énergie disponible dans le système. Donc en fonction de l'énergie disponible dans le système il va décider de l'exécution des jobs bleus ou pas de façon à assurer l'exécution des jobs rouges dans le respect de leurs échéances. Ainsi, quand la capacité de la batterie est grande, la QoS se verra améliorée. Par contre si la batterie est petite le nombre de jobs bleus exécutés sera moindre et donc la QoS s'approchera de

celle obtenue avec Green-RTO.
- EDeg requiert une batterie de capacité plus grande que celle requise avec Green-RTO ou Green-BWP, pour assurer le bon fonctionnement du système.

5.3.3 Etude du taux d'oisiveté

5.3.3.1 Système moyennement chargé temporellement. La Figure 3.20(a) et la Figure 3.20(b) affichent le taux d'oisiveté obtenu dans le cas d'un système moyennement chargé temporellement.



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.20 – Taux de temps oisif ($U_p = 0.6$ et ACET=WCET)

Nous remarquons que :

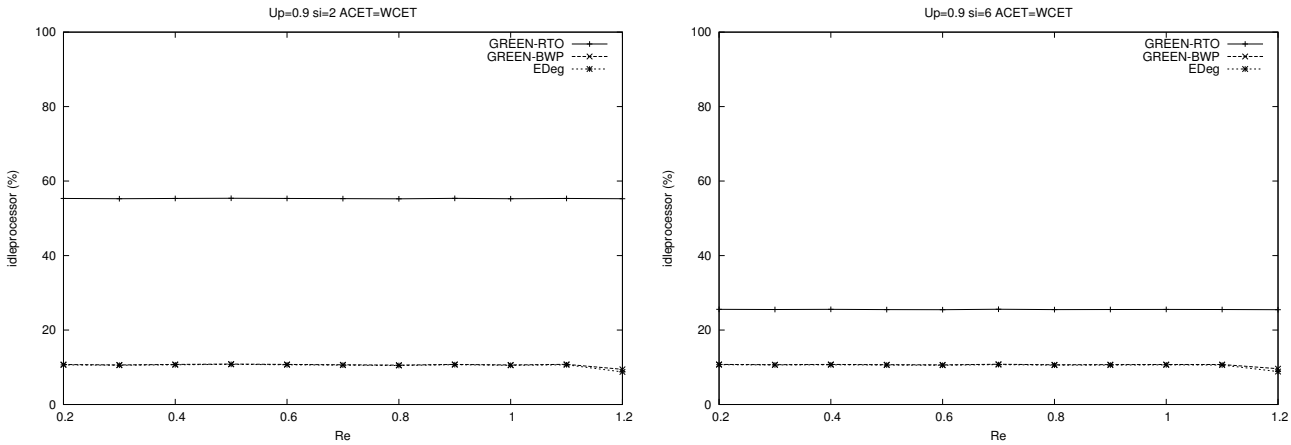
- Green-RTO affiche un taux d'oisiveté constant toujours supérieur à Green-BWP et EDeg.
- Pour $s_i = 2$ (cf. Figure 3.20(a)), quand $R_e = 0.2$, le processeur est inactif pendant 70% du temps avec Green-RTO, 40% du temps avec Green-BWP et EDeg.
- Pour $s_i = 6$ (cf. Figure 3.20(b)), quand $R_e = 0.2$, le processeur est inactif pendant 50% du temps avec Green-RTO. L'écart observé entre le taux d'oisiveté obtenus avec Green-RTO et ceux obtenus avec Green-BWP et EDeg est de 10%.

5.3.3.2 Système fortement chargé temporellement. La Figure 3.21(a) et la Figure 3.21(b) représentent le taux d'oisiveté obtenu dans le cas d'un système fortement chargé temporellement ($U_p = 0.9$).

Nous remarquons que :

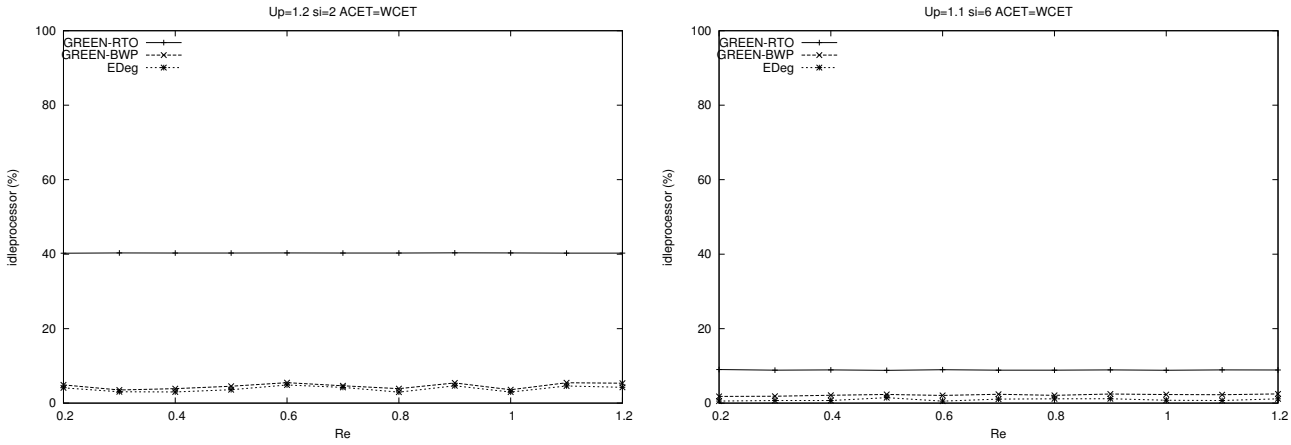
- Green-BWP et EDeg affichent des taux d'oisiveté similaires.
- Green-RTO affiche un taux d'oisiveté constant toujours supérieur à Green-BWP et EDeg.
- Pour $s_i = 2$ (cf. Figure 3.21(a)), l'écart observé entre le taux d'oisiveté avec Green-RTO et les taux obtenus avec Green-BWP et EDeg est égal à 45%.
- Pour $s_i = 6$ (cf. Figure 3.21(b)), l'écart observé entre le taux d'oisiveté avec Green-RTO et les taux obtenus avec Green-BWP et EDeg est égal à 16%.

5.3.3.3 Système surchargé temporellement. Les Figures 3.22(a) et 3.22(b) illustrent le taux d'oisiveté dan le cas d'un système surchargé temporellement ($U_p = 1.2$). Nous remarquons que :



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.21 – Taux de temps oisif ($U_p = 0.9$ et ACET=WCET)



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.22 – Taux de temps oisif ($U_p = 1.2$ (a), $U_p = 1.1$ (b)) et ACET=WCET

- Pour $s_i = 2$ (cf. Figure 3.22(a)), et $R_e = 0.2$, le processeur est inactif 40% du temps avec Green-RTO, 5% du temps avec Green-BWP et 4% du temps avec EDeg.
- Pour $s_i = 6$ (cf. Figure 3.22(b)), et $R_e = 0.2$, le processeur est inactif 9% du temps avec Green-RTO. L'écart observé entre le taux d'oisiveté avec Green-RTO et EDeg est environ égal à 8%.
- Green-BWP présente un taux d'oisiveté légèrement supérieur à EDeg. Ceci est dû au fait que les jobs bleus ne sont exécutés que s'ils ne compromettent pas l'exécution des jobs rouges et donc dans le cas contraire Green-BWP ne tente pas de les exécuter. Ceci dit, EDeg ne contrôle pas l'exécution des jobs périodiques jugeant qu'ils sont tous importants. Par conséquent, le processeur est plus sollicité.

5.3.3.4 Synthèse sur l'étude du taux d'oisiveté

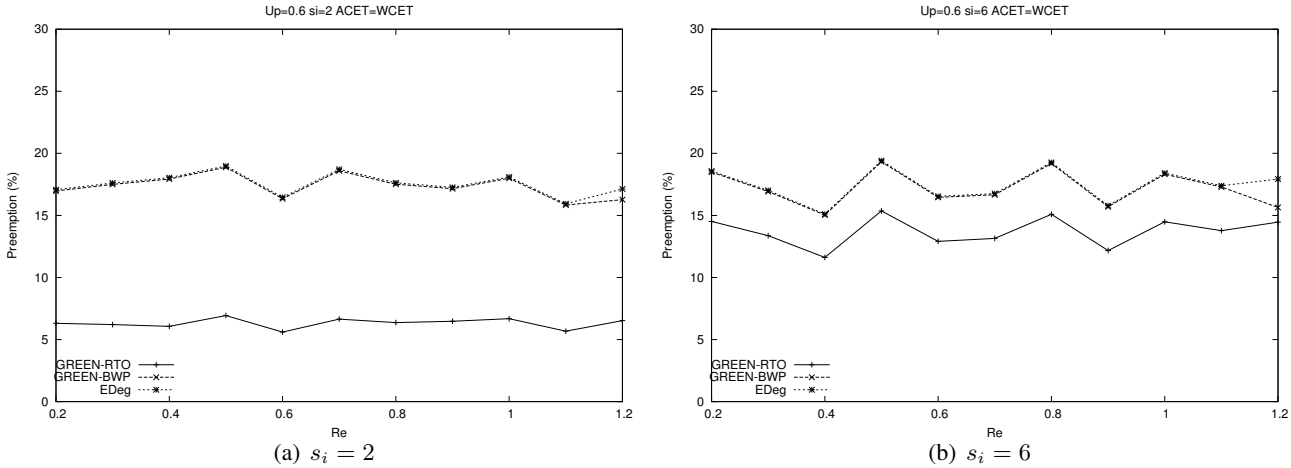
Nous concluons que :
Dans un système **non surchargé ou surchargé temporellement** :

- Les taux d'oisiveté obtenus avec Green-RTO, Green-BWP et EDeg sont d'autant plus petits que la charge processeur augmente. Ils dépendent du paramètre U_p fixé et sont indépendants de la criticité énergétique du système.
- Green-BWP et EDeg affichent des taux d'oisiveté très proches qui tendent vers zéro quand le système est en surcharge temporelle.

- Green-RTO affiche le plus grand taux d'oisiveté par rapport aux autres algorithmes et ce, quelle que soit la charge processeur. En effet, Green-RTO n'exécute que les jobs rouges. Par conséquent, le nombre de jobs qu'il exécute est inférieur au nombre de jobs exécutés avec Green-BWP ou EDeg et donc le processeur est moins sollicité.
- Quand le paramètre de pertes s_i augmente, le nombre de jobs rouges à exécuter augmente. Par conséquent, le taux d'oisiveté obtenu avec Green-RTO diminue et s'approche des taux d'oisiveté observés avec Green-BWP et EDeg.
- L'avantage de Green-RTO est qu'il peut délivrer une QoS minimale acceptable avec un processeur ayant une capacité de traitement plus petite que celle requise par EDeg.
- Green-BWP s'adapte à la capacité de traitement du processeur de sorte à offrir la meilleure QoS possible suivant la charge processeur. Ainsi plus la capacité de traitement du processeur est importante, plus la QoS obtenue est meilleure. Ceci dit, pour une faible capacité de traitement, Green-BWP se comporte comme Green-RTO car ne pouvant plus exécuter les jobs bleus, il se contente de n'exécuter que les jobs rouges.
- Lorsque la puissance de traitement disponible est insuffisante, il faut soit réduire la charge du processeur, soit augmenter la capacité de traitement pour garantir le bon fonctionnement du système. La réduction de la charge CPU est une opération ne nécessitant aucun coût supplémentaire. L'astuce consiste à identifier les aspects de la charge du système qui peuvent être réduits. À cet égard, Green-RTO et Green-BWP présentent le grand avantage de réduire la charge processeur en ayant le contrôle des jobs abandonnés et en permettant ainsi de garantir une QoS acceptable. Par contre, avec EDeg la seule solution existante dans le cas de surcharge de traitement est d'augmenter la capacité de traitement du processeur.

5.3.4 Etude du taux de préemption

5.3.4.1 Système moyennement chargé temporellement. Les Figures 3.23(a) et 3.23(b) montrent le taux de préemption dans le cas d'un système moyennement chargé temporellement ($U_p = 0.6$).
Nous remarquons que :

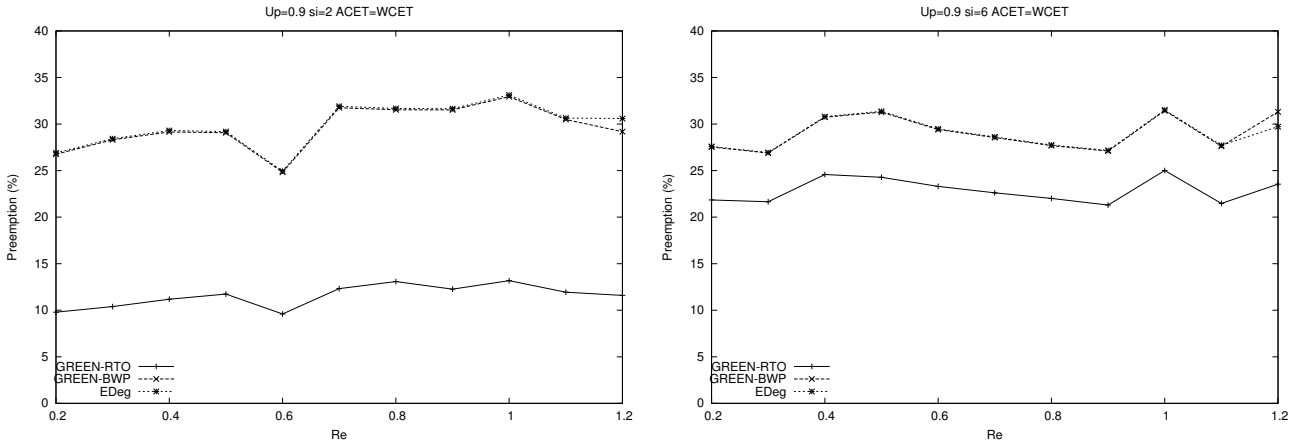


(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.23 – Taux de préemption ($U_p = 0.6$ et ACET=WCET)

- Pour $s_i = 2$, les taux de préemption obtenus avec Green-BWP et EDeg sont similaires et varient entre 15% et 18%. Les taux de préemption obtenus avec Green-RTO varient entre 5% et 6%. Par conséquent, Green-RTO présente trois fois moins de préemptions que Green-BWP et EDeg.
- Pour $s_i = 6$, les taux de préemption obtenus avec Green-BWP et EDeg sont similaires et varient entre 15% et 19%. Les taux de préemption obtenus avec Green-RTO varient entre 11% et 15%. Green-RTO engendre un quart de préemptions en moins que Green-BWP et EDeg.

5.3.4.2 Système fortement chargé temporellement. Les Figures 3.24(a) et 3.24(b) montrent le taux de préemption dans un système fortement chargé temporellement ($U_p = 0.9$).
Nous remarquons :

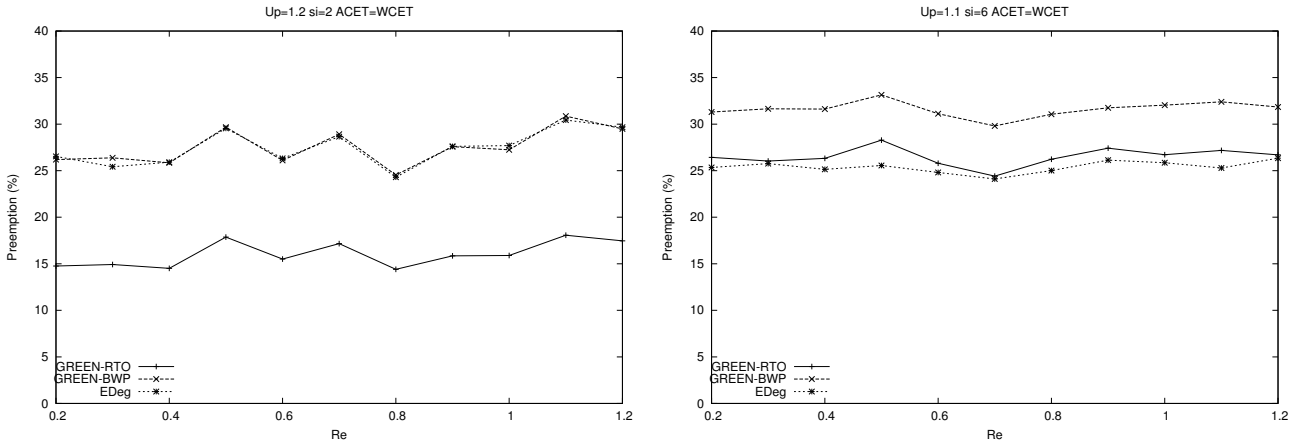


(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.24 – Taux de préemption ($U_p = 0.9$ et ACET=WCET)

- Pour $s_i = 2$ (cf. Figure 3.24(a)), les taux de préemption obtenus avec Green-BWP et EDeg sont très proches et varient entre 25% et 33%. Les taux de préemption obtenus avec Green-RTO varient entre 11% et 15%. Green-RTO affiche un taux de préemption qui représente $\frac{2}{5}$ de ceux obtenus avec Green-BWP et EDeg.
- Pour $s_i = 6$ (cf. Figure 3.24(b)), les taux de préemption obtenus avec Green-BWP et EDeg sont très similaires et varient entre 15% et 19%. Green-RTO affiche un taux de préemption qui représente $\frac{4}{5}$ de ceux obtenus avec Green-BWP et EDeg.
- Le fait que Green-RTO engendre le moins de préemptions parmi les ordonnanceurs étudiés permet de déduire que les changements de contexte sont moindres. Ceci signifie que Green-RTO a un faible overhead en plus de s'implémenter plus facilement que Green-BWP et EDeg.

5.3.4.3 Système surchargé temporellement. Les Figures 3.25(a) et 3.25(b) présentent le taux de préemption dans un système surchargé temporellement ($U_p = 1.2$).



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.25 – Taux de préemption ($U_p = 1.2$ (a), $U_p = 1.1$ (b)) et ACET=WCET

- Pour $s_i = 2$ et $U_p = 120\%$ (cf. Figure 3.25(a)), les taux de préemption obtenus avec Green-BWP et EDeg sont très proches et varient entre 24% et 29%. Les taux de préemption obtenus avec Green-RTO varient entre 14% et 17%. Green-RTO affiche des taux de préemption deux fois moindres que ceux obtenus avec Green-BWP et EDeg.

- Pour $s_i = 6$ et $U_p = 1.1$(cf. Figure 3.25(b)), les taux de préemption obtenus avec Green-BWP et EDeg sont très proches et varient entre 24% et 29%. Green-RTO affiche des taux de préemption qui varient entre 14% et 17% et qui représente $\frac{3}{5}$ de ceux obtenus avec Green-BWP et EDeg.

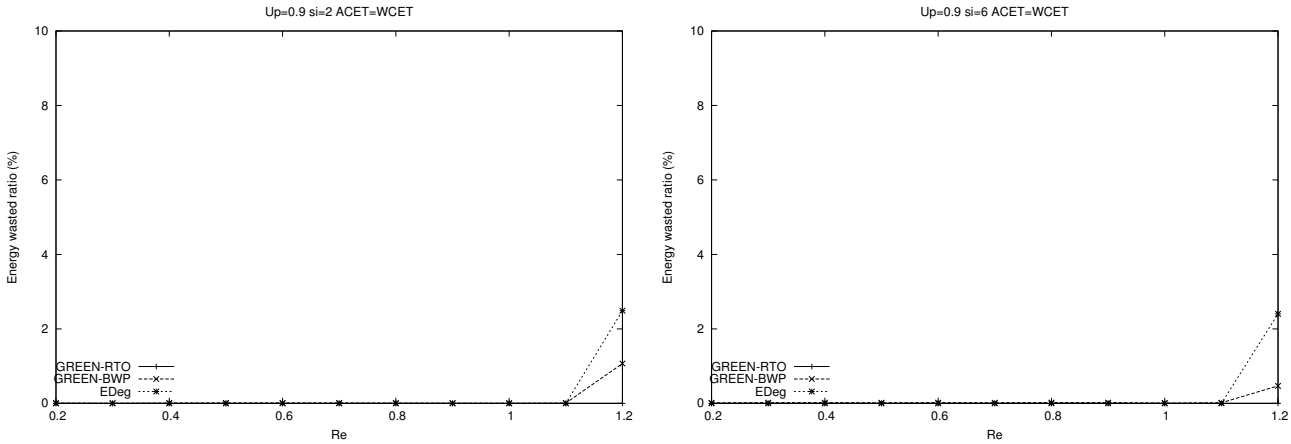
5.3.4.4 Synthèse sur l'étude du taux de préemption

Nous concluons que :
- Green-BWP et EDeg induisent des taux de préemption similaires et toujours supérieurs à ceux obtenus avec Green-RTO. Ceci est dû au fait que EDeg et Green-BWP exécutent plus de jobs que Green-RTO. Ceci étant, quand le paramètre de pertes augmente, le taux de préemption avec Green-RTO augmente et s'approche des taux de préemption obtenus avec Green-BWP et EDeg.
- Plus le taux de préemption est petit, moins il y a de changements de contexte engendrés. Green-RTO délivre une QoS minimale donc provoque moins de changements de contexte que Green-BWP et EDeg.

5.3.5 Etude du ratio d'énergie gaspillée

Dans cette partie nous considérons un système fortement chargé temporellement et un système surchargé temporellement et nous observons le ratio d'énergie gaspillée avec Green-RTO, Green-BWP et EDeg en fonction de la criticité énergétique.

5.3.5.1 Système fortement chargé temporellement. Les Figures 3.26(a) et 3.26(b) présentent le ratio d'énergie gaspillée dans un système fortement chargé temporellement ($U_p = 0.9$).
On remarque que :



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.26 – Ratio d'énergie gaspillée ($U_p = 0.9$ et ACET=WCET)

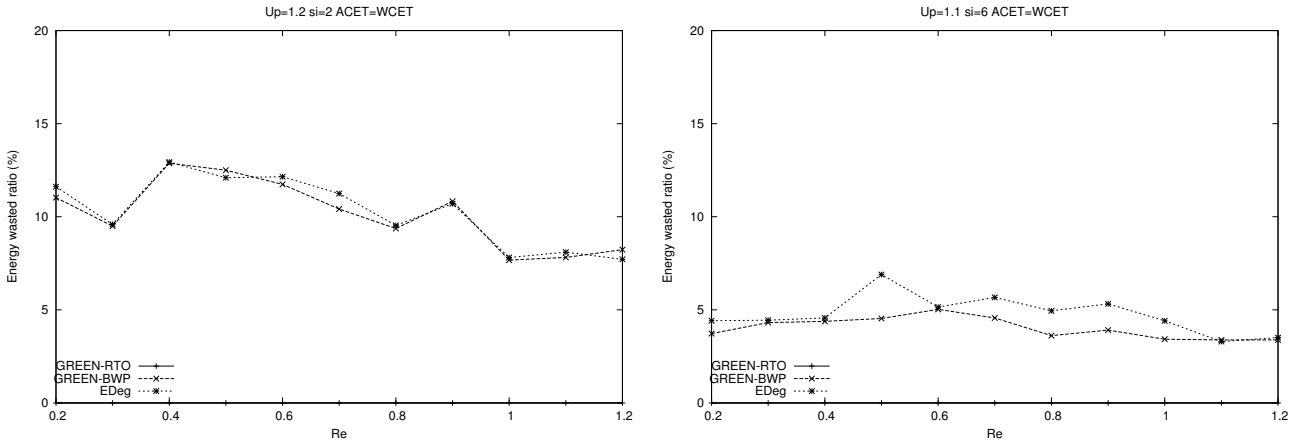
- Green-RTO n'engendre pas de gaspillage d'énergie quelle que soit la criticité énergétique car il exécute uniquement les jobs rouges.

- Quand le système n'est pas surchargé énergétiquement, il n'y a pas de gaspillage d'énergie observé avec EDeg. On peut cependant observer un léger gaspillage d'énergie causé par Green-BWP qui est dû à l'abandon de jobs bleus dû à l'exécution de jobs rouges toujours plus prioritaires.

- Quand le système est surchargé énergétiquement, les ratios de gaspillage d'énergie respectivement affichés avec Green-BWP et EDeg sont de 0.9% et 2% pour $R_e = 1.2$ et $s_i = 2$ (cf. Figure 3.26(a)).

- Quand le système est surchargé énergétiquement, les ratios de gaspillage d'énergie affichés avec Green-BWP et EDeg sont respectivement de 0.7% et 2% pour $R_e = 1.2$ et $s_i = 6$ (cf. Figure 3.26(b)).

5.3.5.2 Système surchargé temporellement. La Figure 3.27(a) et la Figure 3.27(b) présentent le ratio d'énergie gaspillée dans un système surchargé temporellement.



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.27 – Ratio d'énergie gaspillée ($U_p = 1.2$ (a), $U_p = 1.1$ (b)) et ACET=WCET

Nous remarquons que :

- Green-RTO n'engendre pas de gaspillage d'énergie quelle que soit la criticité énergétique car il exécute uniquement les jobs rouges.
- Pour $s_i = 2$ (cf. Figure 3.27(a)), quand le système n'est pas surchargé énergétiquement, des gaspillages d'énergies similaires sont observés avec EDeg et Green-BWP qui augmentent progressivement quand la criticité énergétique augmente et qui atteignent un pic de 9% pour $R_e = 0.7$. Ensuite, nous observons que le taux de gaspillage diminue progressivement pour atteindre 5% quand $R_e = 1.2$. Ceci s'explique par le fait qu'il y a moins de jobs qui sont exécutés par manque de temps et d'énergie.
- Pour $s_i = 6$ (cf. Figure 3.27(b)), quand le système n'est pas surchargé énergétiquement, des gaspillages d'énergie similaires sont observés avec EDeg et Green-BWP qui augmentent progressivement quand la criticité énergétique augmente et qui atteignent un pic de 13% pour $R_e = 0.5$. Ensuite, nous observons que le taux de gaspillage diminue progressivement pour atteindre 8% quand $R_e = 1.2$. Ceci s'explique par le fait qu'il y a moins de jobs qui sont exécutés par manque de temps et d'énergie.
- Quand le paramètre de pertes augmente, le taux de gaspillage est plus élevé.

5.3.5.3 Synthèse sur l'étude du ratio d'énergie gaspillée

Nous concluons que :

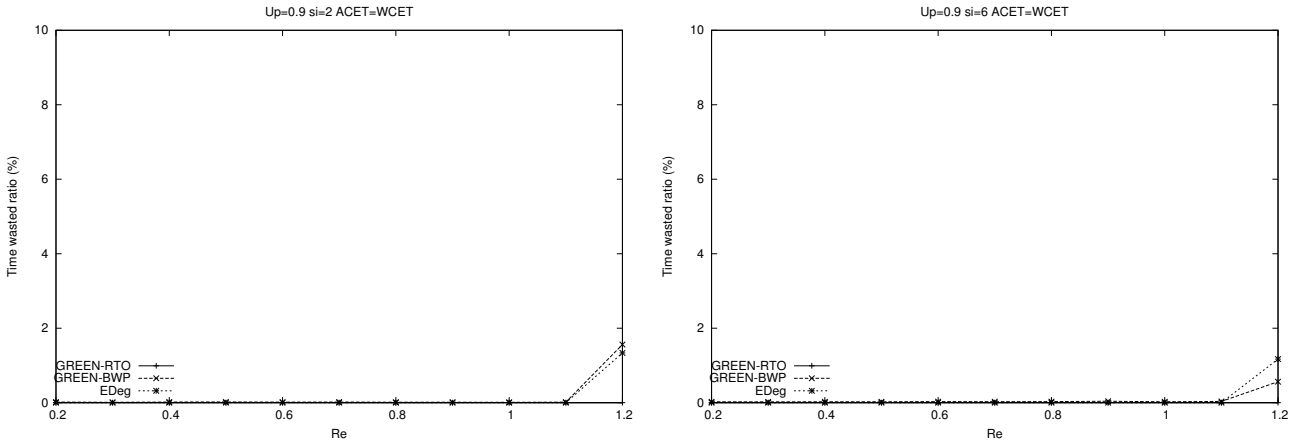
- Green-RTO n'engendre pas de gaspillage d'énergie quel que soit le profil énergétique et temporel étudié.
- Dans un système non surchargé temporellement, Green-BWP et EDeg affichent des gaspillages d'énergie à partir du moment où le système est en surcharge énergétique. Ceci est dû à des violations d'échéances de jobs partiellement exécutés et qui sont en manque d'énergie.
- Dans un système surchargé temporellement, Green-BWP et EDeg induisent des gaspillages d'énergie quel que soit le profil énergétique. Ce gaspillage d'énergie est dû à des violations d'échéances de jobs partiellement exécutés et qui sont en manque de temps et il a tendance à augmenter quand le paramètre de pertes augmente.

- Le ratio d'énergie gaspillé le plus important est observé avec Green-BWP et EDeg quand le système est en surcharge temporelle et moyennement chargé énergétiquement.

5.3.6 Etude du ratio de temps gaspillé

Dans cette partie, nous étudions le ratio de temps gaspillé dans l'exécution de jobs non complètement exécutés avant leur échéance.

5.3.6.1 Système fortement chargé temporellement. Les Figures 3.28(a) et 3.28(b) présentent le ratio de temps gaspillé dans un système fortement chargé temporellement ($U_p = 0.9$).



(a) $s_i = 2$ (b) $s_i = 6$

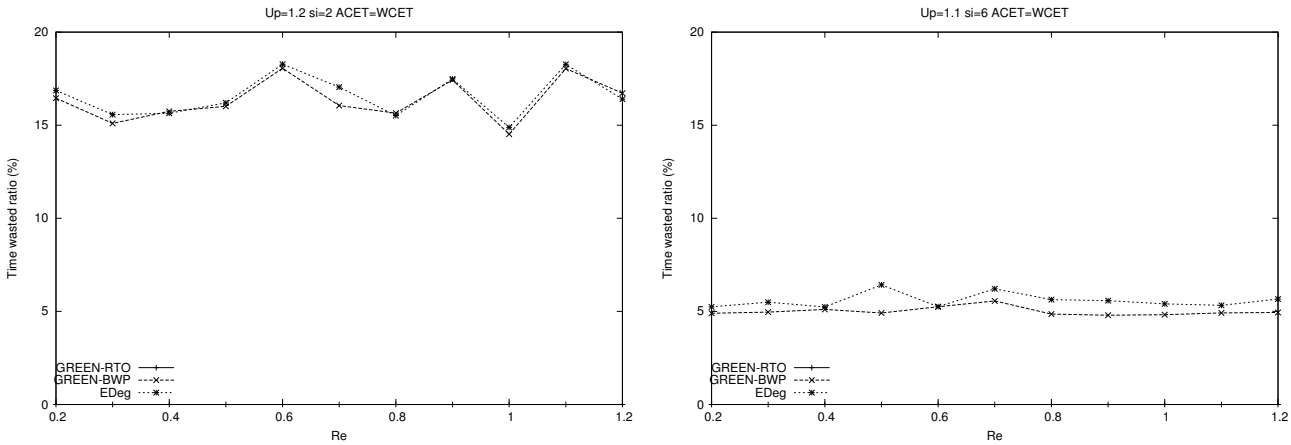
FIGURE 3.28 – Ratio de temps gaspillé ($U_p = 0.9$ et ACET=WCET)

Nous remarquons que :

- Green-RTO n'engendre pas de gaspillage de temps quelle que soit la criticité énergétique car il exécute tous les jobs rouges dans le respect de leur échéance.

- Quand le système n'est pas surchargé énergétiquement :
  - Pour $s_i = 2$ (cf. Figure 3.28(a)), des gaspillages de temps négligeables de l'ordre de 1% sont observés avec Green-BWP.
  - Pour $s_i = 6$ (cf. Figure 3.28(b)), des gaspillages de temps négligeables de l'ordre de 2% sont observés avec Green-BWP.
  - EDeg n'affiche pas de gaspillage de temps.
  - Quand le système est surchargé énergétiquement ($R_e = 1.2$), les ratios de gaspillage de temps affichés avec Green-BWP et EDeg sont respectivement de 50% et 80% pour $s_i = 2$ et de 15% et 50% pour $s_i = 6$.
  - EDeg affiche un gaspillage d'énergie trois fois supérieur à Green-BWP quand le paramètre de pertes vaut 6, ceci est dû au fait qu'avec EDeg il n'y a pas de contrôle sur les jobs abandonnés. Sur l'hyperpériode équivalente H^*, dans le cas de surcharge, EDeg subit beaucoup plus de violations d'échéances que Green-BWP car il n'y a pas de contrôle sur les violations d'échéances de jobs qui subissent l'effet domino. Le job ayant manqué d'énergie est abandonné et par conséquent, l'énergie utilisée a été gaspillée. Cette énergie gaspillée aurait pu être utilisée pour l'exécution d'un futur job plus important. Ceci met en avant la performance de Green-BWP qui traite au moins tous les jobs rouges en garantissant le respect de leur échéance. La QoS obtenue est d'autant meilleure quand des jobs bleus sont complètement exécutés. Les jobs bleus qui manque leur échéance ont certes gaspillée de l'énergie toutefois ce gaspillage reste inférieur à celui observé avec EDeg.

5.3.6.2 Système surchargé temporellement. Les Figures 3.29(a) et 3.29(b) permettent d'évaluer le ratio de temps gaspillé dans le cas d'un système surchargé temporellement.



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.29 – Ratio de temps gaspillé ($U_p = 1.2$ (a), $U_p = 1.1$ (b)) et ACET=WCET

Nous remarquons que :

- Green-RTO n'engendre pas de gaspillage de temps quelle que soit la criticité énergétique car il exécute tous les jobs rouges dans le respect de leur échéance.
- EDeg et Green-BWP présentent des ratios de gaspillage de temps très proches.
- Pour $s_i = 2$ et $U_p = 120\%$, le ratio de gaspillage avec EDeg et Green-BWP augmente suivant la criticité énergétique et atteint un pic qui vaut 8% quand $R_e = 0.7$ puis il diminue. Ceci est dû au fait qu'il y a moins de jobs qui débutent leur exécution car le système devient de plus en plus chargé énergétiquement.
- Pour $s_i = 6$ et $U_p = 110\%$, le ratio de gaspillage avec EDeg et Green-BWP augmente suivant la criticité énergétique et atteint un pic qui vaut 9% quand $R_e = 0.6$.

5.3.6.3 Synthèse sur l'étude du ratio de temps gaspillé

Nous concluons que :
- Green-RTO n'engendre pas de gaspillage d'énergie quel que soit le profil énergétique et temporel étudié.
- Dans un système non surchargé temporellement :
si le système est **non surchargé énergétiquement** ($R_e \leq 1$) :

- le ratio de temps gaspillé est négligeable avec Green-BWP.
- le ratio de temps gaspillé est nul avec EDeg.

si le système est surchargé énergétiquement :

- Green-BWP présente des gaspillages de temps dû à des violations d'échéance de jobs bleus partiellement exécutés.
- EDeg présente des gaspillages de temps dû à des violations d'échéance de jobs partiellement exécutés.
- Quand le paramètre de pertes augmente, le gaspillage de temps augmente.

- Dans un système **surchargé temporellement** :

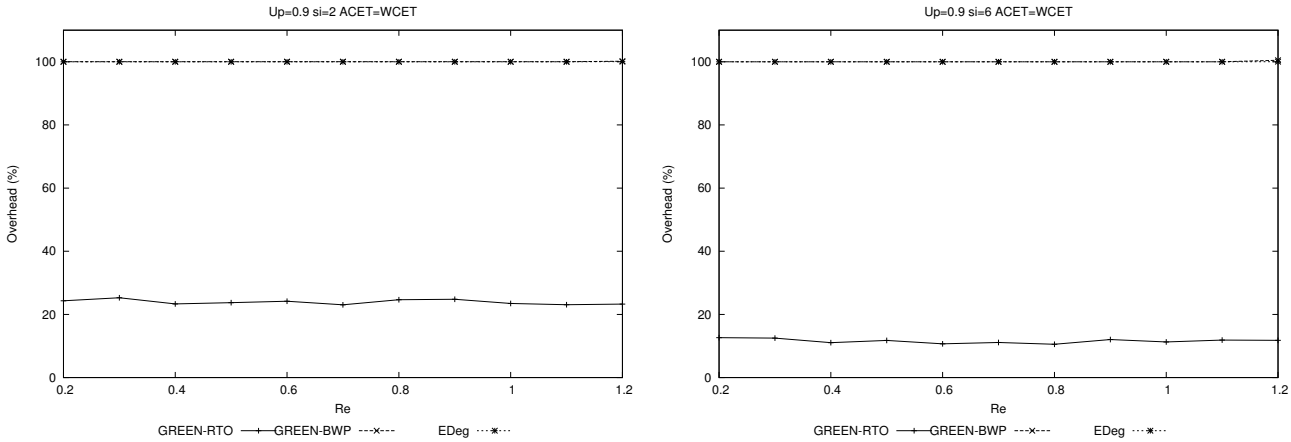
- Green-BWP présente des gaspillages d'énergie dû à des violations d'échéance de jobs bleus partiellement exécutés.
- EDeg présente des gaspillages d'énergie dû à des violations d'échéance de jobs partiellement exécutés.

- Quand le paramètre de pertes augmente, le gaspillage d'énergie a tendance à augmenter avec EDeg et Green-BWP.

- Le taux de gaspillage d'énergie avec EDeg et Green-BWP est plus important quand le ratio de criticité énergétique augmente jusqu'à atteindre un ratio de gaspillage de temps maximum quand le système est moyennement chargé énergétiquement. Ensuite ce ratio de gaspillage d'énergie diminue suivant que le système devient fortement chargé énergétiquement car il y a moins de jobs qui peuvent démarrer leur exécution par manque d'énergie.

5.3.7 Etude de l'overhead

L'overhead calculé ici correspond au nombre de fois où le calcul de la laxité énergétique a été sollicité par rapport au nombre total de jobs exécutés.



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.30 – Taux d'Overhead ($U_p = 0.9$ et ACET=WCET)

5.3.7.1 Système fortement chargé temporellement. Nous remarquons sur la Figure 3.30(a) et la Figure 3.30(b) que :

- Green-BWP et EDeg affichent un overhead maximal de 100% quelle que soit la criticité énergétique.

- L'overhead obtenu avec Green-RTO vaut environ 23% quelle que soit la criticité énergétique et est environ 4 fois moindre qu'avec Green-BWP et EDeg pour $s_i = 2$ (cf. Figure 3.30(a)).

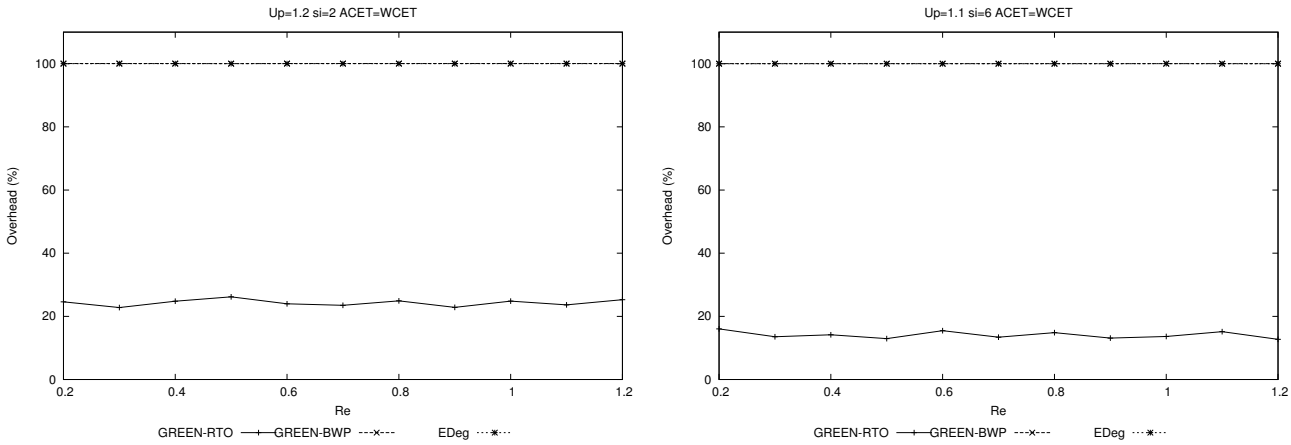
- L'overhead obtenu avec Green-RTO vaut environ 13% quelle que soit la criticité énergétique et est 7 fois plus petit qu'avec Green-BWP et EDeg pour $s_i = 6$ (cf. Figure 3.30(b)).

5.3.7.2 Système surchargé temporellement. Nous remarquons à partir de la Figure 3.31(a) et de la Figure 3.31(b) que :

- Green-BWP et EDeg affichent un overhead qui vaut 100% quelle que soit la criticité énergétique.

- L'overhead obtenu avec Green-RTO vaut environ 15% quelle que soit la criticité énergétique et est 6 fois moindre qu'avec Green-BWP et EDeg pour $s_i = 2$ (cf. Figure 3.31(a)).

- L'overhead obtenu avec Green-RTO vaut environ 25% quelle que soit la criticité énergétique et est 4 fois moindre qu'avec Green-BWP et EDeg pour $s_i = 6$ (cf. Figure 3.31(b)).



(a) $s_i = 2$ (b) $s_i = 6$

FIGURE 3.31 – Taux d'Overhead ($U_p = 1.2$ (a), $U_p = 1.1$ (b)) et ACET=WCET

5.3.7.3 Synthèse sur l'étude de l'overhead

Nous concluons que :
Dans un système **non surchargé ou surchargé temporellement** :
- L'overhead causé par le calcul de la laxité énergétique est plus important avec EDeg et Green-BWP qu'avec Green-RTO.
- Avec EDeg et Green-BWP, l'overhead vaut 100% car le calcul de la laxité énergétique est sollicité à chaque début d'exécution d'un job.
- Avec Green-RTO, l'overhead est beaucoup moins important qu'avec Green-BWP et EDeg, car premièrement le nombre de jobs à exécuter est inférieur au nombre de jobs exécutés avec Green-BWP ou EDeg. Deuxièmement le calcul calcul de la laxité énergétique se fait sur des intervalles de temps et non à chaque début d'exécution d'un job.

5.4 Variation de la capacité de la batterie

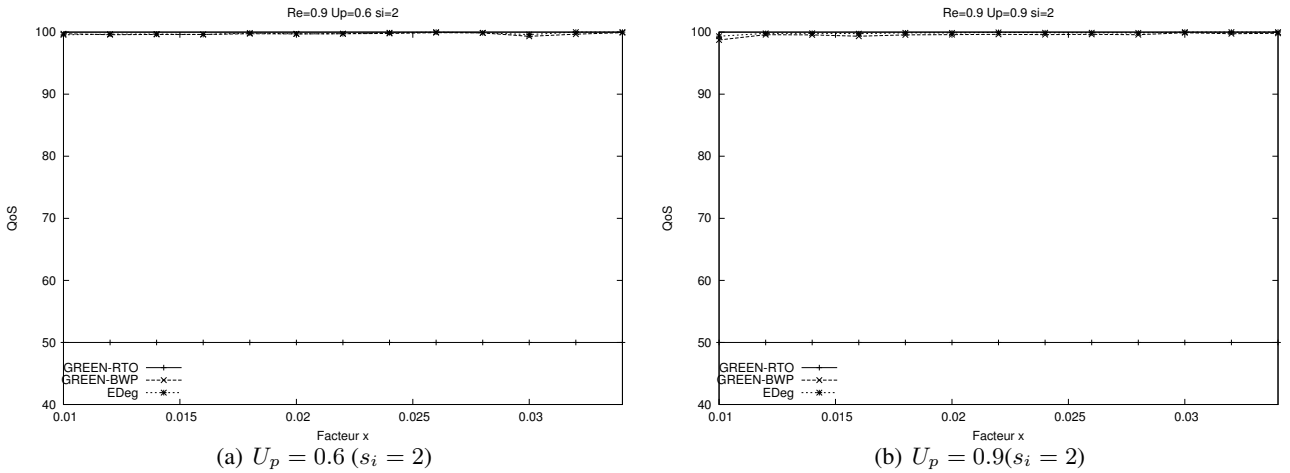
Dans cette partie, nous considérons deux types de systèmes chacun caractérisé par une charge processeur U_p et un ratio de criticité énergétique R_e. Nous supposons que la capacité de la batterie vaut $C = H.P_r.x$ sachant que x est un nombre réel qui varie dans l'intervalle $[0, 1]$ et nous évaluons le taux de respect obtenu avec Green-RTO, Green-BWP et EDeg en fonction de x. Les simulations sont effectuées sur 6 hyperpériodes $H^* = PPCM(T_1s_1, ...,T_ns_n)$. A chaque valeur de x, 100 configurations de tâches différentes sont générées et chaque configuration est constituée de 10 tâches périodiques temps réel fermes ayant un paramètre de pertes égal à 2. Dans un premier temps, nous traitons ensuite le cas d'un système fortement chargé puis nous traitons le cas d'un système en surcharge énergétique.

5.4.1 Système fortement chargé énergétiquement

Dans cette partie, nous considérons un système fortement chargé énergétiquement en fixant $R_e = 0.9$.

Sur la Figure (cf. 3.32(a)) et la Figure (cf. 3.32(b)), le système est fortement chargé énergétiquement : - Avec Green-RTO, le taux de respect est constant et vaut 50% pour toutes les valeurs de x.
- Les taux de respect obtenus avec EDeg et Green-BWP sont croissants et augmentent en fonction de la capacité de la batterie.
- Pour $x = 0.004$, EDeg et Green-BWP affichent respectivement des taux de respect de 99% et 98%.
- EDeg affiche un taux de respect égal à 100% à partir de $x = 0.48$.

On conclut que :
- Les taux de respect obtenus avec Green-BWP sont très proches de ceux obtenus avec EDeg et ils augmentent progressivement avec la taille de la batterie.
- Green-RTO permet de garantir la plus petite QoS pour une taille de batterie 12 fois plus petite que celle requise

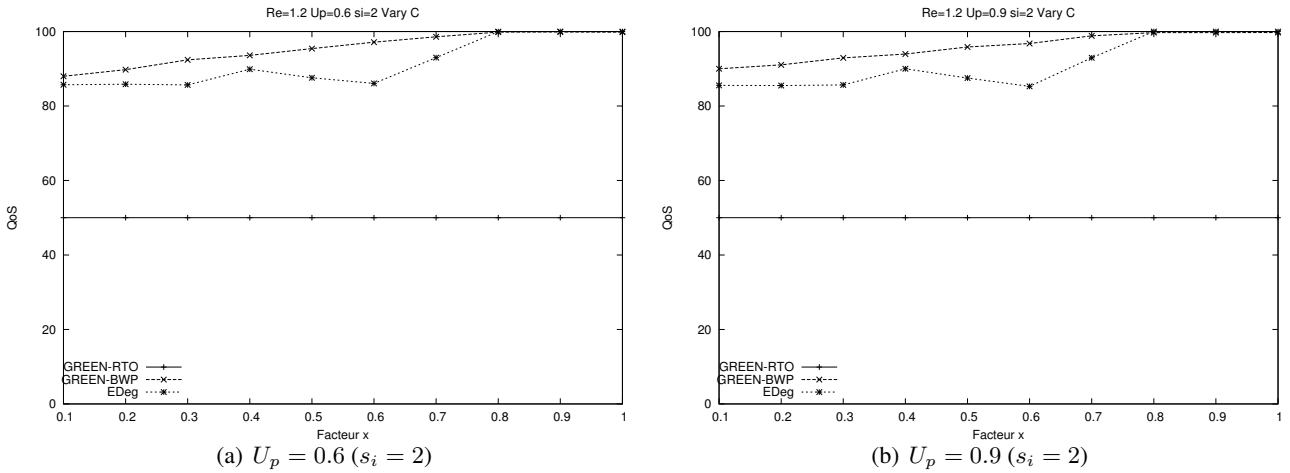


(a) $U_p = 0.6$ ($s_i = 2$) (b) $U_p = 0.9$($s_i = 2$)

FIGURE 3.32 – Taux de respect ($R_e = 0.9$ et ACET=WCET)

par EDeg pour garantir le bon fonctionnement du système.
- L'avantage de Green-BWP réside dans le fait que même si la QoS n'est pas de 100%, cela n'a pas d'effet sur le fonctionnement du système car les jobs abandonnés sont les jobs bleus qui ne sont pas importants. Par conséquent, plus la capacité de la batterie est grande, plus Green-BWP offre une meilleure QoS.
- Green-BWP peut se contenter d'une batterie de faible capacité et offrir une QoS acceptable ; dans le pire-cas, il n'exécute aucun job bleu et offre une QoS égale à celle fournie par Green-RTO.
- Avec EDeg quand x est faible (la capacité est faible), le taux de respect est inférieur à 100%. Ceci implique que des jobs périodiques importants n'ont pas été exécutés dans le respect de leur échéance et ceci peut engendrer de graves conséquences vis-à-vis du fonctionnement du système.
- Pour une batterie 12 fois plus petite que celle requise par EDeg, Green-BWP offre une QoS de 98%.
- Green-RTO et Green-BWP sont plus performants que EDeg en termes de taille de batterie requise car la QoS reste acceptable avec une batterie de capacité inférieure à celle nécessaire avec EDeg pour garantir le bon fonctionnement du système.

5.4.2 Système en surcharge énergétique



(a) $U_p = 0.6$ ($s_i = 2$) (b) $U_p = 0.9$ ($s_i = 2$)

FIGURE 3.33 – Taux de respect ($R_e = 1.2$ et ACET=WCET)

Sur la Figure 3.33(a) et la Figure 3.33(b), nous remarquons que dans le cas d'un système fortement chargé énergétiquement, les courbes ont les mêmes allures et ne sont pas influencées par le fait que le système soit moyennement ou fortement chargé temporellement :

- Le taux de respect obtenu avec Green-BWP est constant et vaut 50% pour toutes les valeurs de x.
- Les taux de respect obtenus avec EDeg et Green-BWP sont croissants et augmentent au fur et à mesure que la capacité de la batterie augmente.
- EDeg et Green-BWP affichent respectivement un taux de respect égal à 85% et 88% pour $x = 0.1$.
- EDeg et Green-BWP affichent un taux de respect de 100% à partir de $x = 0.9$.

Nous concluons que :
- Green-RTO garantit une QoS minimale pour une batterie de capacité 9 fois inférieure à celle requise par EDeg pour délivrer une QoS de 100%.
- Les taux de respect obtenus avec Green-BWP sont très proches de ceux obtenus avec EDeg et ils augmentent progressivement avec la taille de la batterie.
- L'avantage de Green-BWP réside dans le fait que les jobs abandonnés sont les jobs bleus qui ne sont pas importants.
- Avec EDeg quand x est faible (la capacité est alors faible), le taux de respect est inférieur à 100%. Ceci implique que des jobs périodiques importants n'ont pas été exécutés dans le respect de leur échéance.
- Même si Green-BWP affiche des taux de respect inférieurs à ceux obtenus avec EDeg, il a l'avantage d'offrir une QoS fiable et acceptable dans le sens où l'on sait que les jobs abandonnés sont les jobs bleus qui ne sont pas importants. De plus, quand le système est surchargé énergétiquement EDeg ne fonctionne pas correctement tant que la batterie a une capacité inférieure à $0.9H.P_r$. Par conséquent, avec Green-RTO et Green-BWP, nous pouvons utiliser des batteries beaucoup plus petites qu'avec EDeg tout en offrant une QoS acceptable et garantissant le bon fonctionnement du système.

5.4.2.1 Synthèse sur la variation de la capacité de la batterie

Lors de la conception d'un système, il faut prévoir une batterie de taille suffisante de manière à garantir, avant la mise en opération du système, que celui-ci ne sera pas à court d'énergie. Un manque d'énergie peut entraîner des violations d'échéance entraînant la chute du système dans certains cas.

EDeg ordonnance des tâches périodiques temps réel strictes sous contraintes énergétiques et temporelles. Ainsi toutes les tâches doivent être exécutées et aucune violation d'échéance de tâches n'est tolérée. Contrairement, Green-RTO et Green-BWP ordonnancent des tâches périodiques temps réel fermes en tolérant un seuil de pertes qui est défini par le paramètre de pertes propre à chaque tâche. Avec Green-RTO, la QoS du système est constante (et minimale) alors qu'avec Green-BWP la QoS dépend du nombre de jobs bleus exécutés en plus des jobs rouges. En utilisant Green-RTO ou Green-BWP, nous pouvons être sûrs que le résultat est toujours acceptable même si la QoS est dégradée. De ce point de vue, avec Green-RTO ou Green-BWP, nous pouvons réduire la taille de la batterie et continuer à avoir un résultat acceptable contrairement à EDeg qui a besoin d'une taille prédéfinie à l'avance de sorte à ne pas manquer d'énergie en cas de surcharge.

5.4.3 Synthèse

Le but de ces simulations a été d'évaluer les performances des deux algorithmes proposés dans ce chapitre à savoir Green-RTO, Green-BWP par rapport à l'algorithme EDeg. Nous résumons l'étude faite au travers des tableaux 3.1 et 3.2. Dans ces tableaux, un "*" représente les bonnes performances d'un algorithme d'ordonnancement par rapport à un critère donné. Les critères sont la QoS délivrée, la complexité algorithmique, le surcoût de calcul, la taille de batterie (elle se traduit par la taille de batterie requise pour garantir le bon fonctionnement du système), la vitesse de traitement du processeur (elle se traduit par le niveau de puissance de traitement requis pour garantir le bon fonctionnement du système). Dans le tableau 3.1, nous considérons un système sous-chargé énergétiquement et temporellement.

Dans le Tableau 3.2, nous considérons un système surchargé énergétiquement et/ou temporellement.

Critères	Green-RTO	Green-BWP	EDeg
QoS	**	***	***
Complexité algorithmique	***	**	***
Surcoût de calcul	***	*	*
Taille de la batterie	***	***	*
Vitesse de traitement du processeur	***	***	*

TABLE 3.1 – Performances en sous-charge énergétique et sous charge temporelle.

Critères	Green-RTO	Green-BWP	EDeg
QoS	**	***	*
Complexité algorithmique	***	**	***
Surcoût de calcul	***	*	*
Taille de la batterie	***	***	*
Vitesse de traitement du processeur	***	***	*

TABLE 3.2 – Performances en surcharge énergétique et/ou temporelle.

6 Conclusion

Dans ce chapitre, nous avons dans un premier temps rappelé le principe de l'algorithme EDeg qui permet l'ordonnancement d'une configuration de tâches périodiques sans pertes soumise à des contraintes énergétiques et temporelles. Puis, nous avons décrit en détails les algorithmes RTO et BWP du modèle Skip-Over. Ensuite, nous avons présenté et décrit le modèle du système que nous considérons ainsi que les deux algorithmes Green-RTO et Green-BWP que nous avons proposés dans le cadre de cette thèse. Enfin, ce chapitre s'est terminé par une étude comparative des algorithmes Green-RTO, Green-BWP et EDeg pour évaluer leur performance suivant que le système est moyennement chargé, fortement chargé ou surchargé temporellement.

En termes de QoS, les simulations montrent que Green-RTO offre la QoS minimale acceptable et ce, avec un faible overhead par rapport à Green-BWP et EDeg. De plus, Green-RTO peut se contenter d'une batterie de faible capacité et d'un processeur à faible puissance de traitement tout en garantissant cette QoS minimale.

Dans le cas d'un système en sous-charge énergétique et temporelle, EDeg offre une QoS de 100%. Par ailleurs, Green-BWP offre aussi une QoS qui tend vers 100%. Cependant, l'avantage de Green-RTO et Green-BWP par rapport à EDeg réside dans la taille de la batterie qui peut être 9 à 10 fois plus petite que celle requise par EDeg pour garantir le bon fonctionnement du système.

Dans le cas d'un système en surcharge temporelle et/ou énergétique, EDeg n'est pas fiable dans le sens où un nombre non contrôlé de jobs peuvent être rejetés par manque de temps de traitement ou d'énergie et ceci pourra avoir de graves conséquences sur le système. Par conséquent, pour éviter la chute du système dans le cas de surcharge énergétique ou temporelle, nous avons proposé Green-RTO et Green-BWP qui traitent le problème de surcharge en respectant toujours un seuil de QoS. Green-RTO garantit toujours une QoS minimale quel que soit le profil temporel ou énergétique du système. En outre, Green-BWP permet d'améliorer la QoS obtenue avec Green-RTO essayant de maximiser le nombre de jobs exécutés en fonction du profil énergétique et temporel. Ainsi, il garantit l'exécution des jobs rouges sans violation de leur échéance et exécute les jobs bleus quand cela est possible. Par conséquent, Green-BWP a le principal avantage de s'adapter aux contraintes temporelles et énergétiques et permet d'offrir la meilleure QoS possible.

Même si Green-BWP s'avère être l'ordonnanceur offrant la meilleure QoS par rapport au profil du système, Green-RTO a l'avantage d'avoir un overhead en termes de calcul de la laxité énergétique, 4 fois plus faible qu'avec Green-BWP et EDeg et évite le gaspillage d'énergie et de temps. Ce qui n'est pas le cas de Green-BWP et EDeg qui présentent des overheads supérieurs à Green-RTO et qui engendrent tous les deux des gaspillages d'énergie et de temps dûs à des violations d'échéance en cas de surcharge.

Pour conclure, pour assurer l'autonomie du système, il est préférable d'utiliser soit Green-RTO soit Green-BWP. Pour faire le choix parmi l'un d'eux, il faut trouver le bon compromis en fonction du type d'application, du niveau de QoS voulu et des contraintes du système (coût processeur, coût batterie, etc.).

Dans le chapitre suivant, nous nous intéressons à l'algorithme Green-BWP et nous proposons des stratégies d'ordonnancement des jobs bleus dans le but d'améliorer la stabilité du système.

Chapitre 4

Stabilité et robustesse des systèmes autonomes en énergie

Sommaire

Ce chapitre a pour objectif d'étudier la "robustesse" et la "stabilité" de l'algorithme Green-BWP. Dans un premier temps, nous introduisons les définitions relatives aux notions de robustesse et de stabilité. Puis nous soulignons le défaut de stabilité inhérent à Green-BWP. Nous présentons alors deux variantes appelées MS (Minimum success) et LF (Last Failure). Elles permettent d'améliorer la stabilité du système par de nouvelles politiques d'ordonnancement des jobs bleus. Enfin, par le biais de simulations, nous évaluons de manière comparative les performances en termes de stabilité et de robustesse des ordonnanceurs Green-BWP, Green-BWP-MS et Green-BWP-LF.

1 Définitions et terminologie

La performance d'un ordonnanceur sous contraintes de QoS implique d'analyser sa robustesse et le comportement de chaque tâche, caractérisée par son taux de respect individuel. Dans la suite, nous décrivons les notions de stabilité et de robustesse d'un algorithme.

1.1 Notion de stabilité

Soit une configuration de n tâches périodiques. On parle de stabilité quand l'exécution des jobs de chaque tâche se fait avec équité : il doit y avoir un équilibre des taux de réussite individuels des tâches périodiques de la

configuration. Dans nos travaux, nous tiendrons compte de la définition du critère de stabilité défini initialement comme suit :

Définition 1.1 *[SC96] Un algorithme d'ordonnancement X est plus stable qu'un algorithme d'ordonnancement Y si la plus grande différence entre les taux de respect individuels des tâches avec X est inférieure à la plus grande différence entre les taux de respect individuels des tâches avec Y.*

La stabilité d'un système peut alors être définie comme suit :

Définition 1.2 *[Mar06] Un système temps réel est dit stable si au bout d'un temps fini, la proportion de jobs réussis par rapport au nombre total de jobs engendrés pour une tâche est la même pour toutes les tâches, dans une fourchette ϵ prédéfinie.*

Cependant le fait de considérer chaque tâche individuellement peut engendrer la dégradation du taux de respect global des tâches, tout en conservant toujours des taux de respect individuels équilibrés. Par conséquent, la stabilité doit si possible être accompagnée de robustesse.

1.2 Notion de robustesse

Nous parlons de robustesse lorsque l'on s'intéresse à la performance globale du système en termes de QoS. En effet l'étude de la robustesse consiste à évaluer le taux de succès global (i.e. le nombre de jobs exécutés dans le respect de leur échéance sur le nombre total de jobs) de la configuration de tâches. Nous reprenons la définition suivante de la robustesse d'un système temps réel :

Définition 1.3 *[SC96] Un algorithme d'ordonnancement X est plus robuste qu'un algorithme d'ordonnancement Y si le taux de respect global des tâches avec X est supérieur au taux de respect global des tâches avec Y.*

La robustesse d'un système surchargé représente sa capacité à ordonnancer le plus grand nombre de jobs.

2 Analyse Green-RTO et Green-BWP en termes de stabilité et de robustesse

Les algorithmes Green-RTO et Green-BWP permettent d'ordonnancer des tâches périodiques fermes sous contraintes temporelles et énergétiques. Comme expliqué dans le chapitre précédent, chaque tâche se compose de jobs rouges et bleus. Green-RTO n'exécute que les jobs rouges selon EDF. Par contre, Green-BWP exécute les jobs rouges et les jobs bleus sachant que les jobs rouges sont les plus prioritaires et doivent impérativement être exécutés avant leur échéance alors que les jobs bleus peuvent être abandonnés pour plusieurs raisons : soit il y a eu un manque d'énergie, soit un manque de temps.

2.1 Green-RTO

Avec Green-RTO, la performance globale du système reste médiocre en termes de taux de respect. Elle dépend directement du paramètre de pertes des tâches (plus s_i est petit, plus le taux de respect global des tâches est faible). De plus, avec Green-RTO, les taux de respect individuels sont égaux uniquement si toutes les tâches possèdent le même paramètre de pertes s_i ; dans le cas contraire, les taux de respect individuels sont très différents.

Exemple illustratif :
La Figure 4.1 illustre l'ordonnancement avec Green-RTO d'une configuration de 3 tâches temps réel fermes profondément rouges telle que $\mathcal{T}^*$=$\{\tau_i(C_i,\ D_i,\ T_i,\ s_i,\ E_i)\}$ = $\{\tau_1(5,\ 10,\ 10,\ 2,\ 16)$, $\tau_2(4,\ 15,\ 15,\ 2,\ 14)$ et $\tau_3(2,\ 6,\ 6,\ 2,\ 7)\}$.
Nous supposons que la puissance reçue sur une hyperpériode H^* est constante avec $P_r = 3$ et que la batterie

est initialement chargée avec *E(0)*=$E_{max} = \frac{H*P_r}{20} = 9$.

$U_p = \sum_{i=1}^{n} \frac{C_i}{T_i} = 1.1$ et $R_e = \frac{P_e}{P_r} = 1.23$. Comme $U_p > 1$ et $R_e > 1$, le système est en surcharge énergétique et temporelle.
$U_p^* = 0,733 < 1$.
$U_e^* = 0.698 < 1$. Comme $U_p^* < 1$ et $U_e^* < 1$, les jobs rouges sont ordonnançables considérant leurs contraintes temporelles et énergétiques.

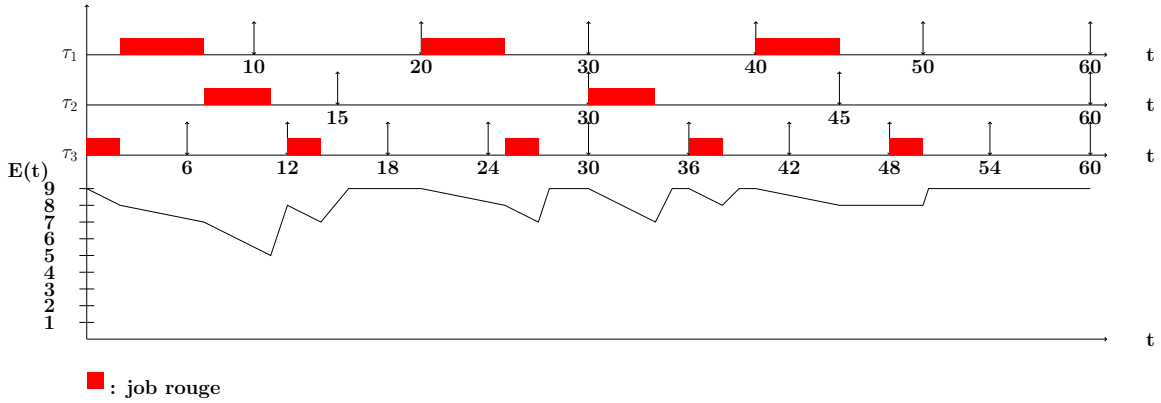


FIGURE 4.1 – Taux de respect global, $R_e = 0.3$

Nous remarquons que sur la Figure 4.1, tous les jobs ont le même paramètre de pertes qui vaut 2. Chaque tâche présente un taux de respect individuel égal à 50%. Par conséquent, dans le cas où le paramètre de pertes fixé est identique pour toutes les tâches du système, Green-RTO est un algorithme très stable.

Sur la Figure 4.2, nous considérons la même configuration de tâches que celle de la Figure 4.1 sauf que la tâche τ_1 a un paramètre de pertes qui vaut 3 telle que $\mathcal{T}^*$=$\{\tau_i(C_i,\ D_i,\ T_i,\ s_i,\ E_i)\}$ = $\{\tau_1(5,\ 10,\ 10,\ 3,\ 16)$, $\tau_2(4,\ 15,\ 15,\ 2,\ 14)$ et $\tau_3(2,\ 6,\ 6,\ 2,\ 7)\}$.
$U_p^* = 0,90 < 1$.
$U_e^* = 0.86 < 1$.
Comme $U_p^* < 1$ et $U_e^* < 1$, les jobs rouges sont ordonnançables considérant leurs contraintes temporelles et énergétiques.

Nous remarquons que les taux de respect individuels des tâches valent respectivement 66.66%, 50% et 50%. Avec Green-RTO, le taux de respect d'une tâche dépend directement de son paramètre de pertes. Par conséquent, Green-RTO n'est plus un algorithme stable dans le cas où l'application possède des tâches ayant des paramètres de pertes différents.

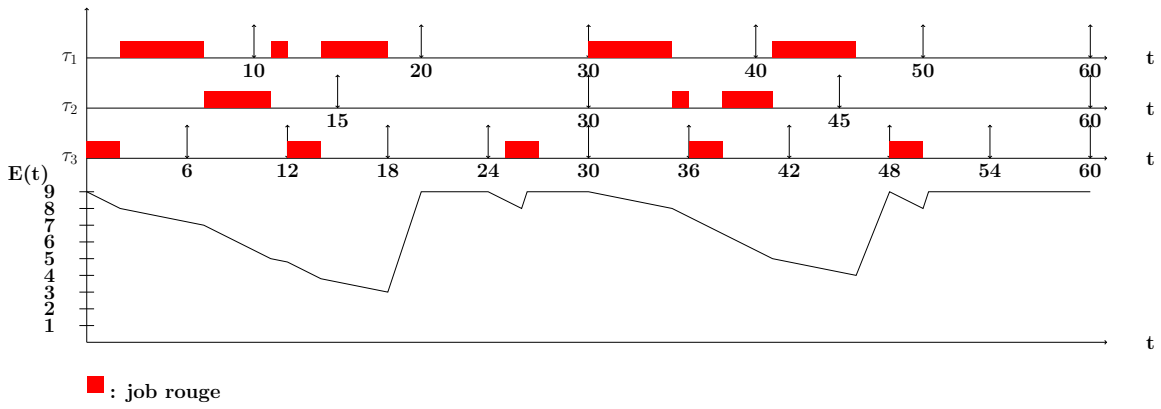


FIGURE 4.2 – Taux de respect global, $R_e = 0.3$

2.2 Green-BWP

Avec Green-BWP, les jobs bleus sont conventionnellement exécutés selon EDF quand cela s'avère possible. Par suite, le taux de respect individuel (i.e. ratio de jobs totalement exécutés sur le nombre total de jobs qui devrait être exécutés pour chaque tâche) diffère d'une tâche à une autre en fonction du nombre de jobs bleus exécutés. Dans ce chapitre nous nous inspirons de précédents travaux effectués pour des systèmes soumis à des contraintes temporelles [Ely91, SC96, Mar06].

Selon Green-BWP, les jobs bleus sont exécutés suivant la règle définie par EDF (Earliest Deadline First). Par conséquent à un instant *t*, s'il n'y a aucun job rouge prêt, le job bleu ayant la plus proche échéance sera élu pour exécution. Il ne sera évidemment exécuté que si la laxité énergétique du système est positive et si le niveau d'énergie dans la batterie est suffisant.

Exemple illustratif Considérons une configuration de 3 tâches temps réel fermes profondément rouges $\mathcal{T}^*$=$\{\tau_i(C_i,\ D_i,\ T_i,\ s_i,\ E_i)\}$ = $\{\tau_1(5,\ 10,\ 10,\ 2,\ 16), \tau_2(4,\ 15,\ 15,\ 2,\ 14)$ et $\tau_3(2,\ 6,\ 6,\ 2,\ 7)\}$.
Nous supposons que la puissance reçue sur une hyperpériode H^* est constante avec $P_r = 3$ et que la batterie est initialement chargée avec *E(0)*=$E_{max} = \frac{H*P_r}{20} = 9$.

$U_p = \sum_{i=1}^{n} \frac{C_i}{T_i} = 1.1$ et $R_e = \frac{P_e}{P_r} = 1.23$. Comme $U_p > 1$ et $R_e > 1$, le système est en surcharge énergétique et temporelle.
$U_p^* = 0,733 < 1$. $U_e^* = 0.698$. Comme $U_p^* < 1$ et $U_e^* < 1$, les jobs rouges sont ordonnançables considérant leurs contraintes temporelles et énergétiques.

La Figure 4.3 illustre l'ordonnancement de la configuration $\mathcal{T}^*$ selon Green-BWP. Nous pouvons remarquer

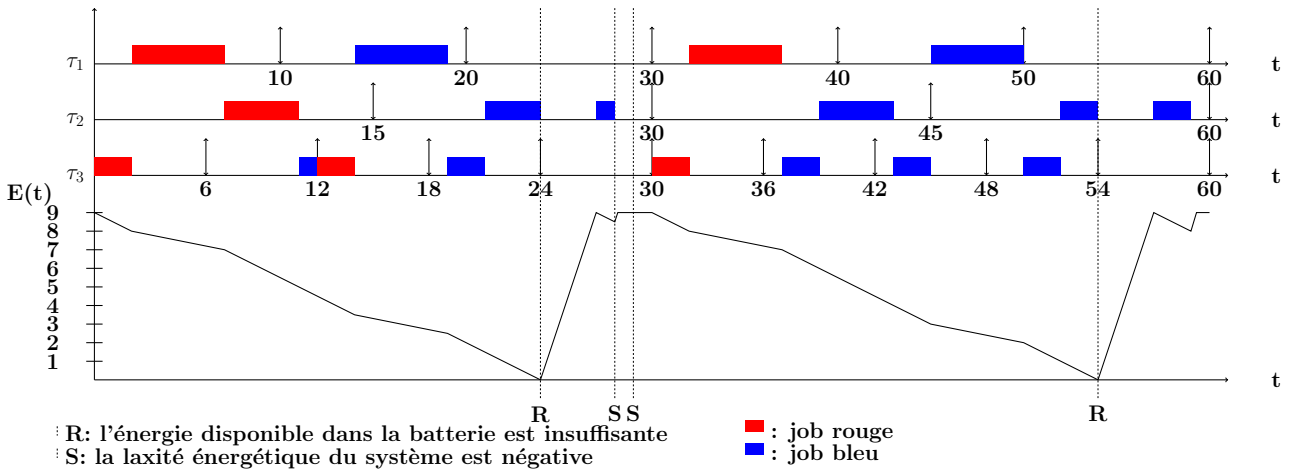


FIGURE 4.3 – Ordonnancement selon Green-BWP

que pour les tâches τ_1, τ_2 et τ_3, les taux de respect individuels sont respectivement égaux à 67%, 100% et 70%. Le taux de respect global quant à lui vaut 75%. Les taux de respect individuels affichent un écart maximal de 33%, ce qui nous permet de conclure que Green-BWP n'est pas stable : les tâches ne sont pas servies de manière équitable lorsque les jobs bleus sont ordonnancés selon l'algorithme *EDF*.

Dans la suite, nous proposons deux autres stratégies d'ordonnancement nommées Green-BWP-LF et Green-BWP-MS. Puis, nous analysons leurs performances en termes de robustesse et stabilité.

3 Variantes de Green-BWP orientées stabilité

3.1 Green-BWP-MS(Minimum Success) :

Cette stratégie diffère de Green-BWP (sous-entendu Green-BWP-EDF) par l'ordonnancement des jobs bleus. Selon Green-BWP-MS, le job bleu prêt avec le plus faible taux de respect observé depuis l'instant

initial obtiendra la plus haute priorité.

Exemple illustratif La Figure 4.4 illustre l'ordonnancement de la configuration $\mathcal{T}^*$ suivant Green-BWP-MS. Nous pouvons remarquer que pour les tâches τ_1, τ_2 et τ_3 les taux de respect individuels sont respectivement

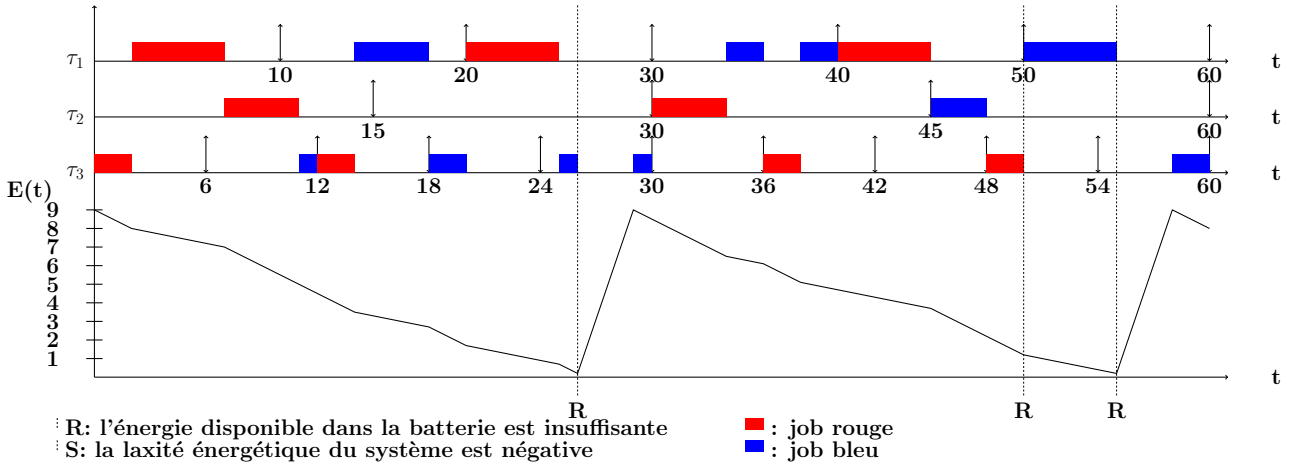


FIGURE 4.4 – Ordonnancement selon Green-BWP-MS

égaux à 66.66%, 50% et 70%. Le taux de respect global quant à lui vaut 75%. Les taux de respect individuels affichent un écart maximal de 20%, ce qui nous permet de conclure que Green-BWP-MS est plus stable que Green-BWP. Ceci est dû au fait que la tâche qui possède le taux de respect le plus faible à un instant t est toujours la plus prioritaire. Par conséquent, les tâches sont servies de manière plus équitable avec Green-BWP-MS qu'avec Green-BWP.

3.2 Green-BWP-LF(Last Failure) :

L'algorithme appelé Last Failure first *(LF)* consiste à exécuter à tout instant *t*, le job bleu prêt dont le nombre de succès successifs mesuré depuis le dernier échec, est le plus petit. L'échéance la plus proche est utilisée pour résoudre les conflits entre deux jobs bleus de priorités identiques.

Ainsi selon *Green-BWP-LF*, s'il n'y a aucun job rouge à l'état prêt, le job bleu prêt d'une tâche ayant le plus petit nombre de succès successifs mesuré depuis le dernier échec est le job le plus prioritaire parmi les jobs bleus prêts. Il sera par conséquent exécuté si la laxité énergétique du système est positive et si le niveau d'énergie dans la batterie est suffisant.

Exemple illustratif La Figure 4.5 illustre l'ordonnancement selon *Green-BWP-LF* de la configuration $\mathcal{T}^*$ de l'exemple précedent. Dans ce cas, nous observons un meilleur équilibre des taux de respect individuels avec *Green-BWP-LF* par rapport à ceux obtenus avec *Green-BWP*. Les taux de respect individuels obtenus pour les tâches τ_1, τ_2 et τ_3 sont respectivement 83.33%, 75% et 70%. La différence maximale observée entre les taux de respect individuels est 13.33%.

Par ailleurs, nous constatons que les taux de respect globaux obtenus avec Green-BWP, Green-BWP-MS et Green-BWP-LF sont respectivement 75% 65% et 70%. Ceci nous permet de dire que sur la configuration de tâches considérée dans l'exemple, Green-BWP est le plus robuste parmi tous les ordonnanceurs étudiés.

Dans la suite, nous étudions la stabilité et robustesse de différents profils de systèmes au travers de plus amples simulations.

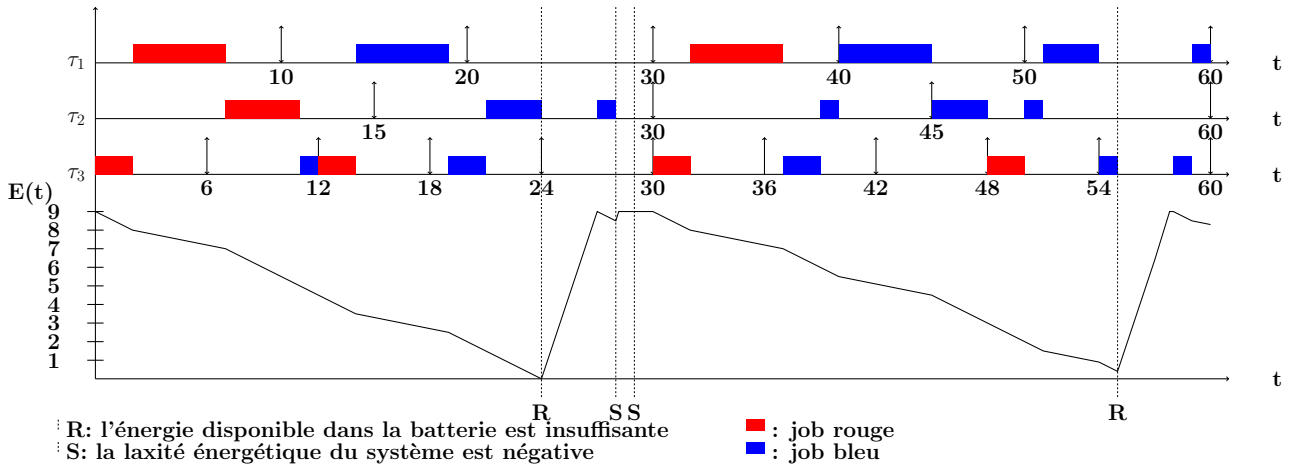


FIGURE 4.5 – Ordonnancement selon Green-BWP-LF

4 Résultats de simulation

Nous avons fait des simulations pour analyser la stabilité et la robustesse de Green-BWP, Green-BWP-LF et Green-BWP-MS. Le générateur de configurations de tâches périodiques prend en paramètres :
- n le nombre de tâches dans la configuration, le ppcm des périodes des tâches,
- U_p la charge du processeur, $\bar{P_e}$ la puissance moyenne énergétique des tâches,
- s_i le paramètre de perte des tâches.

Les tâches sont préemptables, indépendantes et à échéances sur requêtes et leurs périodes sont harmoniques. Les durées d'exécution des tâches C_i sont générées aléatoirement et dépendent directement de U_p de sorte que $U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$. De même, les demandes énergétiques des tâches E_i sont générées aléatoirement et dépendent de $\bar{P_e}$ de sorte que $\bar{P_e} = \sum_{i=1}^{n} \frac{E_i}{T_i}$.

Dans la suite, nous considérons trois types de systèmes caractérisés par un ratio de criticité énergétique et par une tolérance aux pertes telle que $s_i = 2$. La batterie est initialement remplie telle que $E(0) = \frac{H^*.P_r}{10}$ et le temps de simulation est réglé à 10 hyperpériodes. Nous allons faire varier la charge du processeur U_p et pour chaque valeur, 100 configurations de 10 tâches périodiques seront générées. Nous allons analyser les taux de respect individuels des tâches de sorte à étudier la stabilité et la robustesse des ordonnanceurs selon différents profils temporel et énergétique.

4.1 Système faiblement chargé énergétiquement

Dans cette partie, nous considérons un système faiblement chargé énergétiquement tel que $R_e = 0.3$.

4.1.1 Analyse de la stabilité

Les courbes des Figures 4.6, 4.7 et 4.8 affichent les taux de respect individuels des tâches obtenus respectivement avec les ordonnanceurs Green-BWP, Green-BWP-LF et Green-BWP-MS en fonction de la valeur de U_p.

On constate que :

- Quand le système n'est pas surchargé temporellement, Green-BWP-LF est le moins stable. On observe par exemple que la distance entre les courbes des taux de respect individuels des tâches τ_1 et τ_{10} est de 12.93% quand $U_p = 1$. Pour $U_p \leq 1$, Green-BWP et Green-BWP-MS sont stables car la distance maximale entre les taux de respect individuels des tâches τ_1 et τ_{10} est proche de 0%.

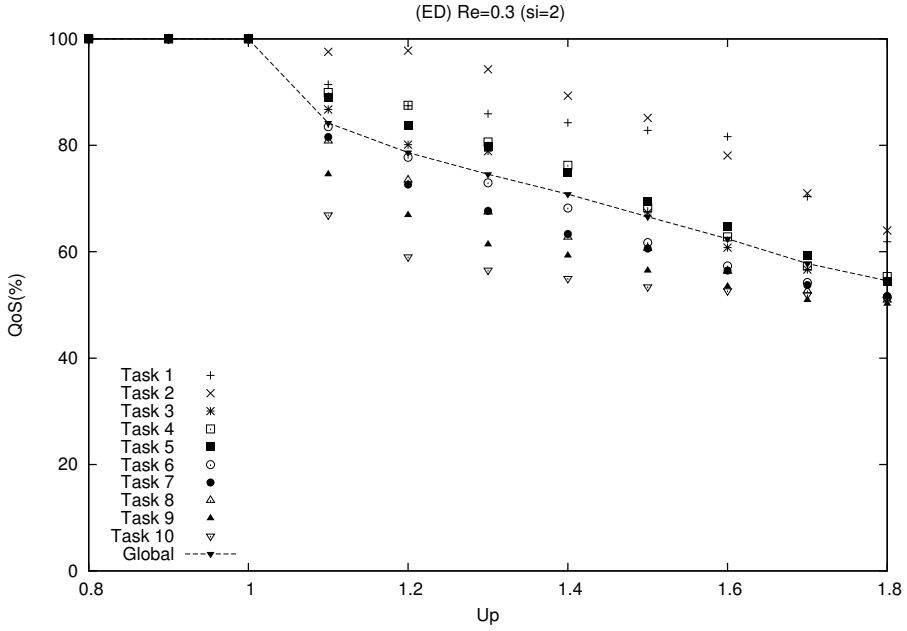


FIGURE 4.6 – Taux de respect individuels obtenus avec Green-BWP, $R_e = 0.3$

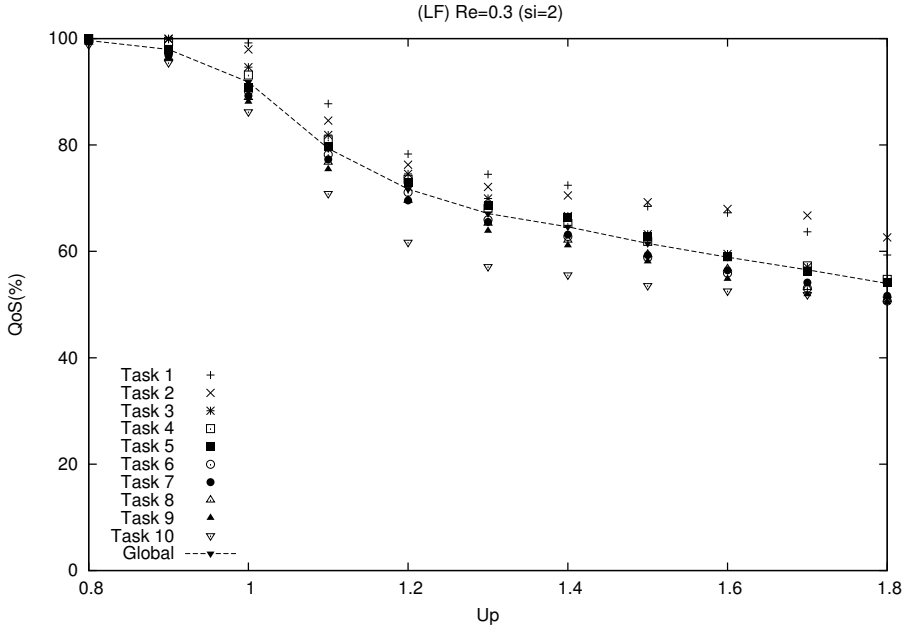


FIGURE 4.7 – Taux de respect individuels obtenus avec Green-BWP-LF, $R_e = 0.3$

- Quand le système est surchargé temporellement, Green-BWP est le moins stable. Il affiche la plus grande distance maximale entre les taux de respect individuels (jusqu'à 29% quand $U_p = 1.5$). Avec Green-BWP-LF, la distance maximale entre les taux de respect individuels est de 17% pour $U_p = 1.3$. Green-BWP-MS est le plus stable avec une distance maximale entre les taux de respect individuels des tâches τ_1 et τ_{10} égale à 10% pour $U_p = 1.1$. Pour $U_p > 1.1$, la distance maximale entre les taux de respect individuels des tâches est négligeable et quasi-nulle.

4.1.2 Analyse de la robustesse

La Figure 4.9 nous montre le taux de respect global pour les 3 algorithmes Green-BWP, Green-BWP-LF et Green-BWP-MS en fonction de la charge U_p du système. On observe que Green-BWP-LF et Green-BWP-MS sont moins robustes globalement que Green-BWP. Le fait d'ordonnancer les jobs bleus selon *EDF* crée certes un problème de stabilité mais néanmoins le taux de respect global du système est meilleur. A l'opposé,

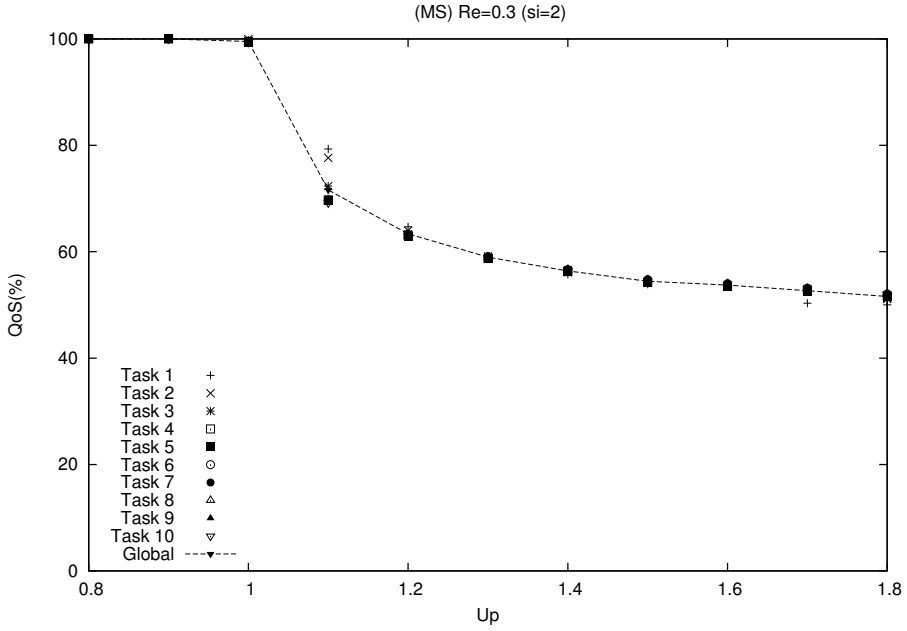


FIGURE 4.8 – Taux de respect individuels obtenus avec Green-BWP-MS, $R_e = 0.3$

Green-BWP-MS offre les performances les plus médiocres en termes de robustesse dès lors que le système est surchargé temporellement. En revanche, lorsque le système n'est pas surchargé temporellement ($U_p \leq 1$). Green-BWP-MS affiche les mêmes résultats de performance que Green-BWP. Green-BWP-LF offre le même taux de respect que Green-BWP pour $U_p = 0.8$. Cependant ses performances de stabilité se dégradent pour des valeurs de U_p supérieures à 0.8. Pour $U_p > 1$, Green-BWP-LF offre un taux de respect global intermédiaire, situé entre celui de Green-BWP et Green-BWP-MS.

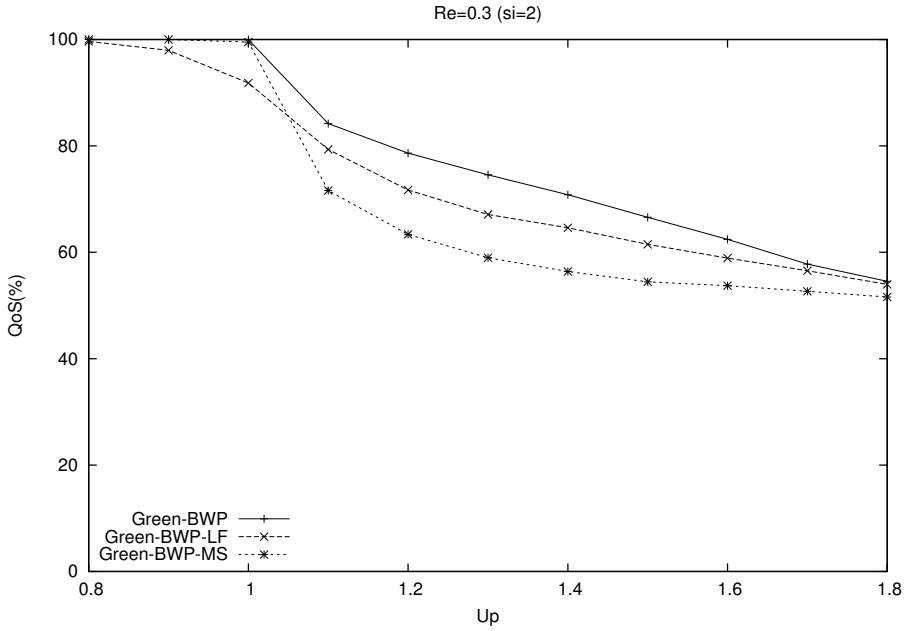


FIGURE 4.9 – Taux de respect global, $R_e = 0.3$

En résumé, en termes de robustesse, nous concluons que :
- Quand le système n'est pas surchargé temporellement, Green-BWP et Green-BWP-MS sont les ordonnanceurs les plus robustes car leur taux de respect global tend vers 100% quand $U_p \leq 1$. Green-BWP-LF est par contre le moins robuste des variantes orientées stabilité, affichant un taux de respect global qui décroît quand la charge processeur augmente.

- Quand le système est surchargé temporellement Green-BWP est le plus robuste affichant le meilleur taux de respect global parmi Green-BWP-LF et Green-BWP-MS, et Green-BWP-MS est le moins robuste.

4.2 Système fortement chargé énergétiquement

Considérons un système fortement chargé énergétiquement tel que $R_e = 0.9$.

4.2.1 Analyse de la stabilité

Les Figures 4.10, 4.11 et 4.12 affichent les taux de respect individuels des tâches en fonction de U_p respectivement pour Green-BWP, Green-BWP-LF et Green-BWP-MS.

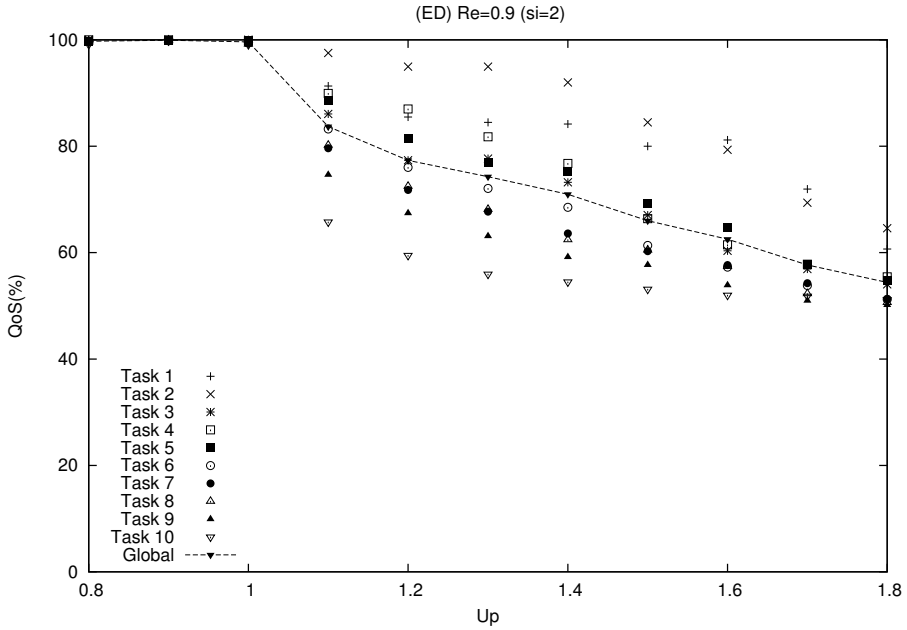


FIGURE 4.10 – Taux de respect individuels obtenus avec Green-BWP, $R_e = 0.9$

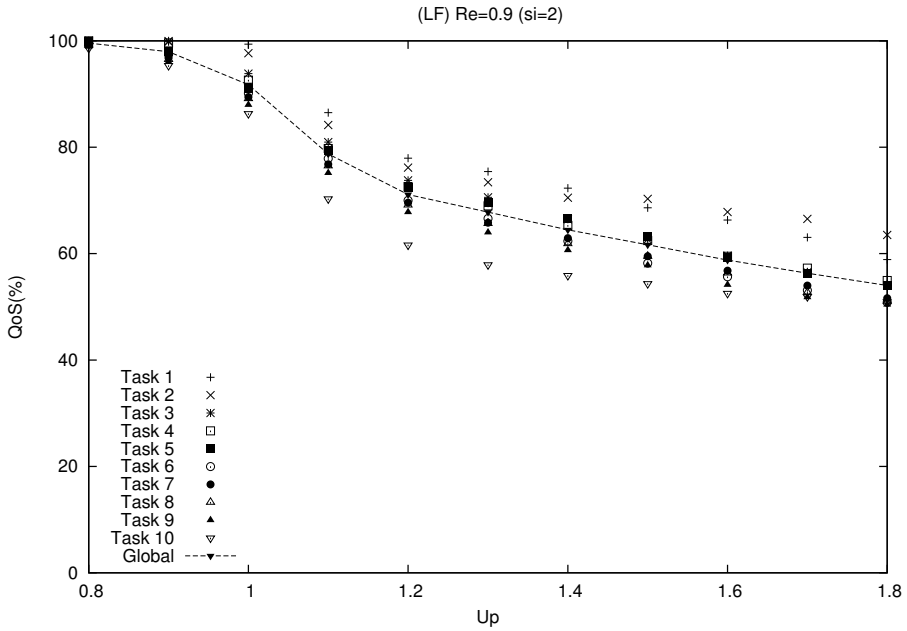


FIGURE 4.11 – Taux de respect individuels obtenus avec Green-BWP-LF, $R_e = 0.9$

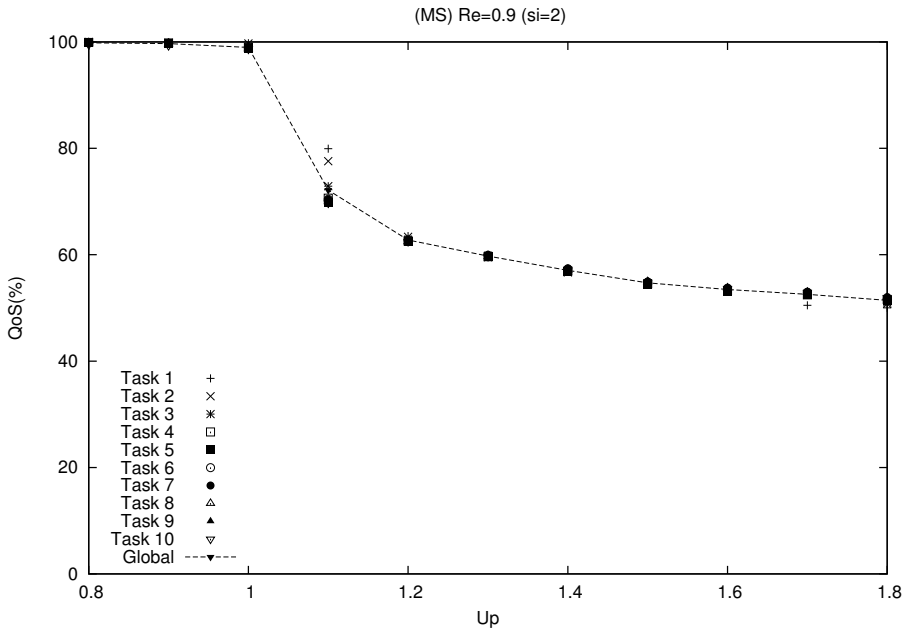


FIGURE 4.12 – Taux de respect individuels obtenus avec Green-BWP-MS , $R_e = 0.9$

On constate que :

- Quand le système n'est pas surchargé temporellement, Green-BWP-LF est le moins stable. En effet, la plus grande distance entre les taux de respect individuels est observée pour les tâches τ_1 et τ_{10} et est égale à 13.07% quand $U_p = 1$. Pour $U_p \leq 1$, Green-BWP et Green-BWP-MS sont stables car la distance maximale entre les taux de respect individuels des tâches est proche de zéro.

- Quand le système est surchargé temporellement, Green-BWP est le moins stable car la distance maximale entre les taux de respect individuels des tâches atteint 30% quand $U_p = 1.4$. La distance maximale est de 17% avec Green-BWP-LF pour $U_p = 1.3$. Green-BWP-MS est le plus stable avec une distance maximale entre les taux de respect individuels des tâches égal à 10% pour $U_p = 1.1$. Pour $U_p > 1.1$, la distance maximale entre les taux de respect individuels des tâches est négligeable et quasi-nulle.

4.2.2 Analyse de la robustesse

Sur la Figure 4.13, on observe les courbes correspondant aux taux de respect global obtenus en fonction de U_p pour les 3 algorithmes Green-BWP, Green-BWP-MS et Green-BWP-LF.

On remarque que :

- Quand le système n'est pas surchargé temporellement, Green-BWP et Green-BWP-MS sont les ordonnanceurs les plus robustes car le taux de respect global est proche des 100% quand $U_p \leq 1$. Green-BWP-LF est par contre le moins robuste parmi les deux autres ordonnanceurs affichant un taux de respect global qui décroît quand la charge du processeur augmente.

- Quand le système est surchargé temporellement, Green-BWP est l'ordonnanceur le plus robuste. En revanche, Green-BWP-MS offre la robustesse la plus faible.

4.3 Système surchargé énergétiquement

Considérons un système en surcharge de point de vue énergétique tel que $R_e = 1.2$.

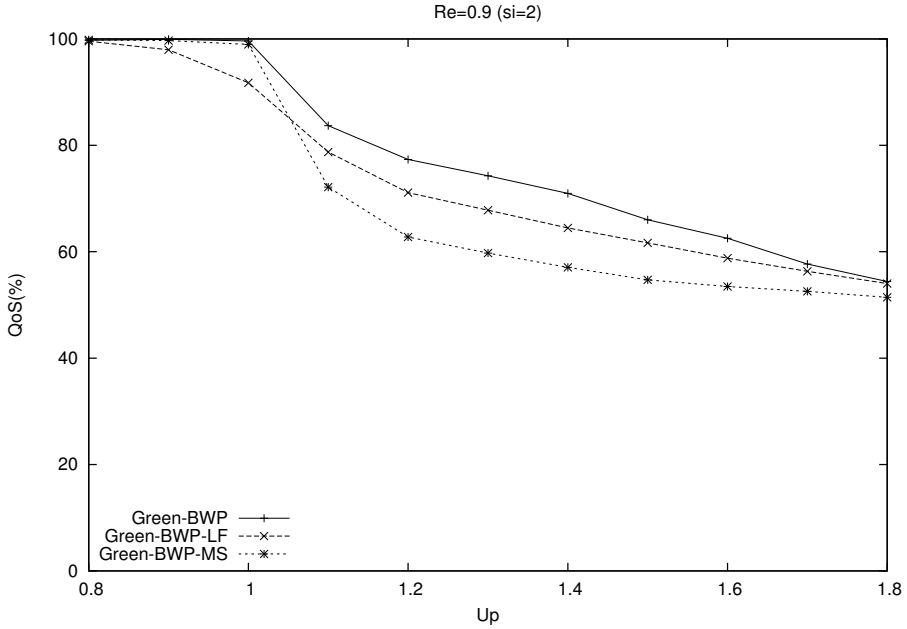


FIGURE 4.13 – taux de respect global, $R_e = 0.9$

4.3.1 Analyse de la stabilité

Sur les Figures 4.14, 4.15 et 4.16, on observe respectivement les taux de respect individuels des tâches pour Green-BWP, Green-BWP-LF et Green-BWP-MS en fonction de U_p.

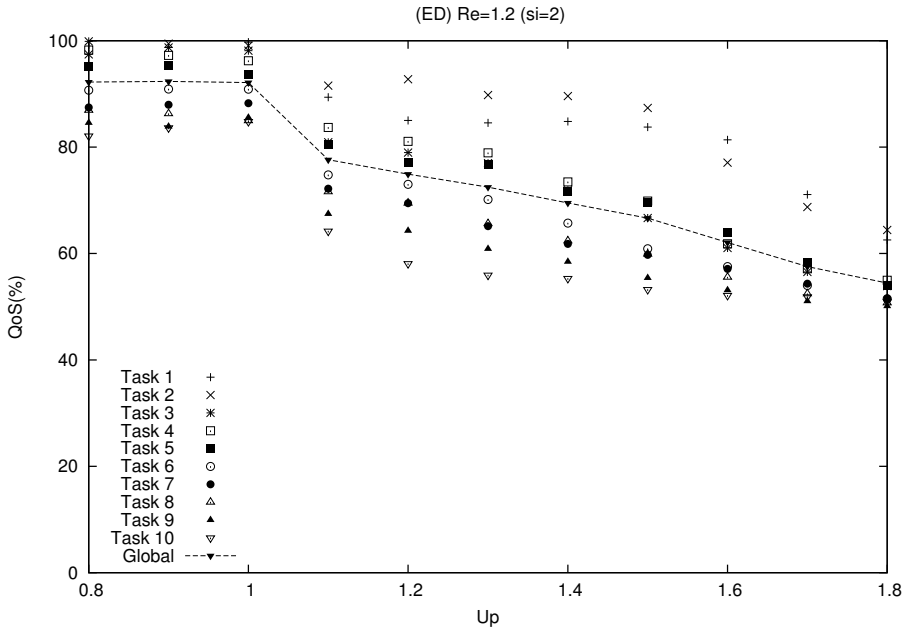


FIGURE 4.14 – Taux de respect individuels obtenus avec Green-BWP, $R_e = 1.2$

On remarque que :

- Quand le système n'est pas surchargé temporellement, Green-BWP-LF et Green-BWP sont les ordonnanceurs les moins stables. Les distances entre les taux de respect individuels des tâches peuvent atteindre 16% avec Green-BWP-LF quand $U_p = 0.9$ et 18% avec Green-BWP quand $U_p = 0.8$. Green-BWP-MS est le plus stable avec une distance maximale observée entre les taux de respect individuels égale à 7% quand $U_p = 0.9$.

- Quand le système est surchargé temporellement, Green-BWP est alors le moins stable car la distance

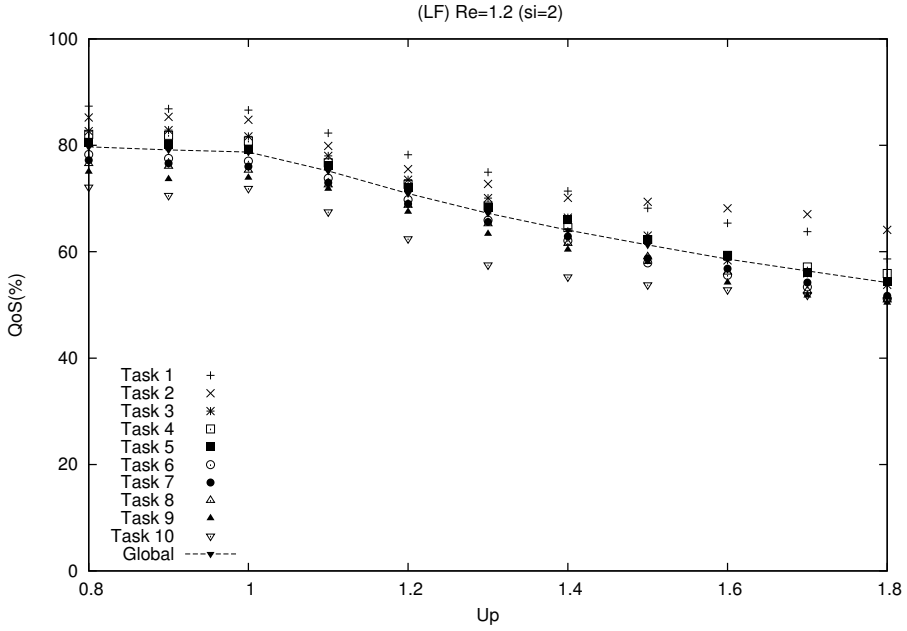


FIGURE 4.15 – Taux de respect individuels obtenus avec Green-BWP-LF, $R_e = 1.2$

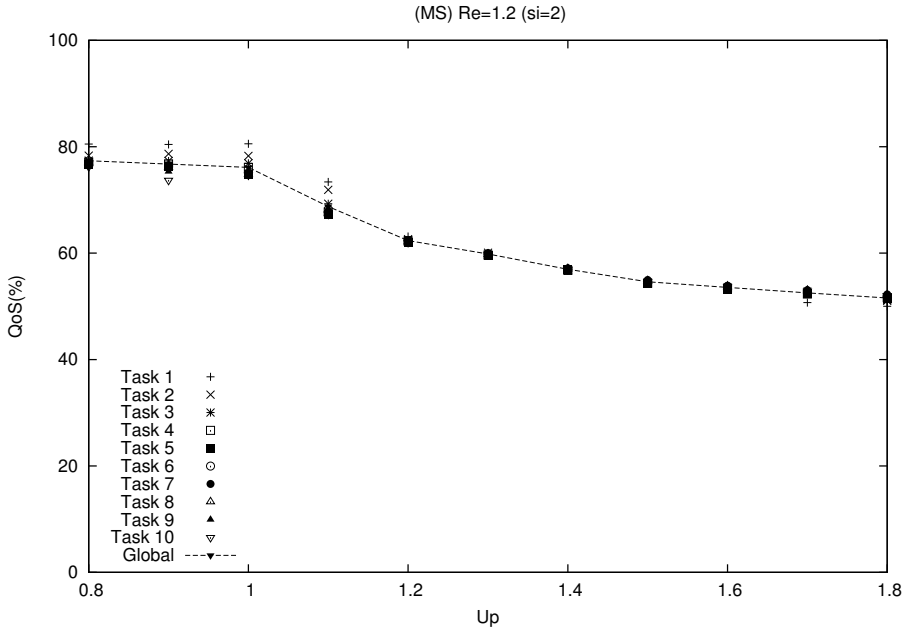


FIGURE 4.16 – Taux de respect individuels obtenus avec Green-BWP-MS, $R_e = 1.2$

maximale entre les taux de respect individuels des tâches est de 30% quand $U_p = 1.5$. Avec Green-BWP-LF, la distance maximale entre les taux de respect individuels est de 17% pour $U_p = 1.3$. Green-BWP-MS reste le plus stable avec une distance entre les taux de respect individuels des tâches égale à 6% pour $U_p = 1.1$. Pour $U_p > 1.1$, la distance maximale entre les taux de respect individuels des tâches est proche 0%.

4.3.2 Analyse de la robustesse

Sur la Figure 4.17, on observe les taux de respect global en fonction de la valeur de U_p pour Green-BWP, Green-BWP-LF et Green-BWP-MS.

On remarque :

- Quand le système n'est pas surchargé temporellement Green-BWP est le plus robuste. En effet, il offre un

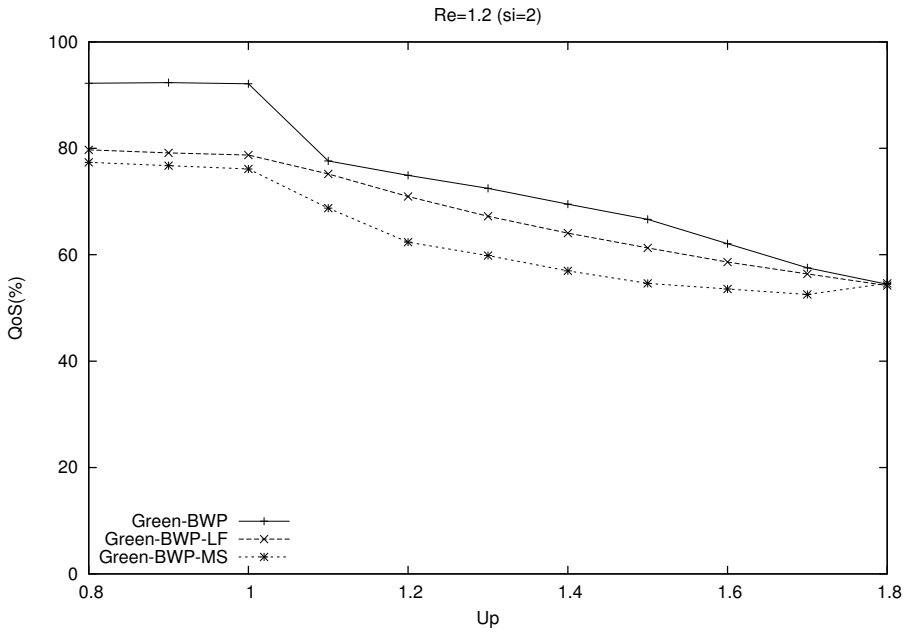


FIGURE 4.17 – Taux de respect global , $R_e = 1.2$

taux de respect global de 92% quand $U_p \leq 1$. Par contre, avec Green-BWP-LF et Green-BWP-MS, les taux de respect sont moindres respectivement 79% et 77%.

- Quand le système est surchargé temporellement Green-BWP reste plus robuste que Green-BWP-LF et Green-BWP-MS. Cependant, plus la charge processeur augmente, plus les courbes convergent pour atteindre un taux de respect inférieur à 50%.

Nous pouvons conclure qu'en fonction du système considéré et des contraintes temporelles et énergétiques, une de ces trois stratégies peut être favorisée par rapport aux autres en fonction des résultats requis par l'application.

4.4 Synthèse

Au vu des résultats de simulation présentés précédemment, nous avons pu constater le gain de stabilité obtenu grâce aux variantes LF et MS. Le tableau 4.1 synthétise la performance en termes de stabilité des différents ordonnanceurs étudiés selon le niveau de contrainte énergétique du système.

Les métriques observées sont les suivantes :
- d_{max}, la plus grande distance entre les taux de respect individuels,
- σ, l'écart-type calculé sur les taux de respect individuels des tâches.

L'écart-type va nous permettre d'évaluer la dispersion des taux de respect individuels autour du taux de respect global calculé pour chaque valeur de U_p. Plus l'écart-type calculé est faible moins les taux de respect individuels sont dispersés autour du taux de respect global et plus l'ordonnanceur est stable.

L'analyse du tableau 4.1, nous permet d'établir les conclusions suivantes :

- Green-BWP-MS affiche l'écart-type minimal quelle que soit la charge énergétique du système. Par conséquent, l'ordonnanceur Green-BWP-MS est le plus stable quel que soit le profil énergétique du système.
- Green-BWP affichant les plus grandes valeurs de distances maximales et d'écart-type, est l'ordonnanceur le moins stable.
- Green-BWP-LF dont les performances se situent entre Green-BWP et Green-BWP-MS peut être vu comme un ordonnanceur intermédiaire en termes de robustesse et de stabilité.

TABLE 4.1 – Comparaison de la stabilité

	Ordonnanceur	d_{max}	σ
$R_e = 0.3$	Green-BWP	29.41	**11.54**
	Green-BWP-LF	17.37	5.04
	Green-BWP-MS	10.11	**3.75**
$R_e = 0.9$	Green-BWP	29.69	**11.63**
	Green-BWP-LF	17.54	4.99
	Green-BWP-MS	10.22	**3.64**
$R_e = 1.2$	Green-BWP	30.55	**11.42**
	Green-BWP-LF	17.42	5.18
	Green-BWP-MS	6.73	**2.15**

On en conclut que, l'algorithme le plus stable est Green-BWP-MS. Il est suivi par Green-BWP-LF qui reste plus stable que Green-BWP. Cependant, même si Green-BWP est l'algorithme le moins stable, il reste le plus performant en terme de robustesse. En effet, le gain de stabilité acquis avec Green-BWP-LF et Green-BWP-MS cause des pertes de robustesse. Par conséquent, le choix de l'algorithme dépendra de l'équilibre requis par l'application en termes de robustesse et de stabilité. Il faudra trouver le bon compromis pour garantir le niveau de QoS souhaité par le concepteur de l'application au niveau des tâches (globalement et individuellement).
Le tableau 4.2 résume l'adéquation des 3 algorithmes vis-à-vis des critères de stabilité et de robustesse.

	Green-BWP	Green-BWP-LF	Green-BWP-MS
Stabilité	*	**	***
Robustesse	***	**	*

TABLE 4.2 – Performances des ordonnanceurs Green-BWP, Green-BWP-LF et Green-BWP-MS

4.5 Evaluation du temps moyen d'obtention de la stabilité

Nous évaluons à présent la performance dynamique de Green-BWP-MS et Green-BWP-LF, en mesurant la durée au bout de laquelle le ratio du nombre de jobs réussis par rapport au nombre de jobs réveillés devient très proche (voire identique) pour toutes les tâches. Nous fixons la charge périodique à $U_p = 1.4$ et le paramètre de pertes à $s_i = 2$. Nous observons les taux de respect individuels avec Green-BWP-LF et Green-BWP-MS sur une durée égale à 10 hyperpériodes.

4.5.1 Système faiblement chargé énergétiquement

Les Figures 4.18 et 4.19 représentent les variations dans le temps des taux de respect individuels des tâches, pour Green-BWP-LF et Green-BWP-MS respectivement, pour un système faiblement chargé énergétiquement. Les courbes nous montrent que les variations observées au niveau des taux de respect individuels sont minimales au-delà de la $4^{ème}$ et $3^{ème}$ hyperpériode de simulation respectivement pour Green-BWP-LF et Green-BWP-MS. L'écart maximal entre les taux de respect avec Green-BWP-MS est toujours beaucoup plus petit que celui obtenu avec Green-BWP-LF.

4.5.2 Système fortement chargé énergétiquement

Les Figures 4.20 et 4.21 illustrent les variations des taux de respect individuels des tâches dans le temps, pour Green-BWP-LF et Green-BWP-MS respectivement, pour un système fortement chargé énergétiquement. Nous fixons cette fois-ci $R_e = 0.9$. Dans la Figure 4.21, la distance entre les taux de respect individuels devient constante au bout de 3 hyperpériodes avec Green-BWP-MS.

Dans la Figure 4.20, la distance entre les taux de respect individuels ne varie plus au bout de 4 hyperpériodes avec Green-BWP-LF.

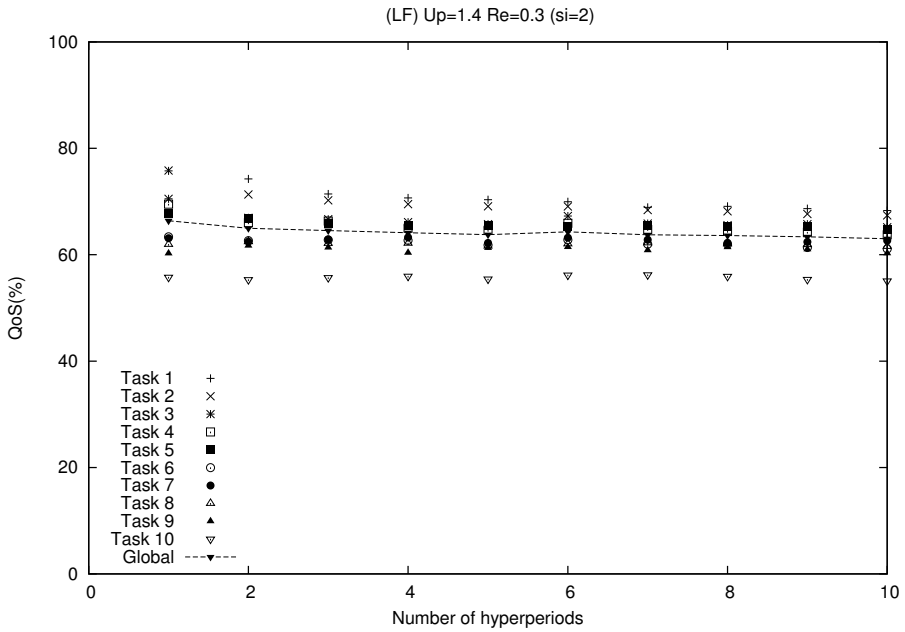


FIGURE 4.18 – Taux de respect individuels avec Green-BWP-LF, $R_e = 0.3$

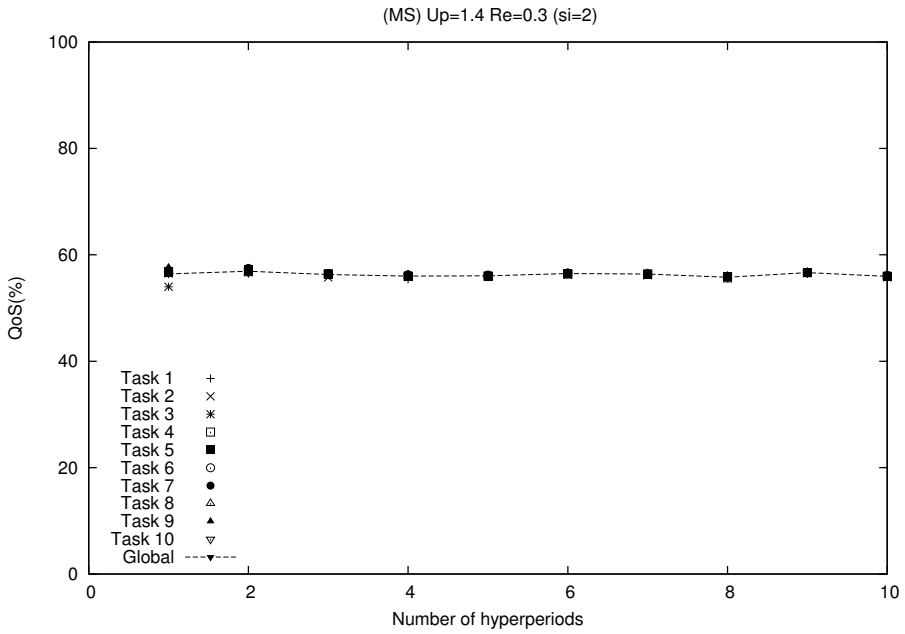


FIGURE 4.19 – Taux de respect individuels avec Green-BWP-MS, $R_e = 0.3$

4.5.3 Système surchargé énergétiquement

Les Figures 4.22 et 4.23 affichent les variations des taux de respect individuels des tâches dans le temps, pour Green-BWP-LF et Green-BWP-MS respectivement, dans un système surchargé énergétiquement. Nous fixons ici $R_e = 1.2$.

Dans la Figure 4.23, la distance entre les taux de respect individuels se stabilise au bout de 3 hyperpériodes avec Green-BWP-MS. Dans la Figure 4.22, la distance entre les taux de respect individuels ne varie plus au bout de 6 hyperpériodes avec Green-BWP-LF. Nous concluons donc que Green-BWP-MS atteint la stabilité dans un délai plus court que Green-BWP-LF. De plus, nous remarquons que l'écart maximal entre les taux de respect avec Green-BWP-MS est toujours beaucoup plus petit que celui obtenu avec Green-BWP-LF.

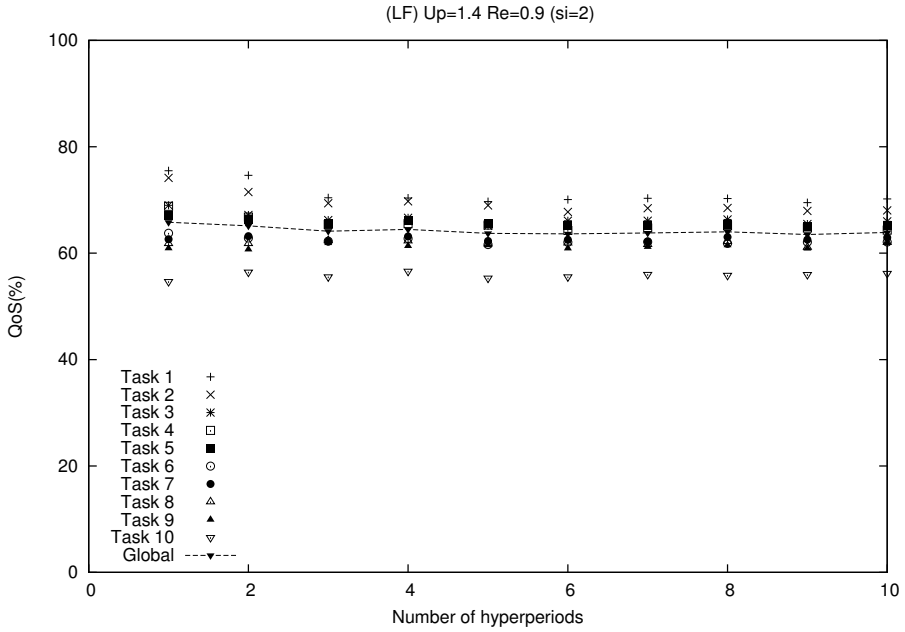


FIGURE 4.20 – Taux de respect individuels avec Green-BWP-LF, $R_e = 0.9$

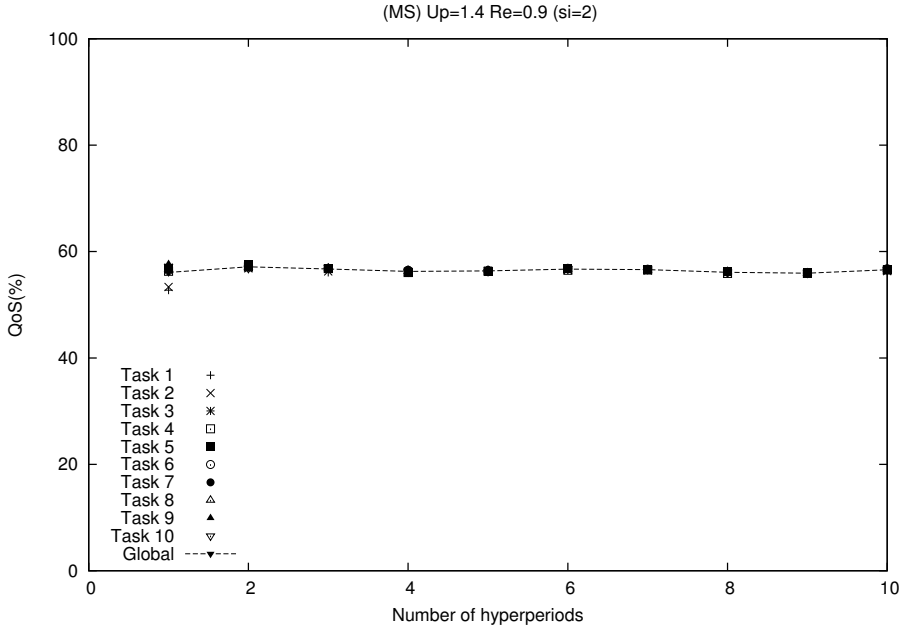


FIGURE 4.21 – Taux de respect individuels avec Green-BWP-MS, $R_e = 0.9$

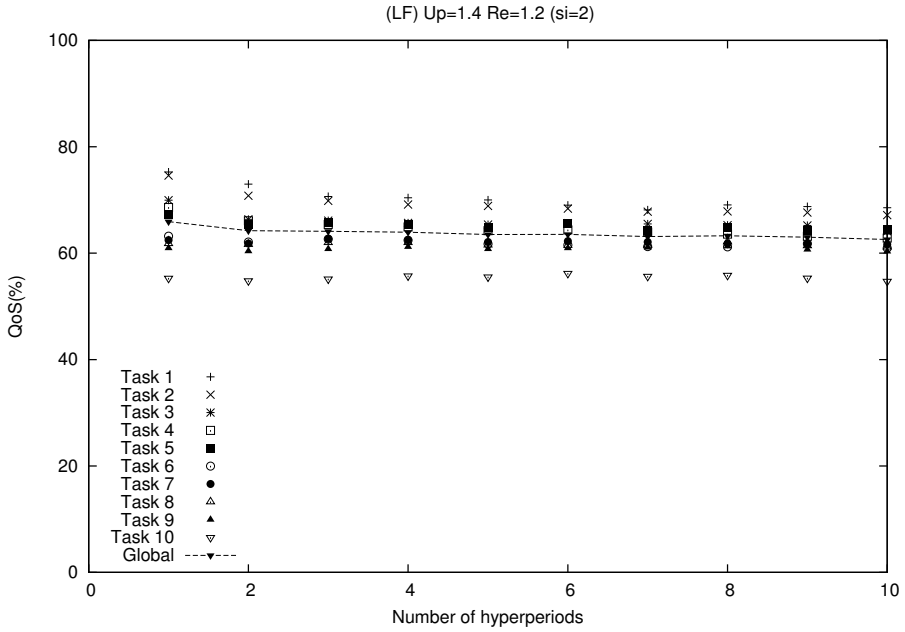


FIGURE 4.22 – Taux de respect individuels avec Green-BWP-LF, $R_e = 1.2$

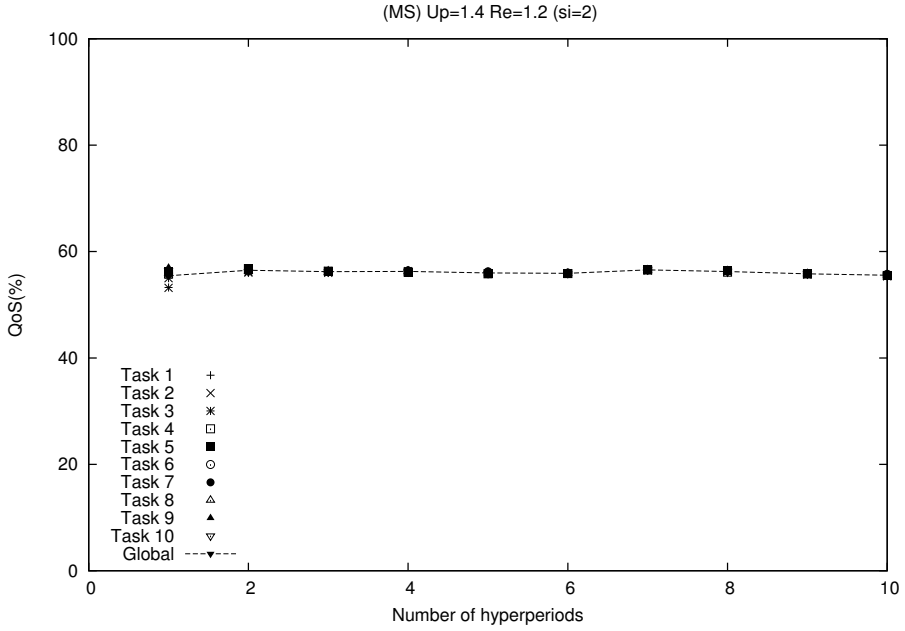


FIGURE 4.23 – Taux de respect individuels avec Green-BWP-MS, $R_e = 1.2$

5 Conclusion

Dans ce chapitre, nous avons dans un premier temps étudié la stabilité et la robustesse de l'ordonnanceur Green-BWP pour lequel tous les jobs sont ordonnancés selon la règle EDF. Après avoir traité un exemple soulignant son manque de stabilité, nous avons proposé deux nouveaux ordonnanceurs, à savoir Green-BWP-LF et Green-BWP-MS.

L'évaluation menée par le biais de diverses simulations, nous a permis d'observer le gain de stabilité obtenu grâce à ces nouveaux ordonnanceurs. Cependant, nous avons pu remarquer que le gain de stabilité engendrait une légère dégradation de la QoS globale du système. Nous pouvons alors déduire que le gain de stabilité s'assortit d'une perte de robustesse du système.

Le choix du meilleur ordonnanceur sera donc directement dicté par les spécifications de l'application concernée. Il peut être parfois plus judicieux de privilégier la stabilité par rapport à la robustesse, ou l'inverse, voire de trouver un compromis satisfaisant entre robustesse et stabilité.

Dans le chapitre suivant, nous traitons l'ordonnancement de tâches périodiques temps réel strictes conjointement à des tâches apériodiques sur un monoprocesseur. Notre objectif est alors de minimiser le temps de réponse des tâches apériodiques tout en assurant l'exécution des tâches périodiques dans le respect de leur échéance.

Troisième partie

Contribution à l'ordonnancement de tâches apériodiques

Chapitre 5

Ordonnancement de tâches apériodiques sans gestion de surcharge

Sommaire

Dans ce chapitre, le problème traité est d'ordonnancer des tâches apériodiques non critiques conjointement à des tâches périodiques sans pertes. L'objectif consiste à minimiser le temps de réponse des tâches apériodiques. Dans un premier temps nous proposons de nouveaux serveurs de tâches apériodiques appelés BG-EDeg, TBS-EDeg et TBS_{op}-EDeg qui se basent sur les serveurs bien connus BG et TBS. Dans un deuxième temps, nous établissons une étude comparative des performances des différents algorithmes proposés.

1 Terminologie et modèles de tâches

Nous considérons un système temps réel dur et nous nous intéressons au problème de l'ordonnancement monoprocesseur de tâches périodiques et de tâches apériodiques non critiques sous contraintes temporelles et énergétiques.

1.1 Modèle de tâches périodiques

Nous considérons une configuration de tâches périodiques temps réel strictes $\mathcal{T}$ dans laquelle chaque tâche τ_i est caractérisée par :

- r_i : sa date de réveil,
- C_i : sa durée d'exécution au pire cas (WCET),
- D_i : son échéance relative,
- d_i : son échéance absolue,
- T_i : sa période,
- E_i : sa demande énergétique au pire cas (WCEC).

Les tâches périodiques sont indépendantes, synchrones et à échéances sur requêtes ($D_i = T_i$). Elles sont ordonnancées avec EDeg (Earliest Deadline with energy guarantee) [CG09].

1.2 Modèle de tâches apériodiques non critiques

Nous considérons de plus une configuration de tâches apériodiques non critiques φ dans laquelle une tâche apériodique φ_i est caractérisée par :

- r_i^a : sa date d'arrivée
- C_i^a : sa durée d'exécution au pire cas
- E_i^a : sa demande énergétique

A partir des caractéristiques propres à chaque tâche apériodique citées ci-dessus, nous nous intéressons aux variables suivantes :

- G_i^a : sa gigue de démarrage (ou Jitter en anglais)
- f_i^a : sa date de fin d'exécution
- TR_i, le temps de réponse : $TR_i = f_i^a - r_i^a$,
- TG_i, le temps de gigue de démarrage : $TG_i = G_i^a - r_i^a$,
- TRn_i, le temps de réponse normalisé : $TRn_i = \frac{C_i}{TR_i}$,
- TGn_i, le temps de gigue de démarrage normalisé : $TGn_i = \frac{TG_i}{f_i^a - r_i^a}$,

Les tâches apériodiques sont rangées selon un ordre FIFO (le premier arrivé est le premier servi).

Par la suite nous proposons différents algorithmes appelés serveurs permettant d'exécuter conjointement les tâches périodiques et apériodiques tout en garantissant le respect des échéances des tâches périodiques.

2 Méthode basée sur le serveur BG

Nous proposons un nouvel algorithme nommé BG-EDeg qui se base sur le principe de l'algorithme BG (Background Server) [LSS87]. Les jobs périodiques sont toujours les plus prioritaires par rapport aux tâches apériodiques. Par conséquent, les tâches apériodiques ne sont exécutées que pendant les laxités temporelles et énergétiques existantes dans le système. Ainsi à un instant *t*, s'il n'y a aucun job périodique à l'état prêt et s'il existe des tâches apériodiques en attente, une tâche apériodique est choisie selon *FIFO* pour être exécutée. Cependant, cette tâche ne débutera son exécution que si elle ne compromet pas l'exécution de jobs périodiques futurs. Pour cela, notre algorithme fait en sorte d'exécuter partiellement ou totalement la tâche apériodique s'assurant que la batterie soit toujours pleine au prochain réveil de futurs jobs périodiques.

2.1 Description de l'algorithme

L'algorithme BG-EDeg est caractérisé par les données suivantes :

- $\mathcal{T}(t)$ la liste des tâches périodiques prêtes à l'instant t.
- $\varphi(t)$ la liste des tâches apériodiques prêtes à l'instant t.
- $idle$ un booléen prenant la valeur *FAUX* si le processeur est *actif* et *VRAI* si le processeur est *inactif*.
- E_{min} : le niveau d'énergie minimale dans la batterie.
- E_{max} : le niveau d'énergie maximale dans la batterie.
- $E_i^a(t)$: l'énergie que la tâche apériodique φ_i a consommé pendant l'intervalle $[r_i^a, t]$.
- $C_i^a(t)$: la durée exécutée pour la tâche apériodique φ_i pendant l'intervalle $[r_i^a, t]$.
- $E_{disp}(t)$: la quantité maximale d'énergie pouvant être utilisée par une tâche apériodique à l'instant *t* de façon à ne pas compromettre l'exécution de jobs périodiques futurs.

Dans l'algorithme BG-EDeg (cf. Algorithme 6) nous ne présentons que le traitement fait sur les tâches apériodiques. Les jobs périodiques quant à eux sont ordonnancés selon l'algorithme EDeg présenté dans le chapitre 3. En résumé, les jobs périodiques prêts sont ordonnancés selon la règle *EDF*. Ainsi à chaque instant *t*, le job périodique ayant la plus proche échéance est choisi pour être exécuté. Cependant, celui-ci ne sera exécuté que si l'énergie disponible dans la batterie à l'instant t est suffisante et si la laxité énergétique du système à l'instant t est positive, de façon à ne pas compromettre l'exécution de futurs jobs périodiques.

Rappelons que la laxité énergétique du système à l'instant t, $SlackEnergy(t)$, représente le surplus d'énergie dans le système qui peut être consommé à l'instant *t*. Soit d, l'échéance du job le plus prioritaire à l'instant t, elle correspond à la laxité énergétique minimale des jobs périodiques prêts et ayant leur échéance inférieure à d.

La laxité énergétique d'un job $\tau_{i,k}$ ($k^{ème}$ job de la tâche τ_i) à l'instant *t* est le surplus d'énergie qui peut être consommé entre t et son échéance $d_{i,k}$ et elle est calculée comme suit :

$$SlackEnergy(\tau_{i,k}, t) = E(t) + \int_t^{d_{i,k}} P_r(x)dx - A_{i,k} \tag{5.1}$$

où, l'on représente par :
- $E(t)$, l'énergie restante dans la batterie à l'instant t,
- $P_r(x)$, la puissance reçue à l'instant x,
- $A_{i,k}$, la demande énergétique des jobs réveillés après $r_{i,k}$ et d'échéance inférieure ou égale à $d_{i,k}$. Donc $A_{i,k} = \sum_{r_j \geq t, d_j \leq d_{i,k}} E_j$,
- E_j, l'énergie requise pour exécuter la tâche périodique τ_j.

Dans le cas où l'énergie disponible dans la batterie n'est pas suffisante ou la laxité énergétique du système est nulle, la laxité temporelle est calculée avec l'algorithme EDL à l'instant *t*.

2.2 Exemple illustratif avec BG-EDeg

On considère une configuration de tâches périodiques $\mathcal{T}$ telle que $\tau_1(3, 12, 12, 7)$ et $\tau_2(3, 15, 15, 8)$. $P_r = 2$ et $E(0) = 6$. Comme $U_p = 0.45$ et $U_e = 0.495$, le système est moyennement chargé temporellement et énergétiquement. La Figure 5.1 illustre l'ordonnancement de la configuration $\mathcal{T}$ suivant BG-EDeg. Une tâche apériodique, $\alpha(4, 4, 12)$ arrive à *t=4*. Elle n'est exécutée que s'il n'y a aucun job périodique à l'état prêt et que s'il y a suffisament d'énergie dans la batterie de sorte qu'au prochain réveil d'un job périodique, la batterie soit pleine (i.e à $r_{min} = 12$, $E(12) = E_{max} - E_{min} = 6$).
A l'instant *t=6*, il n'y a aucun job périodique à l'état prêt. On calcule $E_{disp}(6)$ qui vaut 6. Comme $E_{disp}(6)$ est inférieur à E_{ap}^1 (i.e l'énergie demandée par α), α ne sera pas entièrement exécutée entre *t=4* et r_{min}. La tâche

Algorithm 6 BG-EDeg

t : temps courant
$\mathcal{T}(t)$: liste des tâches périodiques prêtes
$\varphi(t)$: liste des tâches apériodiques prêtes
$idle$: état du processeur
$E_{disp}(t)$ = 0
while VRAI **do**
  idle ← VRAI
  if $\mathcal{T}(t) \neq (\phi)$ **then**
    $E_{disp}(t)$ = 0
    Exécuter la tâche périodique la plus prioritaire selon EDeg
  else if $\mathcal{T}(t) = (\phi)$ **then**
    Soit φ_i la tâche apériodique en tête de la liste $\varphi(t)$
    Soit r_{min} : le plus proche réveil parmi les tâches périodiques.
    if $E_{disp}(t)$ = 0 **then**
      $E_{disp}(t) = E(t) + (r_{min} - t).P_r - (E_{max} - E_{min})$
    end if
    if $E_{disp}(t) \neq 0$ **then**
      idle ← FAUX
    end if
    if $idle$ = FAUX **then**
      Exécuter une unité de temps de φ_i
      Mettre à jour $E_i^a(t)$
      Mettre à jour $C_i^a(t)$
      Mettre à jour $E_{disp}(t)$
      if $C_i^a(t) = 0$ **then**
        enlever φ_i de φ
        $E_{disp}(t)$ = 0
      end if
    end if
  end if
end while

est exécutée tant que la quantité d'énergie E_{disp} est positive. α sera à nouveau exécutée à *t=18* et finira son exécution à *t=19*.
Dans cet exemple, le système est moyennement chargé temporellement et énergétiquement. Le temps de réponse de α est de 15 unités de temps et son temps de réponse normalisé vaut $\frac{4}{15} = 0.26$ ce qui montre que la durée d'attente est importante. En conséquence, nous pouvons penser que dans un système fortement chargé énergétiquement et/ou temporellement, le temps de réponse risque d'être beaucoup plus important. C'est pourquoi, dans la suite, nous proposons d'autres algorithmes dans le but d'améliorer le temps de réponse des tâches apériodiques.

3 Algorithmes basés sur le serveur TBS

Dans cette section, nous présentons TBS-EDeg et TBS_{op}-EDeg, deux nouveaux algorithmes utilisant le principe du serveur *TBS* [SB94] permettant d'ordonnancer conjointement les tâches périodiques et apériodiques non critiques. Le serveur TBS [SB94] est un mécanisme simple qui revient à transformer les tâches apériodiques non critiques en tâches apériodiques à contraintes strictes, par le calcul d'une échéance fictive. Cette échéance fictive est calculée à chaque fois qu'une tâche apériodique survient. Cette échéance permettra à la tâche de s'exécuter avec un temps de réponse minimal. La tâche apériodique est ensuite exécutée conjointement aux tâches périodiques selon l'algorithme *EDF*.

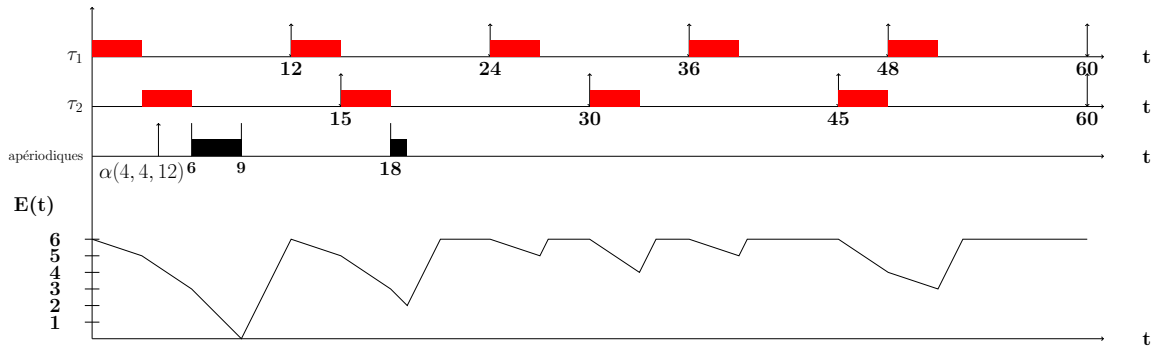


FIGURE 5.1 – Ordonnancement selon BG-EDeg

3.1 Définitions et tests d'ordonnançabilité

3.1.1 Point de vue temporel

Spuri et Buttazo ont proposé et démontré le théorème suivant :

Théorème 3.1 *[SB94] Soient une configuration de n tâches périodiques, avec un facteur d'utilisation U_p, et une configuration de tâches apériodiques servies par un serveur TBS, avec un facteur d'utilisation du serveur U_S. Le système global est ordonnançable si et seulement si :*

$$U_p + U_S \leq 1 \tag{5.2}$$

Un deuxième théorème proposé par Spuri et Butazzo, s'énonce comme suit :

Théorème 3.2 *[SB94] Soient une configuration de n tâches périodiques, avec un facteur d'utilisation U_p, et une configuration de tâches apériodiques servies par un serveur TBS, avec un facteur d'utilisation du serveur U_S. Dans chaque intervalle de temps $[t_1, t_2]$, si C_{ape} est la demande totale de temps processeur exigée par les tâches apériodiques avec une date d'arrivée supérieure ou égale à t_1 et une date d'échéance inférieure ou égale à t_2, alors $C_{ape} \leq (t_2 - t_1)U_S$.*

Spuri et Buttazo proposent une formule permettant le calcul de l'échéance fictive en fonction de sa largeur de bande CPU, U_S [SB94] :

$$d_k^a = max(r_k^a, d_{k-1}^a) + \frac{C_k^a}{U_S} \tag{5.3}$$

où,
C_k^a est la durée d'exécution de la requête apériodique courante,
U_S est le facteur d'utilisation du serveur tel que $U_S \leq 1 - U_p$,
Par définition, $d_0^a = 0$.

Par la suite, nous proposons des théorèmes et formules qui prennent en compte la contrainte énergétique .

3.1.2 Point de vue énergétique

Nous supposons que la puissance reçue par l'environnement est constante, $P_r(t) = P_r$, pendant une hyperpériode $H = PPCM(T_1, .., T_i, .., T_n)$. Nous proposons le théorème suivant :

Théorème 3.3 *Soient une configuration de n tâches périodiques, avec un facteur d'utilisation énergétique U_e, et une configuration de tâches apériodiques servies par un serveur TBS, avec une largeur de bande énergétique du serveur U_{es}. Dans chaque intervalle de temps $[t_1, t_2]$, considérons que E_{ape} est la demande totale d'énergie requise par les tâches apériodiques avec une date d'arrivée supérieure ou égale à t_1 et avec une date d'échéance inférieure ou égale à t_2, alors il est nécessaire que :*

$$E_{ape} \leq (E(t_1) + (t_2 - t_1)P_r)U_{es} \tag{5.4}$$

Preuve :
Considérons que pendant l'intervalle $[t_1, t_2]$, il existe m tâches apériodiques ayant une date de réveil supérieure ou égale à t_1 et une échéance inférieure ou égale à t_2. Nous supposons que la puissance reçue, P_r, est constante. Par conséquent, l'énergie disponible fournie pendant l'intervalle $[t_1, t_2]$ est égale à $E(t_1)+(t_2-t_1)P_r$. Comme U_{es} est la proportion d'énergie disponible pour les tâches apériodiques, la quantité d'énergie maximale pouvant être consommée par les tâches apériodiques est $(E(t_1)+(t_2-t_1)P_r)U_{es}$. Par conséquent, soit E_{ape} la quantité d'énergie consommée par les tâches apériodiques sur l'intervalle $[t_1, t_2]$ telle que $E_{ape} = \sum_{i=1,(r_i^a \geq t_1),(d_i^a \leq t_2)}^{m}$ alors $E_{ape} \leq (E(t_1) + (t_2 - t_1)P_r)U_{es}$.□

Théorème 3.4 *Soient une configuration de n tâches périodiques, avec un facteur d'utilisation énergétique U_e, et une configuration de m tâches apériodiques servies par un serveur TBS. On définit U_{es} le facteur d'utilisation énergétique du serveur. Le système global est ordonnançable du point de vue énergétique seulement si :*

$$U_e + U_{es} \leq 1 \tag{5.5}$$

Preuve :
Supposons que $U_e + U_{es} \leq 1$ et qu'une violation d'échéance survienne par manque d'énergie à un instant $t = t_v$. Cette violation est précédée par l'exécution de tâches périodiques et/ou apériodiques. Par conséquent, soit un instant $t = t_1$ inférieur à t. Seules les tâches périodiques et apériodiques ayant leur instant de réveil supérieur ou égal à t_1 et une échéance inférieure ou égale à $t = t_v$ sont exécutées sur l'intervalle $[t_1, t_v]$.
Soit E_g, la demande d'énergie globale des tâches s'exécutant sur l'intervalle $[t_1, t_v]$. S'il y a eu un manque d'énergie à un instant t_v, ceci s'explique par le fait que la demande d'énergie sur l'intervalle $[t_1, t_v]$ excède l'énergie disponible entre t_1 et t_v :
$E_g > E(t_1) + (t_v - t_1)P_r$.

Remarquons aussi que $E_g \leq g(t_1, t_v) + E_{ape}$.
où :
- $g(t_1, t_v)$ est la demande énergétique des tâches périodiques ayant leur instant de réveil supérieur ou égal à t_1 et une échéance inférieure ou égale à $t = t_v$ telle que $g(t_1, t_v) = \sum_{i=1}^{n} g_i(t_1, t_v) = \sum_{i=1}^{n} \left\lfloor \frac{t_v - t_1}{T_i} \right\rfloor E_i$,
- E_{ape} est la demande énergétique des tâches apériodiques ayant leur instant de réveil supérieur ou égal à t_1 et une échéance inférieure ou égale à $t = t_v$.

Ceci implique que :
$E_g \leq \sum_{i=1}^{n} \left\lfloor \frac{t_v - t_1}{T_i} \right\rfloor E_i + (E(t_1) + (t_v - t_1)P_r)U_{es}$
$\leq (E(t_1) + (t_v - t_1)P_r)U_e + (E(t_1) + (t_v - t_1)P_r)U_{es}$
$\leq (E(t_1) + (t_v - t_1)P_r)(U_e + U_{es})$
Par suite, $U_e + U_{es} > 1$. Ce qui contredit notre supposition.□

Théorème 3.5 *Soit U_{es} la largeur de bande énergétique telle que $U_{es} = 1 - U_e$. L'échéance fictive d'une tâche apériodique φ_k en fonction de la largeur de la bande énergétique U_{es} est calculée comme suit :*

$$d_k^a = max(r_k^a, d_{k-1}^a) + \left\lceil \frac{\frac{E_k^a}{U_{es}} - E(r_k^a)}{P_r} \right\rceil \tag{5.6}$$

Preuve : Le calcul de l'échéance fictive d_k^a, pour une tâche apériodique φ_k, est fait à l'instant de son réveil r_k^a. Considérons que φ_k a une demande énergétique qui vaut E_k^a. Par conséquent l'échéance fictive doit être calculée de sorte que la tâche puisse disposer de cette quantité d'énergie. Selon la condition 5.4, E_k^a ne doit pas excéder l'énergie disponible pendant l'intervalle $[r_k^a, d_k]$. Par suite, $E_k^a \leq (E(r_k^a) + (d_k^a - r_k^a)P_r)U_{es}$. On obtient l'inégalité suivante $d_k^a \geq \frac{\frac{E_k^a}{U_{es}} - E(r_k^a)}{P_r} + r_k^a$.

Nous considérons qu'une tâche apériodique n'est jamais préemptée par une autre tâche apériodique donc $d_{k-1}^a < d_k^a$. De plus, dans le pire-cas, φ_{k-1} se termine à d_{k-1}^a. Donc la tâche apériodique φ_k suivante peut commencer son exécution à l'instant $t = max(r_k^a, d_{k-1}^a)$ et donc l'inégalité devient : $d_k^a \geq \frac{\frac{E_k^a}{U_{es}} - E(r_k^a)}{P_r} + max(r_k^a, d_{k-1}^a)$. □

3.1.3 Point de vue temporel et énergétique

Théorème 3.6 *Soient une configuration de n tâches périodiques et une configuration de tâches apériodiques servies par un serveur TBS, l'échéance fictive d'une tâche apériodique φ_k est calculée comme suit :*

$$d_k^a = max(r_k^a, d_{k-1}^a) + max(\left\lceil \frac{C_k^a}{U_S} \right\rceil, \left\lceil \frac{\frac{E_k^a}{U_{es}} - E(r_k^a)}{P_r} \right\rceil) \tag{5.7}$$

Preuve : La preuve découle des formules 5.3 et 5.6.□

Par la suite, nous proposons deux algorithmes basés sur TBS : TBS-EDeg et TBS_{op}-EDeg. Ces deux algorithmes diffèrent uniquement par le calcul de l'échéance fictive :
- avec TBS-EDeg, on considère que l'énergie dans la batterie à l'instant $t = max(r_k, d_{k-1})$ n'est pas connue et donc la formule de l'échéance fictive pour une tâche φ_k devient :

$$d_k^a = max(r_k^a, d_{k-1}^a) + max(\left\lceil \frac{C_k^a}{U_S} \right\rceil, \left\lceil \frac{E_k^a}{U_{es}.P_r} \right\rceil) \tag{5.8}$$

- avec TBS_{op}-EDeg, l'énergie dans la batterie à l'instant $t = max(r_k, d_{k-1})$ est connue et donc l'échéance fictive pour une tâche φ_k est donnée par :

$$d_k^a = max(r_k^a, d_{k-1}^a) + max(\left\lceil \frac{C_k^a}{U_S} \right\rceil, \left\lceil \frac{\frac{E_k^a}{U_{es}} - E(max(r_k^a, d_{k-1}^a))}{P_r} \right\rceil) \tag{5.9}$$

Que ce soit avec TBS-EDeg ou TBS_{op}-EDeg, le principe est le même : une fois que l'échéance fictive de la tâche apériodique est calculée, celle-ci est ajoutée à la liste des tâches apériodiques prêtes. A chaque instant *t*, la tâche, parmi la liste des tâches périodiques prêtes et la liste des tâches apériodiques prêtes, ayant la plus proche échéance sera choisie pour être exécutée. Celle-ci sera exécutée seulement si la laxité énergétique du système est positive et si l'énergie disponible dans la batterie est suffisante.

Comme les algorithmes TBS-EDeg et TBS_{op}-EDeg ont le même principe et diffèrent seulement par la formule du calcul de l'échéance fictive, ils se résument en un seul algorithme que l'on décrit ci-après.

3.2 Description de l'algorithme

L'algorithme 7 est caractérisé par les données suivantes :

- $\mathcal{T}(t)$, la liste de tâches périodiques prêtes à l'instant *t*.
- $\varphi(t)$, la liste de tâches apériodiques prêtes à l'instant *t*.
- $idle$, un booléen prenant la valeur FAUX quand le processeur est actif et VRAI quand le processeur est inactif.
- $SlackTime(t)$, la laxité temporelle disponible à l'instant t en utilisant l'algorithme EDL(Earliest Deadline as Late as possible).
  C'est le temps disponible pour recharger la batterie sans pour autant compromettre l'exécution des tâches périodiques.

  $$SlackTime(t) = \Omega_t^{EDL}(t, d) \tag{5.10}$$

  où d correspond à la plus petite échéance parmi les jobs périodiques et apériodiques prêts et $\Omega_t^{EDL}(t, d)$ représente le temps creux disponible calculé avec EDL entre l'instant *t* et *d*.
- $SlackEnergy(t)$, la laxité énergétique du système qui représente la quantité d'énergie maximale dans le système qui peut être consommée à l'instant *t*.
  Sa valeur est déterminée en calculant la plus petite laxité énergétique disponible parmi tous les jobs périodiques prêts à l'instant supérieur à *t* et ayant leur échéance inférieure à *d* (i.e. échéance de la tâche

périodique ou apériodique la plus prioritaire à l'instant *t*).

$$SlackEnergy(t) = min(SlackEnergy(\tau_{i,k}, t)) \tag{5.11}$$

La laxité énergétique d'un job $\tau_{i,k}$ ($k^{ème}$ job de la tâche périodique τ_i) à l'instant t est le surplus d'énergie qui peut être consommé entre t et son échéance $d_{i,k}$ et elle est calculée comme suit :

$$SlackEnergy(\tau_{i,k}, t) = E(t) + \int_t^{d_{i,k}} P_r(x)dx - A_{i,k} \tag{5.12}$$

où, l'on représente par :
- $E(t)$, l'énergie restante dans la batterie à l'instant t,
- $P_r(x)$, la puissance reçue à l'instant x,
- $A_{i,k}$, la demande énergétiques des jobs réveillés après $r_{i,k}$ et d'échéance inférieure ou égale à $d_{i,k}$. Donc $A_{i,k} = \sum_{r_j \geq t, d_j \leq d_{i,k}} E_j$,
- E_j, l'énergie requise pour s'exécuter la tâche périodique τ_j.

3.3 Exemple illustratif d'ordonnancement avec TBS-EDeg

On reprend le même exemple que celui avec BG-EDeg.

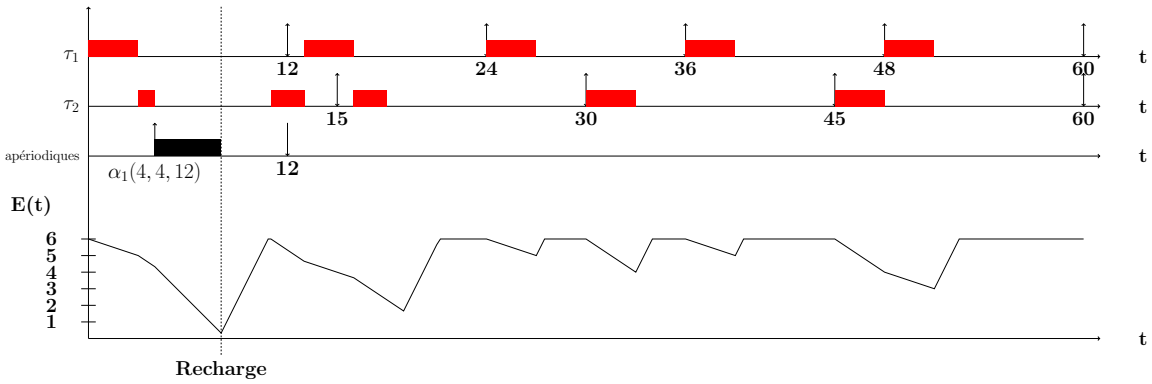


FIGURE 5.2 – Ordonnancement selon $TBS - EDeg$

La Figure 5.2 illustre l'ordonnancement de la configuration $\mathcal{T}$ suivant TBS-EDeg. Dans cet exemple, une tâche apériodique α arrive à *t=4*, son échéance fictive est calculée et lui est assignée, $d_1^a = 12$. Puis cette tâche est ajoutée à la liste des tâches prêtes et elle est exécutée conjointement avec les tâches périodiques. Avec TBS-EDeg, le temps de réponse de α vaut 4 unités de temps. Ceci montre une nette amélioration de 11 unités de temps par rapport à BG-EDeg. De plus son temps de réponse normalisé vaut $\frac{4}{4} = 1$. Ceci signifie ainsi que α a été servie immédiatement.

3.4 Exemple illustratif d'ordonnancement avec TBS_{op}-EDeg

On reprend le même exemple que celui avec BG-EDeg. La Figure 5.3 illustre l'ordonnancement de la configuration $\mathcal{T}$ suivant TBS_{op}-EDeg. Dans cet exemple, une tâche apériodique α_1 arrive à *t=4*, une échéance fictive est calculée et lui est assignée, $d_1^a = 10$. Cette tâche est ajoutée à la liste des tâches prêtes et elle sera exécutée conjointement avec les tâches périodiques.

Dans cet exemple, il n'y a aucun job périodique ou apériodique prêt entre l'intant *t=4* et l'instant *t=12* ayant une échéance inférieure à 12. L'échéance fictive calculée avec TBS_{op}-EDeg vaut 10 et est plus courte que celle calculée avec TBS-EDeg. Que ce soit avec TBS-EDeg ou avec TBS_{op}-EDeg, la tâche est complètement exécutée sans interruption, et le temps de réponse avec TBS_{op}-EDeg est de 4 unités de temps comme celui obtenu avec TBS-EDeg et est meilleur que celui obtenu avec BG-EDeg.

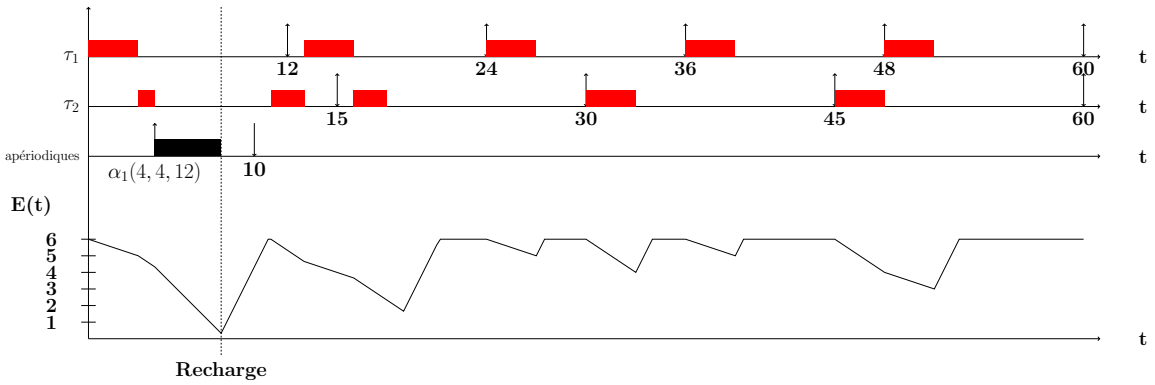


FIGURE 5.3 – Ordonnancement selon $TBS_{op} - EDeg$

4 Evaluation des performances

L'objectif de ce chapitre est d'évaluer comparativement les performances des algorithmes BG-EDeg, TBS-EDeg et TBS_{op}-EDeg par le biais de simulations. Pour cette étude, un générateur de tâches périodiques et un générateur de tâches apériodiques sont implémentés en langage C.

Le générateur de tâches périodiques prend en entrée les paramètres suivants :

- *n* : le nombre de tâches périodiques constituant la configuration,
- *PPCMmax* : le ppcm maximal des périodes des tâches périodiques constituant la configuration,
- U_p : la charge processeur associée aux tâches périodiques,
- $\bar{P}_e$: la puissance moyenne consommée par les tâches périodiques,

En sortie, le générateur génère une configuration de tâches sans pertes $\mathcal{T} = \{\tau_i(C_i, T_i, E_i),\ i = 1 \text{ à } n\}$.

- Les tâches périodiques sont indépendantes, préemptables et à échéances sur requêtes ($D_i = T_i$).
- Les périodes des tâches T_i sont aléatoirement choisies et leur choix dépend du $PPCMmax$ donné en entrée.
- Les durées d'exécution des tâches au pire-cas C_i sont générées aléatoirement et le choix dépend de $U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$.
- Les demandes énergétiques des tâches E_i sont générées aléatoirement et le choix dépend de $\bar{P}_e = \sum_{i=1}^{n} \frac{E_i}{T_i}$.
- Il n'y a aucune corrélation entre E_i et C_i.

Le générateur de tâches apériodiques prend en entrée les paramètres suivant :

- *m* : le nombre de tâches apériodiques générées chaque hyperpériode,
- *H* : la valeur de l'hyperpériode des tâches périodiques,
- U_{ap} : la charge temporelle dédiée aux tâches apériodiques,
- U_{ape} : la charge énergétique dédiée aux tâches apériodiques.

En sortie, ce générateur renvoie une configuration de m tâches apériodiques non critiques $\varphi = \{\varphi_i(r_i^a, C_i^a, E_i^a),\ i = 1 \text{ à } m\}$.

- Les instants de réveil r_i^a sont calculés de façon à ce que la tâche φ_i soit réveillée et exécutée avant la fin de l'hyperpériode.
- Les durées d'exécution des tâches au pire-cas C_i^a sont générées aléatoirement et le choix dépend de U_{ap}.
- Les demandes énergétiques des tâches E_i^a sont générées aléatoirement et le choix dépend de U_{ape}.

Le simulateur consiste à ordonnancer en-ligne une configuration de n tâches périodiques et m tâches apériodiques non critiques selon BG-EDeg, TBS-EDeg et TBS_{op}-EDeg. Nous considérons successivement un système faiblement/moyennement/fortement chargé temporellement par les tâches périodiques, et les critères de performances suivants :

1. **Temps de réponse normalisé moyen** : Le temps de réponse TR_i d'une tâche φ_i représente la durée prise par φ_i depuis son réveil pour être exécutée ($TR_i = f_i^a - r_i^a$). Nous évaluons le temps de réponse normalisé moyen qui représente la moyenne, sur l'ensemble de tâches apériodiques, des durées d'exécution au pire-cas des tâches apériodiques normalisées par rapport aux temps de réponse.

$$Temps\_reponse\ moyen = \frac{\sum_{i=1}^{m} \frac{C_i^a}{f_i^a - r_i^a}}{m} \tag{5.13}$$

   Plus le temps de réponse normalisé est proche de 1, plus le temps de réponse est court par rapport à la durée d'exécution. A contrario, si le temps de réponse normalisé est proche de zéro ceci implique que le temps de réponse de la tâche est très long.

2. **Temps de gigue normalisé moyen** : Le temps de gigue de démarrage représente l'écart entre l'instant de réveil de la tâche apériodique et son début d'exécution ($f_i^a - G_i^a$). Si le temps de gigue d'une tâche φ_i est très proche de son temps de réponse, ceci implique que la tâche a été servie rapidement sans être bloquée. Par conséquent, nous choisissons d'étudier le temps de gigue normalisé moyen qui représente la moyenne, sur l'ensemble de tâches apériodiques, des temps de gigue de démarrage rapportés aux temps de réponse.

$$Temps\_gigue\ moyen = \frac{\sum_{i=1}^{m} \frac{f_i^a - G_i^a}{f_i^a - r_i^a}}{m} \tag{5.14}$$

   Cette métrique permet de visualiser le temps durant lequel la tâche apériodique est bloquée par une autre tâche et attend pour être servie. Elle permet de déterminer quel ordonnanceur choisir quand le résultat fourni après l'exécution d'une tâche apériodique est pertinent et est requis dans les plus brefs délais. **Quand le temps de gigue normalisé affiche une valeur proche de 1 ceci veut dire que la tâche a été immédiatement servie et que le temps de blocage avant exécution est nul.**

Nous présentons dans la suite, les résultats de simulation obtenus suivant différents profils énergétiques et temporels du système. 100 configurations de tâches périodiques et apériodiques différentes sont générées. Chaque configuration de tâches périodiques est constituée de 10 tâches avec un ppcm maximal égal à 3600. De plus, 5 tâches apériodiques sont générées sur chaque hyperpériode.
Les simulations sont effectuées sur 10 hyperpériodes. La batterie est supposée initialement pleine telle que $E(0) = H.P_r$.

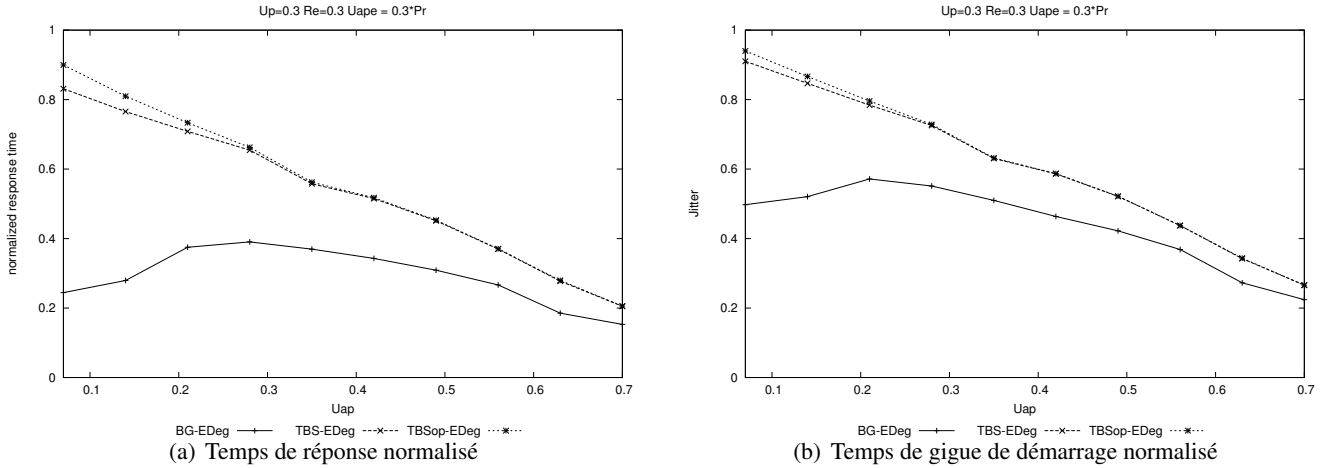
Pour chaque cas étudié, nous faisons varier en abscisse la charge temporelle des tâches apériodiques U_{ap}.

4.1 Système faiblement chargé temporellement

4.1.1 Système faiblement chargé énergétiquement

Dans cette partie, nous considérons que le système est en sous-charge temporelle et en sous-charge énergétique ($U_p = 0.3$ et $R_e = 0.3$). Nous ajoutons à ce système des tâches apériodiques ayant une charge énergétique $U_{ape} = 0.3P_r$. Par conséquent, le système global devient moyennement chargé énergétiquement.

Sur la Figure 5.4(a) et sur la Figure 5.4(b), on remarque que :



(a) Temps de réponse normalisé (b) Temps de gigue de démarrage normalisé

FIGURE 5.4 – $U_p = 0.3$, $R_e = 0.3$, $U_{ape} = 0.3P_r$

- Les temps de réponse des tâches apériodiques normalisés obtenus avec BG-EDeg, TBS-EDeg et TBS_{op}-EDeg diminuent en fonction de la charge temporelle des tâches apériodiques c'est-à-dire que plus la charge temporelle des tâches apériodiques augmente, plus leur temps de réponse est long.

- BG-EDeg offre le temps de réponse le plus long et ce quelle que soit la charge temporelle des tâches apériodiques. Le temps de réponse normalisé varie entre 20% et 40%. Ceci implique que le temps de réponse obtenu avec BG-EDeg représente 2.5 à 5 fois le temps d'exécution moyen des tâches apériodiques.

- La plus grande différence de temps de réponse normalisé est observée entre BG-EDeg et TBS_{op}-EDeg pour $U_{ap} = 7\%$. Ainsi, le temps de réponse est 4 fois plus long avec BG-EDeg qu'avec TBS_{op}-EDeg . Cependant plus la charge temporelle des tâches apériodiques est importante, plus l'écart entre le temps de réponse normalisé obtenu avec BG-EDeg et celui obtenu avec TBS-EDeg ou TBS_{op}-EDeg est petit.

- Quand la charge temporelle des tâches apériodiques est inférieure à 30% ($U_{ap} \leq 30\%$), le meilleur temps de réponse est obtenu avec TBS_{op}-EDeg. Pour $U_{ap} = 7\%$, TBS-EDeg possède un temps de réponse normalisé inférieur de 10% à celui obtenu avec TBS_{op}-EDeg.

- Quand la charge temporelle des tâches apériodiques augmente, TBS_{op}-EDeg et TBS-EDeg offrent des temps de réponse de plus en plus proches.

- Lorsque U_{ap} dépasse 35%, TBS_{op}-EDeg et TBS-EDeg offrent les mêmes performances en termes de temps de réponse et de temps de gigue de démarrage. En effet, C_k^a devient plus important que E_k^a et donc $\frac{C_k^a}{U_S} >> \frac{E_k^a}{U_{es}}$. Par conséquent, le calcul de l'échéance fictive dépend de $\frac{C_k^a}{U_S}$ ce qui implique que l'échéance calculée avec TBS_{op}-EDeg et TBS-EDeg est la même.

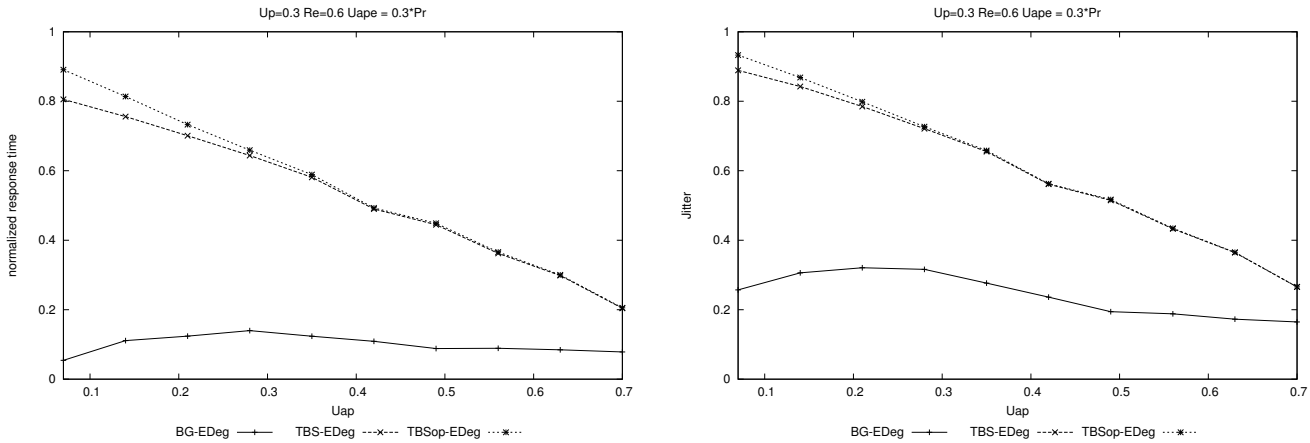
- Quand la charge temporelle des tâches apériodiques est inférieure à 30% ($U_{ap} \leq 30\%$), TBS_{op}-EDeg offre les meilleurs temps de gigue de démarrage. A titre illustratif, un écart environ égal à 5% est observé entre TBS_{op}-EDeg et TBS-EDeg pour $U_{ap} = 7\%$.

- Le temps de gigue normalisé le plus médiocre est observé avec BG-EDeg. Nous observons que pour $U_{ap} = 7\%$, l'écart entre BG-EDeg et TBS_{op}-EDeg vaut environ 40%. Ceci implique qu'avec BG-EDeg, les tâches apériodiques attendent le plus longtemps, parmi les autres ordonnanceurs, pour commencer leur exécution.

- Plus la charge temporelle des tâches apériodiques augmente plus le temps de gigue normalisé diminue ce qui implique que le temps de gigue augmente avec U_{ap} quel que soit l'ordonnanceur étudié.

4.1.2 Système moyennement chargé énergétiquement

Dans cette partie, nous considérons que le système est en sous-charge temporelle et est moyennement chargé énergétiquement ($U_p = 0.3$ et $R_e = 0.6$). Nous ajoutons à ce système des tâches apériodiques ayant une charge énergétique $U_{ape} = 0.3P_r$. Par conséquent, le système global devient fortement chargé énergétiquement.
Sur la Figure 5.5(a) et la Figure 5.5(b), on remarque que :



(a) Temps de réponse normalisé

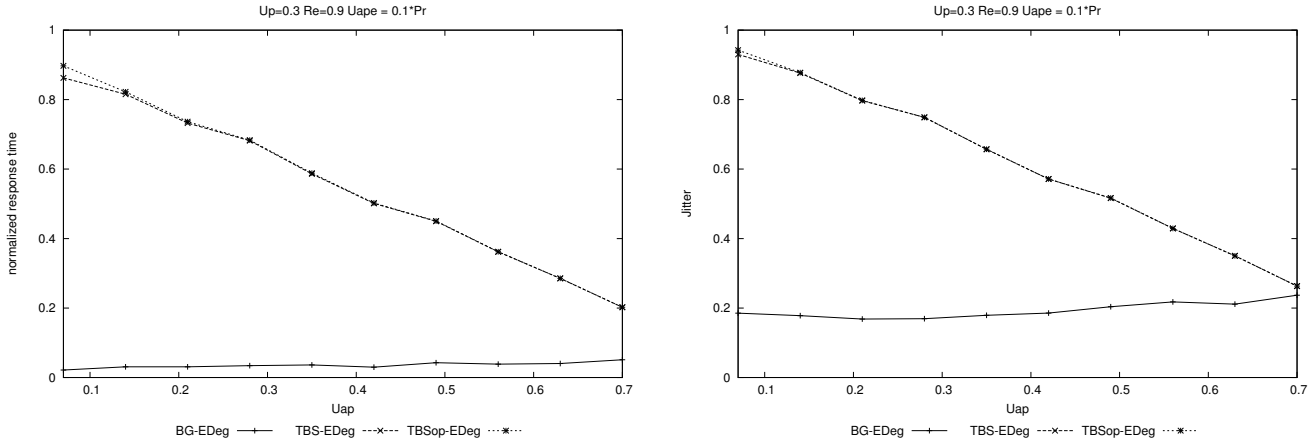
(b) Temps de gigue de démarrage normalisé

FIGURE 5.5 – $U_p = 0.3$, $R_e = 0.6$, $U_{ape} = 0.3P_r$

- Les temps de réponse des tâches apériodiques obtenus avec TBS-EDeg et TBS_{op}-EDeg augmentent avec la charge temporelle des tâches apériodiques.
- BG-EDeg offre le temps de réponse le plus médiocre et ce, quelle que soit la charge temporelle des tâches apériodiques. A titre illustratif, pour $U_{ap} = 7\%$, TBS_{op}-EDeg et TBS-EDeg servent respectivement les tâches apériodiques 9 et 8 fois plus vite que BG-EDeg.
- Quand la charge temporelle des tâches apériodiques est inférieure à 35% ($U_{ap} < 35\%$), les meilleurs temps de réponse et temps de gigue de démarrage sont observés avec TBS_{op}-EDeg. Pour $U_{ap} = 7\%$, TBS_{op}-EDeg sert les tâches apériodiques 1.2 fois plus vite que TBS-EDeg avec un retard de début d'exécution légèrement moins prononcé que celui de TBS-EDeg.
- Plus la charge temporelle des tâches apériodiques augmente plus les performances de TBS_{op}-EDeg et TBS-EDeg en termes de temps de réponse et de temps de gigue de démarrage sont proches pour devenir similaires à partir de $U_{ap} = 35\%$.
- Avec BG-EDeg, le temps de gigue de démarrage est le plus long parmi les ordonnanceurs étudiés. Nous observons que pour $U_{ap} = 7\%$, les tâches apériodiques sont servies avec un retard qui vaut 4 fois le retard observé avec TBS_{op}-EDeg.
- Plus la charge temporelle des tâches apériodiques augmente plus le temps de réponse et temps de gigue augmentent quel que soit l'ordonnanceur étudié. Ceci veut dire que les tâches apériodiques sont bloquées plus longtemps avant de débuter leur exécution.

4.1.3 Système fortement chargé énergétiquement

Dans cette partie, nous considérons que le système est en sous-charge temporelle et est fortement chargé énergétique ($U_p = 0.3$ et $R_e = 0.9$). Nous ajoutons à ce système des tâches apériodiques ayant une charge énergétique $U_{ape} = 0.1P_r$. Par conséquent, le système global devient très chargé énergétiquement.



(a) Temps de réponse normalisé (b) Temps de gigue de démarrage normalisé

FIGURE 5.6 – $U_p = 0.3$, $R_e = 0.9$, $U_{ape} = 0.1P_r$

Nous remarquons sur la Figure 5.6(a) que :
- Le temps de réponse obtenu avec BG-EDeg est très long car le temps de réponse normalisé vaut environ 5%. Ceci implique que le temps de réponse obtenu avec BG-EDeg représente environ 20 fois la durée d'exécution des tâches.
- TBS-EDeg et TBS_{op}-EDeg offrent les meilleurs temps de réponse. Pour une faible charge apériodique $U_{ap} = 7\%$, BG-EDeg prend 18 fois plus de temps que TBS_{op}-EDeg pour servir les tâches apériodiques.
- Avec TBS-EDeg et TBS_{op}-EDeg, les temps de réponse sont très proches avec un écart d'environ 5% pour $U_{ap} = 7\%$ puis ils deviennent similaires à partir de $U_{ap} = 14\%$.

Nous remarquons sur la Figure 5.6(b) que :
- Les tâches apériodiques commencent leur exécution plus rapidement avec TBS-EDeg et TBS_{op}-EDeg comparativement à BG-EDeg, i.e. le temps de gigue est moins élevé.
- Pour une charge apériodique $U_{ap} = 7\%$, avec TBS-EDeg et TBS_{op}-EDeg, les tâches apériodiques commencent leur exécution 4.5 fois plus rapidement qu'avec BG-EDeg. Ceci s'explique par le fait que BG-EDeg n'exécute les tâches apériodiques que s'il n'y a aucun job périodique prêt et que l'énergie disponible est suffisante pour garantir qu'au prochain réveil d'un job périodique, la batterie sera pleine.
- Plus la charge temporelle des tâches apériodiques est grande, plus le temps de gigue augmente avec TBS-EDeg et TBS_{op}-EDeg et tend vers celui obtenu avec BG-EDeg.

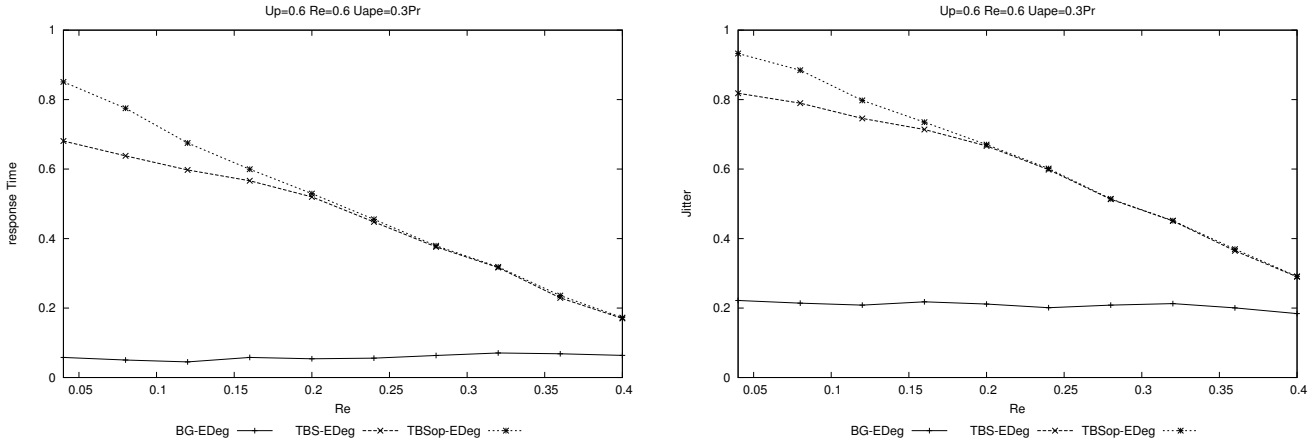
4.2 Système moyennement chargé temporellement

4.2.1 Système moyennement chargé énergétiquement

Dans cette partie, nous considérons que le système est moyennement chargé temporellement et énergétiquement ($U_p = 0.6$ et $R_e = 0.6$). Nous ajoutons à ce système des tâches apériodiques avec une charge énergétique $U_{ape} = 0.3P_r$. Par conséquent, le système global devient fortement chargé énergétiquement.

Sur la Figure 5.7(a) et 5.7(b), nous remarquons que :

- BG-EDeg offre des temps de réponse et des temps de gigue de démarrage plus longs que ceux obtenus avec TBS-EDeg et TBS_{op}-EDeg.
- TBS-EDeg et TBS_{op}-EDeg offrent les meilleurs temps de réponse et temps de gigue de démarrage quelle que soit la charge temporelle des tâches apériodiques U_{ap}. A titre illustratif pour une faible charge temporelle des tâches apériodiques, $U_{ap} = 20\%$, le temps de réponse obtenu avec BG-EDeg vaut 14 fois celui obtenu avec TBS-EDeg et TBS_{op}-EDeg.
- Pour $U_{ap} < 20\%$, TBS_{op}-EDeg affiche le meilleur temps de réponse. On observe un écart d'environ 20% avec TBS-EDeg pour $U_{ap} = 4\%$. Cependant les temps de réponse obtenus avec TBS_{op}-EDeg et



(a) Temps de réponse normalisé (b) Temps de gigue de démarrage normalisé

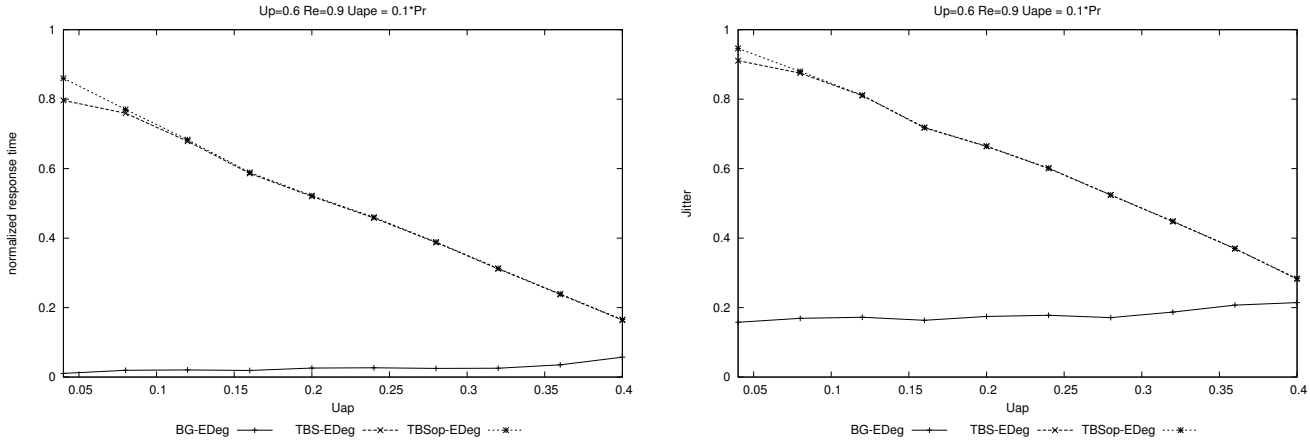
FIGURE 5.7 – $U_p = 0.6$, $R_e = 0.6$, $U_{ape} = 0.3P_r$

TBS-EDeg deviennent très proches pour devenir similaires à partir de $U_{ap} = 20\%$.

- Le temps de gigue de démarrage est le meilleur avec TBS_{op}-EDeg pour $U_{ap} < 20\%$.
- Pour $U_{ap} = 20\%$, TBS_{op}-EDeg et TBS-EDeg offrent des temps de gigue de démarrage 4 fois plus courts que ceux observés avec BG-EDeg.
- Quand le système global est très chargé temporellement, le temps de gigue de démarrage obtenu avec tous les algorithmes sont très proches. Cependant TBS_{op}-EDeg et TBS-EDeg continuent à offrir un meilleur temps de réponse qui est 4 fois inférieur à celui obtenu avec BG-EDeg.

4.2.2 Système fortement chargé énergétiquement

Dans cette partie, nous considérons que le système est moyennement chargé temporellement et est fortement chargé énergétiquement ($U_p = 0.6$ et $R_e = 0.9$). Nous ajoutons à ce système des tâches apériodiques ayant une charge énergétique $U_{ape} = 0.1P_r$. Par conséquent, le système devient très fortement chargé énergétiquement.



(a) Temps de réponse normalisé (b) Temps de gigue de démarrage normalisé

FIGURE 5.8 – $U_p = 0.6$, $R_e = 0.9$, $U_{ape} = 0.1P_r$

Sur les Figures 5.8(a) et 5.8(b), nous remarquons que :

- BG-EDeg offre des temps de réponse et des temps de gigue de démarrage très longs : le temps de réponse normalisé est proche de zéro quelle que soit la charge temporelle des tâches apériodiques U_{ap}.

- $U_{ap} < 10\%$, TBS_{op}-EDeg affiche le meilleur temps de réponse. On observe un écart d'environ 5% avec TBS-EDeg pour $U_{ap} = 4\%$. Cependant les temps de réponse obtenus avec TBS_{op}-EDeg et TBS-EDeg s'approchent pour devenir similaires à partir de $U_{ap} \geq 10\%$.

- Le temps de gigue de démarrage est le meilleur avec TBS_{op}-EDeg pour $U_{ap} < 10\%$. Au-delà de $U_{ap} = 10\%$, TBS_{op}-EDeg et TBS-EDeg offrent les mêmes temps de gigue de démarrage qui sont plus courts que ceux observés avec BG-EDeg.

- Pour un système moyennement chargé temporellement tel que la charge temporelle globale vaut $U = U_p + U_{ap} \cong 72\%$, BG-EDeg a un temps de réponse qui est 65 fois plus long que TBS_{op}-EDeg et TBS-EDeg et un temps de gigue de démarrage 4 fois plus long que TBS_{op}-EDeg et TBS-EDeg.

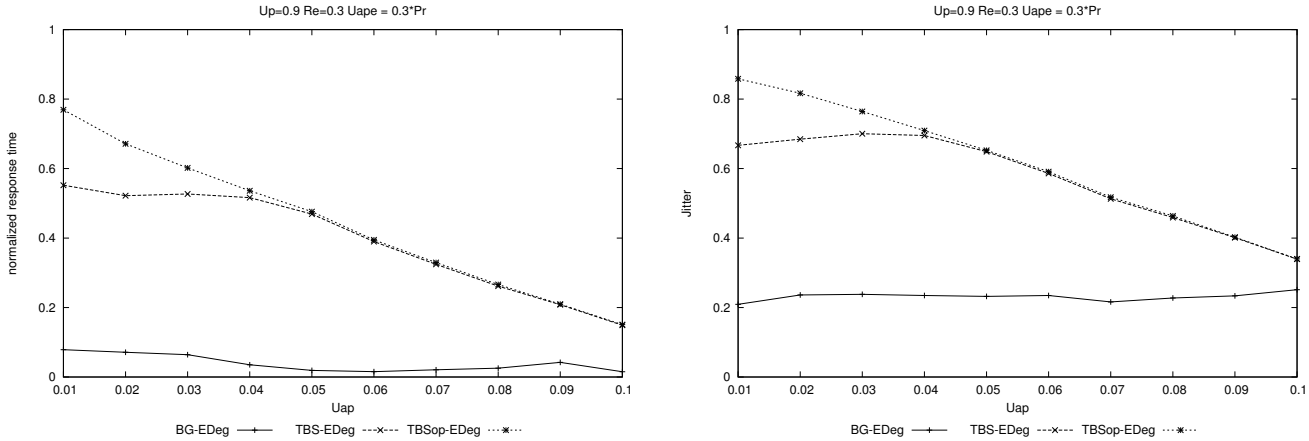
- TBS-EDeg et TBS_{op}-EDeg offrent les meilleurs temps de réponse et temps de gigue de démarrage quelle que soit U_{ap}.

- Plus la charge temporelle des tâches apériodiques augmente, plus TBS-EDeg et TBS_{op}-EDeg offrent des temps de réponse et des temps de gigue de démarrage de plus en plus longs.

4.3 Système fortement chargé temporellement

4.3.1 Système faiblement chargé énergétiquement

Dans cette partie, nous considérons que le système est fortement chargé temporellement et est faiblement chargé énergétiquement ($U_p = 0.9$ et $R_e = 0.3$). Nous supposons que des tâches apériodiques arrivent avec une charge énergétique $U_{ape} = 0.3P_r$. Par conséquent, le système devient moyennement chargé énergétiquement.



(a) Temps de réponse normalisé

(b) Temps de gigue de démarrage normalisé

FIGURE 5.9 – $U_p = 0.9$, $R_e = 0.3$, $U_{ape} = 0.3P_r$

Sur la Figure 5.9(a) et 5.9(b), nous remarquons que :

- BG-EDeg offre des temps de réponse et des temps de gigue de démarrage très longs. A contrario, TBS-EDeg et TBS_{op}-EDeg offrent les meilleurs temps de réponse et temps de gigue de démarrage quelle que soit U_{ap}.

- Pour $U_{ap} < 5\%$, TBS_{op}-EDeg affichent le meilleur temps de réponse. Pour $U_{ap} = 5\%$, on observe une différence d'environ 20% entre les temps de réponse normalisés obtenus avec TBS_{op}-EDeg et TBS-EDeg. TBS_{op}-EDeg est 1.5 plus rapide que TBS-EDeg pour exécuter les tâches apériodiques.

- Les temps de réponse obtenus avec TBS_{op}-EDeg et TBS-EDeg s'approchent pour devenir similaires à partir de $U_{ap} = 5\%$.

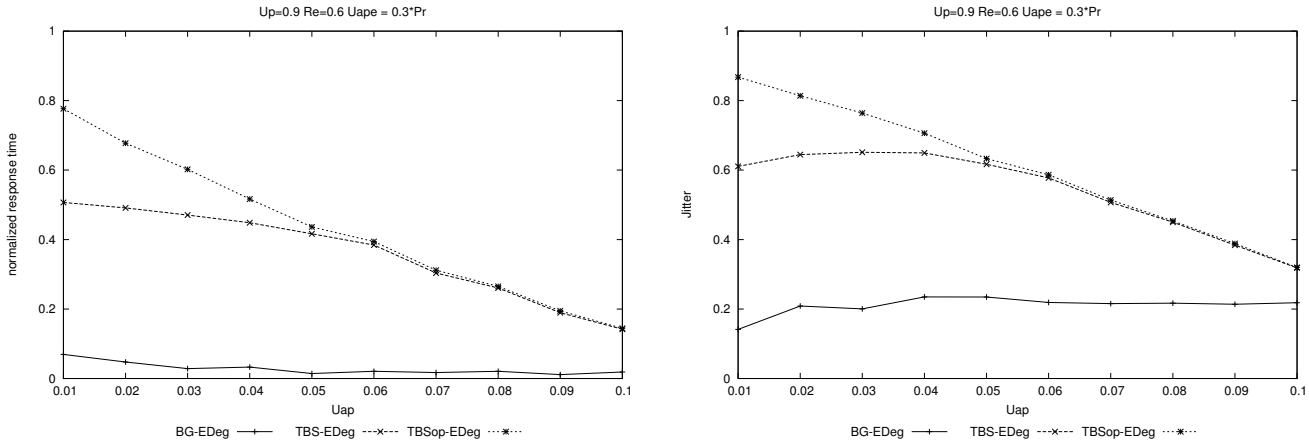
- Le temps de gigue de démarrage observé est le meilleur avec TBS_{op}-EDeg pour $U_{ap} < 5\%$.

- Au delà de $U_{ap} = 0.05$, TBS_{op}-EDeg et TBS-EDeg offrent les mêmes temps de gigue de démarrage qui sont plus courts que ceux observés avec BG-EDeg.

- Plus la charge temporelle des tâches apériodiques augmente, plus TBS-EDeg et TBS_{op}-EDeg offrent des temps de réponse et des temps de gigue de démarrage longs.

4.3.2 Système moyennement chargé énergétiquement

Dans cette partie, nous considérons que le système est fortement chargé temporellement et moyennement chargé énergétiquement ($U_p = 0.9$ et $R_e = 0.6$). Nous supposons que des tâches apériodiques arrivent avec une charge énergétique $U_{ape} = 0.3P_r$. Par conséquent, le système global devient fortement chargé énergétiquement.
Sur la Figure 5.10(a) et 5.10(b), nous remarquons que :



(a) Temps de réponse normalisé

(b) Temps de gigue de démarrage normalisé

FIGURE 5.10 – $U_p = 0.9$, $R_e = 0.6$, $U_{ape} = 0.3P_r$

- BG-EDeg offre des temps de réponse et des temps de gigue de démarrage très longs alors que TBS-EDeg et TBS_{op}-EDeg offrent des temps de réponse et temps de gigue de démarrage beaucoup plus rapides quelle que soit la charge temporelle des tâches apériodiques U_{ap}.

- $U_{ap} < 6\%$, TBS_{op}-EDeg affiche le meilleur temps de réponse. Pour $U_{ap} = 6\%$, TBS_{op}-EDeg offre un temps de réponse 1.5 fois plus rapide que celui obtenu avec TBS-EDeg.

- Le meilleur temps de gigue de démarrage est observé avec TBS_{op}-EDeg pour $U_{ap} < 6\%$.

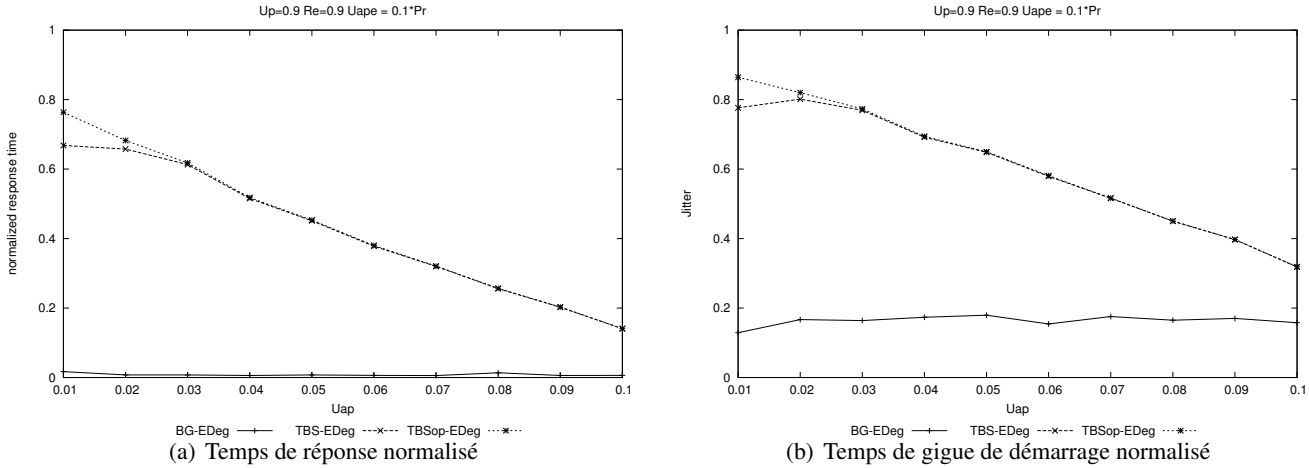
- Au delà de $U_{ap} = 6\%$, TBS_{op}-EDeg et TBS-EDeg offrent les mêmes temps de réponse et temps de gigue de démarrage qui sont toujours plus courts que ceux observés avec BG-EDeg.

- Plus la charge temporelle des tâches apériodiques augmente, plus TBS-EDeg et TBS_{op}-EDeg offrent des temps de réponse et des temps de gigue de démarrage longs.

4.3.3 Système fortement chargé énergétiquement

Dans cette partie, nous considérons que le système est fortement chargé temporellement et énergétiquement ($U_p = 0.9$ et $R_e = 0.9$). Nous supposons que des tâches apériodiques arrivent avec une charge énergétique $U_{ape} = 0.1P_r$. Par conséquent, le système global devient très fortement chargé énergétiquement.
Sur les Figures 5.11(a) et 5.11(b), nous remarquons que :

- BG-EDeg offre des temps de réponse et des temps de gigue de démarrage très longs : les temps de réponse normalisés sont quasiment nuls.



(a) Temps de réponse normalisé

(b) Temps de gigue de démarrage normalisé

FIGURE 5.11 – $U_p = 0.9$, $R_e = 0.9$, $U_{ape} = 0.1P_r$

- TBS-EDeg et TBS_{op}-EDeg sont beaucoup plus performants que BG-EDeg en termes de temps de réponse et de temps de gigue de démarrage quelle que soit la charge temporelle U_{ap}. A titre illustratif, pour $U_{ap} = 3\%$, TBS-EDeg et TBS_{op}-EDeg offrent un temps de réponse 60 fois plus court que BG-EDeg et un temps de gigue de démarrage 4 fois plus court que BG-EDeg.
- Si $U_{ap} < 3\%$, TBS_{op}-EDeg affiche le meilleur temps de réponse et le meilleur temps de gigue de démarrage.
- On observe sur la Figure 5.11(a), que TBS_{op}-EDeg est 12% plus rapide que TBS-EDeg pour $U_{ap} = 3\%$. Cependant les temps de réponse obtenus avec TBS_{op}-EDeg et TBS-EDeg sont de plus en plus proches et deviennent similaires à partir de $U_{ap} = 3\%$.
- Le meilleur temps de gigue de démarrage est observé avec TBS_{op}-EDeg pour $U_{ap} < 3\%$ (cf. Figure 5.11(b)). Au delà de $U_{ap} = 3\%$, TBS_{op}-EDeg et TBS-EDeg offrent les mêmes temps de gigue de démarrage qui sont beaucoup plus courts que ceux observés avec BG-EDeg.
- Plus la charge temporelle des tâches apériodiques augmente, plus TBS-EDeg et TBS_{op}-EDeg offrent des temps de réponse et des temps de gigue de démarrage plus longs mais qui restent meilleurs que ceux obtenus avec BG-EDeg. Pour $U_{ap} = 10\%$, les temps de réponse obtenus avec TBS_{op}-EDeg et TBS-EDeg sont 15 fois moins longs que ceux obtenus avec BG-EDeg.

5 Synthèse

A partir de l'étude de simulation faite nous pouvons conclure que :

- Avec BG-EDeg, les performances sont médiocres en termes de temps de réponse et de temps de gigue de démarrage quelle que soit la charge temporelle ou énergétique. En outre, si les charges énergétiques et temporelles du système augmentent, le temps de réponse et le temps de gigue de démarrage s'allongent. L'avantage avec BG-EDeg réside dans sa simplicité de réalisation et dans le fait qu'il sollicite moins souvent le calcul de la laxité énergétique que TBS-EDeg et TBS_{op}-EDeg.

- TBS_{op}-EDeg et TBS-EDeg présentent de meilleures performances que BG-EDeg en termes de temps de réponse et de temps de gigue de démarrage.

- Pour un système global faiblement chargé temporellement et énergétiquement, TBS_{op}-EDeg et TBS-EDeg ont des temps de réponse 4 fois plus courts et des temps de gigue de démarrage 2 fois plus courts c'est-à-dire que les tâches apériodiques commencent et finissent leur exécution plus rapidement qu'avec BG-EDeg.

De plus quand le système devient de plus en plus chargé les performances ont tendance à se dégrader avec tous les algorithmes. Cependant, TBS_{op}-EDeg et TBS-EDeg sont toujours plus performants que BG-EDeg.

- TBS_{op}-EDeg et TBS-EDeg présentent les mêmes performances en termes de temps de réponse et de temps de gigue de démarrage quand la charge temporelle des apériodiques U_{ap} est plus significative par rapport à la charge énergétique des apériodiques U_{ape}.

- Pour une charge temporelle des tâches apériodiques faible par rapport à la charge énergétique des tâches apériodiques, les échéances fictives calculées avec TBS_{op}-EDeg et TBS-EDeg sont différentes car le calcul de l'échéance est influencé par la contrainte énergétique. Par conséquent, nous pouvons constater que TBS_{op}-EDeg offre de meilleurs temps de réponse et temps de gigue de démarrage que TBS-EDeg car les échéances fictives calculées avec TBS_{op}-EDeg sont plus courtes que celle calculées avec TBS-EDeg.

- TBS_{op}**-EDeg est l'ordonnanceur le plus performant parmi les ordonnanceurs étudiés dans le cas d'un système faiblement à moyennement chargé temporellement et/ou énergétiquement. Cependant, il présente un inconvénient : à chaque calcul de l'échéance fictive il suppose que l'on doit connaitre le niveau d'énergie dans la batterie à l'instant de calcul**.

Nous résumons l'étude faite par les tableaux 5.1 et 5.2. Dans ces tableaux, un "*" représente les bonnes performances d'un algorithme d'ordonnancement par rapport à un critère donné.

Le tableau 5.1 étudie le cas d'un système global faiblement et/ou moyennement chargé énergétiquement et temporellement.

Critères	BG-EDeg	TBS-EDeg	TBS_{op}-EDeg
Complexité algorithmique	***	**	*
Surcoût de calcul	***	*	*
Temps de réponse	*	**	***
Temps de blocage avant de commencer l'exécution	*	**	***

TABLE 5.1 – Comparatif de performances d'un système moyennement chargé.

Le tableau 5.2 étudie le cas d'un système global fortement chargé énergétiquement et temporellement.

Critères	BG-EDeg	TBS-EDeg	TBS_{op}-EDeg
Complexité algorithmique	***	**	*
Surcoût de calcul	***	*	*
Temps de réponse	*	**	**
Temps de blocage avant de commencer l'exécution	*	**	**

TABLE 5.2 – Comparatif de performances d'un système fortement chargé.

6 Conclusion

Dans ce chapitre, nous avons dans un premier temps décrit de nouveaux ordonnanceurs BG-EDeg, TBS-EDeg et TBS_{op}-EDeg. Ces algorithmes ont été conçus pour traiter le problème de l'ordonnancement conjoint de tâches périodiques et apériodiques en garantissant le respect des contraintes temporelles et énergétiques des tâches périodiques. Notre objectif étant de minimiser le temps de réponse des tâches apériodiques, les résultats de simulation nous ont permis de vérifier que TBS-EDeg et TBS_{op}-EDeg sont les plus performants en termes de temps de réponse et temps de gigue de démarrage. BG-EDeg, a le seul avantage d'être simple à réaliser, mais il affiche des temps de réponse et de gigue de démarrage très longs qui augmentent quand le système est de plus en plus chargé temporellement ou énergétiquement.

Dans le chapitre suivant, nous allons présenter de nouveaux serveurs qui permettent de tolérer des pertes au niveau des tâches périodiques dans le but minimiser le temps de réponse des tâches apériodiques non critiques dans des systèmes en surcharges temporelle et/ou énergétique.

Algorithm 7 TBS-EDeg

t : le temps courant
$\mathcal{T}(t)$: liste des tâches périodiques prêtes à l'instant *t*
$\varphi(t)$: liste des tâches apériodiques prêtes à l'instant *t*
$idle$: état du processeur
ex : tâche périodique ou apériodique sélectionnée pour être exécutée
while VRAI **do**
  $ex \leftarrow$ Null
  $idle \leftarrow$ VRAI
  if (une tâche apériodique φ_i arrive) **then**
    Calculer une échéance fictive et la rajouter aux caractéristiques de φ_i
    Ajouter φ_i dans la liste $\varphi(t)$
  end if
  if ($\mathcal{T}(t) \neq \phi$) OU ($\varphi(t) \neq \phi$) **then**
    Soit ex le job parmi la liste $\mathcal{T}(t)$ et $\varphi(t)$ ayant la plus proche échéance
  end if
  if ($E(t) > E_{min}$) **then**
    Calculer $SlackEnergy(t)$
    if ($SlackEnergy(t) \geq 0$) **then**
      $idle \leftarrow$ FAUX
    else
      $SlackEnergy(t) < 0$ *
      if (E(t)=E_{max}) **then**
        $idle \leftarrow$ FAUX
      else
        $E_{min} < E(t) < E_{max}$ *
        Calculer $SlackTime(t)$
        if ($SlackTime(t) = 0$) **then**
          $idle \leftarrow$ FAUX
        end if
      end if
    end if
  end if
  if ($idle$ = FAUX and $SlackEnergy(t) \geq 0$) **then**
    if ($ex \neq Null$) **then**
      Exécute ex
    end if
  end if
end while

Chapitre 6

Ordonnancement de tâches apériodiques avec gestion de surcharge

Sommaire

Dans ce chapitre, notre objectif est d'exploiter les pertes autorisées au niveau des tâches périodiques dans le but de minimiser le temps de réponse des tâches apériodiques non critiques. Dans un premier temps, nous proposons de nouveaux serveurs de tâches apériodiques qui se basent sur les algorithmes Green-RTO et Green-BWP et les serveurs bien connus BG et TBS. Dans un deuxième temps, nous rapportons les résultats d'une étude comparative des performances des différents algorithmes proposés.

1 Terminologie et modèles de tâches

Nous considérons un système temps réel ferme muni d'un processeur qui est alimenté par une batterie rechargeable. Cette batterie est rechargée par une énergie ambiante. Nous considérons dans nos travaux que la puissance reçue par l'environnement est constante. Et nous nous intéressons au problème de l'ordonnancement conjoint de tâches périodiques et apériodiques. Par la suite nous présentons les modèles de tâches considérés.

1.1 Modèle de tâches périodiques

Nous considérons une configuration $\mathcal{T}^*$ de tâches périodiques temps réel fermes au sens Skip-Over dans laquelle chaque tâche τ_i est caractérisée par :

- r_i : sa date de réveil,

- C_i : sa durée d'exécution au pire cas (WCET),
- D_i : son échéance relative,
- d_i : son échéance absolue,
- s_i : son paramètre de pertes, $2 \leq s_i \leq \infty$
- T_i : sa période,
- E_i : sa demande énergétique au pire cas (WCEC).

Les tâches périodiques sont indépendantes, synchrones, profondément rouges et à échéances sur requêtes ($D_i = T_i$).

1.2 Modèle de tâches apériodiques non critiques

Nous considérons de plus une configuration de tâches apériodiques non critiques φ dans laquelle une tâche apériodique φ_i est caractérisée par :

- r_i^a : sa date d'arrivée
- C_i^a : sa durée d'exécution au pire cas
- E_i^a : sa demande énergétique

A partir des caractéristiques propres à chaque tâche apériodique citées ci-dessus, nous nous intéressons aux variables suivantes :

- G_i^a : sa gigue de démarrage (ou Jitter en anglais)
- f_i^a : sa date de fin d'exécution
- TR_i, le temps de réponse : $TR_i = f_i^a - r_i^a$,
- TG_i, le temps de gigue de démarrage : $TG_i = G_i^a - r_i^a$,
- TRn_i, le temps de réponse normalisé : $TRn_i = \frac{C_i}{TR_i}$,
- TGn_i, le temps de gigue de démarrage normalisé : $TGn_i = \frac{TG_i}{f_i^a - r_i^a}$,

Les tâches apériodiques sont rangées dans une liste selon l'ordre $FIFO$.

Par la suite, nous proposons différents algorithmes appelés serveurs permettant de minimiser le temps de réponse des tâches apériodiques tout en essayant d'optimiser la QoS observée au niveau des tâches périodiques constituant le système.

2 Méthodes basées sur l'algorithme Green-RTO

2.1 BG-Green-RTO

2.1.1 Principe de fonctionnement

Nous proposons un nouvel algorithme d'ordonnancement nommé BG-Green-RTO qui est basé sur le principe de l'algorithme BG-EDeg que nous avons décrit dans le chapitre 5. Les tâches périodiques sont ordonnancées selon Green-RTO, c'est-à-dire que seuls les jobs rouges sont exécutés et les jobs bleus sont systématiquement rejetés.

Les jobs rouges sont toujours les plus prioritaires par rapport aux tâches apériodiques. Ainsi, à un instant *t*, s'il n'y a aucun job rouge à l'état prêt et s'il existe des tâches apériodiques en attente, une tâche apériodique est choisie selon *FIFO* pour être exécutée. Cette tâche ne débutera son exécution que si elle ne compromet pas l'exécution de jobs rouges futurs. La tâche apériodique est partiellement ou totalement exécutée selon l'énergie disponible de sorte que la batterie soit toujours pleine au prochain réveil de futurs jobs rouges.

2.1.2 Exemple illustratif avec BG-Green-RTO

On considère une configuration de tâches périodiques temps réel fermes à échéances sur requêtes telle que $\mathcal{T}$=$\tau_i(C_i, D_i, T_i, s_i, E_i)$ avec $\tau_1(3, 12, 12, 2, 7)$ et $\tau_2(3,\ 15,\ 15,\ 2,\ 8)$. La Figure 6.1 illustre l'ordonnancement de la configuration $\mathcal{T}$ suivant BG-Green-RTO sur la première sous-hyperpériode H=$PPCM(T_1,\ T_2) = 60$ (l'hyperpériode $H^* = PPCM(T_1 s_1,\ T_2 s_2) = 120$).

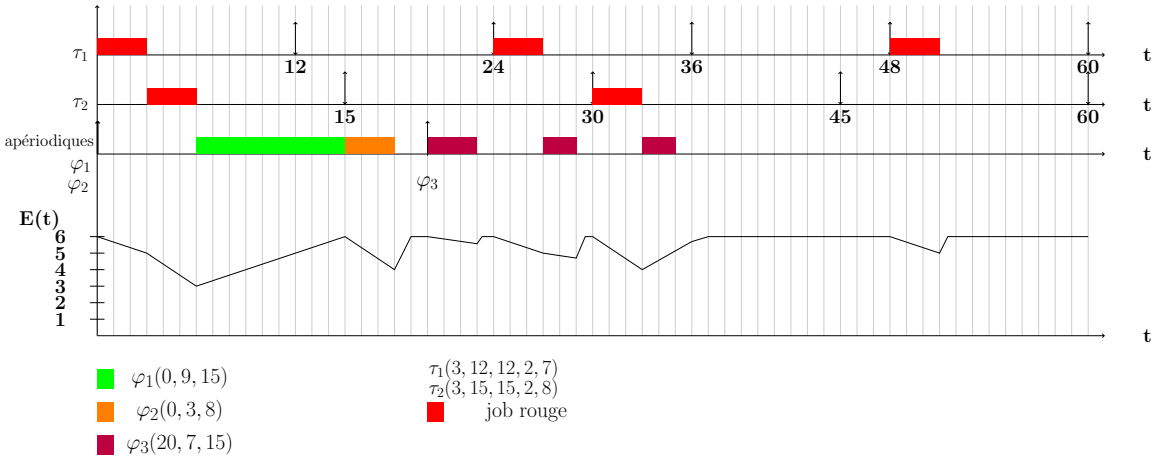


FIGURE 6.1 – Ordonnancement selon BG-Green-RTO

On suppose dans l'exemple que $P_r = 2$ et que la batterie est initialement pleine telle que $E(0) = \frac{H.P_r}{20} = 6$. $U_p = 0.45$ et $U_e = 0.54$. Donc le système considéré n'est pas surchargé temporellement et énergétiquement. De plus, $U_p^* = 0.4$ et $U_e^* = 0.45$.
Nous supposons que trois tâches apériodiques arrivent pendant la première sous-hyperpériode H : $\mathcal{T}$=$\{\varphi_i(r_i^a,\ C_i^a,\ E_i^a)\}$ = $\{\varphi_1(0,\ 9,\ 15),\ \varphi_2(0,\ 3,\ 8),\ \varphi_3(20, 7, 15)\}$. La charge temporelle apériodique est alors $U_{ap} = \frac{9+3+7}{120} = 0.15$ et la charge énergétique des tâches apériodiques est $U_{ape} = \frac{15+8+15}{120} = 0.158.P_r$. Nous notons que nous utiliserons cet exemple pour tous les algorithmes étudiés dans ce chapitre.

φ_1 et φ_2 arrivent à l'instant *t=0*. Comme les jobs rouges $\tau_{1,1}$ et $\tau_{2,1}$ sont prêts, ils sont exécutés selon Green-RTO. A l'instant *t=6*, il n'y a aucun job rouge prêt. Tant que l'énergie disponible jusqu'au prochain réveil d'un job rouge est positive et tant qu'il n'y a pas de job rouge prêt, on peut exécuter des tâches apériodiques.

L'énergie disponible jusqu'au prochain réveil d'un job rouge, $r_{min} = 24$, est calculée ainsi : $E_{disp}(6) = E(6)+(r_{min}-6)P_r-C = 3+36-6 = 33$. Comme $E_{disp}(6)$ est positive, φ_1 commence son exécution à l'instant *t=6*, et continue à s'exécuter tant que l'énergie disponible est positive et qu'aucun job rouge n'est réveillé. Elle finit son exécution à l'instant *t=15*. Comme aucun job rouge n'est prêt et qu'il y a une tâche apériodique en attente, l'énergie disponible est évaluée. Celle-ci est positive. Par conséquent, φ_2 commence son exécution à l'instant *t=15* et finit son exécution à l'instant *t=18*. A l'instant *t=20*, φ_3 arrive et l'énergie disponible est positive ($E_{disp}(20) = 8$). φ_3 commence donc son exécution. Cependant à l'instant *t=23*, l'énergie disponible n'est pas suffisante pour continuer l'exécution de φ_3 et donc celle-ci arrête son exécution. Pendant une unité de temps le processeur est inactif, et la batterie se recharge et est pleine à l'instant *t=24*. A l'instant *t=24*, le job rouge $\tau_{1,2}$ s'exécute pendant 3 unités de temps. A l'instant *t=26*, $E_{disp}(26)$ est à nouveau calculée pour s'assurer que la batterie sera pleine au réveil de $\tau_{2,2}$. Comme $E_{disp}(26)$ est positive, φ_3 continue son exécution. Dans cet exemple, φ_1, φ_2 et φ_3 ont des temps de réponse respectifs de 15, 18 et 15 unités de temps. Si l'on rapporte les durées d'exécution aux temps de réponse des tâches apériodiques, les temps de réponse normalisés obtenus avec φ_1, φ_2 et φ_3 sont respectivement de 0.6, 0.16 et 0.46. Ceci montre que malgré le fait que le système soit moyennement chargé temporellement et énergétiquement, les temps de réponse sont relativement longs.

2.2 Algorithmes basés sur le serveur TBS

2.2.1 Principe de fonctionnement

Dans cette section, nous présentons TBS-Green-RTO et TBS_{op}-Green-RTO qui sont deux algorithmes basés sur le principe du serveur *TBS* [SB94,CB97] permettant d'ordonnancer conjointement les jobs rouges des tâches périodiques et les tâches apériodiques non critiques. L'idée est d'exploiter les temps creux gagnés par l'abandon des jobs bleus afin de minimiser les temps de réponse des tâches apériodiques. Les tâches apériodiques non critiques deviennent des tâches apériodiques à contraintes strictes en leur affectant une échéance fictive. En effet, à chaque fois qu'une tâche apériodique survient, une échéance fictive est calculée de sorte de ne pas compromettre l'exécution de jobs rouges dans le respect de leurs échéances. Ensuite, la tâche apériodique est ordonnancée conjointement aux jobs rouges des tâches périodiques selon l'algorithme *EDF*.

2.2.2 Définitions et tests d'ordonnançabilité

2.2.2.1 Point de vue temporel

Des travaux établis par Caccamo et Buttazo [CB97] reprennent les travaux de Spuri et Buttazo [SB94] mais dans un système à contraintes temporelles fermes. Ces auteurs se sont intéressés à la minimisation du temps de réponse des tâches apériodiques par l'utilisation des temps libérés par les jobs bleus non exécutés. Ainsi Cacamo et Buttazo ont proposé une condition suffisante d'ordonnançabilité énoncée dans le théorème suivant :

Théorème 2.1 *[CB97] Soient une configuration de n tâches périodiques, avec un facteur d'utilisation équivalent U_p^*, et une configuration de tâches apériodiques servies par un serveur TBS, avec un facteur d'utilisation du serveur U_S. Le système global contenant un ensemble hybride de tâches est ordonnançable par RTO ou BWP si :*

$$U_p^* + U_S \leq 1 \tag{6.1}$$

Le théorème 2.1 garantit une largeur de bande temporelle minimale $U_{Smin} = 1 - U_p^*$ pendant laquelle peuvent s'exécuter les tâches apériodiques. En fait, une largeur de bande CPU additionnelle $U_p - U_p^*$ est créée par l'abandon de jobs bleus et peut être utilisée pour servir les tâches apériodiques. En effet, les pertes induisent des temps creux discontinus ou des temps creux uniformément distribués sur la séquence d'ordonnancement des tâches périodiques. Or, les temps creux discontinus créent des trous dans la séquence d'ordonnancement qui peuvent (mais pas toujours) être exploités par une tâche apériodique. Par conséquent, si une tâche apériodique survient dans un des temps creux, elle peut exploiter une largeur de bande temporelle supérieure à $1 - U_p^*$. Ainsi, une largeur de bande temporelle maximale au delà de laquelle la séquence n'est plus ordonnançable est donnée par le théorème suivant :

Théorème 2.2 *[CB97] Soient une configuration de n tâches périodiques fermes avec un facteur d'utilisation équivalent du processeur U_p^*, et une configuration de tâches apériodiques servies par un serveur TBS avec un facteur d'utilisation du processeur U_S. Une condition nécessaire à l'ordonnançabilité du système entier s'exprime comme suit :*

$$U_S \leq U_{Smax} \tag{6.2}$$

où $U_{Smax} = 1 - U_p + \sum_{i=1}^{n} \frac{C_i}{T_i.s_i}$

Dans le cas où U_{Smin} = U_{Smax}, alors les pertes sont toutes exploitées par les tâches apériodiques.

La condition $U_{Smin} < U_{Smax}$ implique que les pertes ne sont pas uniformément distribuées. Le temps $U_{Smax} - U_{Smin}$ ne pouvant pas être exploité par les tâches apériodiques, ces temps peuvent être utilisés pour servir les jobs bleus. Cependant, Green-RTO ne tente pas d'exécuter les jobs bleus. Par contre, Green-BWP aura l'avantage d'utiliser ces temps pour l'exécution de jobs bleus et d'améliorer la QoS observée au niveau des tâches périodiques constituant le système.

Caccamo et Buttazo ont montré que la largeur de bande additionnelle engendrée par les sauts de jobs bleus permet d'avancer l'exécution des tâches apériodiques. La formule permettant le calcul de l'échéance fictive en

fonction de sa largeur de bande CPU, U_S, est toujours donnée par [CB97] :

$$d_k^a = max(r_k^a, d_{k-1}^a) + \frac{C_k^a}{U_s} \tag{6.3}$$

où,
C_k^a est la durée d'exécution de la requête apériodique courante,
U_S est le facteur d'utilisation du serveur tel que $U_S \leq 1 - U_p^*$,
Par définition, $d_0^a = 0$.

Par la suite, nous avons proposé des théorèmes et formules qui étendent ces résultats en prenant en compte la contrainte énergétique.

2.2.2.2 Point de vue énergétique

Dans un premier temps, nous proposons une limitation de la consommation énergétique des tâches apériodiques dans le théorème suivant :

Théorème 2.3 *Soient une configuration de n tâches périodiques temps réel fermes, avec un facteur d'utilisation énergétique équivalent U_e^*, et une configuration de tâches apériodiques servies par un serveur TBS, avec un facteur d'utilisation énergétique du serveur U_{es}. Dans chaque intervalle de temps $[t_1, t_2]$, considérons que E_{ape} est la demande totale d'énergie requise par les tâches apériodiques ayant leur date d'arrivée supérieure ou égale à t_1 et leur date d'échéance inférieure ou égale à t_2, alors il est nécessaire que :*

$$E_{ape} \leq (E(t_1) + (t_2 - t_1)P_r)U_{es} \tag{6.4}$$

Preuve :
Considérons que pendant l'intervalle $[t_1, t_2]$, il existe m tâches apériodiques possédant une date de réveil supérieure ou égale à t_1 et une échéance inférieure ou égale à t_2. Nous supposons que la puissance reçue, P_r, est constante. Par conséquent, l'énergie disponible fournie pendant l'intervalle $[t_1, t_2]$ est égale à $E(t_1) + (t_2 - t_1)P_r$. Comme U_{es} est la proportion d'énergie disponible pour les tâches apériodiques, la quantité d'énergie maximale pouvant être consommée par les tâches apériodiques est $(E(t_1) + (t_2 - t_1)P_r)U_{es}$. Par conséquent, soit E_{ape} la quantité d'énergie consommée par les tâches apériodiques sur l'intervalle $[t_1, t_2]$ telle que $E_{ape} = \sum_{i=1,(r_i^a \geq t_1),(d_i^a \leq t_2)}^{m} E_i$ alors $E_{ape} \leq (E(t_1) + (t_2 - t_1)P_r)U_{es}$. □

Ceci nous amène à proposer, le théorème suivant :

Théorème 2.4 *Soient une configuration de n tâches périodiques, avec un facteur d'utilisation énergétique équivalent U_e^*, et une configuration de m tâches apériodiques servies par un serveur TBS. Soit U_{es} le facteur d'utilisation énergétique du serveur. Le système global est ordonnançable du point de vue énergétique seulement si :*

$$U_e^* + U_{es} \leq 1 \tag{6.5}$$

où, U_e^*, est l'utilisation équivalente énergétique associée aux tâches périodiques et est définie dans le chapitre 3. Nous rappelons que nous le calculons comme suit :

$$U_e^* = max_{L \geq 0} \left\{ \frac{\sum_{i=1}^{n} g_i^*(0, L)}{E(0) + E_r(0, L)} \right\} \tag{6.6}$$

E(0) représente le niveau initial d'énergie dans la batterie, $E_r(0, L)$ représente l'énergie reçue pendant l'intervalle $[0, L($.
On suppose que la puissance reçue par l'environnement est constante pendant une hyperpériode H^*=$PPCM($ $T_1.s_1$, .., $T_i.s_i$, .., $T_n.s_n$). Comme $P_r(t) = P_r \ \forall t$, $E_r(0, L) = P_r.L$, où L représente les échéances des jobs rouges pendant l'intervalle $[0, H($.

La demande énergétique des jobs rouges d'une tâche périodique τ_i est calculée comme suit :

$$g_i^*(0, L) = (\left\lfloor \frac{L}{T_i} \right\rfloor - \left\lfloor \frac{L}{T_i.s_i} \right\rfloor)E_i \tag{6.7}$$

Preuve :
Supposons que $U_e^* + U_{es} \leq 1$ et qu'une violation d'échéance survienne par manque d'énergie à un instant $t = t_v$. Cette violation est précédée par l'exécution de jobs rouges périodiques et/ou tâches apériodiques. Par conséquent, soit un instant $t = t_1$ inférieur à t. Seuls les jobs rouges des tâches périodiques et les tâches apériodiques ayant leur instant de réveil supérieur ou égal à t_1 et une échéance inférieure ou égale à $t = t_v$ sont exécutées sur l'intervalle $[t_1, t_v]$.
Soit E_g, la demande d'énergie globale des tâches s'exécutant sur l'intervalle $[t_1, t_v]$. S'il y a eu un manque d'énergie à un instant t_v, ceci s'explique par le fait que la demande d'énergie sur l'intervalle $[t_1, t_v]$ excède l'énergie disponible entre t_1 et t_v :

$$E_g > E(t_1) + (t_v - t_1)P_r.$$

Remarquons aussi que $E_g \leq g^*(t_1, t_v) + E_{ape}$.
où :
- $g^*(t_1, t_v)$ est la demande énergétique des jobs rouges ayant leur instant de réveil supérieur ou égal à t_1 et une échéance inférieure ou égale à $t = t_v$ telle que $g^*(t_1, t_v) = \sum_{i=1}^{n} g_i^*(t_1, t_v)$,
- E_{ape} est la demande énergétique des tâches apériodiques ayant leur instant de réveil supérieur ou égal à t_1 et une échéance inférieure ou égale à $t = t_v$.

Ceci implique que :
$E_g \leq g^*(t_1, t_v) + (E(t_1) + (t_v - t_1)P_r)U_{es}$
$\leq (E(t_1) + (t_v - t_1)P_r)U_e^* + (E(t_1) + (t_v - t_1)P_r)U_{es}$
$\leq (E(t_1) + (t_v - t_1)P_r)(U_e^* + U_{es})$
Par suite, $U_e^* + U_{es} > 1$ qui contredit notre supposition.□

Enfin, d'après les formules étudiées dans cette partie, nous proposons une nouvelle formule permettant de calculer l'échéance fictive d'une tâche apériodique φ_k en considérant les contraintes énergétiques du système.

Théorème 2.5 *Soit un serveur TBS avec une largeur de bande énergétique telle que $U_{es} = 1 - U_e^*$. L'échéance fictive d'une tâche apériodique φ_k est calculée comme suit :*

$$d_k^a = max(r_k, d_{k-1}) + \left\lceil \frac{\frac{E_k^a}{U_{es}} - E(r_k)}{P_r} \right\rceil \tag{6.8}$$

Preuve : Le calcul de l'échéance fictive d_k^a, pour une tâche apériodique φ_k, est fait à l'instant de son réveil r_k^a. Considérons que φ_k a une demande énergétique qui vaut E_k^a. Par conséquent l'échéance fictive doit être calculée de sorte que la tâche puisse disposer de cette quantité d'énergie. Selon la condition 6.4, E_k^a ne doit pas excéder l'énergie disponible pendant l'intervalle $[r_k^a, d_k]$. Par suite, $E_k^a \leq (E(r_k^a) + (d_k^a - r_k^a)P_r)U_{es}$. On obtient l'inégalité suivante $d_k^a \geq \frac{\frac{E_k^a}{U_{es}} - E(r_k^a)}{P_r} + r_k^a$.

Une tâche apériodique n'est jamais préemptée par une autre tâche apériodique et dans le pire-cas, une tâche apériodique φ_{k-1} finit son exécution à d_{k-1}^a. Donc une tâche apériodique φ_k en attente commence son exécution à l'instant $t = max(r_k^a, d_{k-1}^a)$ et l'inégalité précédente devient : $d_k^a \geq \frac{\frac{E_k^a}{U_{es}} - E(r_k^a)}{P_r} + max(r_k^a, d_{k-1}^a)$.
□

2.2.2.3 Point de vue temporel et énergétique

Dans cette partie, nous prenons en compte les contraintes temporelles et énergétiques du système. Par conséquent, le calcul de l'échéance fictive devient comme suit.

Théorème 2.6 *Soient une configuration de n tâches périodiques temps réel fermes et une configuration de tâches apériodiques servies par un serveur TBS, l'échéance fictive d'une tâche apériodique φ_k en fonction de la largeur de la bande CPU U_S et la largeur de la bande énergétique U_{es} est calculée comme suit :*

$$d_k^a = max(r_k^a, d_{k-1}^a) + max(\left\lceil \frac{C_k^a}{U_S} \right\rceil, \left\lceil \frac{\frac{E_k^a}{U_{es}} - E(r_k)}{P_r} \right\rceil) \tag{6.9}$$

Preuve : La preuve découle des formules 6.3 et 6.4. □

Nous proposons deux algorithmes basés sur l'algorithme TBS : TBS-Green-RTO et TBS_{op}-Green-RTO. Ces deux algorithmes diffèrent uniquement par le calcul de l'échéance fictive :
- avec TBS-Green-RTO, on considère que l'énergie dans la batterie à l'instant $t = max(r_k, d_{k-1})$ n'est pas connue et donc la formule de l'échéance fictive pour une tâche φ_k devient :

$$d_k^a = max(r_k^a, d_{k-1}^a) + max(\left\lceil \frac{C_k^a}{U_S} \right\rceil, \left\lceil \frac{E_k^a}{U_{es}.P_r} \right\rceil) \tag{6.10}$$

- avec TBS_{op}-Green-RTO, l'énergie dans la batterie à l'instant $t = max(r_k, d_{k-1})$ est connue et donc l'échéance fictive pour une tâche φ_k est donnée par :

$$d_k^a = max(r_k^a, d_{k-1}^a) + max(\left\lceil \frac{C_k^a}{U_S} \right\rceil, \left\lceil \frac{\frac{E_k^a}{U_{es}} - E(max(r_k^a, d_{k-1}^a)}{P_r} \right\rceil) \tag{6.11}$$

Avec TBS-Green-RTO ou TBS_{op}-Green-RTO, une fois l'échéance fictive de la tâche apériodique arrivant calculée, celle-ci est ajoutée à la liste des tâches apériodiques prêtes. Ainsi à chaque instant *t*, le job dans la liste des jobs rouges prêts et la liste des tâches apériodiques prêtes, ayant la plus proche échéance sera choisi pour être exécuté. Celui-ci sera exécuté seulement si la laxité énergétique du système est positive et si l'énergie disponible dans la batterie est suffisante.

2.2.3 Exemple illustratif de TBS-Green-RTO

Nous reprenons l'exemple illustratif précédent. La Figure 6.2 représente l'ordonnancement des configurations

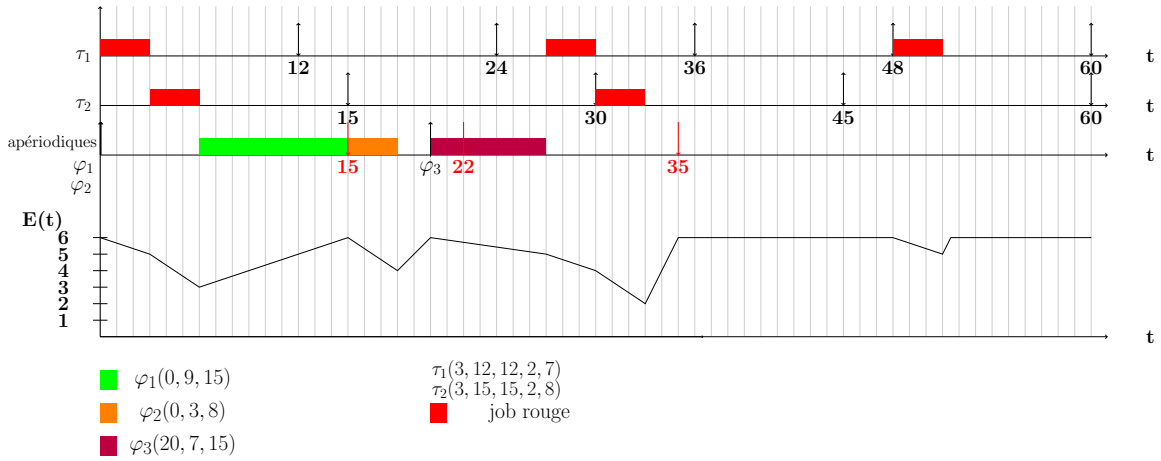


FIGURE 6.2 – Ordonnancement selon TBS-Green-RTO

$\mathcal{T}*$ et φ selon l'algorithme TBS-Green-RTO.
Comme $U_p^* = 0.4$ et $U_e^* = 0.45$, alors $U_S = 0.6$ et $U_{es} = 0.55$. Les tâches apériodiques φ_1 et φ_2 arrivent à l'instant *t=0*, les échéances fictives calculées sont respectivement $d_1^a = 0 + max(\lceil \frac{9}{0.6} \rceil, \lceil \frac{15}{0.55*2} \rceil) = 15$ et $d_1^a = max(0, 15) + max(\lceil \frac{3}{0.6} \rceil, \lceil \frac{8}{0.55*2} \rceil) = 22$.
φ_1 et φ_2 sont ordonnancées conjointement aux jobs rouges selon Green-RTO. Ainsi le job rouge $\tau_{1,1}$ est celui qui a la plus proche échéance. Après avoir calculé la laxité énergétique du système et avoir obtenu une valeur positive égale à 22, le job rouge $\tau_{1,1}$ commence son exécution.
Ensuite à l'instant *t=3*, le job $\tau_{2,1}$ est le plus prioritaire. La laxité énergétique du système est calculée en prenant en considération la consommation des jobs $\tau_{2,1}$ et φ_1 telle que $SlackEnergy(3) = E(3) + (d_{2,1} - 3)P_r - (E_{2,1} + E_1^a) = 5 + 24 - (8 + 15) = 6$. Comme $SlackEnergy(3)$ est positive alors le job rouge $\tau_{2,1}$ commence son exécution. Il termine à l'instant *t=6*, et c'est le job apériodique φ_1 qui est le plus prioritaire. Comme la laxité énergétique a déjà été calculée entre l'instant *t=6* et *t=15* et que celle-ci est positive, et comme l'énergie disponible dans la batterie est suffisante ($E(6) = 3$), φ_1 commence son exécution. φ_1 finit son exécution à l'instant *t=15*. φ_2 ayant l'échéance $d_2^a = 22$ est la plus prioritaire. La laxité énergétique du système à l'instant

t=15 est calculée et vaut $SlackEnergy(15) = E(15) + (22 - 15)P_r - E_2^a = 6 + 14 - 8 = 12$. Comme $SlackEnergy(15)$ est positive, φ_2 commence son exécution à l'instant *t=15*. A l'instant *t=20*, une tâche apériodique φ_3 arrive. L'échéance fictive $d_3^a = 35$ lui est attribuée. Comme φ_3 a la plus proche échéance à l'instant *t=20*, la laxité énergétique du système est calculée. Celle-ci étant positive, φ_3 démarre son exécution dès son arrivée.

Nous remarquons dans cet exemple que le temps de réponse de la tâche φ_3 est nettement meilleur que celui obtenu avec BG-Green-RTO (il y a un gain de 8 unités de temps). On obtient respectivement pour les tâches apériodiques φ_1, φ_2 et φ_3, des temps de réponse égaux à 15, 18, 7. Les temps de réponse normalisés correspondant aux tâches φ_1, φ_2 et φ_3 sont respectivement égaux à 0.6, 0.18 et 1.

2.2.4 Exemple illustratif de TBS_{op}-Green-RTO

Nous reprenons l'exemple illustratif précédent. La Figure 6.3 représente l'ordonnancement des configurations

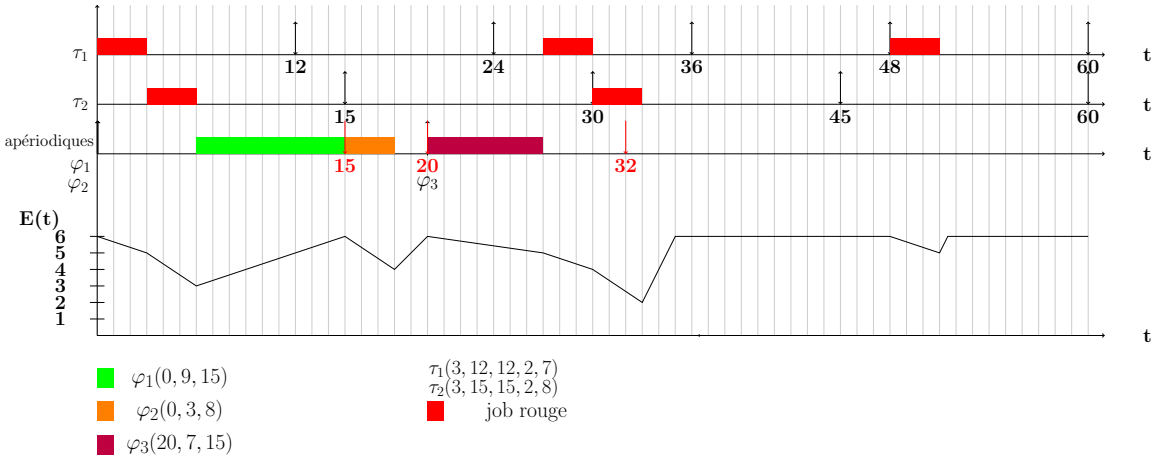


FIGURE 6.3 – Ordonnancement selon TBS_op-Green-RTO

$\mathcal{T}*$ et φ selon l'algorithme TBS_{op}-Green-RTO.
Comme $U_p^* = 0.4$ et $U_e^* = 0.45$, alors $U_S = 0.6$ et $U_{es} = 0.55$.
Les tâches apériodiques φ_1 et φ_2 arrivent à l'instant *t=0*. Les échéances fictives calculées sont respectivement $d_1^a = 0 + max(\lceil \frac{9}{0.6} \rceil, \left\lceil \frac{\frac{15}{0.55} - 6}{2} \right\rceil) = 15$ et $d_1^a = max(0, 15) + max(\lceil \frac{3}{0.6} \rceil, \left\lceil \frac{\frac{8}{0.55} - 6}{2} \right\rceil) = 20$.
φ_1 et φ_2 sont ordonnancées conjointement aux jobs rouges selon Green-RTO. On remarque que φ_2 a une échéance plus courte que celle calculée avec TBS-Green-RTO. Cependant dans cet exemple, il n'y a pas de jobs devant s'exécuter entre $t = 15$ et $t = 22$. Même si l'échéance calculée est plus courte, le temps de réponse de φ_2 ne sera pas amélioré avec TBS_{op}-Green-RTO.
φ_3 arrive à l'instant *t=20* et se voit attribuer l'échéance $d_3^a = 32$. Même si cette échéance est plus courte que celle obtenue avec TBS-Green-RTO, comme aucune tâche n'est plus prioritaire que φ_3 entre *t=20* et *t=35*, le temps de réponse sera le même que celui obtenu avec TBS-Green-RTO.

Par conséquent, dans cet exemple, on peut observer que l'ordonnancement avec TBS_{op}-Green-RTO est similaire à celui obtenu avec TBS-Green-RTO. On observe aussi que les temps de réponse sont meilleurs que ceux observés avec BG-Green-RTO. Ceci dit, comme il n'y a que les jobs rouges des tâches périodiques qui sont exécutés, la QoS globale obtenue au niveau des tâches périodiques est minimale et vaut 50% comme le paramètre de pertes vaut 2 pour τ_1 et τ_2.

3 Algorithmes basés sur Green-BWP

En utilisant les algorithmes basés sur Green-RTO, on minimise le temps de réponse des tâches apériodiques tout en engendrant une dégradation maximale de la QoS du système au niveau des tâches périodiques. Nous proposons une autre solution sans cet inconvénient, basée sur Green-BWP.

3.1 BG-Green-BWP

3.1.1 Principe de fonctionnement

Avec BG-Green-BWP, les jobs bleus sont exécutés quand c'est possible c'est-à-dire quand il n'y a pas de jobs rouges à l'état prêt. Ainsi les jobs rouges et bleus sont ordonnancés selon l'algorithme Green-BWP. Les tâches apériodiques arrivant dans le système commencent leur exécution seulement si aucun job rouge ou bleu n'est prêt et si l'énergie disponible est suffisante de façon à assurer que la batterie soit pleine au prochain réveil d'un job rouge.

3.1.2 Exemple illustratif de BG-Green-BWP

Nous reprenons le même exemple illustratif. La Figure 6.4 illustre l'ordonnancement des tâches périodiques et apériodiques selon BG-Green-BWP.
Les tâches apériodiques φ_1 et φ_2 arrivent à l'instant *t=0*. Comme les jobs rouges $\tau_{1,1}$ et $\tau_{2,1}$ sont prêts, ils sont

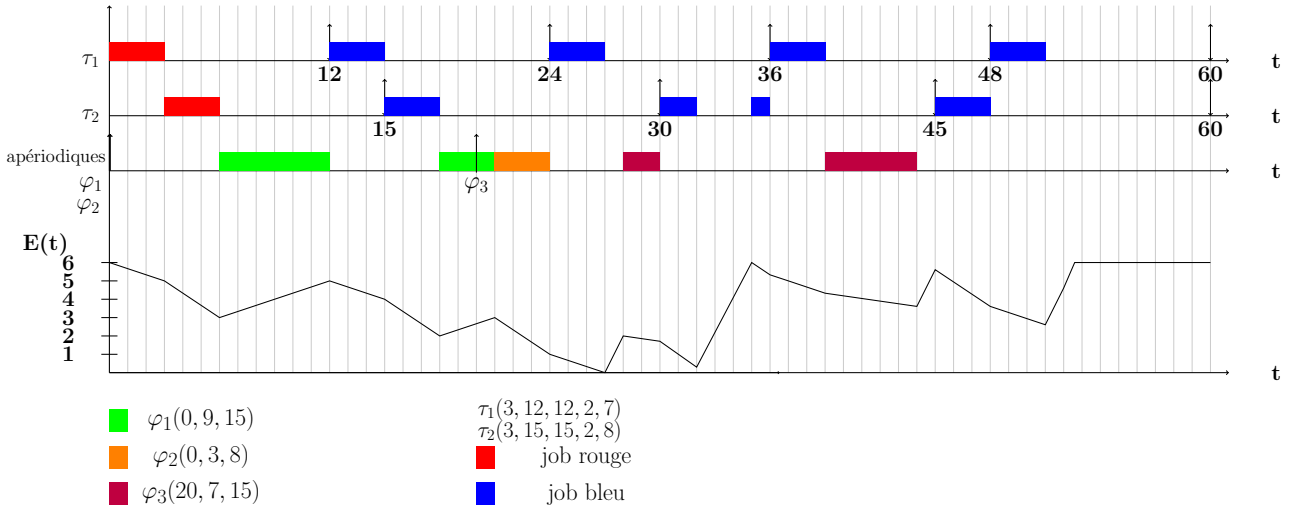


FIGURE 6.4 – Ordonnancement selon BG-Green-BWP

les plus prioritaires et sont exécutés selon Green-BWP. A l'instant *t=6*, il n'y a ni job rouge ni job bleu prêt, on peut exécuter des tâches apériodiques. φ_1 est choisie pour être exécutée. Celle-ci ne sera exécutée que si l'énergie disponible est suffisante de sorte à avoir le niveau de batterie pleine lors du prochain réveil d'un job rouge. BG-Green-BWP suppose le pire-cas qui considère que si l'occurrence d'une tâche est bleue, le prochain job de cette tâche est rouge. Par conséquent, dans notre cas, comme à l'instant *t=6*, les jobs $\tau_{1,2}$ et $\tau_{2,2}$ sont bleus, nous supposons que les prochaines occurrences des tâches sont rouges. Ainsi le prochain réveil est celui du job $\tau_{1,3}$ et vaut $r_{min} = 24$.
L'énergie disponible pour exécuter φ_1 est calculée ainsi : $E_{disp}(6) = E(6) + (r_{min} - 6)P_r - C = 3 + 36 - 6 = 33$. Comme $E_{disp}(6)$ est positive, φ_1 commence son exécution à l'instant *t=6*, et continue à s'exécuter tant que l'énergie disponible est positive et qu'aucun job rouge ou bleu n'est réveillé. A l'instant *t=12*, le job bleu $\tau_{1,2}$ est réveillé et est le plus prioritaire, donc l'exécution de φ_1 est interrompue. La laxité énergétique du système à l'instant *t=12* est calculée telle que $SlackEnergy(12) = E(12) + (24 - 12)P_r - E_1 = 5 + 24 - 7 = 22$. Comme $SlackEnergy(12)$ est positive, $\tau_{1,2}$ commence son exécution.
Ensuite, à l'instant *t=15*, le job bleu $\tau_{2,2}$ est choisi pour être exécuté. Le calcul de la laxité énergétique est fait. Comme $SlackEnergy(15) = 26$, $\tau_{2,2}$ commence son exécution. A l'instant *t=18*, φ_1 est choisie pour être exécutée. Le calcul de l'énergie disponible avant le prochain réveil d'un job rouge est fait. Le prochain réveil d'un job rouge est $r_{min} = 36$. Comme $E_{disp} = E(18) + (36 - 18)P_r - 6 = 32$, φ_1 continue son exécution. Et ainsi de suite..

Dans cet exemple, φ_1, φ_2 et φ_3 ont des temps de réponse respectifs de 21, 24 et 24 unités de temps. Si l'on rapporte les durées d'exécution des tâches par rapport aux temps de réponse obtenus, les temps de réponse

normalisés de φ_1, φ_2 et φ_3 sont respectivement de 0.375, 0.125 et 0.291. Ceci montre que malgré le fait que le système soit moyennement chargé temporellement et énergétiquement, les temps de réponse sont très longs. BG-Green-BWP a l'avantage d'offrir une QoS globale égale à 100% puisque sur cet exemple $U_p < 1$.

Ci-après, nous proposons l'algorithme Green-AWP qui a le même principe que BG-Green-BWP sauf que les tâches apériodiques sont plus prioritaires que les jobs bleus.

3.2 Green-AWP

3.2.1 Principe de fonctionnement

L'algorithme Green-AWP (Aperiodic When Possible) sous-entend que les tâches apériodiques sont exécutées quand il n'y a aucun job rouge à l'état prêt. Il a le même principe que BG-Green-BWP sauf que s'il n'y a aucun job rouge prêt, les tâches apériodiques sont les plus prioritaires. Par conséquent, les jobs bleus sont exécutés seulement s'il n'y a ni job rouge ni tâche apériodique à l'état prêt.

A chaque instant *t*, l'ordonnanceur choisit le job à exécuter :

- S'il existe au moins un job rouge à l'état prêt, celui qui a la plus proche échéance est choisi.
- Sinon si la liste des tâches apériodiques bloquées est non vide, la tâche apériodique la plus ancienne est choisie.
- S'il n'y a pas de jobs rouges prêts et si la liste des tâches apériodiques bloquées est vide, le job bleu prêt qui a la plus proche échéance est choisi.

Ensuite si le job à exécuter est périodique, celui-ci est exécuté selon Green-BWP. Sinon si le job à exécuter est apériodique, la quantité d'énergie disponible avant le prochain réveil d'un job rouge r_{min} est calculée telle que $E_{disp}(t) = E(t) + (r_{min} - t)P_r - (E_{max} - E_{min})$. Si $E_{disp}(t)$ est positive et l'énergie dans la batterie est suffisante, la tâche apériodique sélectionnée est autorisée à s'exécuter sur le processeur.

3.2.2 Exemple illustratif

La Figure 6.5 illustre l'ordonnancement des tâches périodiques et apériodiques selon Green-AWP.
Comme les jobs rouges $\tau_{1,1}$ et $\tau_{2,1}$ sont prêts, ils sont les plus prioritaires et sont exécutés selon Green-BWP.

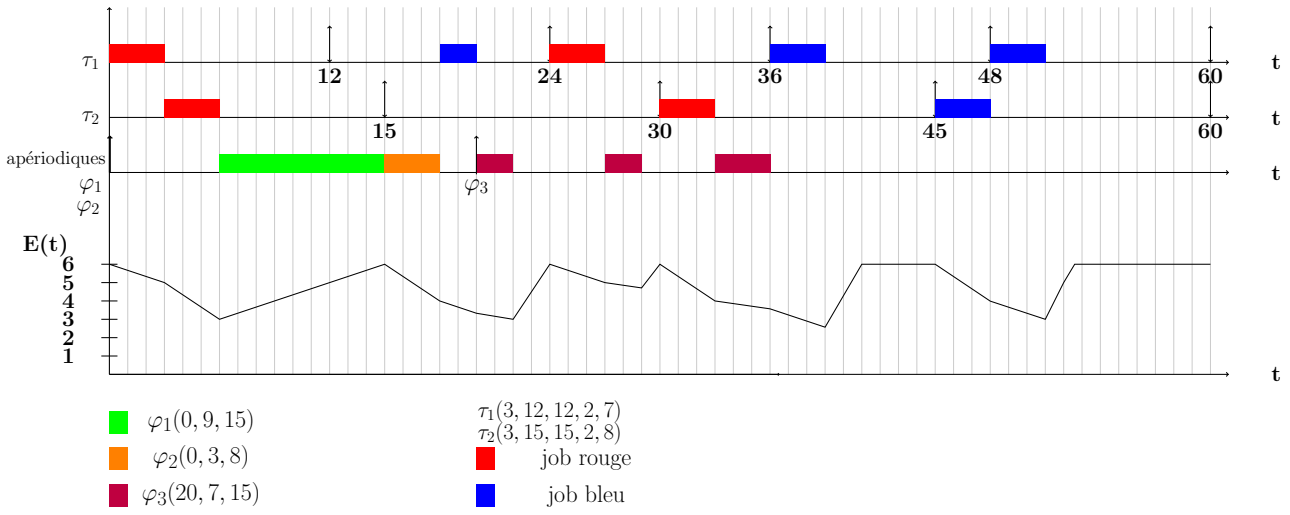


FIGURE 6.5 – Ordonnancement selon Green-AWP

A l'instant *t=6*, il n'y a aucun job périodique prêt. Donc φ_1 est choisie et ne démarre son exécution que si l'énergie disponible jusqu'au prochain réveil d'un job rouge est positive. Green-AWP suppose que les tâches apériodiques sont plus prioritaires que les jobs bleus. Par conséquent, dans notre cas, comme à l'instant *t=6*,

les jobs $\tau_{1,2}$ et $\tau_{2,2}$ sont bleus, nous supposons que les prochaines occurrences des tâches sont rouges. Ainsi le prochain réveil est celui du job rouge $\tau_{1,3}$ et vaut $r_{min} = 24$. L'énergie disponible pour exécuter φ_1 est calculée ainsi : $E_{disp}(6) = E(6) + (r_{min} - 6)P_r - C = 3 + 36 - 6 = 33$. Comme $E_{disp}(6)$ est positive, φ_1 commence son exécution à l'instant *t=6*, et continue à s'exécuter tant que l'énergie disponible est positive et qu'aucun job rouge n'est réveillé. A l'instant *t=12*, le job bleu $\tau_{1,2}$ est réveillé cependant il est moins prioritaire que φ_1. φ_1 continue alors son exécution comme l'énergie dans la batterie est suffisante et l'énergie disponible est toujours positive.
A l'instant *t=15*, φ_1 finit son exécution, et comme il n'y a aucun job rouge prêt, φ_2 est choisie pour être exécutée. L'énergie disponible à l'instant *t=15* est calculée de façon à assurer que la batterie soit pleine au prochain réveil d'un job rouge, $r_{min} = 24$: $E_{disp}(15) = E(15)+(r_{min}-15)P_r-C = 6+18-6 = 18$. Comme $E_{disp}(15)$ est positive, φ_2 commence son exécution. Celle-ci termine son exécution à l'instant $t = 18$, et comme il n'y ni job rouge ni tâche apériodique à l'état prêt, le job bleu $\tau_{1,2}$ est choisi pour être exécuté. La laxité énergétique du système est calculée à l'instant $t = 18$ telle que $SlackEnergy(18) = E(18)+(24-18)P_r-E_1 = 4+12-7 = 9$. Comme $SlackEnergy(18)$ est positive, $\tau_{1,2}$ commence son exécution. Cependant à l'instant $t = 20$, φ_3 est réveillée et interrompt l'exécution de $\tau_{1,2}$. L'énergie disponible est calculée à l'instant $t = 20$ telle qu'au réveil du job rouge $\tau_{1,3}$ la batterie soit pleine, $E_{disp}(20) = E(20) + (r_{min} - 20)P_r - C = 3 + 8 - 6 = 5$. Comme $E_{disp}(20)$ est positive, φ_3 commence son exécution tant qu'aucun job rouge n'est réveillé, que l'énergie disponible est positive et que l'énergie dans la batterie est suffisante. A l'instant $t = 22$, l'énergie disponible n'est plus suffisante donc φ_3 arrête son exécution. Ainsi, pendant deux unités de temps, le processeur est inactif et donc la batterie se recharge jusqu'à devenir complètement pleine au réveil du job rouge $\tau_{1,3}$. Ensuite, à l'instant $t = 27$, $E_{disp}(27) = E(27)+(30-27)P_r-C = 5+6-6 = 5$. Comme $E_{disp}(27)$ est positive, φ_3 continue son exécution. A l'instant $t = 29$, l'énergie disponible restante n'est plus suffisante et donc φ_3 arrête son exécution et le processeur est inactif pour recharger la batterie. A l'instant $t = 30$, la batterie est pleine et le job rouge $\tau_{2,3}$ s'exécute pendant 3 unités de temps. Enfin, après avoir recalculé l'énergie disponible jusqu'au prochain réveil d'un job rouge $r_{min} = 30$, comme $E_{disp}(33)$ est positive, φ_3 continue son exécution et est complètement exécutée à l'instant $t = 36$. Et ainsi de suite..

Dans cet exemple, φ_1, φ_2 et φ_3 ont des temps de réponse respectifs de 15, 18 et 16 unités de temps. Si l'on rapporte les durées d'exécution des tâches par rapport à leur temps de réponse, les temps de réponse normalisés de φ_1, φ_2 et φ_3 sont respectivement de 0.6, 0.16 et 0.43. Ceci montre que les temps de réponse obtenus avec Green-AWP sont nettement meilleurs que ceux obtenus avec BG-Green-BWP. Cependant, ceci a pour conséquence de dégrader la QoS au niveau des tâches périodiques. Avec Green-AWP, on obtient une QoS globale sur cet exemple égale à 77.5%.

3.3 Algorithmes basés sur le serveur TBS

Notre objectif est ici de proposer une solution qui améliore les performances obtenues par Green-AWP.

3.3.1 Principe de fonctionnement

Les algorithmes TBS-Green-BWP et TBS_{op}-Green-BWP fonctionnent en assignant une échéance fictive à toute tâche apériodique non critique arrivant dans le système. Ensuite, ils ordonnancent les tâches apériodiques conjointement aux tâches périodiques telles que les jobs rouges et apériodiques soient ordonnancés selon EDF et soient toujours plus prioritaires que les jobs bleus. A chaque fois, qu'un job périodique ou apériodique veut s'exécuter, il nécessite de calculer la laxité énergétique du système. Si celle-ci est positive et si l'énergie disponible dans la batterie est suffisante, le job est exécuté. Nous rappelons que la différence entre TBS-Green-BWP et TBS_{op}-Green-BWP réside dans le calcul de l'échéance fictive. En effet, TBS_{op}-Green-BWP prend en compte le niveau d'énergie disponible dans la batterie au moment du calcul de l'échéance.

Les conditions d'ordonnançabilité étudiées dans le cas de Green-RTO s'appliquent également dans le cas de Green-BWP. De plus, comme les formules pour assigner une échéance fictive à une tâche apériodique, ne prennent en compte que les jobs rouges dans le système, il en découle que les formules avec Green-BWP sont identiques aux formules 6.10 et 6.11 étudiées dans le cas de Green-RTO (cf. Section 2.2.1). Ci-après, nous présentons des exemples illustratifs d'ordonnancement avec TBS-Green-BWP et TBS_{op}-Green-BWP.

3.3.2 Exemple illustratif avec TBS-Green-BWP

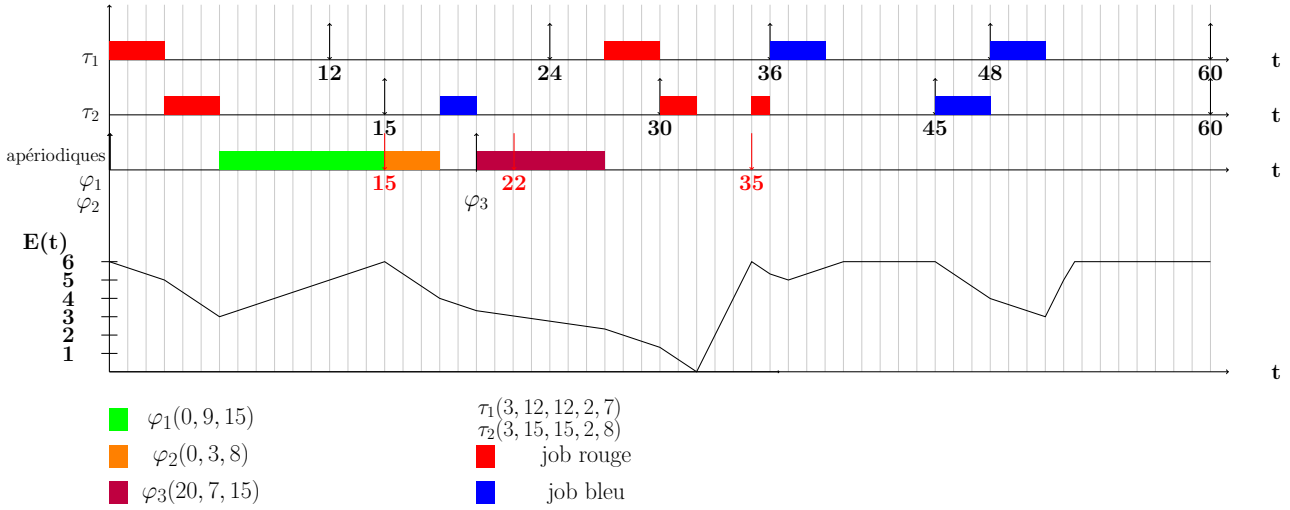


FIGURE 6.6 – Ordonnancement selon TBS-Green-BWP

La Figure 6.6 illustre l'ordonnancement des configurations $\mathcal{T}$ et φ selon TBS-Green-BWP.
$U_p^* = 0.4$ et $U_e^* = 0.45$, alors $U_S = 0.6$ et $U_{es} = 0.55$. A l'instant *t=0*, les tâches apériodiques φ_1 et φ_2 sont réveillées, les échéances fictives calculées sont respectivement $d_1^a = 0 + max(\lceil \frac{9}{0.6} \rceil, \lceil \frac{15}{0.55*2} \rceil) = 15$ et $d_1^a = max(0, 15) + max(\lceil \frac{3}{0.6} \rceil, \lceil \frac{8}{0.55*2} \rceil) = 22$.
Ensuite, φ_1 et φ_2 sont ordonnancées conjointement aux jobs périodiques selon Green-BWP. A l'instant $t = 0$, le job rouge $\tau_{1,1}$ est celui qui a la plus proche échéance. Après avoir calculé la laxité énergétique du système et avoir obtenu une valeur positive égale à 22, le job rouge $\tau_{1,1}$ commence son exécution. Ensuite à l'instant *t=3*, le job $\tau_{2,1}$ est le plus prioritaire, la laxité énergétique du système est calculée en prenant en considération la consommation des jobs $\tau_{2,1}$ et φ_1 telle que $SlackEnergy(3) = E(3) + (d_{2,1} - 3)P_r - (E_{2,1} + E_1^a) = 5 + 24 - (8 + 15) = 6$. Comme $SlackEnergy(3)$ est positive alors le job rouge $\tau_{2,1}$ commence son exécution. Il termine à l'instant *t=6*, et c'est le job apériodique φ_1 qui est le plus prioritaire. Comme la laxité énergétique a déjà été calculée entre l'instant *t=6* et *t=15* et que celle-ci est positive, et puisque l'énergie disponible dans la batterie est suffisante ($E(6) = 3$), φ_1 commence son exécution. φ_1 finit son exécution à l'instant *t=15*. φ_2, a l'échéance $d_2^a = 22$ et est la plus prioritaire. La laxité énergétique du système à l'instant *t=15* est calculée et vaut $SlackEnergy(15) = E(15) + (22 - 15)P_r - E_2^a = 6 + 14 - 8 = 12$. Comme $SlackEnergy(15)$ est positive, φ_2 commence son exécution à l'instant *t=15*.
A l'instant $t = 18$, il n'y a aucun job rouge ou apériodique prêt, le job bleu $\tau_{2,2}$ est choisi pour commencer son exécution. $SlackEnergy(18) = E(18) + (30 - 18)P_r - 8 = 4 + 24 - 8 = 20$ donc $\tau_{2,2}$ commence son exécution. Or à l'instant $t = 22$, la tâche apériodique φ_3 arrive et l'échéance $d_3^a = 35$ lui est attribuée. Par conséquent, $\tau_{2,2}$ arrête son exécution et comme $SlackEnergy(20)$ est positive, φ_3 commence son exécution. Et ainsi de suite..
Nous observons une amélioration de 9 unités de temps, du temps de réponse de φ_3 obtenus avec TBS-Green-BWP par rapport à celui obtenu avec Green-AWP. On obtient respectivement pour les tâches apériodiques φ_1, φ_2 et φ_3, des temps de réponse égaux à 15, 18, 7 qui sont identiques à ceux obtenus avec TBS-Green-RTO.

Les temps de réponse normalisés correspondant aux tâches φ_1, φ_2 et φ_3 sont respectivement égaux à 0.6, 0.18 et 1. TBS-Green-BWP présente un grand avantage par rapport à TBS-Green-RTO au niveau de la QoS globale obtenue. En effet, dans l'exemple 6.2, on obtient la QoS minimale qui vaut 50% alors que TBS-Green-BWP permet de minimiser le temps de réponse tout en garantissant une meilleure QoS qui vaut 77.5%.

3.3.3 Exemple illustratif avec TBS_{op}-Green-BWP

La Figure 6.7 représente l'ordonnancement des configurations $\mathcal{T}^*$ et φ selon l'algorithme TBS_{op}-Green-BWP. Comme $U_p^* = 0.4$ et $U_e^* = 0.45$, alors $U_S = 0.6$ et $U_{es} = 0.55$. Les tâches apériodiques φ_1 et φ_2 arrivent à l'instant *t=0*, les échéances fictives calculées sont respectivement $d_1^a = 0 + max(\lceil \frac{9}{0.6} \rceil, \left\lceil \frac{\frac{15}{0.55} - 6}{2} \right\rceil) = 15$ et

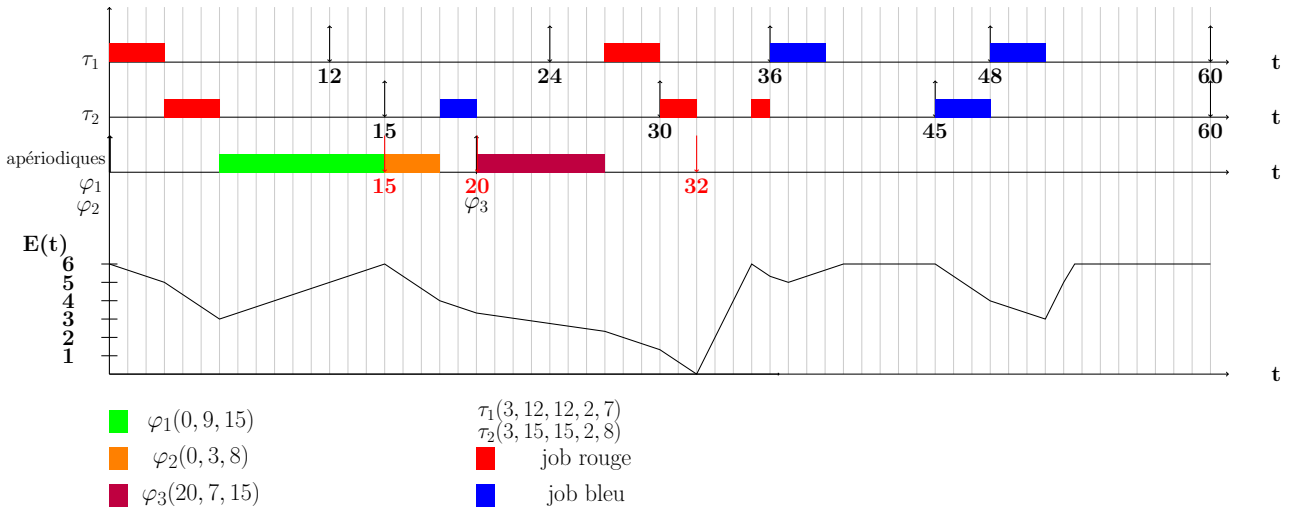


FIGURE 6.7 – Ordonnancement selon TBS_{op}-Green-BWP

$d_1^a = max(0, 15) + max(\lceil \frac{3}{0.6} \rceil, \left\lceil \frac{\frac{8}{0.55} - 6}{2} \right\rceil) = 20$.
φ_1 et φ_2 sont ordonnancées conjointement aux jobs rouges selon Green-BWP. On remarque que φ_2 a une échéance plus courte que celle calculée avec TBS-Green-BWP. Cependant dans cet exemple il n'y a pas de jobs devant s'exécuter entre $t = 15$ et $t = 22$. Par conséquent, l'optimisation de l'échéance ne change pas l'ordonnancement obtenu et donc c'est exactement le même que celui expliqué pour TBS-Green-BWP. φ_3 arrivant à l'instant *t=20* se voit attribuer une échéance $d_3^a = 32$. Même si cette échéance est plus courte que celle obtenue avec TBS-Green-RTO, comme aucune tâche n'est plus prioritaire que φ_3 entre *t=20* et *t=35*, le temps de réponse sera le même que celui obtenu avec TBS-Green-RTO qui est déjà optimisé. L'ordonnancement avec TBS_{op}-Green-RTO étant similaire à celui obtenu avec TBS-Green-RTO, les temps de réponse sont les mêmes.

4 Evaluation des performances

L'objectif de cette section est d'évaluer comparativement les performances des serveurs proposés par le biais de simulations. Notre but est alors de minimiser le temps de réponse des tâches apériodiques tout en veillant à exécuter les jobs rouges dans le respect de leurs contraintes temporelles. De plus, nous tentons de minimiser la dégradation de la QoS au niveau des tâches périodiques en proposant des algorithmes basés sur Green-BWP. Pour cette étude, un générateur de tâches périodiques et un générateur de tâches apériodiques sont implémentés en langage C.

Le générateur de tâches périodiques prend en entrée les paramètres suivants :

- *n* : le nombre de tâches périodiques constituant la configuration,
- *PPCMmax* : le ppcm maximal des périodes des tâches périodiques constituant la configuration,
- U_p : la charge processeur des tâches périodiques,
- $\bar{P_e}$: la puissance moyenne consommée par les tâches périodiques,
- s_i : le paramètre de pertes uniformes à toutes les tâches périodiques,

En sortie, le générateur renvoie une configuration de tâches temps réel fermes $\mathcal{T}^*$ = $\{\tau_i(C_i,\ D_i,\ T_i,\ s_i,\ E_i),\ i = 1 \text{ à } n\}$.

- Les tâches périodiques sont indépendantes, préemptables, profondément rouges et à échéances sur requêtes ($D_i = T_i$).

- Les périodes des tâches T_i sont aléatoirement choisies et leur choix dépend du *PPCMmax* donné en entrée.
- Les durées d'exécution des tâches au pire-cas C_i sont générées aléatoirement et le choix dépend de $U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$.
- Les demandes énergétiques des tâches E_i sont générées aléatoirement et le choix dépend de $\bar{P}_e = \sum_{i=1}^{n} \frac{E_i}{T_i}$.
- Il n'y a aucune corrélation entre E_i et C_i.

Le générateur de tâches apériodiques prend en entrée les paramètres suivants :

- *m* : le nombre de tâches apériodiques générées sur chaque hyperpériode,
- H^* : la valeur de l'hyperpériode équivalente des tâches périodiques avec pertes,
- U_{ap} : la charge temporelle dédiée aux tâches apériodiques,
- U_{ape} : la charge énergétique dédiée aux tâches apériodiques,

En sortie, ce générateur renvoie une configuration de m tâches apériodiques non critiques φ = $\{\varphi_i(r_i^a, C_i^a, E_i^a),\ i = 1\ à\ m\}$.

- Les instants de réveil r_i^a sont calculés de façon à ce que la tâche φ_i soit réveillée et exécutée avant la fin de l'hyperpériode.
- Les durées d'exécution des tâches au pire-cas C_i^a sont générées aléatoirement et le choix dépend de U_{ap}.
- Les demandes énergétiques des tâches E_i^a sont générées aléatoirement et le choix dépend de U_{ape}.

Le simulateur consiste à ordonnancer en-ligne une configuration de n tâches périodiques et m tâches apériodiques non critiques selon les algorithmes décrits précédemment. Nous considérons des systèmes avec différents profils de charge temporelle dûs aux tâches périodiques, et nous évaluons les performances des différents algorithmes vis-à-vis des arrivées de tâches apériodiques et en tenant compte des critères de performances suivants :

1. **Taux de respect des tâches périodiques** : Le nombre de jobs périodiques complètement exécutés avant leur échéance sur le nombre total de jobs périodiques.

$$TQoS = \frac{nombre\ d'échéance\ satisfaites}{nombre\ total\ de\ jobs}.100 \tag{6.12}$$

   Cette métrique permet de visualiser comment le système réagit vis-à-vis des requêtes apériodiques. Elle permet de donner une idée sur le meilleur ordonnanceur à choisir pour servir le plus vite les tâches apériodiques tout en garantissant le moins de dégradation de la QoS au niveau des tâches périodiques de l'application.

2. **Temps de réponse normalisé** : Le temps de réponse TR_i d'une tâche φ_i représente la durée prise par φ_i depuis son réveil pour être exécutée ($TR_i = f_i^a - r_i^a$). Plus le temps de réponse d'une tâche φ_i est proche de sa durée, plus le temps de réponse est optimisé. C'est pourquoi, nous évaluons le temps de réponse normalisé qui représente la moyenne, sur l'ensemble de tâches apériodiques, des durées d'exécution au pire-cas des tâches apériodiques, normalisées par rapport aux temps de réponse des tâches apériodiques.

$$Temps\_réponse\ moyen\ normalisé = \frac{\sum_{i=1}^{m} \frac{C_i^a}{f_i^a - r_i^a}}{m} \tag{6.13}$$

   Plus le temps de réponse normalisé est proche de 1, plus le temps de réponse est proche de la durée d'exécution. A contrario, si le temps de réponse normalisé est proche de zéro ceci implique que le temps de réponse de la tâche s'éloigne de sa durée d'exécution et peut même être infini.

3. **Temps de gigue normalisé** : Le temps de gigue de démarrage représente l'écart entre l'instant de réveil de la tâche apériodique et son début d'exécution ($f_i^a - G_i^a$). Si le temps de gigue d'une tâche φ_i est très proche de son temps de réponse, ceci implique que la tâche a été servie rapidement sans être bloquée. Par conséquent, nous choisissons d'étudier le temps de gigue normalisé qui représente le temps de gigue de démarrage rapporté à son temps de réponse.

$$Temps\_gigue\ moyen\ normalisé = \frac{\sum_{i=1}^{m} \frac{f_i^a - G_i^a}{f_i^a - r_i^a}}{m} \tag{6.14}$$

Cette métrique permet de visualiser la durée durant laquelle la tâche apériodique est bloquée par une autre tâche et attend pour être servie. Elle permet de déterminer quel ordonnanceur choisir quand le résultat fourni après l'exécution d'une tâche apériodique est pertinent et est requis dans les plus brefs délais. **Quand le temps de gigue normalisé affiche une valeur proche de 1 ceci veut dire que la tâche a été immédiatement servie et que le temps de blocage avant exécution est nul.**

Dans cette partie, nous évaluons les performances des algorithmes basés sur Green-RTO puis celles des algorithmes basés basés sur Green-BWP suivant différents profils énergétiques et temporels du système considéré. Pour chaque pas de simulation, 100 configurations de tâches périodiques et apériodiques différentes sont générées. Chaque configuration de tâches périodiques est constituée de 10 tâches avec un ppcm maximal égal à 3600. De plus, 5 tâches apériodiques sont générées sur une hyperpériode H^*.
Les simulations sont effectuées sur 10 hyperpériodes. La batterie est supposée initialement pleine telle que $E(0) = H^*.P_r$.

Pour chaque cas étudié, nous faisons varier en abscisse la charge temporelle des tâches apériodiques U_{ap} et nous étudions les métriques décrites ci-dessus.

4.1 Etude avec Green-RTO

Dans cette partie, nous étudions les performances des algorithmes proposés qui se basent sur l'algorithme Green-RTO pour ordonnancer les tâches périodiques. Nous ne présentons pas le graphique relatif au taux de respect des tâches périodiques car Green-RTO offre le taux de respect minimal qui dépend du paramètre de pertes des tâches périodiques. Comme le paramètre de pertes est uniforme pour toutes les tâches périodiques, le taux de respect global obtenu est de 50% quand $s_i = 2$ et 83% quand $s_i = 6$.

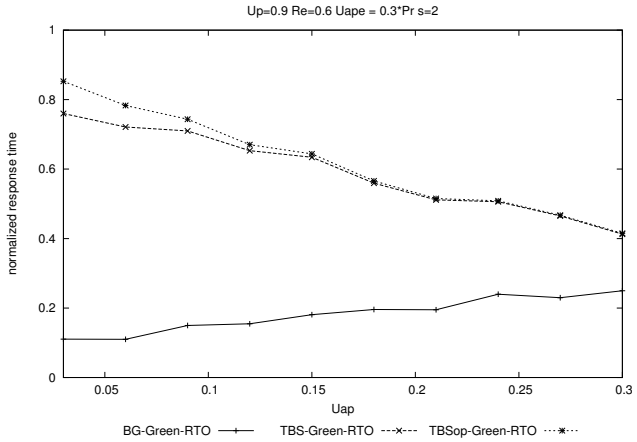
4.1.1 Système fortement chargé temporellement

Dans cette partie, nous considérons que le système est fortement chargé temporellement ($U_p = 0.9$) et moyennement chargé énergétiquement ($R_e = 0.6$). Nous ajoutons à ce système des tâches apériodiques avec une charge énergétique $U_{ape} = 0.3P_r$. Par conséquent, le système global devient fortement chargé énergétiquement.

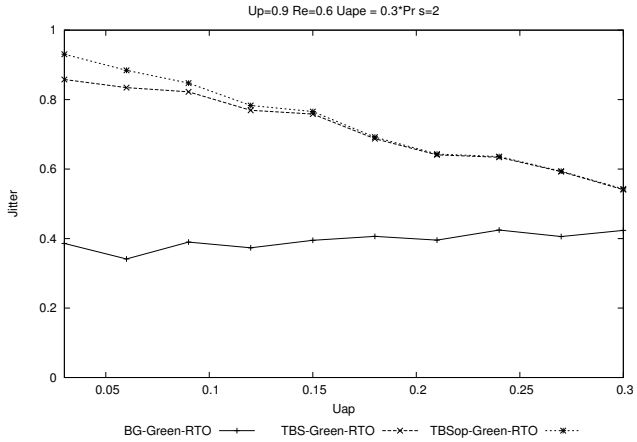
Nous remarquons sur les Figures 6.8(a), 6.9(a), 6.8(b) et 6.9(b) que :

- BG-Green-RTO offre les plus médiocres temps de réponse et temps de gigue de démarrage et ce, quelle que soit la charge temporelle. TBS_{op}-Green-RTO et TBS-Green-RTO offrent des temps de réponse et des temps de gigue de démarrage nettement meilleurs que ceux de BG-Green-RTO. A titre illustratif, pour $U_{ap} = 15\%$, TBS_{op}-Green-RTO et TBS-Green-RTO offrent des temps de réponse 6 fois plus faibles que BG-Green-RTO.

- Quand $U_{ap} \leq 15\%$, TBS_{op}-Green-RTO est le plus performant en termes de temps de réponse et de temps de gigue de démarrage. A titre illustratif, pour $U_{ap} = 3\%$, TBS_{op}-Green-RTO offre un temps de réponse et un temps de gigue supérieur de 10% à celui obtenu avec TBS-Green-RTO. Cependant, plus la charge temporelle des apériodiques est grande, plus l'écart observé entre TBS_{op}-Green-RTO et TBS-Green-RTO, au niveau du temps de réponse et du temps de gigue de démarrage, diminue.

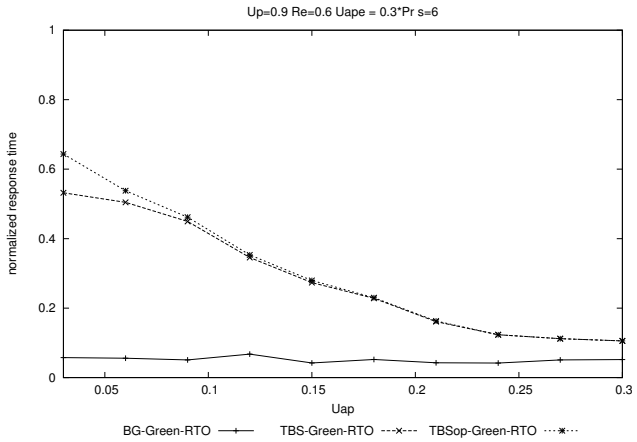


(a) Temps de réponse normalisé

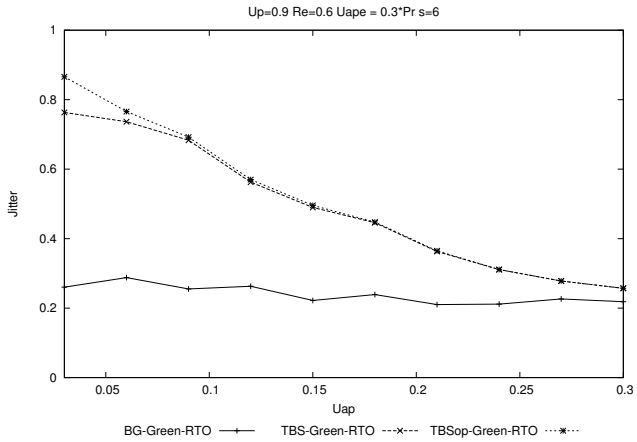


(b) Temps de gigue de démarrage normalisé

FIGURE 6.8 – $U_p = 0.9$, $R_e = 0.6$, $U_{ape} = 0.3P_r$, $s_i = 2$



(a) Temps de réponse normalisé



(b) Temps de gigue de démarrage normalisé

FIGURE 6.9 – $U_p = 0.9$, $R_e = 0.6$, $U_{ape} = 0.3P_r$, $s_i = 6$

- Avec tous les algorithmes, le temps de réponse et de gigue de démarrage sont d'autant plus longs quand le paramètre de pertes des tâches périodiques est grand. A titre illustratif, pour $U_{ap} = 15\%$, le temps de réponse subit une augmentation d'un facteur de 2 quand le paramètre de pertes passe de 2 à 6.

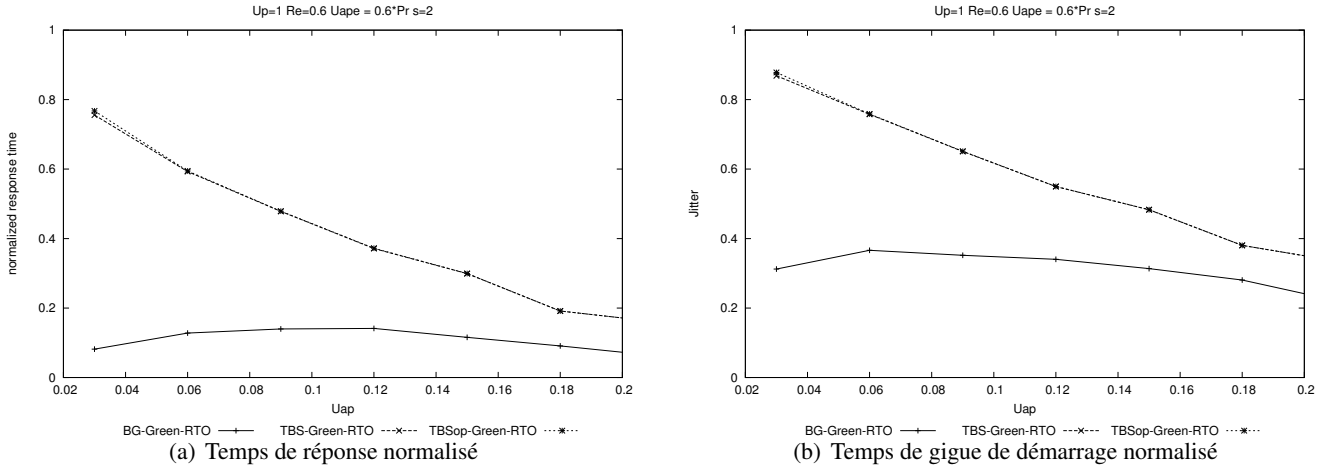
- Quand la charge temporelle des apériodiques augmente, les temps de réponse et les temps de gigue de démarrage augmentent et ce, avec tous les algorithmes. Cependant BG-Green-RTO reste toujours l'algorithme le moins performant en termes de temps de réponse : pour $U_{ap} = 30\%$, le temps de réponse est deux fois plus long avec BG-Green-RTO qu'avec TBS_{op}-Green-RTO et TBS-Green-RTO.

4.1.2 Système très fortement chargé

Dans cette partie, nous considérons que le système a une charge processeur maximale ($U_p = 1$) et qu'il est moyennement chargé énergétiquement ($R_e = 0.6$). Nous ajoutons à ce système des tâches apériodiques ayant une charge énergétique $U_{ape} = 0.6P_r$. Par conséquent, le système global devient surchargé énergétiquement.

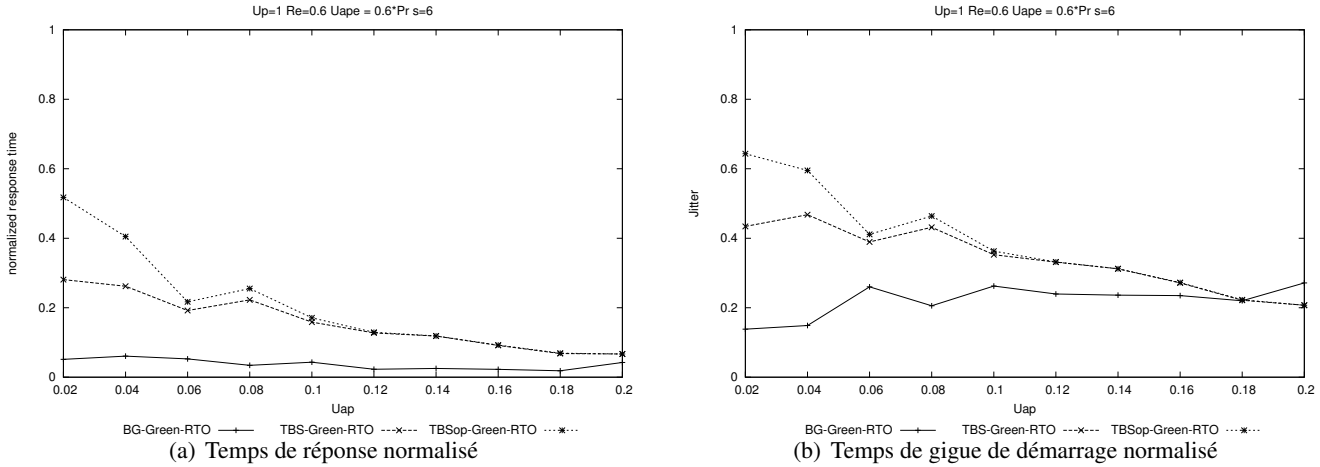
Nous remarquons sur les Figures 6.10(a), 6.11(a), 6.10(b) et 6.11(b) que :

- BG-Green-RTO offre le temps de réponse et le temps de gigue les plus médiocres et ce, quelle que soit la charge temporelle des tâches apériodiques.



(a) Temps de réponse normalisé (b) Temps de gigue de démarrage normalisé

FIGURE 6.10 – $U_p = 1$, $R_e = 0.6$, $U_{ape} = 0.6P_r$, $s_i = 2$



(a) Temps de réponse normalisé (b) Temps de gigue de démarrage normalisé

FIGURE 6.11 – $U_p = 1$, $R_e = 0.6$, $U_{ape} = 0.6P_r$, $s_i = 6$

- Quand $s_i = 2$, TBS_{op}-Green-RTO et TBS-Green-RTO offrent les mêmes performances en termes de temps de réponse et de temps de gigue de démarrage quelle que soit la charge temporelle des tâches apériodiques.

- Avec BG-Green-RTO, TBS_{op}-Green-RTO et TBS-Green-RTO, le temps de réponse et le temps de gigue de démarrage sont d'autant plus longs que le paramètre de pertes fixé est grand. A titre illustratif, pour $U_{ap} = 10\%$, le temps de réponse subit une augmentation d'un facteur de 2.2 quand le paramètre de pertes passe de 2 à 6.

- Pour $s_i = 6$, TBS_{op}-Green-RTO offre de meilleurs temps de réponse et temps de gigue de démarrage que TBS-Green-RTO pour $U_{ap} \leq 10\%$. Les largeurs de bande temporelle et énergétique U_S et U_{es} dépendent de U_p^* et de U_e^*. Par conséquent, U_S et U_{es} diminuent quand le paramètre de pertes des tâches périodiques est grand. Par conséquent, l'échéance fictive calculée est plus longue quand le paramètre de pertes est grand.

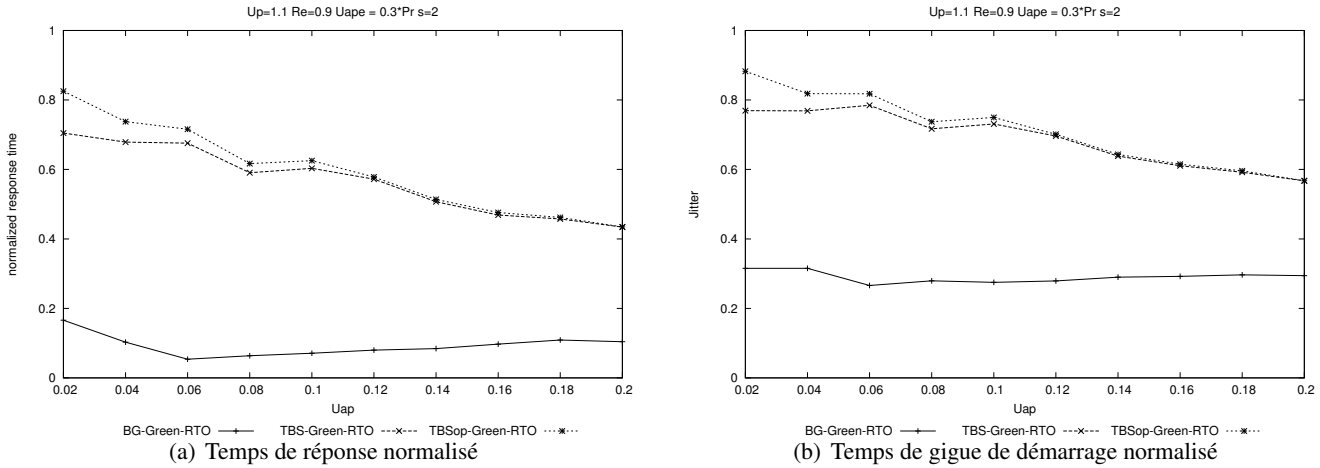
- TBS_{op}-Green-RTO est le plus performant en termes de temps de réponse et de temps de gigue de démarrage pour $U_{ap} \leq 10\%$. La différence de performance entre TBS-Green-RTO et TBS_{op}-Green-RTO dépend du calcul de l'échéance fictive. Ainsi TBS_{op}-Green-RTO se démarque de TBS-Green-RTO en calculant une échéance fictive plus courte si la contrainte énergétique est plus dominante que la contrainte

temporelle, c'est-à-dire si $\left\lceil \frac{C_k^a}{U_S} \right\rceil << \left\lceil \frac{\frac{E_k^a}{U_{es}} - E(r_k)}{P_r} \right\rceil$.

- Plus la charge temporelle des apériodiques est importante, plus le facteur total d'utilisation du processeur $U_p^* + U_{ap}$ augmente et s'approche de 100%. Par conséquent, tous les algorithmes offrent des temps de réponse et des temps de gigue de démarrage médiocres. Ainsi si le paramètre de pertes vaut 6 quand $U_{ap} = 20\%$, les temps de réponse normalisés et les temps de gigue de démarrage normalisé obtenus avec tous les algorithmes sont similaires.

4.1.3 Système surchargé temporellement

Dans cette partie, nous considérons que le système est en surcharge temporelle ($U_p = 1.1$) et qu'il est fortement chargé énergétiquement ($R_e = 0.9$). Nous ajoutons à ce système des tâches apériodiques ayant une charge énergétique $U_{ape} = 0.3P_r$. Par conséquent, le système global devient surchargé temporellement et énergétiquement.
Sur les Figures 6.12(a) et 6.12(b), nous remarquons que :



(a) Temps de réponse normalisé

(b) Temps de gigue de démarrage normalisé

FIGURE 6.12 – $U_p = 1.1$, $R_e = 0.9$, $U_{ape} = 0.3P_r$, $s_i = 2$

- BG-Green-RTO offre le temps de réponse et le temps de gigue les plus médiocres et ce, quelle que soit la charge temporelle. A titre illustratif, pour $U_{ap} = 12\%$, BG-Green-RTO présente un temps de réponse 6 fois plus long que celui obtenu avec TBS-Green-RTO et TBS_{op}-Green-RTO. De plus BG-Green-RTO attend deux fois plus de temps que TBS-Green-RTO et TBS_{op}-Green-RTO pour servir les tâches apériodiques en attente.

- TBS_{op}-Green-RTO offre de meilleurs temps de réponse et temps de gigue de démarrage que TBS-Green-RTO pour $U_p + U_{ap} < 122\%$. Quand la charge temporelle globale devient supérieure à 122% ($U_p + U_{ap} \geq 122\%$), TBS_{op}-Green-RTO et TBS-Green-RTO offrent les mêmes temps de réponse et temps de gigue de démarrage.

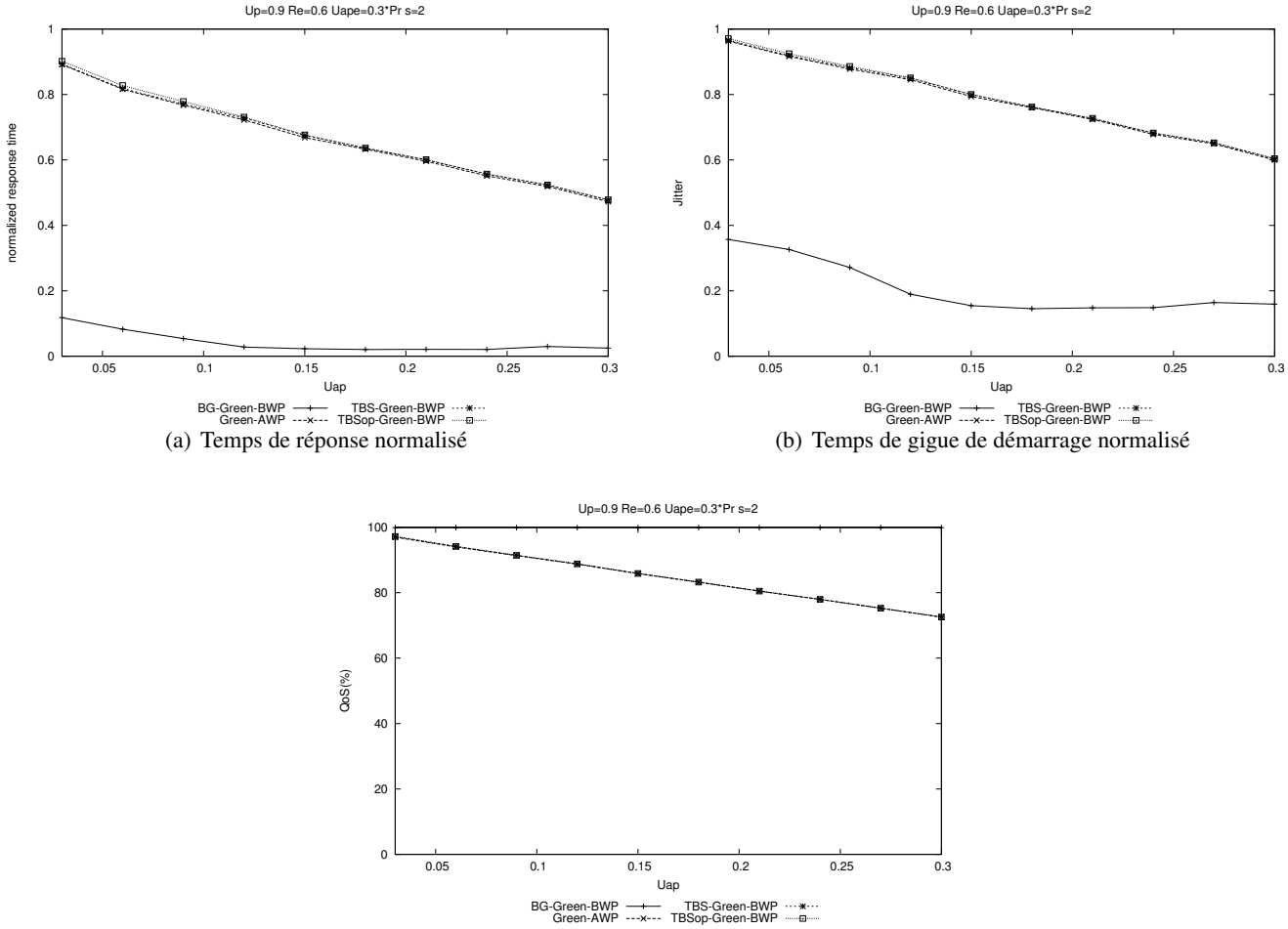
- Les algorithmes engendrent des temps de réponse et des temps de gigue de démarrage des tâches apériodiques de plus en plus longs quand la charge temporelle des tâches apériodiques augmente.

4.2 Etude avec Green-BWP

Dans cette partie, nous étudions les performances des serveurs qui se basent sur l'algorithme Green-BWP pour ordonnancer les tâches périodiques.

4.2.1 Système fortement chargé temporellement

Dans cette partie, nous considérons que le système est fortement chargé temporellement ($U_p = 0.9$) et moyennement chargé énergétiquement ($R_e = 0.6$). Nous ajoutons à ce système des tâches apériodiques ayant une charge énergétique $U_{ape} = 0.3P_r$. Par conséquent, le système global devient fortement chargé énergétiquement.



(a) Temps de réponse normalisé

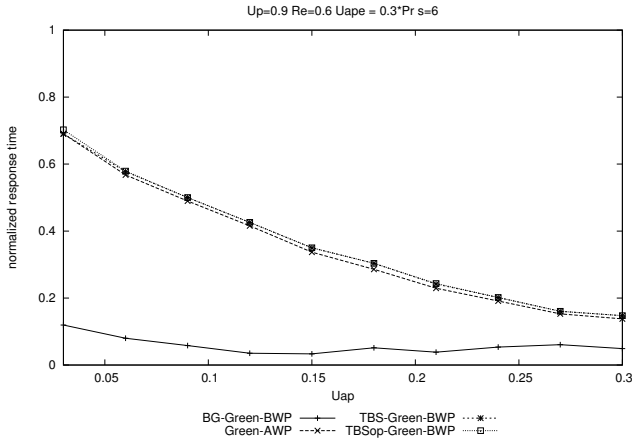
(b) Temps de gigue de démarrage normalisé

(c) Taux de respect des tâches périodiques

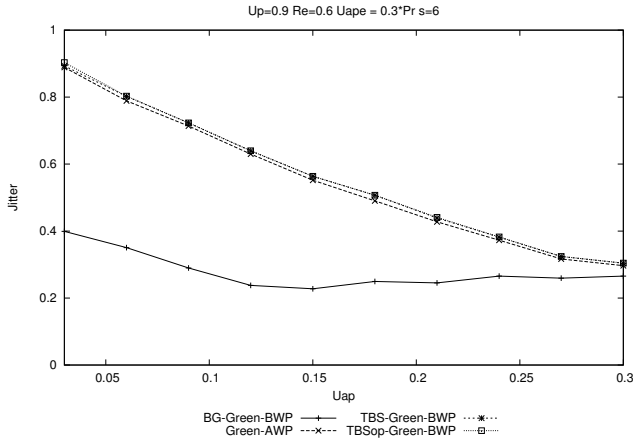
FIGURE 6.13 – $U_p = 0.9$, $R_e = 0.6$, $U_{ape} = 0.3P_r$, $s_i = 2$

Nous remarquons sur les Figures 6.13(a), 6.13(b), 6.14(a)et 6.14(b) que :

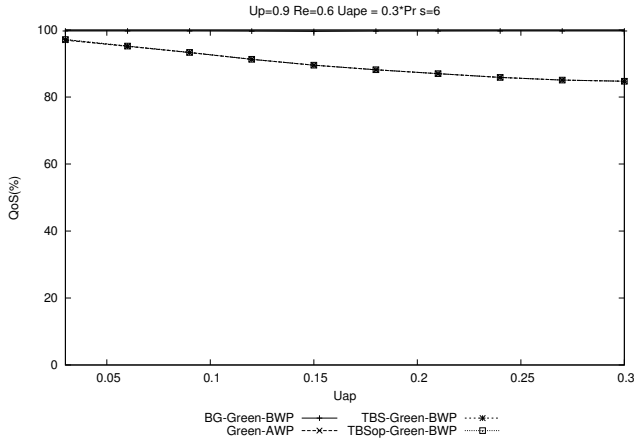
- Les temps de réponse et les temps de gigue de démarrage sont plus longs quand la charge temporelle des tâches apériodiques augmente.
- Plus le paramètre de pertes considéré est grand, plus le nombre de jobs rouges à exécuter est grand et donc le temps de réponse et le temps de gigue de démarrage des tâches apériodiques deviennent plus longs.
- TBS-Green-BWP et TBS_{op}-Green-BWP offrent les mêmes temps de réponse et temps de gigue de démarrage des tâches apériodiques. Dans ce cas de figure, le fait de calculer l'échéance fictive en prenant en compte le niveau d'énergie dans la batterie à l'instant de calcul, n'optimise pas le temps de réponse et le temps de gigue de démarrage.
- Par ailleurs, Green-AWP affiche un temps de réponse et un temps de gigue de démarrage très proches de TBS-Green-BWP et TBS_{op}-Green-BWP avec un écart visible de 1% pour $U_p + U_{ap} = 0.93$.



(a) Temps de réponse normalisé



(b) Temps de gigue de démarrage normalisé



(c) Taux de respect des tâches périodiques

FIGURE 6.14 – $U_p = 0.9$, $R_e = 0.6$, $U_{ape} = 0.3P_r$, $s_i = 6$

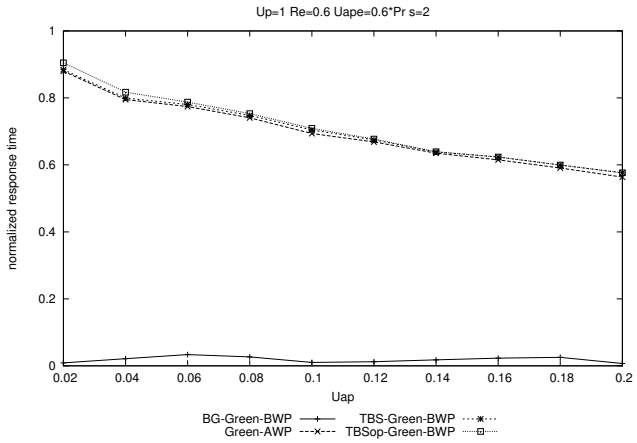
- L'avantage fondamental de Green-AWP par rapport à TBS-Green-BWP et TBS_{op}-Green-BWP est le fait qu'il ne sollicite pas le calcul de la laxité énergétique du système pour exécuter une tâche apériodique ($SlackEnergy(t)$). Ce calcul peut s'avérer coûteux quand le nombre de tâches périodiques est grand.

- BG-Green-BWP est l'algorithme offrant les performances les plus médiocres en termes de temps de réponse et de gigue de démarrage. Pour $U_{ap} = 12\%$, le système a une charge temporelle globale égale à $U_p + U_{ap} = 102\%$: pour $s_i = 2$ et $s_i = 6$, BG-Green-BWP présente respectivement un temps de réponse des tâches apériodiques 37 fois et 14 fois plus long qu'avec Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP.

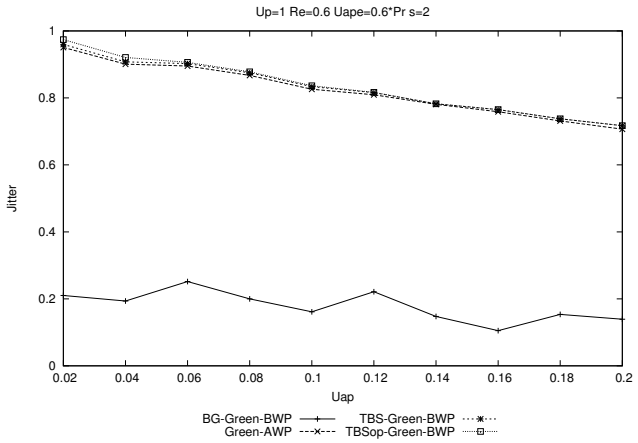
Avec BG-Green-BWP, les temps de réponse et les temps de gigue de démarrage se révèlent extrêmement longs. Cependant les Figures 6.13(c) et 6.14(c) montrent que c'est la méthode la plus performante en termes de QoS au niveau des tâches périodiques. En effet, il permet de garantir la meilleure QoS qui est de 100% dans un système non surchargé temporellement et énergétiquement. Cette QoS est indépendante de la charge temporelle des tâches apériodiques. Ceci est dû au fait qu'avec BG-Green-BWP, les jobs rouges et bleus sont toujours plus prioritaires que les tâches apériodiques. A contrario, avec les autres algorithmes, les tâches apériodiques sont plus prioritaires que les jobs bleus. Par conséquent, on peut remarquer que la QoS obtenue avec Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP est inférieure à 100%. A titre illustratif, pour $U_p + U_{ap} = 93\%$, la QoS obtenue avec Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP est de 97% puis celle-ci décroît à mesure que la charge temporelle des apériodiques augmente.

4.2.2 Système très fortement chargé temporellement

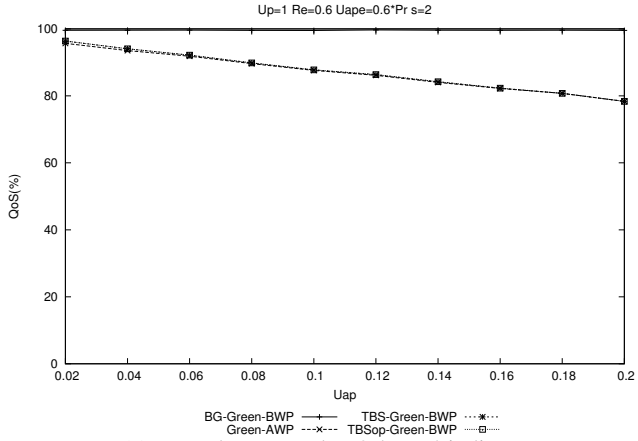
Dans cette partie, nous considérons que le système a une charge processeur maximale ($U_p = 1$) et est moyennement chargé énergétiquement ($R_e = 0.6$). Nous ajoutons à ce système des tâches apériodiques ayant une charge énergétique $U_{ape} = 0.6P_r$. Par conséquent, le système global devient surchargé énergétiquement.



(a) Temps de réponse normalisé



(b) Temps de gigue de démarrage normalisé

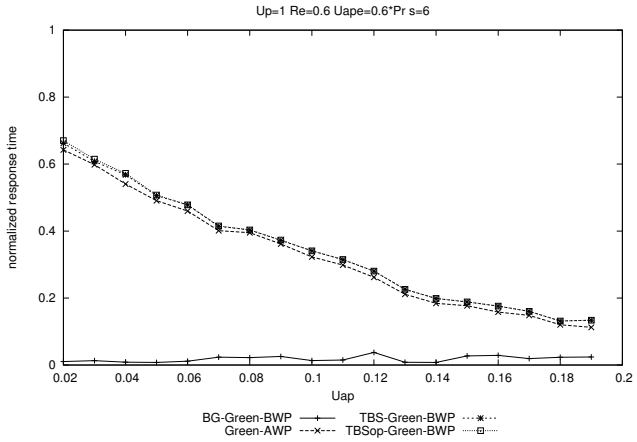


(c) Taux de respect des tâches périodiques

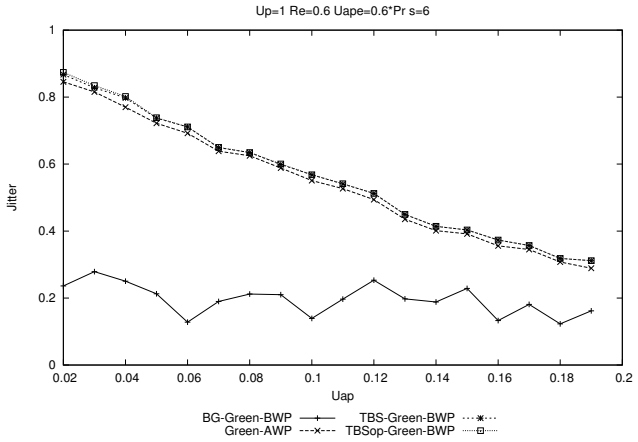
FIGURE 6.15 – $U_p = 1$, $R_e = 0.6$, $U_{ape} = 0.6P_r$, $s_i = 2$

Nous remarquons sur les Figures 6.15(a), 6.16(a), 6.15(b) et 6.16(b) que :

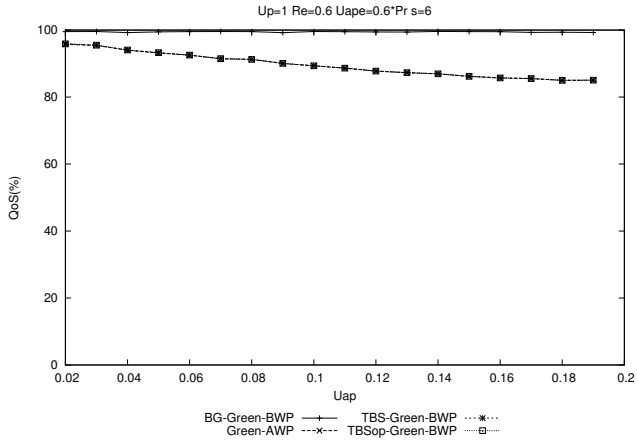
- Plus le paramètre de pertes considéré est grand, plus le temps de réponse et le temps de gigue de démarrage sont longs.
- Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP offrent les meilleurs temps de réponse et temps de gigue de démarrage des tâches apériodiques.
- Par ailleurs, Green-AWP et TBS-Green-BWP présentent des temps de réponse et des temps de gigue de démarrage légèrement inférieurs à TBS_{op}-Green-BWP, l'écart observé étant à peine de 1% pour $U_{ap} \leq 1\%$.
- BG-Green-BWP affiche un temps de réponse très long : le temps de réponse normalisé obtenu vaut $0.01 \cong 0$.
- BG-Green-BWP est l'algorithme le moins performant en termes de temps de réponse et de temps de gigue de démarrage. Pour $U_{ap} = 2\%$, le système a une charge temporelle globale égale à $U_p + U_{ap} = 102\%$ et BG-Green-BWP présente respectivement pour $s_i = 2$ et $s_i = 6$, un temps de réponse des tâches



(a) Temps de réponse normalisé



(b) Temps de gigue de démarrage normalisé



(c) Taux de respect des tâches périodiques

FIGURE 6.16 – $U_p = 1$, $R_e = 0.6$, $U_{ape} = 0.6P_r$, $s_i = 6$

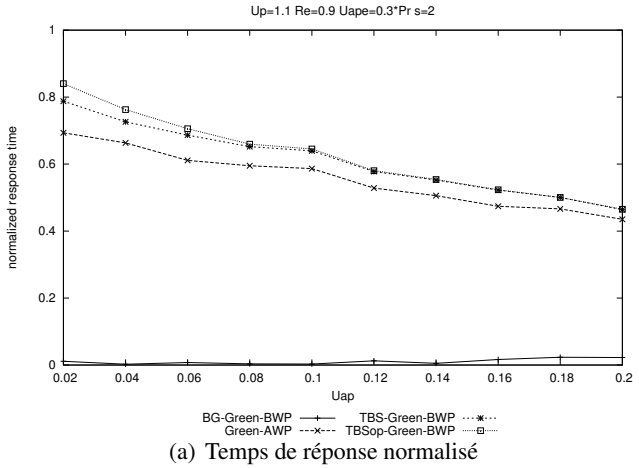
apériodiques 90 fois et 65 fois plus long qu'avec les autres algorithmes. De plus, pour $s_i = 2$, il présente un temps de gigue de démarrage 4 fois plus long qu'avec les autres algorithmes.

Même si les temps de réponse et les temps de gigue de démarrage se révèlent très longs avec BG-Green-BWP, les Figures 6.15(c) et 6.16(c) montrent que BG-Green-BWP est le plus performant en termes de taux de respect des tâches périodiques. En effet, comme le système est non surchargé temporellement et énergétiquement (en faisant abstraction aux charges temporelle et énergétique des tâches apériodiques), la QoS obtenue est de 100%. A contrario, avec les autres algorithmes, comme les tâches apériodiques sont plus prioritaires que les jobs bleus, la QoS avec Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP est la même et est inférieure à 100%. Elle vaut 96% quand le système global est surchargé temporellement et énergétiquement ($U_p + U_{ap} = 102\%$ et $R_e + U_{ape} = 1.2$). Ensuite, si la charge temporelle des tâches apériodiques augmente, la QoS a tendance à diminuer avec Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP.

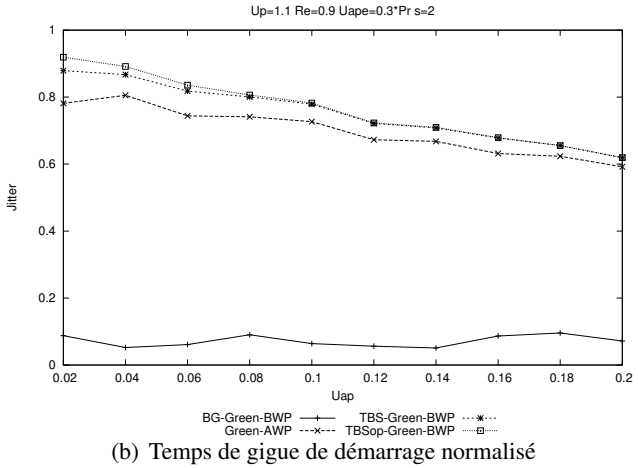
4.2.3 Système surchargé temporellement

Dans cette partie, nous considérons que le système est surchargé temporellement $U_p = 1.1$ et fortement chargé énergétiquement $R_e = 0.9$. Nous ajoutons à ce système des tâches apériodiques ayant une charge énergétique $U_{ape} = 0.3P_r$. Par conséquent, le système global devient surchargé temporellement et énergétiquement.
Sur la Figure 6.17(a), nous remarquons que :

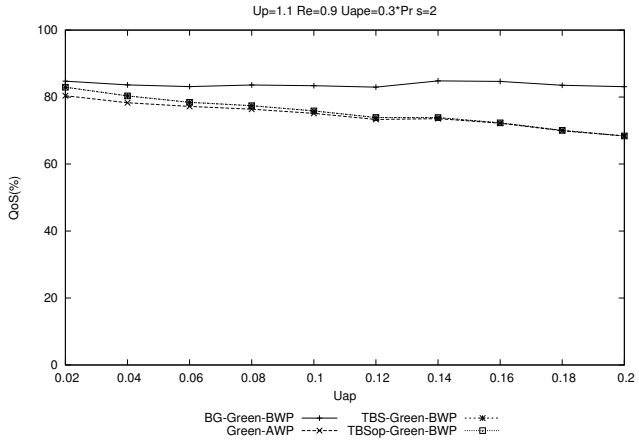
- BG-Green-BWP présente un temps de réponse normalisé quasi-nul. En utilisant BG-Green-BWP, il faut s'attendre à un temps de réponse infiniment long.



(a) Temps de réponse normalisé



(b) Temps de gigue de démarrage normalisé



(c) Taux de respect des tâches périodiques

FIGURE 6.17 – $U_p = 1.1$, $R_e = 0.9$, $U_{ape} = 0.3P_r$, $s_i = 2$

- BG-Green-BWP affiche le plus long temps de gigue de démarrage parmi les algorithmes étudiés dans cette partie. En effet, le temps de gigue de démarrage normalisé varie aux alentours de 0.1.
- Green-AWP s'avère beaucoup plus performant que BG-Green-BWP en termes de temps de réponse et temps de gigue de démarrage. En effet, pour $U_{ap} = 2\%$, il est le temps de réponse 70 fois plus court que BG-Green-BWP.
- Green-AWP affiche un temps de réponse plus long que ceux obtenus avec TBS_{op}-Green-BWP et TBS-Green-BWP. Pour $U_{ap} = 2\%$, l'écart entre le temps de réponse obtenu avec Green-AWP et TBS-Green-BWP est de 15%.
- Pour $U_{ap} < 10\%$, TBS_{op}-Green-BWP affiche les meilleurs temps de réponse et temps de gigue de démarrage qui sont légèrement supérieurs à ceux obtenus avec TBS-Green-BWP. A titre illustratif, pour $U_{ap} = 2\%$, TBS_{op}-Green-BWP présente un temps de réponse inférieur de 5% par rapport à celui obtenu avec TBS-Green-BWP.
- A partir de $U_p + U_{ap} \geq 120\%$, TBS-Green-BWP et TBS_{op}-Green-BWP ont les même performances en termes de temps de réponse et de temps de gigue de démarrage.
- Green-AWP affiche un temps de gigue de démarrage plus long que ceux obtenus avec TBS_{op}-Green-BWP et TBS-Green-BWP. Pour $U_{ap} = 2\%$, l'écart entre le temps de gigue de démarrage obtenu avec Green-AWP et TBS-Green-BWP est de 12%.

Sur la Figure 6.17(c), nous remarquons que :

- BG-Green-BWP est le plus performant en termes de QoS : il offre une QoS constante au niveau des tâches périodiques qui vaut 83%.

- TBS_{op}-Green-BWP et TBS-Green-BWP sont légèrement plus performants que Green-AWP, pour $U_p + U_{ap} \leq 120\%$. Pour $U_p + U_{ap} = 112\%$, la QoS est de 82% avec TBS_{op}-Green-BWP et TBS-Green-BWP et 80% avec Green-AWP.

- Quand la charge temporelle des tâches apériodiques augmente, ceci influence la QoS obtenue avec Green-AWP, TBS_{op}-Green-BWP et TBS-Green-BWP qui diminue progressivement. Pour une charge globale temporelle égale à $U_p + U_{ap} = 120\%$, Green-AWP, TBS_{op}-Green-BWP et TBS-Green-BWP affichent une QoS de 64%.

5 Synthèse

- Pour tous les algorithmes étudiés, le temps de réponse et le temps de gigue de démarrage des tâches apériodiques sont d'autant plus longs que la charge temporelle apériodique est importante ou que le paramètre de pertes appliqué au niveau des tâches périodiques est grand.

- La QoS observée au niveau des tâches périodiques, avec les algorithmes basés sur Green-RTO, est constante et dépend uniquement du paramètre de pertes de tâches périodiques constituant le système.

- BG-Green-BWP a l'avantage d'offrir la meilleure QoS au niveau des tâches périodiques. La QoS obtenue est indépendante de la charge temporelle et énergétique des tâches apériodiques. Ainsi si le système est en sous-charge temporelle et énergétique, la QoS obtenue est de 100%. Par contre, si le système est en surcharge temporelle et/ou énergétique, la QoS obtenue est inférieure à 100%.

- Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP offrent des QoS inférieures à 100% car ils permettent d'exécuter les tâches apériodiques en priorité devant les jobs bleus. Ainsi plus la charge temporelle et/ou énergétique des apériodiques augmente, plus la QoS se trouve dégradée.

- Si l'on considère un système non surchargé temporellement et énergétiquement :

- Dans le chapitre précédent nous avons proposé et décrit les algorithmes BG-EDeg, TBS-EDeg et TBS_{op}-EDeg. Ces algorithmes ordonnancent une configuration de tâches périodiques temps réel strictes conjointement à des tâches apériodiques non critiques en considérant les contraintes temporelles et énergétiques des tâches périodiques. Maintenant, dans ce chapitre, nous considérons une configuration de tâches périodiques temps réel fermes et nous établissons de nouveaux algorithmes basés sur Green-RTO et Green-BWP, permettant d'exploiter les sauts tolérés au niveau des jobs périodiques, pour réduire le temps de réponse des tâches apériodiques.

En effet, si nous nous référons aux résultats de simulation dans un système fortement chargé temporellement et moyennement chargé énergétiquement ($U_p = 0.9$ et $R_e = 0.6$) étudié dans le chapitre 5 et dans ce chapitre, nous pouvons constater que le temps de réponse et le temps de gigue de démarrage obtenus avec Green-RTO (cf. Partie 4.1.1 Figure 6.8(a) et 6.8(b)) sont moins longs que ceux obtenus avec EDeg (cf. Chapitre 5, Section 4, Figure 5.10(a) et Figure 5.10(b)) :

- TBS_{op}-Green-RTO présente un temps de réponse normalisé qui vaut 0.85 et un temps de gigue de démarrage normalisé qui vaut 0.93 pour $s_i = 2$ quand le système a une charge temporelle globale de $U_p + U_{ap} = 0.93$ alors que TBS_{op}-EDeg affiche un temps de réponse normalisé de 0.78 et un temps de gigue de démarrage de 0.85 quand $U_p + U_{ap} = 0.91$.

- Pour $s_i = 6$ (cf. Partie 4.1.1 Figure 6.9(a) et 6.9(b)), TBS_{op}-Green-RTO présente un temps de réponse normalisé qui vaut 0.4 et un temps de gigue de démarrage de 0.62 quand le système a une charge temporelle globale de $U_p + U_{ap} = 100\%$ alors que TBS_{op}-EDeg présente un temps de réponse normalisé qui vaut 0.17 et un temps de gigue de démarrage qui vaut 0.35.

- TBS_{op}-Green-RTO permet de servir et exécuter complètement les tâches apériodiques plus rapidement que TBS_{op}-EDeg.

- BG-Green-RTO offre de meilleurs temps de réponse et temps de gigue de démarrage que BG-EDeg pour $s_i = 2$.

Ceci dit, même si les temps de réponse et les temps de gigue de démarrage observés avec les algorithmes basés sur Green-RTO sont meilleurs que ceux basés sur EDeg, la QoS au niveau des tâches périodiques subit une dégradation avec Green-RTO ce qui n'est pas le cas avec EDeg qui permet d'exécuter 100% des tâches périodiques.
Pour remédier à cette dégradation de la QoS, l'utilisation des algorithmes basés sur Green-BWP peut être une alternative raisonnable permettant de minimiser les temps de réponse et les temps de gigue de démarrage des tâches apériodiques tout en garantissant une QoS au niveau des tâches périodiques acceptable.

- Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP offrent des temps de réponse et des temps de gigue de démarrage des tâches apériodiques similaires largement supérieurs à ceux obtenus avec BG-Green-BWP.

- Si l'on considère un système surchargé temporellement et/ou énergétiquement :

- Les algorithmes proposés dans ce chapitre présentent l'avantage de réagir correctement à une surcharge temporaire dû à un événement extérieur. En effet, ils offrent une QoS acceptable au niveau des tâches périodiques tout en minimisant les temps de réponse des tâches apériodiques. Ceci-dit, les algorithmes basés sur Green-RTO offrent la QoS minimale qui dépend directement du paramètre de pertes alors que ceux basés sur Green-BWP permettent de garder une QoS acceptable supérieure à celle obtenue avec Green-RTO tout en minimisant au mieux les temps de réponse des tâches apériodiques.

- Parmi les algorithmes basés sur Green-RTO, TBS_{op}-Green-RTO offre les meilleurs temps de réponse et temps de gigue de démarrage des tâches apériodiques qui deviennent similaires à ceux obtenus avec TBS-Green-RTO si la contrainte temporelle est dominante.

- TBS_{op}-Green-BWP offre les meilleurs temps de réponse et des temps de gigue de démarrage des tâches apériodiques parmi les algorithmes étudiés basés sur Green-BWP. Cependant, les performances TBS-Green-BWP peuvent s'avérer identiques à ceux de TBS_{op}-Green-BWP dans le cas où le système est plus contraint temporellement qu'énergétiquement.

- Green-AWP offre un temps de réponse et temps de gigue de démarrage des tâches apériodiques légèrement inférieurs à ceux obtenus avec TBS-Green-BWP et TBS_{op}-Green-BWP. Ceci dit, plus le système est surchargé temporellement, plus l'écart entre Green-AWP et les algorithmes TBS-Green-BWP et TBS_{op}-Green-BWP augmente.

- Cependant avec Green-AWP, seules les tâches périodiques sollicitent le calcul de la laxité énergétique du système avant leur exécution, alors qu'avec TBS-Green-BWP et TBS_{op}-Green-BWP, ce calcul est sollicité à chaque fois qu'une tâche périodique ou apériodique veut s'exécuter. Par conséquent, Green-AWP présente l'avantage d'être simple à réaliser et de ne pas engendrer des surcoûts liés au calcul de la laxité énergétique du système.

- Nous avons pu remarquer que les algorithmes Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP offraient de meilleurs temps de réponse et temps de gigue de démarrage que TBS-Green-RTO et TBS_{op}-Green-RTO. A titre illustratif, sur la Figure 6.9(a), pour $U_p + U_{ap} = 120\%$, TBS-Green-RTO et TBS_{op}-Green-RTO affichent un temps de réponse normalisé de 0.4 alors que Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP affichent un temps de réponse normalisé de 0.7 qui est plus proche de 1. En effet, Green-BWP présente l'avantage d'exécuter les jobs bleus quand c'est possible ce qui n'est pas le cas de Green-RTO qui n'exécute que les jobs rouges à des intervalles de temps réguliers. Ceci dit, avec Green-BWP, si un job bleu d'une tâche est complètement exécuté, le prochain job de cette tâche est toujours bleu. Par conséquent, si une tâche apériodique arrive, comme elle est toujours plus prioritaire que les jobs bleus prêts, elle pourra être exécutée s'il y a aucun job rouge prêt. C'est pourquoi, on peut observer que les temps de réponse et les temps de gigue de démarrage obtenus avec Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP sont meilleurs que ceux obtenus avec TBS-Green-RTO et TBS_{op}-Green-RTO pour un même profil temporel et énergétique du système.

Nous résumons l'étude faite par les tableaux 6.1 et 6.2. Dans ces tableaux, un "*" représente les bonnes performances d'un algorithme d'ordonnancement par rapport à un critère donné.

Critères	BG-RTO	TBS-RTO	TBS_{op}-RTO
Complexité algorithmique	***	**	*
Surcoût de calcul	***	*	*
Temps de réponse	*	**	***
Temps de blocage avant de commencer l'exécution	*	**	***
QoS au niveau des tâches périodiques	*	*	*

TABLE 6.1 – Comparatifs de performances avec Green-RTO

Critères	BG-Green-BWP	Green-AWP	TBS-Green-BWP	TBS_{op}-Green-BWP
Complexité algorithmique	***	***	**	*
Surcoût de calcul	***	***	*	*
Temps de réponse	*	**	**	***
Temps de blocage avant de commencer l'exécution	*	**	**	***
QoS au niveau des tâches périodiques	***	**	**	**

TABLE 6.2 – Comparatifs de performances avec Green-BWP

6 Conclusion

Dans ce chapitre, nous avions pour objectif d'exploiter les pertes au niveau des tâches périodiques dans le but de minimiser le temps de réponse des tâches apériodiques. Dans un premier temps, nous avons présenté trois algorithmes basés sur l'algorithme Green-RTO permettant d'exécuter conjointement des tâches apériodiques avec les jobs rouges des tâches périodiques à savoir BG-Green-RTO, TBS-Green-RTO et TBS_{op}-Green-RTO. Ces algorithmes présentent l'avantage de minimiser les temps de réponse des tâches apériodiques en les exécutant pendant les temps creux libérés par la non exécution des jobs bleus. Cependant ils ont l'inconvénient de fonctionner en mode le plus dégradé car ils ne tentent jamais d'exécuter les jobs bleus. Ceci nous a amenés à proposer quatre nouveaux algorithmes basés sur l'algorithme Green-BWP à savoir BG-Green-BWP, Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP. Ils permettent d'exécuter les tâches apériodiques conjointement aux jobs rouges et bleus des tâches périodiques. Par conséquent la QoS observée au niveau des tâches périodiques est améliorée par rapport à celle obtenue avec les algorithmes basés sur Green-RTO.

Ensuite, nous avons mené des simulations dans le but d'évaluer les différentes performances des algorithmes proposés en termes de QoS, de temps de réponse et de temps de gigue de démarrage des tâches apériodiques. Ceci nous a permis de valider que les algorithmes TBS_{op}-Green-RTO et TBS_{op}-Green-BWP sont les algorithmes les plus performants en termes de temps de réponse et de temps de gigue des tâches apériodiques parmi les algorithmes basés sur Green-RTO et Green-BWP quand le système est plus contraint énergétiquement que temporellement. Quand la contrainte temporelle du système est plus dominante que la contrainte énergétique, TBS_{op}-Green-RTO et TBS-Green-RTO affichent les même performances en termes de temps de réponse. Idem pour TBS_{op}-Green-BWP et TBS-Green-BWP.

De plus, dans un système non surchargé temporellement et/ou énergétiquement, Green-AWP offre un temps de réponse très proche de celui obtenu avec TBS_{op}-Green-BWP. Néanmoins, il présente un temps de réponse

légèrement inférieur quand le système est surchargé temporellement et/ou énergétiquement. Cependant, comme il engendre moins de calculs de la laxité énergétique que les algorithmes basés sur le principe TBS, cette approche présente des atouts importants du point de vue de l'implémentation et de sa mise en oeuvre.

Les algorithmes les plus médiocres en termes de temps de réponse et de temps de démarrage pour les tâches apériodiques sont BG-Green-RTO et BG-Green-BWP. Cependant, ils ont l'avantage d'être simple à réaliser et surtout BG-Green-BWP permet d'offrir une meilleure QoS possible au niveau des tâches apériodiques.

Finalement, les questions clés que doit se poser tout concepteur d'application temps réel dans le contexte d'un système autonome en énergie sont les suivantes :

1. Est-il préférable pour l'application de privilégier le temps de réponse moyen des tâches apériodiques sur la Qualité de Service des tâches périodiques ou l'inverse ?

2. Quel est le niveau de charge temporelle et énergétique du système ?

Selon la réponse à ces questions, il conviendra de se référer à la synthèse des résultats de simulation figurant à la fin de ce chapitre.

Conclusion générale

Contexte de l'étude

De nos jours, les réseaux de capteurs prennent de plus en plus de place dans divers domaines dont celui de la surveillance de l'environnement, du suivi médical, de la domotique, etc.. En effet, ces systèmes embarqués s'intègrent de plus en plus dans les applications de tous les jours, et les contraintes de temps sont le plus souvent associées à des contraintes d'encombrement, de taille mémoire, de capacité énergétique et de puissance de calcul. Nos travaux se sont dirigés vers les systèmes autonomes devant garantir une autonomie durable sinon perpétuelle en fonctionnant à base d'énergie renouvelable. Ceci implique de trouver de nouvelles orientations pour les RTOS en termes d'ordonnancement et de gestion de puissance de façon à gérer les cas de surcharges temporelles et/ou énergétiques dans des systèmes monoprocesseur autonomes. Dans le cadre de cette thèse, nous avons proposé des nouvelles solutions d'ordonnancement en-ligne de tâches temps réel présentant des contraintes de QoS et d'énergie. Ces travaux permettent de répondre aux besoins de nouvelles applications temps réel pour lesquelles l'ordonnancement temps réel à contraintes strictes est trop restrictif et n'est pas adapté aux systèmes soumis à de fortes contraintes énergétiques. Dans ce contexte, nos contributions se sont articulées en deux axes d'études.

Contributions

Dans un premier axe d'étude, nous avons étudié l'ordonnancement de tâches périodiques dans des systèmes autonomes dans le but de gérer les cas de surcharge. Nous avons alors proposé deux algorithmes, Green-RTO et Green-BWP et nous les avons comparés à l'algorithme EDeg. Par la suite d'une étude de simulation, nous avons constaté que dans le cas d'un système en sous-charge énergétique et temporelle, EDeg et Green-BWP sont les algorithmes les plus performants en termes de QoS. Cependant, l'avantage de Green-RTO et Green-BWP par rapport à EDeg réside dans le fait qu'ils ont besoin d'une batterie 9 à 10 fois plus petite que celle requise par EDeg pour garantir le bon fonctionnement du système.

Par ailleurs, dans le cas d'un système en surcharge temporelle et/ou énergétique, EDeg ne peut pas être utilisé car il pourrait engendrer de graves conséquences vis-à-vis du système. En effet, EDeg ne contrôle pas le nombre et le type de jobs rejetés par manque de temps de traitement ou d'énergie. Green-RTO et Green-BWP, quant à eux, traitent le problème de surcharge en contrôlant nécessairement les jobs à abandonner de tout en respectant un seuil de QoS. Green-RTO garantit toujours une QoS minimale quel que soit le profil temporel ou énergétique du système. Il a l'avantage d'avoir un overhead en termes de calcul de la laxité énergétique, 4 fois plus faible que celui observé avec Green-BWP et EDeg. Ceux-ci présentent en effet des overheads supérieurs à Green-RTO et engendrent tous les deux des gaspillages d'énergie et de temps dûs à des violations d'échéance en cas de surcharge.

En outre, Green-BWP permet d'améliorer la QoS obtenue avec Green-RTO en essayant de maximiser le nombre de jobs exécutés suivant le profil énergétique et temporel du système. Par conséquent, Green-BWP a le principal avantage de s'adapter aux contraintes temporelles et énergétiques et permet d'offrir la meilleure QoS possible.

Green-BWP s'avère être l'ordonnanceur offrant la meilleure QoS par rapport au profil du système. Cependant, comme les jobs bleus sont ordonnancés selon EDF, il présente un défaut de stabilité. Nous avons alors présenté deux nouvelles heuristiques à savoir, Green-BWP-MS et Green-BWP-LF, permettant d'améliorer la stabilité du système en spécifiant de nouvelles politiques d'ordonnancement des jobs bleus.

Dans un deuxième axe d'étude, nous avons étudié l'ordonnancement de tâches apériodiques non critiques conjointement aux tâches périodiques dans des systèmes autonomes en énergie. Le but étant de gérer les cas de surcharge tout en minimisant le temps de réponse des tâches apériodiques.

Nous avons considéré, dans un premier temps, un système temps réel à contraintes strictes et nous avons décrit de nouveaux ordonnanceurs BG-EDeg, TBS-EDeg et TBS_{op}-EDeg. Ces algorithmes ont été conçus pour traiter le problème de l'ordonnancement conjoint de tâches périodiques et apériodiques en garantissant le respect des contraintes temporelles et énergétiques des tâches périodiques. Notre objectif étant de minimiser le temps de réponse des tâches apériodiques, les résultats de simulation nous ont permis de vérifier que TBS-EDeg et TBS_{op}-EDeg sont les plus performants en termes de temps de réponse et temps de gigue de démarrage. BG-EDeg, a le seul avantage d'être simple à réaliser, mais il affiche des temps de réponse et de gigue de démarrage élevés, d'autant plus que la charge temporelle et/ou énergétique du système est grande.

Nous avons présenté, dans un deuxième temps, trois algorithmes basés sur Green-RTO permettant d'exécuter conjointement des tâches apériodiques avec les jobs rouges des tâches périodiques à savoir BG-Green-RTO, TBS-Green-RTO et TBS_{op}-Green-RTO. Ces algorithmes présentent l'avantage d'exploiter les temps creux libérés par la non exécution des jobs bleus, en vue de minimiser les temps de réponse des tâches apériodiques. Cependant ils ont l'inconvénient de fonctionner dans le mode le plus dégradé car ils ne tentent jamais d'exécuter les jobs bleus. Ceci nous a amenés à proposer quatre nouveaux algorithmes basés sur Green-BWP à savoir BG-Green-BWP, Green-AWP, TBS-Green-BWP et TBS_{op}-Green-BWP. Ces nouveaux algorithmes permettent d'exécuter les tâches apériodiques conjointement aux jobs rouges et bleus des tâches périodiques tout en essayant d'offrir la meilleure QoS possible.

Les simulations nous ont permis de valider que TBS_{op}-Green-RTO et TBS_{op}-Green-BWP sont les algorithmes les plus performants en termes de temps de réponse et de temps de gigue des tâches apériodiques parmi les algorithmes basés sur Green-RTO et Green-BWP quand le système est plus contraint énergétiquement que temporellement. Quand la contrainte temporelle du système est plus dominante que la contrainte énergétique, TBS_{op}-Green-RTO et TBS-Green-RTO affichent les même performances en termes de temps de réponse, de même que pour TBS_{op}-Green-BWP et TBS-Green-BWP.

Par ailleurs, dans un système non surchargé temporellement et/ou énergétiquement, Green-AWP offre un temps de réponse très proche de celui obtenu avec TBS_{op}-Green-BWP. Néanmoins, il présente un temps de réponse légèrement inférieur quand le système est surchargé temporellement et/ou énergétiquement. Cependant, comme il engendre moins de calcul de la laxité énergétique que les algorithmes basés sur le principe TBS, cette approche présente des atouts importants du point de vue de l'implémentation et de sa mise en oeuvre.

Perspectives

Implémentation sur un RTOS

Les travaux de recherche futurs reposent sur l'intégration des ordonnanceurs développés dans les chapitres 3,4,5 et 6 dans un RTOS comme CLEOPATRE [GMSC03] par exemple. Il faudrait que l'ordonnanceur puisse connaître le niveau d'énergie dans la batterie (ou disposer tout du moins d'une estimation de celui-ci) par le biais d'un module logiciel s'interfaçant avec le matériel pour lui fournir cette information.

Extension à un modèle de système et à un modèle de tâches moins restrictifs

Comme les réserves d'énergies fossiles ne sont pas éternelles, l'utilisation des énergies renouvelables se révèle être une alternative qui mérite d'être exploitée pour alimenter un bon nombre de systèmes. L'étude proposée ici est un travail pionnier dans ce domaine. Nous avons considéré un modèle de tâches assez restrictif celui de tâches périodiques indépendantes, à échéances sur requêtes et pour lesquelles une consommation énergétique est connue. Cependant, nous avons relâché des hypothèses présentes dans des travaux existants, de façon à avoir un modèle plus réaliste. En particulier, rappelons que l'algorithme LSA [MBT+06] suppose que la demande énergétique d'une tâche est proportionnelle à son temps d'exécution. L'algorithme proposé par Allavena et Mossé [AM01] suppose quant à lui que toutes les tâches ont une période commune.

Notre étude mérite d'être étendue à un modèle de configurations de tâches plus fidèle aux applications réelles. En effet, nous considérons des tâches périodiques indépendantes. Cependant dans les applications

réelles les tâches périodiques sont rarement indépendantes. Cette nouvelle hypothèse rajouterait au système de nouvelles contraintes, de type précédence ou synchronisation.

En outre, il conviendrait également d'étendre ces travaux en utilisant l'algorithme ED-H, récemment proposé par Chetto [Che14]. Cet algorithme prouvé optimal est une variante de EDF dédié aux systèmes temps réel soumis à des contraintes énergétiques.

Enfin, dans notre étude, nous supposons que la puissance reçue par la source d'énergie est constante sur une hyperpériode. Il conviendrait de la relaxer même si elle n'est pas si irréaliste dans le sens où sur un intervalle de temps de l'ordre de quelques millisecondes, les variations de puissance reçue par la source ont une dynamique lente. Il serait donc intéressant de considérer une puissance variable.

Liste de publications

1. M. Abdallah, M. Chetto, A. Queudet et R. Hage Chehade, **Quality of service facilities for firm real-time energy harvesting systems**. Proceedings of "The IEEE International Conference on Electronics, Circuits, and Systems (ICECS) ", beyrouth, Liban (2011). (IEEE Xplore)

2. M. Abdallah, M. Chetto et A. Queudet, **Scheduling with Quality of Service requirements in Real-Time Energy Harvesting sensors** . Proceedings of "The IEEE International Conference on Green Computing and Communications" (GreenCom 2012), Besançon, France (2012). (Taux d'acceptation de 35.9%) (IEEE Xplore)

3. M. Abdallah, M. Chetto et A. Queudet, **Energy-aware schedulers for Real-Time Energy Harvesting systems with Quality of Service requirements**. Proceedings of "The International Conference on Advances in Computational Tools for Engineering Applications" (ACTEA2012), Beyrouth, Liban (2012). (IEEE Xplore)

4. M. Abdallah, M. Chetto, A. Queudet et R. Hage Chehade,**Stability and Robustness issues in Real-time Sustainable Wireless Sensors**. Proceedings of the "The IEEE International Conference on Green Computing and Communications" (GreenCom 2013), Beijing, China (2013). (Taux d'acceptation de 39%) (IEEE Xplore)

Bibliographie

[AA05] T. AlEnawy and H. Aydin. Energy-constrained scheduling for weakly-hard real-time systems. In *IEEE Real-Time Systems Symposium*, 2005. 60

[AM01] A. Allavena and D. Mossé. Scheduling of frame-based embedded systems with rechargeable batteries. *Workshop on Power Management for Real-Time and Embedded Systems (in conjunction with RTAS)*, 2001. 52, 170

[AMMMA01] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Euromicro Conference on Real-Time Systems*, pages 225–232, 2001. 52

[AP91] A.Dorseuil and P. Pillot. *Le temps-réel en milieu industriel, concepts, environnements,multitâches*. Bordas, 1991. 21

[BAM00] L. Benini, A.Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 8, n°3*, pages 299–316, 2000. 51

[BB97] G. Bernat and A. Burns. Combining (n,m)-hard deadlines and dual-priority scheduling. In *the 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 46–57. IEEE, 1997. 39

[BBL01] G. Bernat, A. Burns, and A. Llamosi. Weakly-hard real-time systems. *IEEE Transactions on Computers*, 50 :308–321, 2001. 20

[BMR90] S. Baruah, A. Mok, , and L. Rosier. The preemptive scheduling of sporadic real-time tasks on one processor. *Proceedings of the 11th Real-Time Systems Symposium*, 1990. 29

[BPB+00] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Stringini. The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture*, 46 :305–325, 2000. 37

[BS93] G. Buttazzo and J. Stankovic. Red robust earliest deadline scheduling. In *The 3rd International Workshop on Responsive Computing Systems*, 1993. 38

[BS99] G.-C. Buttazzo and F. Sensini. Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments. *IEEE Transactions on Software Engineering*, pages 22–32, 1999. 35

[BTM+07] S.P. Beeby, R.N. Torah, M.J.Tudor, P.Glynne-Jones, T.O. Donnel, C.R. Saha, and S. Roy. A micro electromagnetic generator for vibration energy harvesting. *Institute of physics Publishing, J. of Micromech. Microeng*, 2007. 49

[CB97] M. Caccamo and G. Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsivess. In *the 18th IEEE Real-Time Systems Symposium (RTSS'97)*. IEEE, 1997. 40, 68, 144, 145

[CC89] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, pages 1261–1269, 1989. 30, 32, 33, 34, 41, 58

[CC05] R. Casas and O. Casas. Battery sensing for energy-aware system design. In *Computer, vol.38, no. 11*, pages 48–54, 2005. 46

[CDK94] G. Coulouris, J. Dollimore, and T. Kindberg. Distributed systems-concepts and design. *2nd Ed, Addison-Wesley Publishers Ltd*, 1994. 25

[CDKZ00] F. Cottet, J. Delacroix, C. Kaiser, and Z.Mammeri. *Ordonnancement temps réel*. Ed. Hermes, 2000. 22

[CG09] M. Chetto and H. El Ghor. Real-time scheduling of periodic tasks in a monoprocessor system with a rechargeable battery. *17th IEEE International Symposium on Real Time Systems (RTSS 2009), wip session*, 2009. 53, 57, 58, 122

[CHB79] R.-H. Campbell, K.-H. Horton, and G.-G. Belford. Simulations of a fault-tolerant deadline mechanism. *Digest of papers FTCS-9*, pages 95–101, 1979. 38

[Che14] M. Chetto. Optimal scheduling for real-time jobs in energy harvesting computing systems. *IEEE Transactions on Emerging Topics in Computing (2014)*, 2014. 171

[Dar03] M. Dardenne. Ordonnancement à réservations dans un linux temps-réel. *Mémoire Ingénieur Civil en Informatique, Département d'ingénierie informatique, Université catholique de Louvain*, 2003. 37

[DCJ+07] G. Despesse, J.J Chaillout, T. Jager, F. Cardot, and A. Hoogerwerf. Innovative structure for mechanical energy scavenging. *International Solid-State Sensors, Actuators and Microsystems Conference (TRANSDUCERS 2007)*, pages 895–898, 2007. 49

[Der74] M.-L. Dertouzos. Control robotics : the procedural control of physical processes. *Information Processing 74, North-Holland Publishing Compagny*, pages 2 :237–250, 1974. 28, 29

[DlJ03] N. J. Dudney and Y. l. Jang. Analysis of thin-film lithium batteries with cathodes of 50 nm to 4 mm thick licoo2. In *Journal of Power Sources, Vol. 119-121*, pages 300–304, 2003. 46

[DM89] M. Dertouzos and A. Mok. Multiprocessor on-line scheduling of hard real-time tasks. *IEEE Transactions on Software Engineering*, 1989. 29

[DW95] R. Davis and A. Wellings. Dual priority scheduling. In *the 16th IEEE Real-Time Systems Symposium (RTSS'95)*. IEEE, 1995. 39

[Ely91] N. Elyounsi. Ordonnancement et reconfiguration dynamique dans un système temps-réel réparti à contraintes strictes. *Thèse de Doctorat, Ecole Supérieure de Mécanique, Université de Nantes*, 1991. 104

[GCC11] H. El Ghor, M. Chetto, and R. Hage Chehade. A real-time scheduling framework for embedded systems with environmental energy harvesting. *Computers & Electrical Engineering*, 37(4) :498–510, 2011. 59

[GCC13] H. El Ghor, M. Chetto, and R. Hage Chehade. A nonclairvoyant real-time scheduler for ambient energy harvesting sensors. *International Journal of Distributed Sensor Networks*, 2013. 59

[Gho12] H. El Ghor. *Ordonnancement monoprocesseur pour les applications temps réel récupérant l'énergie ambiante*. Thèse de Doctorat de l'Université de Nantes, 2012. 59

[GMSC03] T. Garcia, A. Marchand, and M. Silly-Chetto. Cleopatre : a r&d project for providing new real-time functionalities to linux/rtai. 2003. 170

[GN06] B. Gaujal and N. Navet. *Ordonnancement temps réel et minimisation de la consommation d'énergie*, volume 2 of *Systèmes temps réel*. Hermès, 2006. 51

[Gru02] F. Gruian. Energy-centric scheduling for real-time systems. *PhD thesis, Lund Institute of Technology*, 2002. 51

[HPP09] D Hardy, T Piquet, and I Puaut. Using bypass to tighten wcet estimates for multi-core processors with shared instruction caches. In *Real-Time Systems Symposium*, pages 21–25, 2009. 23

[HR95] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Transactions on Computers*, pages 44(4),1443–1451, 1995. 39

[ISO95] ISO. Open distributed processing - quality of service framework - basic framework. ISO/IEC 10746-2, International Organization for Standardization, 1995. 20

[Jac55] J.-R. Jackson. Scheduling a production line to minimize maximum tardiness. *Research Report 43, Management Science Research Project, University of California, Los Angeles*, 1955. 28

[JNC+89] E.-D. Jensen, J.-D. Northcutt, R.-K. Clark, S.-E. Shipman, F.-D. Reynolds, D.-P. Maynard, and K.-P. Loepfere. Alpha : An operating system for the mission-critical integration and operation of large, complex, distributed real-time systems - an overview. *OSMCC*, 1989. 37

[KKPG98] J. Kymissis, C. Kendall, J. Paradiso, and N. Gershenfeld. Parasitic power harvesting in shoes. In *IEEE International conference on wearable computing*, pages 132–139, 1998. 48

[KS93] G. Koren and D. Shasha. d^{over} : An optimal online scheduling algorithm for overloaded realtime systems. In *IEEE Real-Time Systems Symposium*, pages pages 290–299. IEEE, 1993. 38

[KS95] G. Koren and D. Shasha. Skip-over algorithms and complexity for overloaded systems that allow skips. *inproceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, 1995. 4, 40, 57, 60, 61, 62, 69

[Lab74] J. Labetoulle. Un algorithme optimal pour la gestion des processus en temps réel. *Revue Francaise d'Automatique, Informatique et Recherche Opérationnelle*, pages 11–17, 1974. 28

[LIF10] THERMO LIFE. Technical data and typical parameters, technical data and performance of prototype. *Thermo Life Energy Corp. Laboratory*, 2010. 49

[Liu00] J. W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1 edition, 2000. 20

[LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM, 20(1)*, pages 46–61, 1973. 22, 27, 29, 52

[LLB+94] J.-W.-S. Liu, K.-J. Lin, R. Bettati, D. Hull, and A. Yu. Use of imprecise computation to enhance dependability of real-time systems. *Gary M. Koob and Clifford G. Lau, editors, Foundations of Dependable Computing : Paradigms for Dependable Applications- Kluwer Academic Publishers*, pages 157–182, 1994. 39

[LLN87] J.W.S. Liu, J.K. Lin, and S. Natarajan. Scheduling algorithms for multiprogramming in a hard real-time environment. *the 8th real-time system symposium*, pages 252–260, 1987. 39

[LLS+91] Jane W.-S. Liu, Kwei-Jay Lin, Wei Kuan Shih, Albert Chuang shi Yu, Jen-Yao Chung, and Wei Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, pages 24(5) :58–68, 1991. 39

[LM80] J. Leung and M. Merril. A note on preemprive scheduling of periodic real-time tasks. *Information Processing Letters*, pages 11(3) : 115–118, 1980. 26

[LNL87] J.K. Lin, S. Natarajan, and J.W.S. Liu. Condor : A distributed system making use of imprecise results. In *COMPSAC'87*, 1987. 39

[LRT92] J.-P. Lehozcky and S. Ramos-Thuel. An optimal algorithm for scheduling soft aperiodic tasks in fixed-priority preemptive systems. *inproceedings of the 13th IEEE Real-Time Systems Symposium*, pages 110–123, 1992. 30, 32

[LSD89] J.-P. Lehozcky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm :exact characterization and average case behaviour. *inproceedings of the IEEE Real-Time Systems Symposium*, pages 166–171, 1989. 28

[LSS87] J.-P. Lehozcky, L. Sha, and K.-K. Strosnider. Enhanced aperiodic responsivess in hard real-time environments. *inproceedings of the 13th IEEE Real-Time Systems Symposium*, pages 261–270, 1987. 30, 31, 122

[LW82] J.-Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, pages 2 :237–250, 1982. 28

[MAB07] M. Marzencki, Y. Ammar, and S. Basrour. Integrated power harvesting system including a mems generator and a power management circuit. *International Solid-State Sensors, Actuators and Microsystems Conference (TRANSDUCERS 2007)*, pages 887–890, 2007. 49

[Mar06] A. Marchand. *Ordonnancement temps réel avec contraintes de qualité de service*. Thèse de Doctorat de l'Université de Nantes, 2006. 41, 61, 62, 63, 64, 65, 66, 102, 104

[MaWX+07] N. Min-allah, Y. Wang, J. Xing, W. Nisar, and A. Kazmi. Towards dynamic voltage scaling in real-time systems - a survey. In *IJCSES International journal of Computer Sciences and Engineering Systems, Vol.1, No.2, CSES International ISSN 0973-4406*, 2007. 52

[MBT+06] C. Moser, D. Brunelli, L. Thiele, , and L. Benini. Real-time scheduling with regenerative energy. In *18th Euromicro Conference Real-Time Systems*, 2006. 53, 170

[MC96] A. K. Mok and D. Chen. A general model for real-time tasks. *Technical Report CS-TR-96-24*, 1996. 37

[Mok83] A.-K. Mok. Fundamental design problems of distributed systems for the hard real-time environment. *PhD Dissertation*, 1983. 23, 29

[MSC05] A. Marchand and M. Silly-Chetto. Rlp : Enhanced qos support for real-time applications. In *The 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pages 241–246. IEEE, 2005. 41

[NG06] N. Navet and B. Gaujal. *Ordonnancement temps réel et minimisation de la consommation d'énergie*, volume 2 of *Ordonnancement, réseaux, qualité de service - Systèmes temps réel*. Hermès, 2006. 51

[Niu10] L. Niu. Energy efficient scheduling for real-time embedded systems with qos guarantee. In *IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications*, 2010. 60

[NQ06] L. Niu and G. Quan. System-wide dynamic power management for portable multimedia devices. In *Multimedia, 2006. ISM'06. Eighth IEEE International Symposium*, 2006. 60

[NQ07] L. Niu and G. Quan. Peripheral-conscious scheduling on energy minimization for weakly hard real-time systems. In *Design, Automation & Test in Europe Conference & Exhibition, DATE '07*, 2007. 60

[Pap94] C. Papadimitriou. Computational complexity. *Addison Wesley*, 1994. 26

[PS05] J.A. Paradiso and T. Starner. Energy scavenging for mobile and wireless electronics. In *Pervasive computing*, 2005. 48

[Ram99] P. Ramanathan. Overload management in real-time control applications using (m,k)-firm guarantees. *IEEE Transactions on Parallel and Distributed Systems*, pages Vol 10 No 6, p549–559, 1999. 39

[RBP+04] J. Randall, N.B. Bharatula, N. Perera, T. Von Buren, S. Ossevoort, and G. Troster. Indoor tracking using solar cell powered system : Interpolation of irradiance. Technical report, Wearable Computing Laboratory, Department of Electronics Swiss Federal Institute of Technology, ETH Zürich, 2004. 48

[Rei02] A. Reinders. Options for photovoltaic solar energy systems in portable products. *TCME Fourth International symposium*, 2002. 48

[RJ02] J.F. Randall and J. Jacot. The performance and modelling of 8 photovoltaicc materials under variable light intensity and spectra. In *LPM,IPR,STI,EPFL,CH-1015 Lausanne,Switzerland*, 2002. 48

[RJ03] J.F. Randall and J. Jacot. Is am 1.5 applicable in practice ? modelling eight photovoltaic material with respect to light intensity and two spectra. *Elsevier Renewable Energy*, 2003. 48

[SB94] M. Spuri and G.-C. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. *inproceedings of the IEEE Real-Time Systems Symposium*, 1994. 30, 35, 124, 125, 144

[SB96] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Journal of Real-Time Systems*, 10 :179–210, 1996. 30, 35

[SBLS88] Sprunt, B., J.-P Lehoczky, and L. Sha. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. *inproceedings 9th IEEE Real-Time System Symposium*, pages p. 251–258, 1988. 30

[SC93] M. Silly-Chetto. Sur la problèmatique de l'ordonnancement dans les systèmes informatiques temps-réel. Rapport d'HDR, Université de Nantes, 1993. 21

[SC96] M. Silly-Chetto. On the stability of scheduling algorithms for real-time control. *IMACS/IEEE-SMC Computational Engineering in Systems Applications Multiconference*, 1996. 102, 104

[Sch06] A. Schneuwly. Designing auto power systems with ultracapacitors. In *Embedded systems conference*, 2006. 47

[SFR03] M.E. Salhiene, L. Fesquet, and M. Renaudin. Adaptation dynamique de la puissance des systèmes embarqués : les systèmes asynchrones surclassent les systèmes synchrones. *4ième journée d'études Faible Tension Faible Consommation (FTFC'03)*, pages 51–58, 2003. 51

[SGR88] L. Sha, J. Goodenough, and T. Ralya. An analytic approach to real-time software engineering. *Softw. Engin. Inst. Draft Repor*, 1988. 30

[Sil86] M. Silly. La tolérance aux fautes dans un système temps réel à contraintes strictes. *INRIA, rapport de recherche*, 1986. 38

[Sil99] M. Silly. The edl server for scheduling periodic and soft aperiodic tasks with resource constraints. *The Journal of Real-Time Systems*, pages 17 : 1–25, 1999. 33

[SK04] D. Shin and J. Kim. Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems. In *ASPDAC'03*, pages 653–658, 2004. 52

[SL95] W.-K. Shih and W.-S. Liu. Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error. In *the IEEE Transactions on Computers*, page 44(3). IEEE, 1995. 39

[SLS95] J.-K. Strosnider, J.-P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computer*, 1995. 30, 31

[SP01] N.S. Shenck and J.A. Paradiso. Energy scavenging with shoe-mounted piezoelectrics. In *IEEE Micro, Vol. 21. No. 3*, pages 30–42, 2001. 48

[SP04] T. Starner and J. Paradiso. Human generated power for mobile electronics. In *Low-Power Electronics Design*, 2004. 48

[SPH+05] J. Souyris, E. Le Pavec, G. Himbert, V. Jégu, and G. Borios. Computing the worst case execution time of an avionics program by abstract interpretation. In *the 5th Intl Workshop on Worst-Case Execution Time (WCET) Analysis*, pages 21–25, 2005. 23

[SSD+94] J. Stankovic, M. Spuri, M. DiNatale, , and G. Buttazzo. Implications of classical scheduling results for real-time systems. *Technical Report UMCS-1994-089, U. Mass*, pages 16–25, 1994. 37

[SSL89] B. Sprunt, L. Sha, and J.-P Lehoczky. Aperiodic task scheduling for hard real-time systems. *The Journal of Real-Time Systems*, pages 1 : 27–60, 1989. 30

[Sta88] J. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, pages 10–19, 1988. 19

[TDUIH03] Y. Yoshida T. Douseki, F. Utsunomiya, N. Itoh, and N. Hama. A batteryless wireless system uses ambient heat with a reversible-powersource compatible cmos/soi dc-dc converter. In *IEEE International Solid-State Circuits Conference, Vol. 1*, pages 388–398, 2003. 49

[Wal11] G. Waltisperger. *Architectures intégrées de gestion de l'énergie pour les microsystèmes autonomes*. PhD thesis, Université de Grenoble, 2011. 13, 49

[WEE+08] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem-overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 2008. 23

[Wol00] W. Wolf. Computers and components : Principles of embedded computing system design. *Morgan Kaufman Publishers*, 2000. 43

[WSP03] R. West, K. Schwan, and C. Poellabauer. Dynamic window-constrained scheduling for real-time media streaming. *IEEE Transactions on Computers*, 53 :744–759, 2003. 60

[WWWR02] W. West, J. Whitacre, V. White, and V. Ratnakumar. Fabrication and testing of all solid-state microscale lithium batteries for microspacecraft applications. In *Journal of Micromechanics and Microengineering, Vol. 12*, pages 52–62, 2002. 46

[YDS95] F. Yao, A. J. Demers, , and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Symposium on Foundations of Computer Science*, pages 374–382. IEEE, 1995. 51

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Maïssa ABDALLAH

Ordonnancement temps réel pour l'optimisation de la Qualité de Service dans les systèmes autonomes en énergie

Real-time scheduling for the optimization of quality of service in autonomous systems

Résumé

Dans le cadre de cette thèse, nous nous intéressons aux applications temps réel qualifiées de fermes car acceptant de ne pas satisfaire la totalité des contraintes temporelles. Celles-ci s'expriment par des échéances c'est à dire des dates avant lesquelles les jobs de l'application se doivent de terminer leur exécution. Les applications temps réel concernées sont très diverses : on peut citer les applications multimédia mais aussi les réseaux de capteurs où l'on tolère occasionnellement la perte de données capteurs.
Notre objectif est de proposer et valider par le biais de la simulation, de nouvelles stratégies d'ordonnancement en vue d'optimiser la Qualité de Service(le ratio de contraintes satisfaites). Ce travail constitue une extension de travaux précédents entrepris dans le laboratoire qui ont porté sur les systèmes autonomes en énergie non surchargés temporellement et énergétiquement.
Notre contribution concerne les systèmes entièrement autonomes car alimentés par l'énergie ambiante qui sont soumis à la fois à des contraintes temporelles et énergétiques. Nous considérons un système monoprocesseur monofréquence, alimenté par un réservoir d'énergie approvisionné par une source environnementale. Dans un premier temps, nous considérons qu'il exécute uniquement des tâches périodiques et nous proposons une solution à la gestion de surcharge de traitement d'une part et aux pénuries temporaires d'énergie d'autre part, en se basant sur le modèle dit Skip-Over. Dans un deuxième temps, nous étendons notre modèle au cas de tâches apériodiques non critiques. Nous apportons une solution au problème lié à la minimisation du temps de réponse de ces dernières.

Mots clés

Système temps réel embarqué, ordonnancement de tâches, qualité de service, système autonome, contraintes énergétiques

Abstract

In this thesis, we focus on firm real-time applications allowing some timing constraints not to be meet (the ratio of satisfied constraints represents the level of Quality of Service provided by the system). These are expressed by deadlines i.e. the dates by which the jobs of the application must have completed their execution.
Targeted real-time applications are very diverse such as multimedia ones or sensor networks which can occasionally tolerate some data loss.
The aim of this thesis is to propose and validate through simulation, new scheduling strategies to optimize the Quality of Service. This work is based on previous works undertaken in the laboratory that focused on both real-time systems without energy consideration but subject to processing overload and autonomous energy systems without overload situations. Our contribution concerns fully autonomous systems powered by ambient energy and subject to both timing and energy constraints.
Firstly, we consider a single-frequency uniprocessor system that only schedules periodic tasks (e.g. monitoring / control), powered by an energy reservoir which is charged through an ambient energy source. The proposed Skip-Over model-based methods provide a solution to the management of both processing overload situations and energy starvation cases.
Secondly, we extend our model to handle aperiodic non-critical tasks in our system. We provide a solution to the problem of minimizing the response time.

Key Words

Embedded real-time system, task scheduling, quality of service, autonomous system, energy constraints.