UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS

Ríad Mattos Nassiffe

RECONFIGURAÇÃO DINÂMICA EM SISTEMAS DE TEMPO REAL COM RESTRIÇÕES DE CONSUMO DE ENERGIA

Florianópolis(SC)

2011

Ríad Mattos Nassiffe

RECONFIGURAÇÃO DINÂMICA EM SISTEMAS DE TEMPO REAL COM RESTRIÇÕES DE CONSUMO DE ENERGIA

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas para a obtenção do Grau de mestre.
Orientador: Prof. Dr. Eduardo Camponogara
Coorientador: Prof. Dr. George Marconi de Araújo Lima

Florianópolis(SC)

2011

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informações em:
http://www.bu.ufsc.br/design/Catalogacao.html

Ríad Mattos Nassiffe

RECONFIGURAÇÃO DINÂMICA EM SISTEMAS DE TEMPO REAL COM RESTRIÇÕES DE CONSUMO DE ENERGIA

Esta Dissertação foi julgada aprovada para a obtenção do Título de "mestre", e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Florianópolis(SC), 03 de Março 2011.

Prof. Dr. José Eduardo Ribeiro Cury
Coordenador do Curso

Prof. Dr. Eduardo Camponogara
Orientador

Prof. Dr. George Marconi de Araújo Lima
Coorientador

Banca Examinadora:

Prof. Dr. Eduardo Camponogara
Presidente

Prof. Dr. Leandro Buss Becker

Profª. Drª. Luciana Salete Buriol

Prof. Dr. Rômulo Silva de Oliveira

Dedico esse trabalho a todas as pessoas que de alguma forma participaram dele.

AGRADECIMENTOS

Felizmente muitas pessoas fizeram parte da minha vida acadêmica e todas de alguma forma contribuíram para meu desenvolvimento. Sem o apoio destas com certeza eu não seria capaz de realizar este trabalho.

Em primeiro lugar agradeço aos meus pais, irmãos e familiares, que não somente durante os dois anos do mestrado, mas todos os outros que antecederam o sempre acreditaram e me incentivaram a lutar para alcançar meus objetivos, me apoiando como possível.

Relembrando o tempo de graduação, gostaria de agradecer ao meu antigo orientador Edeyson Gomes e aos professores Adolfo Duran e Cláudio Amorim que acreditaram na minha capacidade em fazer o mestrado e enviaram cartas de recomendação, para que fosse possível minha inscrição no programa de mestrado.

Agradeço ao meu orientador Eduardo Camponogara e coorientador George Lima, que me ajudaram, incentivando e guiando na construção desse trabalho com absoluta dedicação e paciência.

A todos os colegas que me apoiaram de alguma forma durante esta jornada que foi o mestrado, onde com certeza não tive apenas crescimento acadêmico mas também um crescimento pessoal. Em especial aos colegas B2, Cassiano, Codas, Helton, Robinho, Toscano, Tiago, Maradona, Árabe, Lange, Vanessa e Tânia que ajudaram a fazer esse momento mais fácil.

E a todos professores que conheci, por terem dedicado um pouco do seu tempo e conhecimento para que eu chegasse até aqui.

Uma longa viagem começa com um único passo.

Lao-Tsé

RESUMO

Esta dissertação propõe uma infraestrutura para alocação dinâmica de tempo de processador em sistemas de tempo real multi-modais, sob restrições de escalonabilidade e consumo de energia. Tal infraestrutura é adequada para sistemas de tempo real não crítico e sistemas embarcados que necessitam de garantia de economia de energia. A alocação dinâmica é modelada como um problema de otimização inteira para o qual foram propostos dois algoritmos. O primeiro é baseado no princípio recursivo da programação dinâmica, enquanto o segundo é uma heurística do tipo gulosa baseada na relaxação Lagrangeana e *surrogate*. Os algoritmos foram avaliados em uma aplicação real de codificação de vídeo. Embora o problema formulado seja NP-Difícil, análises experimentais mostraram que a heurística alcança resultados próximos da solução ótima e com baixo custo computacional.

Palavras-chave: Escalonamento tempo real; reconfiguração dinâmica; economia de energia com QoS.

ABSTRACT

This dissertation proposes a framework for dynamic, value-based processor time allocation in multi-modal real-time applications under schedulability and energy consumption constraints. The framework is suitable for adaptive real-time embedded systems which need guarantees of energy savings. The dynamic allocation is formulated as a discrete optimization problem for which two algorithms are proposed. The first algorithm is based on the recursive principle of dynamic programming, whereas the second is a greedy heuristic based on Lagrangian and surrogate relaxations. As a means of evaluation, the framework was applied to a real multimedia encoder. Although the formulated problem is NP-Hard, the experimental analysis has shown that the derived heuristic achieves very good approximation results with low computational cost.

Keywords: real time scheduling, dynamic reconfiguration, energy saving with QoS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LISTA DE SÍMBOLOS

SUMÁRIO

1 INTRODUÇÃO

Neste Capítulo é apresentada a motivação e as contribuições pretendidas com esse trabalho. A estrutura do documento é apresentada na sequência.

1.1 MOTIVAÇÃO

O uso de sistemas controlados por computadores embarcados cresce à medida que novos avanços tecnológicos são realizados, como pode ser observado em telefones móveis, *PDAs*, TVs, tocadores de *DVD*, câmeras, carros, utensílios domésticos, *etc.* (BUTTAZZO, 2006).

Muitos destes sistemas possuem restrições em comum, tais como consumo de energia, condições temporais e adaptação. O primeiro caso é de fácil percepção em sistemas embarcados móveis, que são alimentados exclusivamente por baterias que limitam o tempo de uso do sistema. Condições temporais podem ser claramente percebidas em sistemas que interagem com o meio, pois estes precisam responder a um estímulo recebido dentro de um tempo específico, chamado de *deadline*, que não sendo obedecido pode tornar a resposta inválida. A terceira restrição mencionada, a de adaptação, ocorre quando a modificação de um ou mais parâmetros ocasionam alterações na carga de trabalho a ser processada, fazendo-se necessário a adaptação do sistema para suportar a nova carga. Qualquer violação dessas restrições pode gerar uma falha no sistema.

Tendo em vista as restrições citadas anteriormente, este trabalho tem como foco as aplicações de tempo real com garantia de economia de energia e reconfiguração dinâmica do sistema para um determinado conjunto de tarefas. Esta dissertação tem como objetivo o desenvolvimento de modelos e algoritmos para maximizar a utilização da *CPU* em relação ao consumo de energia em sistemas de tempo real adaptativos.

Um possível cenário para aplicação desse trabalho seria em satélites. Quando uma fonte de energia está disponível para recarregar a bateria, o sistema pode trabalhar em modo de qualidade máxima. Por outro lado, quando não for possível recarregar a bateria, pode-se escolher um modo de operação degradado, ou até mesmo encerrar aplicações que não sejam vitais ao funcionamento correto do satélite. Este tipo de comportamento cíclico de baixa e alta disponibilidade de energia é também observado em *notebooks*, servidores, carros, *etc.*

Visando dar suporte a mecanismos de economia de energia, foram lançadas novas gerações de processadores no mercado (BINI; BUTTAZZO; LIPARI, 2009) que permitem a alteração da frequência e voltagem em tempo de execução. Esse tipo de tecnologia já esta sendo usada por sistemas operacionais como Windows, Linux, MacOS e outros no ajuste da frequência de *CPU* em resposta às variações na carga de trabalho. A regulação de frequência e voltagem da *CPU* permite que o processador economize energia, operando em um modo degradado quando a carga de processamento for reduzida.

1.2 CONTRIBUIÇÃO

Este trabalho busca resolver um problema de reconfiguração dinâmica de escalonadores usando o modelo de servidores com reserva e garantia de economia de energia. O mecanismo de reconfiguração proposto escolhe um nível de degradação (modo) de cada tarefa e a frequência do processador para executá-la. Cada combinação entre modo e frequência representa uma configuração de execução. O objetivo do sistema ao escolher uma configuração para uma tarefa é maximizar um critério de desempenho global ao mesmo tempo que garante escalonabilidade e economia de energia do sistema de tempo real.

O problema abordado neste trabalho pode ser visto como uma generalização do problema da mochila, com duas mochilas, no qual uma delas é estabelecida pela limitação de escalonabilidade do processador e a outra pelo consumo de energia. Portanto, o problema de reconfiguração é NP-difícil.

Em um problema clássico da mochila são dados diversos objetos, cada um com um valor de benefício e peso agregado. Uma mochila suporta até um limite de peso determinado, que não comporta todos os objetos. Surge assim o problema de selecionar os objetos que ao serem acomodados na mochila maximizem o valor do benefício agregado dos objetos escolhidos sem ultrapassar a capacidade de peso da mochila.

Para resolver o problema de reconfiguração dinâmica de escalonadores, foram propostos dois algoritmos, sendo o primeiro baseado no princípio de recursividade da programação dinâmica e o segundo baseado em uma heurística gulosa que utiliza as relaxações Lagrangeana e surrogate.

1.3 ESTRUTURA

Este documento esta estruturado da seguinte forma: o Capítulo 2 faz uma revisão de assuntos importantes ao entendimento do trabalho desenvolvido, tais como: escalonamento em tempo real, o problema do tratamento de tarefas não-críticas, técnicas de economia de energia e formas de modelagem, mecanismos de troca de modos de tarefa e, por último, uma revisão de trabalhos relacionados.

O Capítulo 3 contém a descrição do modelo usado para representar o problema, a descrição de uma instância exemplo usada para ilustrar o funcionamento dos algoritmos propostos. Os algoritmos apresentados foram os de programação dinâmica e o baseado em heurística gulosa.

No Capítulo 4 é realizada uma análise numérica de todos os algoritmos propostos no trabalho, com objetivo de verificar o comportamento em diferentes cenários. Além disso, uma simulação foi realizada com a infraestrutura de aplicação multimídia com dados retirados da execução de um codificador, buscando avaliar a sua habilidade de tratar sobrecargas no escalonador.

Por último, o Capítulo 5, apresenta as conclusões e propostas de trabalhos futuros que foram percebidos durante o desenvolvimento desta dissertação.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo é um resumo de assuntos necessários ao entendimento do trabalho realizado. Começará com uma definição de sistema de tempo real, seguido por uma revisão sobre duas das políticas de escalonamento mais utilizadas, o tratamento de tarefas não críticas, a alta variabilidade de carga computacional, o consumo de energia e a reconfiguração dinâmica. Todos os problemas associados ao trabalho serão apresentados juntamente com as abordagens sugeridas pela literatura para resolvê-los.

2.1 SISTEMAS DE TEMPO REAL

Um Sistema de Tempo Real (STR) deve reagir de acordo com estímulos oriundos do seu ambiente, executando tarefas em prazos específicos (FARINES; FRAGA; OLIVEIRA, 2000). O funcionamento correto do sistema não depende exclusivamente da integridade dos resultados, mas também do tempo utilizado para alcançá-los. Assim, a violação da condição temporal pode gerar desde a perda de qualidade até a falha do serviço em questão. Esse tipo de sistema está presente em aplicações de telecomunicação, monitoramento e processamento de sinais.

Um STR é composto por um conjunto de tarefas as quais, individualmente, são responsáveis por prover alguma funcionalidade específica ao sistema. Uma tarefa pode ser executada mais de uma vez, sendo cada execução chamada de instância. Assim sendo, uma tarefa τ responsável pela codificação de vídeo em um STR pode ser definida como um conjunto de instâncias que visam codificar quadros para realizar a funcionalidade da tarefa.

Sistemas de tempo real podem ser classificados em dois tipos: *hard real-time* e *soft real-time* (BUTTAZZO, 1997). No primeiro caso não é permitido a perda de prazos de execução em nenhuma hipótese, como por exemplo um sistema aviônico, que pode causar um acidente se não responder no tempo determinado pelo projetista. O outro tipo tolera perda de *deadline*, como durante a execução de uma aplicação multimídia, onde a demora na codificação de um quadro diminui a qualidade de exibição do vídeo e não o invalida.

As tarefas que compõem um sistema de tempo real podem ser separadas em dois tipos quanto ao cumprimento do seu deadline: críti-

cas e não críticas (LIU, 2000). No primeiro tipo, como sugere o nome, a perda de *deadline* irá causar falha. A existência de uma tarefa desse tipo implica que o sistema é crítico, enquanto no segundo a perda do prazo pode implicar em perda de qualidade do serviço.

Em relação às características temporais as tarefas podem ser divididas em periódicas, esporádicas ou aperiódicas (LIU, 2000). No primeiro caso as tarefas têm valores de tempo de ativação, momento em que uma tarefa é disponibilizada para execução, e o período, tempo entre as ativações das instâncias, conhecidos e fixos. No caso das esporádicas, não é conhecido o tempo de ativação, mas o tempo mínimo entre duas ativações consecutivas é conhecido. No último caso não se conhece o tempo de ativação ou período, pois a ativação desse tipo de tarefa ocorre devido a alguma característica aleatória. Um sistema *hard real-time* é composto por tarefas periódicas enquanto um sistema *soft real-time* pode ter qualquer um desses tipos.

Para execução correta de um sistema de tempo real é necessário que as tarefas sejam executadas em uma ordem específica, para garantir o funcionamento sem perda de *deadline*. O procedimento de ordenar as tarefas é chamado de escalonamento, sendo desempenhado pelo escalonador que é um componente do sistema responsável pela gestão do processador (FARINES; FRAGA; OLIVEIRA, 2000).

Um modelo para representar o problema de escalonar tarefas foi definido em (LIU; LAYLAND, 1973). Esse artigo serviu como base para vários trabalhos na área de tempo real, tendo ele estabelecido as seguintes características e condições para sistemas *hard real-time*:

- Todas as tarefas são periódicas.
- Cada instância deve ser executada antes da próxima chegar.
- As tarefas são independentes.
- O tempo de execução de cada tarefa é constante.
- Qualquer tarefa aperiódica no sistema é tratada de forma especial e não possui um *deadline* crítico.

Um sistema é constituído por um conjunto de tarefas $\tau_i \in \{\tau_1, ..., \tau_n\}$, sendo que cada uma possui um tempo de computação C_i e um período previsto para o lançamento de uma nova instância da tarefa no sistema chamado de T_i. O valor de execução geralmente é mensurado de acordo com o pior caso, pois dessa forma é possível garantir o funcionamento do sistema para qualquer situação.

A utilização do processador por uma tarefa é dado pela divisão do C_i por seu T_i, ficando a utilização do conjunto de tarefas expresso por:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2.1}$$

Com essa informação é possível realizar um teste para saber o quanto está sendo utilizado do processador por um conjunto de tarefas e se ele pode ser escalonado.

2.2 TESTES DE ESCALONAMENTO

Um problema de escalonamento pode envolver um ou mais processadores, um conjunto de recursos compartilhados e um conjunto de tarefas especificadas segundo um modelo de tarefas, o qual pode possuir restrições temporais, de precedência e de exclusão, de acordo com a necessidade.

O escalonamento de tarefas em sistemas de tempo real, na sua forma geral, é identificado como um problema intratável ou NP-completo (GAREY; JOHNSON, 1979). Para a resolução desse problema algumas simplificações podem ser realizadas, levando em consideração as características específicas de um problema qualquer em questão, que poderão levar a concepção de uma heurística, a qual não irá garantir a melhor solução mas fornecerá uma solução que obedeça às restrições fixadas pelo problema.

Devido às diferentes necessidades de conjuntos de tarefas e abordagens para resolver o problema, os algoritmos de escalonamento estão separados em classes. Para um conjunto de tarefas fazer parte de uma classe elas devem ter propriedades em comum, por exemplo, o conjunto de tarefas de prioridade fixa forma uma classe, enquanto o subconjunto de tarefas com prioridade fixa onde *deadline* é igual ao período forma outra classe.

Um algoritmo pode ser considerado ótimo por dois motivos: se minimiza algum custo ou métrica, como achar uma solução com maior utilização de *CPU*, com relação aos outros algoritmos que são aplicados naquela classe ou caso ele consiga encontrar um resultado que obedeça às restrições de uma classe de tarefas sempre que houver um, caso não consiga encontrar a solução e ela exista todos os outros algoritmos também não poderão encontrar para ele ser considerado ótimo (FARINES; FRAGA; OLIVEIRA, 2000).

No artigo (LIU; LAYLAND, 1973) foram apresentadas duas das

políticas de escalonamento mais utilizadas para tempo real, o *Rate Monotonic* e o *Earliest Deadline First*. A seguir será mostrado o resumo das principais características e um exemplo de cada uma das políticas.

2.2.1 *Rate Monotonic*

Rate Monotonic (RM) é uma política de escalonamento de prioridade fixa, que atribui prioridade de acordo com o período das tarefas, quanto menor o período maior a prioridade. Dessa forma fazendo com que todas as prioridades atribuídas em tempo de projeto sejam mantidas ao longo do tempo de execução do sistema. Essa estratégia de escalonamento faz com que as tarefas de baixa prioridade não interfiram nas de alta. Contudo, o *RM* é intrinsecamente preemptivo e uma tarefa em execução pode ser interrompida pela chegada de uma de menor período.

No artigo (LIU; LAYLAND, 1973) é provado que o *RM* é ótimo entre os métodos de prioridade fixa, pois nenhum outro algoritmo de sua classe consegue escalonar um conjunto de tarefas que ele não possa. Ainda de acordo com o artigo, existe um limite superior de utilização, U_{lub}, para qualquer política de escalonamento de prioridade fixa, sendo ele igual a:

$$U_{lub} = n(2^{\frac{1}{n}} - 1) \tag{2.2}$$

Essa relação é considerada suficiente, entretanto não necessária para provar a escalonabilidade do sistema. Outro detalhe importante sobre *RM* é que ele pode conseguir 100% da utilização de *CPU* caso os períodos das tarefas sejam múltiplos entre si, de outra forma não será feito o uso completo da capacidade de processamento do processador.

A Figura 1, retirada do livro (FARINES; FRAGA; OLIVEIRA, 2000) mostra um exemplo de uma situação escalonável pela política *Rate Monotonic*. As tarefas do exemplo, τ_1, τ_2 e τ_3, são descritas na Tabela 1. A prioridade das tarefas é dada na seguinte ordem primeiro τ_1 com prioridade máxima, depois τ_2 e por último τ_3. Pode-se constatar de acordo com a última linha que a utilização total do processador pelo conjunto de tarefas em questão é de $0,753$, e o limite superior dado pela Equação (2.2) com $n = 3$ é de aproximadamente $0,78$, condição suficiente para provar que o sistema é escalonável segundo a política *RM*.

O sistema *RM*, ilustrado na Figura 1, começa a funcionar recebendo uma instância de cada tarefa, a primeira a ser executada é a τ_1 pois tem maior prioridade. Depois a τ_2 começa a usar o processa-

dor, por ter a segunda maior prioridade, com o término da execução a τ_3 assume. Entretanto, antes que ela termine de executar, uma nova instância de τ_1 é lançada, o que faz com que τ_3 pare de executar em $t = 100$ e realize uma preempção para que τ_1 que tem maior prioridade comece a executar. Após a execução de τ_1, τ_3 volta a executar até ser interrompida por novas ativações de τ_2 em $t = 150$ e τ_1 em $t = 200$.

Tabela 1: Tarefas do exemplo do escalonamento por RM.

	C_i	T_i	u_i
τ_1	20	100	0,2
τ_2	40	150	0,267
τ_3	100	350	0,286
U			0,753

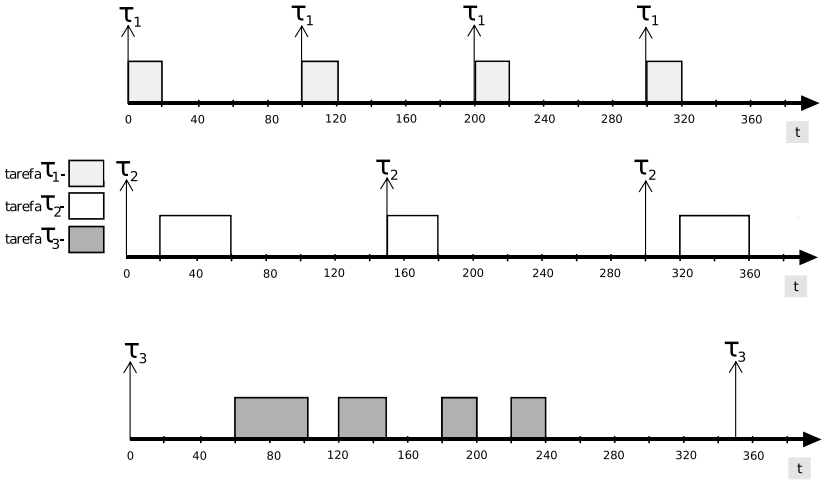

Figura 1: Exemplo de um conjunto de tarefas escalonáveis pelo *RM*.

2.2.2 *Earliest Deadline First*

Earliest Deadline First (EDF) é uma política de escalonamento com prioridade dinâmica. Nas tarefas escalonadas pelo *EDF* a prioridade é dada de acordo com os *deadlines*, quanto mais próxima uma tarefa estiver de alcançar seu prazo de execução, maior sua prioridade,

sendo que a tarefa com maior prioridade preempta uma de menor. Assim como o *RM*, esse modelo é também preemptivo, quando uma tarefa com *deadline* mais próximo é ativada.

A política *EDF* não faz nenhuma suposição sobre a periodicidade das tarefas para atribuir prioridade, ao contrário do *RM* que faz, permitindo seja empregada no escalonamento de tarefas aperiódicas e periódicas. A prioridade das tarefas é dada de acordo com o *deadline*.

Para um sistema ser escalonável por *EDF*, basta que o sistema tenha utilização de *CPU* no pior caso menor ou igual a 1, como mostra a equação a seguir:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{2.3}$$

A Figura 2, retirada do livro (BUTTAZZO, 1997) mostra um exemplo de uma situação escalonável pela política *EDF*. Por outro lado, a política *Rate Monotonic* não garante escalonabilidade, já que com as informações de τ_1 e τ_2 disponíveis na Tabela 2, pode-se perceber que a utilização total do sistema é de $0,97$, enquanto o limite superior de utilização é aproximadamente de $0,82$. Na Figura 2, na unidade de tempo 7 ocorre a perda de *deadline* no conjunto de tarefas escalonado pelo modelo *RM*, enquanto o conjunto que usa o modelo *Earliest Deadline First* continua a execução sem falhas.

Tabela 2: Tarefas do exemplo do escalonamento por *RM* e *EDF*.

	C_i	T_i	U_i
τ_1	2	5	0,40
τ_2	4	7	0,57
Total			0,97

Apesar de conseguir utilizar o máximo da *CPU*, a política de escalonamento *EDF* (assim como outras políticas de prioridade dinâmica) tem como desvantagem a possibilidade de qualquer tarefa forçar preempção sobre qualquer outra do sistema. Caso ocorra um erro na medição do C_i, por subestimação que leve a Equação (2.3) a ser desrespeitada, será difícil prever o comportamento do sistema.

2.3 SERVIDORES

Os algoritmos citados anteriormente pressupõem que o conjunto de tarefas em questão seja homogêneo quanto ao nível de criticidade,

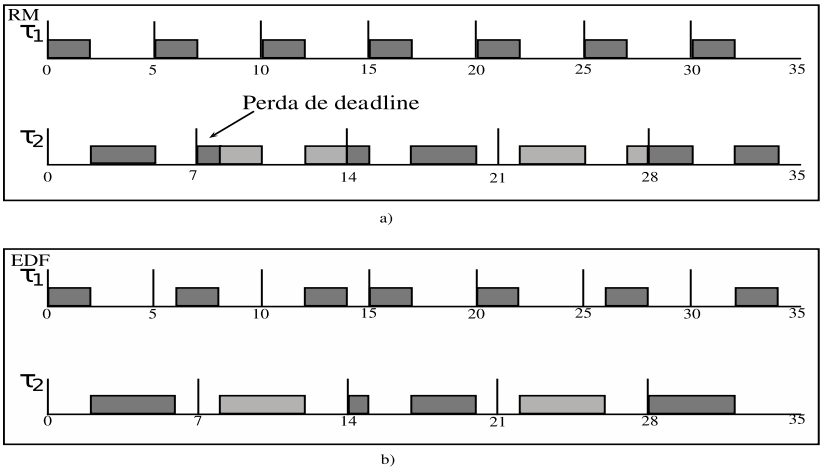

Figura 2: Exemplo de uma situação onde a política EDF consegue escalonar e a RM falha.

ou seja, todas as tarefas são do tipo crítica ou nenhuma é considerada crítica. Contudo, muitos dos sistemas existentes trabalham com tarefas de diferentes níveis de criticidade (BUTTAZZO, 1997). Esse tipo de sistema pode ser escalonado com o uso de servidores de tarefas aperiódicas, que na realidade é uma tarefa periódica que tem a função de executar as instâncias das tarefas aperiódicas da forma mais eficiente possível.

Um servidor pode ser responsável pela execução de uma ou mais tarefas, sendo um servidor representado por S_i, com $i \in \{1, ..., n\}$, definido por uma tupla (Q_i, T_i). Cada servidor tem até Q_i unidades de tempo, prevista em cada intervalo T_i, para execução das tarefas que ele for responsável.

Em um sistema com tarefas críticas e não críticas, híbridas, é necessário garantir o escalonamento das tarefas críticas de acordo com o pior caso possível para o valor do Q_i, já no caso das não críticas pode se tornar interessante garantir o escalonamento para o caso mais comum de ocorrer, o caso médio. De acordo com a Figura 3, que mostrada o gráfico da distribuição normal, é possível ver que o pior caso é muito difícil de ocorrer e tratá-lo poderá exigir um processador com mais capacidade de processamento.

A utilização do processador por um servidor S_i é expressa por $u_i = Q_i/T_i$. Assim como as outras tarefas periódicas, a utilização de um

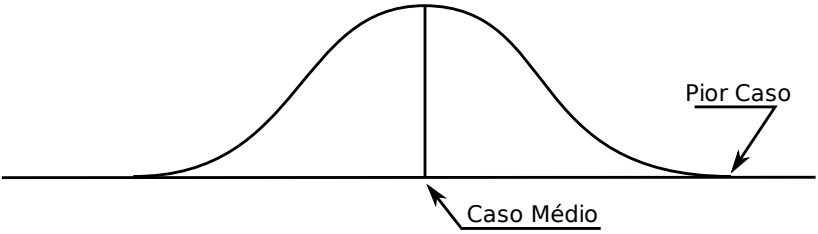

Figura 3: Gráfico da distribuição normal, mostrando a probabilidade de ocorrência do caso médio e pior caso do uso do processador.

servidor pertence ao intervalo $[0, 1]$. Cada servidor possui uma fila para colocar as tarefas que recebe e executá-las de acordo com a política de escalonamento utilizada.

Assim como as políticas de escalonamento de tarefas, os servidores de tarefas podem ser divididos em dois grupos, os de prioridade fixa e os de prioridade dinâmica.

Dentre os servidores de prioridade fixa, o *Background Scheduling* (BUTTAZZO, 1997) é considerado o mais simples e consiste em deixar as tarefas com maior prioridade (que são as críticas) executarem primeiro, só permitindo que as tarefas de baixa prioridade (não críticas geralmente) seja escalonadas quando não houver nenhuma tarefa de alta prioridade na fila. Este comportamento pode causar tempos de resposta muito ruins para as tarefas pertencentes aos servidores de baixa prioridade.

Os servidores de prioridade dinâmica são caracterizados por possuírem limites de escalonabilidade maiores, o que permite uma melhor utilização do processador. A justificativa para tal afirmação está no fato dos servidores de prioridade fixa serem baseados em políticas de escalonamento que obedecem a um limite superior de utilização, enquanto os de prioridade dinâmica podem usar a política de escalonamento *EDF* que pode chegar a utilizar 100% do processador. Os servidores podem ser avaliados quanto a três propriedades:

- **Viabilidade**: nenhum *deadline* estabelecido pelo escalonador pode ser perdido.

- **Eficiência**: tarefas devem ser executadas de forma a minimizar os atrasos. É caracterizado como atraso quando uma tarefa é entregue fora do período dela, mas esteja dentro do *deadline* estabelecido pelo servidor.

- **Flexibilidade**: servidores devem gerenciar tarefas sem saber o pior caso de execução ou o tempo de intervalo de chegada das instâncias.

A seguir serão explicados respectivamente os algoritmos *Dynamic Sporadic Server (DSS)* e *Constant Bandwidth Server (CBS)*. Ambos utilizam uma política *EDF* para escalonar as tarefas.

2.3.1 *Dynamic Sporadic Server (DSS)*

O *DSS* é uma estratégia de serviços aperiódicos proposta por Spuri e Buttazzo (SPURI; BUTTAZZO, 1994) (SPURI; BUTTAZZO, 1996) que é uma extensão do *Sporadic Server* (SPRUNT; SHA; LEHOCZKY, 1989) criado inicialmente para políticas de prioridade fixa. Existe também um outro algoritmo similar ao *DSS* que foi desenvolvido em paralelo chamado de *Deadline Sporadic Server* (GHAZALIE; BAKER, 1995) tendo sido estendido da mesma base.

Como muitos dos algoritmos de servidores existentes para tratar o problema de escalonamento, o *DSS* tem definido um período T_i e o tempo de execução C_i para cada servidor. Sua diferença para muitos outros está no fato que o valor de C_i só é recarregado quando é totalmente consumido e as regras de recarregamento satisfeitas, por causa disso que é possível conseguir 100% de uso do processador. As regras de recarga são:

- Quando o servidor é criado ele inicia com C_i no seu valor máximo.
- O tempo de recarga (RT) e o *deadline* d_i corrente de um servidor são estabelecidos assim que $C_i > 0$ e não existe nenhuma tarefa aperiódica pendente. Ficando o valor de $RT = d_i = t_A + T_i$, sendo t_A igual ao instante em que a tarefa é lançada no sistema.
- O valor da recarga de C_i é chamado de RA . Ele deve ser gasto até o tempo RT, sendo computado quando $C_i = 0$ ou a última tarefa aperiódica ativada é terminada.

Esse algoritmo consegue garantir utilização total do processador, contudo sua implementação não é muito simples e assim como vários outros necessita do conhecimento do pior tempo de execução de uma tarefa ou tempo mínimo de intervalo entre a chegada das instâncias (ABENI; BUTTAZZO, 2004), ou seja, não obedece à propriedade de flexibilidade.

2.3.2 *Constant Bandwidth Server (CBS)*

O *CBS* é inspirado nos servidores *Total Bandwidth Server (TBS)* (BUTTAZZO, 1997) e *DSS*. O diferencial desse servidor em relação aos que inspiraram ele é o fato dele obedecer as três propriedades e assegurar o isolamento temporal entre as tarefas críticas e não críticas de forma a garantir que, em caso de sobrecarga no sistema, uma tarefa não interfira na execução das outras.

Para conseguir obedecer as três propriedades, o *CBS* usa uma política de recarga de orçamento a qual garante que sua utilização máxima não ultrapassará o limite definido em tempo de planejamento. Ele possui somente dois parâmetros: o orçamento máximo Q_i e o período T_i. Toda vez que sua carga atual c_i é recarregada ela recebe o valor de recarga Q_i. A razão Q_i/T_i define a utilização u_i do servidor, que é utilizada para o teste de escalonabilidade da política *EDF*. Um servidor *CBS* então pode ser caracterizado pelos parâmetros a seguir:

- c_i é a carga atual do servidor.
- Q_i é o tempo de execução ou orçamento máximo do servidor.
- T_i é o período.
- utilização da *CPU* por um servidor é dada por $u_i = \frac{Q_i}{T_i}$.
- $d_{i,k}$ é o $k^{ésimo}$ *deadline* marcado pelo servidor.

Um sistema que implementa o *CBS* é formalmente definido por servidores S_i, indexados por $i = 1, \ldots, n$, cada um com valores de Q_i e T_i pré-definidos. O *deadline* $d_{i,k}$ e o orçamento atual do servidor, c_i, são modificados à medida que instâncias da tarefa são executadas.

Em tempo de projeto, um servidor *CBS* S_i recebe um valor Q_i e T_i, sendo eles estipulados de acordo com a necessidade do sistema. Ao receber uma instância, o servidor a coloca numa fila e vai executando à medida do possível, garantindo a propriedade de eficiência. Toda vez que $c_i = 0$, ele é recarregado e o *deadline* atualizado para $d_{i,k+1} = d_{i,k} + T_i$. Devido a esta regra de recarga pode-se ter certeza que qualquer tarefa será executada respeitando o *deadline* dado pelo servidor, mesmo que perca seu prazo.

A recarga do servidor pode ocorrer por dois motivos, o primeiro seria na chegada da tarefa e o outro quando o valor de $c_i = 0$, como foi descrito no parágrafo acima. No primeiro caso, quando uma tarefa chega e não existe nenhuma outra em execução ou na fila e a condição

$r+(c_i/Q_i)T_i \geq d_{i,k}$ for verdadeira, sendo r o tempo atual, o *deadline* tem que ser atualizado para $d_{i,k+1} = r+T_i$ e o valor do tempo de execução recarregado. No último caso, quando há uma tarefa em execução ou tarefas na fila o valor de c_i é recarregado e o *deadline* é atualizado para $d_{i,k+1} = d_{i,k}+T_i$.

O funcionamento de um sistema que implementa o *CBS* é ilustrado na Figura 4 onde existem duas tarefas, uma não crítica (τ_1, com sete instâncias) e outra crítica (τ_2 com quatro instâncias). A Tabela 3 mostra os valores dos U_i onde $\sum_{i=1}^{n} U_i \approx 0,96$. No exemplo em questão, o período possível para execução da tarefa crítica é maior e por consequência ela geralmente terá uma prioridade menor durante a maior parte do tempo de execução do sistema.

Tabela 3: Tarefas do exemplo do escalonamento com *CBS*.

	C_i	T_i	U_i
$CBS(\tau_1)$	2	3	≈0,67
τ_2	2	7	≈0,29
Total			0,96

A primeira linha da Figura 4 corresponde à tarefa não crítica, τ_1, que será escalonada pelo servidor *CBS*. Cada instância terá seu tempo de computação representado por $C_{1,l}$, onde l representa a instância em execução, essa variando entre 1 e 7. O *CBS* tem a variação de orçamento mostrada na terceira linha, tendo sido alocado para as tarefas que ele é responsável com $Q_i = 2$ e $T_i = 3$.

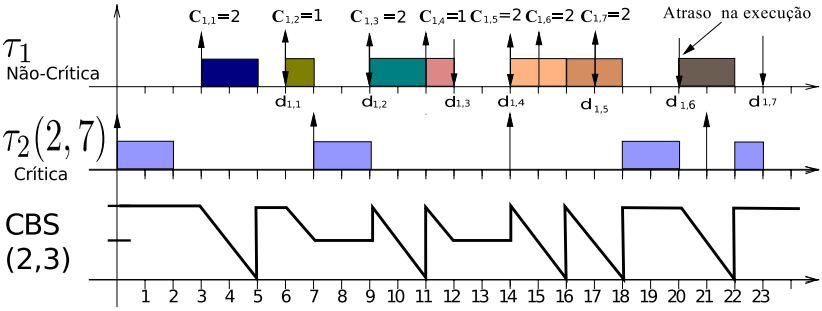

Figura 4: Execução das tarefas τ_1 e τ_2 em um sistema híbrido usando servidor CBS para a tarefa não crítica.

A segunda linha da Figura 4 se refere à tarefa crítica, τ_2, que tem $C_2 = 2$ e $T_2 = 7$, iniciando no instante $r = 0$. Como não há outras

tarefas concorrendo pelo uso da *CPU*, a τ_2 começa a executar, termina no instante $r = 2$ e depois aguarda a chegada da próxima instância.

No intervalo de tempo entre a primeira e a segunda instância de τ_2, chega a primeira instância de τ_1. A execução de τ_1 é controlada pelo servidor *CBS*. Uma instância de τ_1 chega em $r = 3$, com $C_{1,1} = 2$. O servidor não tem nenhuma outra instância em execução ou na fila, calculando $d_{1,1} = 6$. Como o processador está livre, este começa a executar imediatamente a instância e acaba por gastar toda a carga atual, com isso ele tem que ser recarregado adiando o *deadline* para $d_{1,2} = d_{1,1} + T_1 = 9$. Em $r = 6$ chega a segunda instância de τ_1, que precisa de uma unidade de tempo para executar e, como o processador está ocioso ela é imediatamente executada.

No instante $r = 7$ o processador está ocioso e executa a instância de τ_2, que acaba de receber. Assim que termina é lançado no sistema a terceira instância de τ_1 no instante $r = 9$, com $C_{1,3} = 2$, e como não existe nada na fila ou sendo executado pelo servidor é feita uma verificação da condição de chegada, com isso o orçamento é recarregado e o *deadline* recalculado para $d_{1,3} = d_{1,2} + T_1 = 9 + 3 = 12$. No fim da execução da terceira instância de τ_1 todo o orçamento do servidor é gasto, ocasionando uma nova definição de $d_{1,4} = 15$ e renovação do orçamento. Após isso, em $r = 11$, chega uma quarta instância de τ_1 com $C_{1,4} = 1$, a condição de recarga no servidor não é obedecida, logo o *deadline* não é modificado e a instância executa gastando uma unidade de tempo do c_1.

A execução da tarefa τ_1 ocorre sem problemas até o instante em que o sistema sofre uma sobrecarga com as instâncias de número cinco, seis e sete. No momento $r = 14$ chegam duas instâncias, a quinta instância da tarefa τ_1 e outra de τ_2. A quinta instância de τ_1 tem $C_{1,5} = 2$ e não existem tarefas na fila ou sendo executadas pelo servidor. A condição de recarga é verificada e obedecida, o que gera $d_{1,5} = 17$ e faz c_1 ser recarregado, deixando a prioridade dessa instância maior que a da tarefa crítica que foi lançada no mesmo instante, mas tem *deadline* na unidade de tempo 21.

Antes que a quinta instância de τ_1 termine, chega a sexta a qual necessita de duas unidades de tempo para executar, sendo inserida na fila. Ao terminar de executar $C_{1,5}$ no instante $r = 16$, todo o orçamento do servidor foi consumido e precisa ser recarregado, o que leva o *CBS* a atribuir $d_{1,6} = 20$, fazendo com que a prioridade da tarefa não crítica continue maior que a crítica.

Durante a execução da sexta instância de τ_1, a sétima é lançada no sistema no instante $r = 17$. Uma vez que servidor está ocupado

processando a sexta instância de τ_1, a sétima é colocada na fila para esperar sua vez de executar. Quando a sexta instância termina no instante $r = 18$, ocorre a recarga do servidor e o *deadline* remarcado passa a ser $d_{1,7} = 23$, o que faz a tarefa crítica ter uma prioridade maior e preemptar o sistema, pois seu *deadline* é no instante 21.

A sétima instância de τ_1 executa logo após a tarefa crítica τ_2 que inicia execução em $r = 18$ e termina no instante $r = 20$, começando no instante $r = 20$ e acabando no instante $r = 22$. De acordo com o modelo *CBS* não houve perda de *deadline*, pois para executar a sétima instância de τ_1 o servidor tinha estabelecido um *deadline* $d_{1,7} = 23$ e terminou de executar na unidade de tempo 22. Entretanto a instância que deveria ser resolvida em um intervalo de três unidades após sua chegada foi resolvida em cinco unidades de tempo depois.

Por contemplar as três propriedades listadas sobre servidores, ser de fácil implementação, garantir o isolamento temporal das tarefas e garantir o cumprimento da execução de tarefas críticas no prazo, o *CBS* será usado neste trabalho.

Existem ainda várias outras extensões desse modelo como: CASH (CACCAMO; BUTTAZZO; SHA, 2000), BACKSLASH (LIN; BRANDT, 2005), GRUB (LIPARI; BARUAH, 2000) e BASH (CACCAMO; BUTTAZZO; THOMAS, 2005). Cada um desses foi criado visando melhorar o *CBS* para um caso específico.

2.4 ENERGIA

A economia de energia em um sistema computacional pode ser realizada por meio do gerenciamento de operações de *IO*, de acesso à memória, do processador entre outras formas. Em escalonadores, a economia está restrita ao controle de energia gasto pelo processador, pois ele é responsável por dividir o tempo de *CPU* entre as tarefas que estão tentando executar.

Normalmente a economia é realizada por meio da variação da sua voltagem e frequência, através da técnica *DVS (Dynamic Voltage Scaling)*. Usualmente, dois modelos são aplicados para modelar o consumo de energia: *DVS* que considera somente o processador e *DPM (Dynamic Power Management)* que considera outros dispositivos além do processador (ZHAO; AYDIN, 2009). Em ambos os métodos, o consumo de energia por unidade de tempo do processador para o correto

funcionamento é dado por:

$$P_{cpu} = fv^2C_{ef} \tag{2.4}$$

onde P_{cpu} indica a potência do processador, f a frequência em utilização, C_{ef} a capacitância necessária para execução de uma instrução e v a voltagem necessária para o funcionamento do sistema em uma determinada frequência.

Para realizar economia de energia é necessário que haja tanto a variação na frequência quanto na voltagem, pois quando a frequência é reduzida, o tempo de execução aumenta da mesma proporção. Se em uma tarefa que tem dois segundos para usar o processador com a *CPU* operando com toda sua capacidade tiver a frequência reduzida pela metade, então esta tarefa terá o tempo de processamento estendido para quatro segundos.

Esta situação é ilustrada na Figura 5 na qual somente a frequência de uma tarefa é modificada e logo nenhuma economia de energia ocorre. Cada retângulo indica a quantidade de energia consumida para executar a tarefa. Quando o processador está usando 100% da sua frequência, ele gasta duas unidades de tempo para executar a tarefa. Quando ele está usando 50%, o tempo dobra. Quando está usando 25%, o tempo de execução quadruplica, ou seja, o que é economizado usando uma frequência menor é gasto pois a tarefa demora mais tempo para ser executada por completo. Este comportamento é comprovado pela equação de consumo de energia:

$$E = \frac{f}{x}v^2C_{ef}t(x) \tag{2.5}$$

com essa fórmula pode-se evidenciar que caso uma frequência seja dividida por x, o tempo t cresce na mesma razão que a frequência decresce, sendo multiplicado por x, o que acaba por não gerar economia nenhuma de energia.

2.4.1 *DVS (Dynamic Voltage Scaling) Considerando Somente o Processador*

DVS é uma técnica que realiza a economia de energia considerando a troca de frequência e voltagem do processador, podendo ser aplicada tanto de forma estática quanto dinâmica (ZHU; MUELLER, 2005; BINI; BUTTAZZO; LIPARI, 2009; PILLAI; SHIN, 2001). De

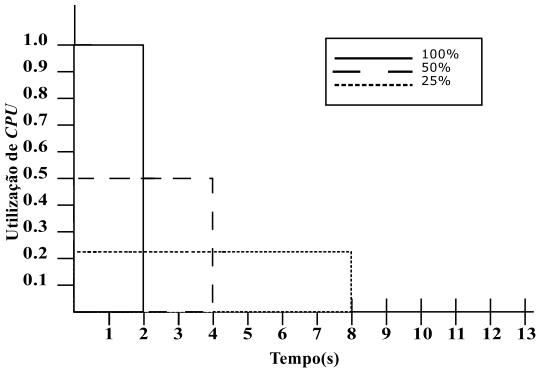

Figura 5: Consumo de energia do processador de acordo com a modificação da frequência. A área de cada retângulo representa a energia consumida.

fato essa técnica inicialmente foi aplicada com um modelo de energia simples considerando somente o uso de energia do processador, ficando representado pela Equação (2.4).

Inicialmente, mesmo com essa forma simples de modelar o consumo de energia, os resultados foram considerados muito eficientes (ZHAO; AYDIN, 2009). Este modelo, apesar de ser adequado para regular o consumo energético do processador, não considera os outros dispositivos utilizados pelo sistema. Portanto, o *DVS* aplicado dessa forma não garante diminuição no consumo energético do sistema, pois devido à diminuição do consumo de energia do processador, outros dispositivos do sistema podem ficar sobrecarregados tendo que funcionar por mais tempo devido à diminuição de frequência do processador.

2.4.2 *DPM (Dynamic Power Management)*

O *DPM* é um modelo energético que regula a frequência e voltagem do processador com o uso do *DVS*, mas também troca os dispositivos do computador de um modo de maior consumo de energia para um de menor, modo *sleep*. Nesta modelagem é considerado que uma tarefa precisa alocar outros dispositivos além da *CPU* enquanto executada e, quanto mais tempo ela levar para executar, mais esses outros dispositivos irão consumir energia.

Assim, pode-se dividir a energia utilizada em dois grupos: *on-*

chip e *off-chip*. O primeiro é influenciado pela frequência do processador, enquanto que o segundo não (AYDIN; DEVADAS; ZHU, 2006; ZHU; MELHEM; MOSSÉ, 2004). O modelo de energia consumido por uma tarefa fica então descrito por:

$$P_{sis} = P^S + (P^I + P^D)h \tag{2.6}$$

onde o $h \in \{0;1\}$ indica se tarefa esta ou não em estado de *sleep*, enquanto os termos P^S, P^I e P^D representam as potências do sistema. P^S é a potência mínima para manter os componentes do sistema ligados em modo *sleep*. P^D e P^I expressam, respectivamente, as potências dependentes e independentes da frequência da *CPU*.

Assume-se que o sistema nunca é desligado em partes ou por completo, pois a reinicialização incorre em um gasto de energia alto e inviável (ELNOZAHY; KISTLER; RAJAMONY, 2003). Como o não desligamento não é possível, P^S é constante e, portanto, a potência do sistema quando ele não esta no modo de *sleep* pode ser escrita como:

$$P_{sis} = (P^I + P^D)h \tag{2.7}$$

simplificando o cálculo da potência do sistema.

O gasto de energia por unidade de tempo do processador é dado de forma aproximada por $C_{ef}v^2 f$ (BURD; BRODERSEN, 1995), onde C_{ef} representa a capacitância do processador e v a voltagem necessária para operar em uma frequência f. Para o correto funcionamento do sistema, a voltagem deve ser reduzida ou aumentada de forma linear à frequência. Com isso tem-se que $v = fV$, na qual V é a voltagem máxima do processador e v a voltagem necessária para uma frequência f selecionada (ZHU; MELHEM; MOSSÉ, 2004). Assim, pode-se escrever:

$$P^D = C_{ef}V^2 f^3 \tag{2.8}$$

Nesse trabalho é usado o modelo *DPM*, pois ele leva em consideração o consumo de energia de vários outros componentes do sistema para garantir a economia de energia. Com esse modelo tem-se a certeza de que uma economia gerada pela diminuição de frequência não irá acarretar o aumento do consumo total do sistema.

2.5 RECONFIGURAÇÃO DINÂMICA DE ESCALONADORES EM TEMPO REAL

As políticas de servidores criadas para tarefas aperiódicas são capazes de prover uma solução aceitável para alguns problemas. Entretanto muitas definições são feitas em etapa de projeto. No caso do *CBS*, a reserva de banda é fixada nesse momento e tem que contemplar todas as situações possíveis em sistemas do tipo multimodal (REAL; CRESPO, 2004), onde uma tarefa pode ter mais de um modo de execução. Tal abordagem pode não ser satisfatória.

Em tarefas multi-modais cada modo de execução é caracterizado por tempos de execução e períodos diferenciados. Tal fato ocorre pois cada modo deve priorizar objetivos diferentes em questões de nível de qualidade e desempenho. Um exemplo seria um robô: quando entra em um novo ambiente é necessário mapeá-lo, depois traçar uma rota e no momento que começar a se locomover e monitorar o ambiente, cada uma dessas ações tem um comportamento diferenciado, pois os seus objetivos são distintos, o que exige uma reconfiguração e troca de modo .

Entretanto esse processo de troca de modo pode ocorrer de formas diferentes e ser tratado em diferentes níveis. Como este trabalho propõe-se a projetar um algoritmo para escolha de modo e frequência de uma tarefa, o escalonador será responsável por realizar as mudanças de versão da tarefa em execução e reserva de banda feita para o servidor.

Neste ponto é importante definir a diferença entre modificar em tempo de execução os parâmetros do escalonador (*i.e.* um processo realizado por um protocolo de mudança de modos) e os métodos de escolha dos valores de cada parâmetro, o que no caso deste trabalho é chamado de reconfiguração dinâmica de escalonadores de tempo real. A decisão de que modo usar (*e.g.*, se o robô está reconhecendo o ambiente ou em movimento) cabe à reconfiguração dinâmica, enquanto a troca de modo no escalonador cabe ao protocolo de mudança de modos.

Os protocolos que realizam a mudança de modos podem ser divididos em síncronos ou assíncronos. No primeiro caso todas as tarefas trocam de modo simultaneamente, enquanto no segundo a mudança de modo das tarefas de um sistema ocorrem de forma independentemente, causando a existência de uma mistura de tarefas em seu modo antigo (esperando a mudança de modo) e tarefas que já fizeram a mudança e se encontram no novo modo.

Para tratar a questão de reconfiguração de escalonadores exis-

tem várias pesquisas e soluções, como por exemplo o uso da teoria de controle em malha fechada e uma política de escalonamento proposta em (BUTTAZZO et al., 2002) capaz de redimensionar as tarefas como se fossem molas. A primeira abordagem é muito complicada de ser colocada em prática, por causa da dificuldade em se determinar uma lei de controle estável para o sistema enquanto a segunda necessita de informações do pior caso. Diferente dessas propostas, esse trabalho busca prover suporte a grandes ajustes aos parâmetros de servidores, sendo baseado em reserva de banda em tempo de execução e levando em conta a garantia na restrição de consumo de energia e sem o conhecimento do pior caso.

2.6 ESCALONADORES PARA TEMPO REAL COM ECONOMIA DE ENERGIA

Tendo em vista a dificuldade do escalonamento em tempo real e a necessidade de economia de energia encontrada em certos sistemas, vários trabalhos estão sendo desenvolvidos sobre este tema.

Dois algoritmos foram propostos em (PILLAI; SHIN, 2001), os quais podem ser aplicados tanto a políticas de escalonamento *RM* quanto *EDF* com tarefas do tipo *hard*. Esses algoritmos trabalham aproveitando o *slack time* para diminuir a frequência da tarefa e funcionam em tempo de execução, podendo se aproveitar de algum tempo que uma tarefa devia usar mas não usou e recalculando o tempo de *CPU* que cada tarefa pode usar.

Alguns *frameworks* para sistemas de tempo real com reconfiguração dinâmica e tarefas multimodais já foram propostos em (OLIVEIRA; CAMPONOGARA; LIMA, 2009; LIMA; CAMPONOGARA; SOKOLONSKI, 2008). Onde foram propostos um modelo probabilístico que garante a escalonabilidade do sistema e um modelo determinístico para escalonar as tarefas usando degradação. No entanto, nenhum desses trabalhos leva em consideração economia de energia.

Em (ZHU; MELHEM; MOSSÉ, 2004), é apresentado o modelo *DPM* para modelar o consumo de energia do sistema. Esse artigo usa o modelo *DPM* para tentar economizar energia com o aproveitamento do *slack time*, tempo o qual o processador deveria ficar ocioso. Nesse tipo de técnica a economia de energia se dá através do melhor esforço, sem garantia de quanto vai economizar. Por isso o sistema foi modelado com duas restrições: a utilização da *CPU* e a energia total consumida pelo sistema. Esta abordagem é voltada a sistemas de tarefas críticas.

Em (ZHU; MUELLER, 2005) foi utilizado um controle que usa o modelo *DVS*, sem levar em conta o consumo total do sistema composto por tarefas críticas. Nesse sistema a execução de uma tarefa é dividida em etapas, as quais servem para evitar que o *deadline* seja perdido, pois se uma parte atrasar o restante da tarefa é executada com 100% de *CPU*, evitando o atraso na finalização da tarefa.

Uma solução visando escalonar tarefas de tempo real críticas com garantia de economia de energia foi apresentada em (RUSU; MELHEM; MOSSÉ, 2003). Nesse trabalho foram desenvolvidos dois algoritmos, um chamado *REW-Pack* e outro *REW-Unpack*, que recebem um grupo de tarefas onde a cada uma é associada uma quantidade de energia necessária para executar em uma frequência específica. Caso o conjunto de tarefas não seja escalonável, as tarefas podem ser descartadas para o sistema ficar escalonável.

A partir deste levantamento bibliográfico, constatou-se que existem trabalhados que gerenciam o consumo de energia com garantia de economia e por meio do melhor esforço em sistemas *hard real-time*. As soluções encontradas que trabalham com garantia de economia de energia como (RUSU; MELHEM; MOSSÉ, 2003) tem um modelo de tarefas muito restrito e um modelo energético simples, onde a cada tarefa e frequência é associado um gasto de energia.

2.7 SUMÁRIO

Este capítulo apresentou os conceitos necessários para o entendimento do trabalho, como a fundamentação de escalonamento de sistemas em tempo real, mostrando duas políticas de escalonamento, uma com prioridade fixa e outra dinâmica. O tratamento de tarefas não-críticas com ênfase no *CBS* que foi detalhado, pois este modelo foi escolhido para os testes práticos. A modelagem energética do sistema procurou ser a mais fiel possível aos sistemas reais. A questão de mudança de modos, necessária à reconfiguração dinâmica dos parâmetros dos servidores, foi abordada e por último foi feita uma revisão sobre trabalhos relacionados ao tema.

3 RECONFIGURAÇÃO DINÂMICA COM RESTRIÇÃO DE ENERGIA

Este capítulo apresenta o modelo e os algoritmos desenvolvidos para o problema de reconfiguração dinâmica com servidores *CBS*, sob restrições de escalonabilidade *EDF* e consumo de energia. O capítulo inicia com a apresentação da notação, seguido do modelo matemático do problema, uma instância exemplo para demonstrar o algoritmo de programação dinâmica e uma heurística gulosa que serão explicados por último.

3.1 NOTAÇÕES

O sistema considerado para esse trabalho é constituído de n tarefas que compartilham um único processador. Cada tarefa τ_i é executada por um servidor S_i definido por uma tupla (Q_i, T_i). Cada servidor S_i possui um orçamento máximo Q_i, previsto em cada intervalo T_i. Assume-se ainda que cada servidor é escalonado pela política *EDF*, com o período T_i igual ao *deadline* do servidor.

Um processador pode operar em um determinado conjunto de frequências $\mathscr{F}$. Sem perda de generalidade, pode-se normalizar o conjunto $\mathscr{F} = \{f_1; ...; f_m\}$, onde $0 < f \leq 1$ para todo $f \in \mathscr{F}$ e m é o número de frequências admitidas pelo processador. Como exemplo, caso a *CPU* opere com as frequências $\{632,5Mhz; 1,265Ghz; 2,53Ghz\}$, tem-se o conjunto normalizado igual a $\mathscr{F} = \{0,25; 0,5; 1\}$.

O orçamento Q_i é expresso com base no tempo de uso da *CPU* e de outros recursos computacionais, que venham a ser utilizados por uma tarefa. Assim, o orçamento fica definido pela soma de Q_i^D e Q_i^I. Sendo Q_i^D o maior tempo que uma tarefa pode consumir de *CPU*, o qual varia dependendo da frequência em uso. Q_i^I é o tempo usado por outros recursos, como memória, disco rígido, *etc.*, os quais são independentes da frequência escolhida. O orçamento total de S_i é expresso por:

$$Q_i = \frac{Q_i^D}{f_j} + Q_i^I \tag{3.1}$$

A utilização do processador é estabelecida pelo somatório das utiliza-

ções de cada servidor S_i, que são calculadas por:

$$u_i = \frac{Q_i^I}{T_i} + \frac{Q_i^D}{T_i f_j} \tag{3.2}$$

Assim, para que as tarefas possam ser executadas em um escalonador *EDF*, tem-se a seguinte condição $\sum_{i=1}^{n} u_i \leq 1$.

As tarefas podem ser executadas diferentes números de vezes durante a execução do sistema, por terem diferentes períodos. Motivo que pode levar tarefas que usam menos energia para executar uma instância a consumir mais que outras durante o funcionamento do sistema.

Para evitar que tarefas com maior custo energético por execução de uma instância, mas que consumam menos energia durante a execução do sistema, sejam penalizadas e ao mesmo tempo garantir a economia de energia desejada, multiplicamos a potência de uma tarefa por sua utilização. Dessa forma o gasto energético de uma tarefa será dado em relação ao seu custo por unidade de tempo de acordo com seu período.

O presente trabalho trata o problema de escolher o nível de degradação das tarefas, modo de execução, sob restrições escalonabilidade e energia disponível como um problema de otimização. Cada combinação de nível de degradação de uma tarefa e frequência disponibilizada pelo processador é chamada de configuração. Cada uma das configurações do servidor S_i tem um valor de benefício agregado, o qual representa o quanto uma configuração é vantajosa para o sistema. O valor do benefício deve ser calculado de forma a ter um valor maior nas configurações mais desejadas.

No desenvolvimento deste trabalho o benefício é dado por $A(u_i, U_i, f_j)$, onde u_i é a utilização da configuração em análise, U_i é a utilização desejada para uma tarefa e f_j é a frequência usada. Para o cálculo do benefício é assumido que os modos de cada tarefa estão ordenados de forma crescente pela utilização, considerando a frequência máxima possível, ficando o valor do benefício definido por:

$$A(u_i^{k,j}, U_i, f_j) = \begin{cases} \frac{\min(u_i^{k,j}, U_i)}{U_i} + (2 f_j) & \text{se } k = \kappa(i) \\ \frac{\min(u_i^{k,j}, U_i)}{U_i} + \left(\frac{\min(u_i^{k+1,j}, U_i)}{U_i} - \frac{\min(u_i^{k,j}, U_i)}{U_i} \right) (f_j) & \text{se } k < \kappa(i) \end{cases} \tag{3.3}$$

Essa forma de calcular o benefício faz com que quanto maior a utilização e frequência usada por uma configuração, maior será o benefício agregado a ela. Para encontrar a melhor configuração para

o sistema, deve-se maximizar $\sum_{S_i \in \mathscr{S}} A(u_i, U_i, f_j)$, sendo $\mathscr{S}$ um conjunto de servidores.

3.2 MODELO

Assumimos que o processador pode operar em uma frequência f de um conjunto finito de frequências $\mathscr{F} = \{f_1; \ldots; f_m\}$, com m sendo o número de frequências possíveis do processador. Há um conjunto $\mathscr{S} = \{S_1; \ldots; S_n\}$ de servidores. Cada servidor S_i pode operar em um conjunto de modos, com cada modo $k \in K_i = \{1; \ldots; \kappa(i)\}$ definido pela tupla $(Q_i^{k,I}, Q_i^{k,D}, P_i^{k,I}, T_i^k)$.

O problema consiste em definir o modo de operação e a frequência do processador para cada um dos servidores, buscando maximizar o benefício agregado do sistema enquanto garante-se a escalonabilidade do sistema sem ultrapassar o consumo de energia máximo definido pelo usuário. Esse é um problema da mochila com duas dimensões (MARTELLO; TOTH, 2003, 1990), pois para escolher a configuração que uma tarefa irá usar tem-se que levar em conta a energia disponibilizada para o sistema e a capacidade de processamento da *CPU*. Matematicamente, o problema é formulado como:

$$P: \quad \max f = \sum_{S_i \in \mathscr{S}} \sum_{(k,j) \in \Omega_i} A_i^{k,j} x_i^{k,j} \tag{3.4a}$$

$$\text{s.a}: \sum_{(k,j) \in \Omega_i} x_i^{k,j} = 1, \; S_i \in \mathscr{S} \tag{3.4b}$$

$$\sum_{S_i \in \mathscr{S}} \sum_{(k,j) \in \Omega_i} u_i^{k,j} x_i^{k,j} \leq 1 \tag{3.4c}$$

$$\sum_{S_i \in \mathscr{S}} \sum_{(k,j) \in \Omega_i} P_i^{k,j} x_i^{k,j} \leq \beta P^{\star} \tag{3.4d}$$

$$x_i^{k,j} \in \{0,1\}, S_i \in \mathscr{S}, (k,j) \in \Omega_i \tag{3.4e}$$

onde:

- $\Omega_i = \{(k,j) : k \in K_i, j \in F\}$ é o conjunto de configurações para o servidor S_i;
- a Equação (3.4a) é a função objetivo que ao ser maximizada encontrá a resposta do problema P;
- a Equação (3.4b) é uma restrição do somatório de $x_i^{k,j}$, a qual as-

sume o valor 1 se o servidor S_i for operar com a configuração (k, j). Esta restrição garante a escolha de somente uma configuração por servidor;

- $F = \{1;2;3;\ldots;m\}$ é o conjunto de índices das frequências do processador, sendo que $j \in F$ se e somente se $f_j \in \mathscr{F}$;

- $u_i^{k,j}$ é a utilização do processador pelo servidor S_i quando no modo k e frequência f_j, sendo definida por:

$$u_i^{k,j} = \frac{Q_i^{k,D}}{T_i^k f_j} + \frac{Q_i^{k,I}}{T_i^k} \tag{3.5}$$

- a Equação (3.4c) limita o uso da *CPU* pela restrição de escalonabilidade do *EDF*;

- a Equação (3.4d) limita a energia consumida por cada servidor S_i, quando no modo k e frequência f_j, usando uma modelagem *DPM* definida por:

$$P_i^{k,j} = (P_i^{k,I} + C_{ef} V^2 f_j) u_i^{k,j} \tag{3.6}$$

onde $P_i^{k,I}$ é a energia consumida por dispositivos que não dependem da frequência do processador e $C_{ef} V^2 f_j$ é a energia consumida pelo processador;

- na Equação (3.4b) o parâmetro $P^\star$ é o consumo de energia máximo do sistema, dado por $P^\star = \sum_{S_i \in \mathscr{S}} \max\{P_i^{k,j} : (k, j) \in \Omega_i\}$ e $\beta \in (0, 1)$ é o parâmetro que define a energia disponível.

3.3 INSTÂNCIA EXEMPLO

A instância a ser usada para ilustrar os algoritmos apresentados nesse capítulo pressupõe uma *CPU* com as seguintes configurações: $V = 1,25$, $\mathscr{F} = \{1;0,5;0,25\}$ com frequência máxima da *CPU* de $2,53Ghz$, $P_i^{k,\mathrm{I}} = 0,438$ (energia utilizada para manter a memória do computador funcionando), $C_{ef} = 6,324 \times 10^{-9}$ e $P^\star = 10,5$. Os valores $Q_i^{k,\mathrm{I}}$ e $Q_i^{k,\mathrm{D}}$ definidos para os conjuntos de tarefas foram gerados pelo algoritmo UUniFast, apresentado em (BINI; BUTTAZZO, 2005). Ele é capaz de gerar cenários de forma eficiente, evitando a formação de cenários muito

otimistas ou pessimistas. Os parâmetros de utilização são gerados de forma aleatória, uniformemente distribuídos no intervalo $[0,1]$.

O sistema em questão é composto por três servidores, cada um responsável por uma tarefa, cujos parâmetros são dados na Tabela 4. Os valores das demandas computacionais são dados na Tabela 5, calculados conforme a Eq. (3.5). A energia consumida é dada na Tabela 6, obtida com base na Eq. (3.6). E por último, os valores dos benefícios associados a cada modo aparecem na Tabela 7, com o uso da Eq. (3.3).

Tabela 4: Parâmetros de configuração dos servidores.

	$(\ Q_i^{k,\mathrm{I}}, Q_i^{k,\mathrm{D}}, P_i^{k,\mathrm{I}}, T_i^k)$		
k	$i=1$	$i=2$	$i=3$
1	(0,1;15,7;0,438;33)	(0,3;7,6;0,438;33)	(2,2;0,03;0,438;33)
2	(0,2;1,9;0,438;67)	(1,9;16,6;0,438;67)	(1,2;1,5;0,438;67)
3	(0,4;8,2;0,438;200)	(0,3;4,8;0,438;200)	(2,7;3,7;0,438;200)

Tabela 5: Utilização da CPU.

	$u_i^{k,j}$		
	$i=1$	$i=2$	$i=3$
$u_i^{1,1}$	47,33%	24,96%	7,71%
$u_i^{1,2}$	63,05%	32,55%	7,74%
$u_i^{1,3}$	94,47%	35,91%	7,81%
$u_i^{2,1}$	3,22%	27,63%	4,1%
$u_i^{2,2}$	4,26%	47,72%	4,9%
$u_i^{2,3}$	6,10%	52,46%	7,4%
$u_i^{3,1}$	4,18%	2,55%	3,19%
$u_i^{3,2}$	5,62%	3,35%	3,80%
$u_i^{3,3}$	8,35%	4,96%	5,02%

Nas duas seções a seguir serão apresentados dois algoritmos propostos para solucionar o problema modelado anteriormente. Primeiro será apresentada a formulação do algoritmo baseado no método de programação dinâmica e o resultado de sua aplicação na instância de exemplo. Por último será apresentado uma heurística e sua aplicação na instância exemplo.

Tabela 6: Energia Consumida.

	$P_i^{k,j}$		
	$i=1$	$i=2$	$i=3$
$P_i^{1,1}$	12,0403	6,3499	1,9602
$P_i^{1,2}$	6,9254	3,5752	0,8504
$P_i^{1,3}$	3,3660	1,7001	0,2783
$P_i^{2,1}$	0,8187	7,0282	1,0565
$P_i^{2,2}$	0,4589	3,9442	0,5401
$P_i^{2,3}$	0,2172	1,8691	0,2296
$P_i^{3,1}$	1,0839	0,6483	0,8115
$P_i^{3,2}$	0,6177	0,3684	0,4176
$P_i^{3,3}$	0,2974	0,1769	0,1790

Tabela 7: Benefício do Sistema

	$A_i^{k,j}$		
	$i=1$	$i=2$	$i=3$
$A_i^{1,1}$	3,0	1,00	3,0
$A_i^{1,2}$	2,5	1,00	2,5
$A_i^{1,3}$	2,0	1,00	2,0
$A_i^{2,1}$	0,09	3,0	1,00
$A_i^{2,2}$	0,08	2,5	1,00
$A_i^{2,3}$	0,079	2,0	0,76
$A_i^{3,1}$	1,00	1,00	0,54
$A_i^{3,2}$	0,77	0,77	0,55
$A_i^{3,3}$	0,55	0,55	0,48

3.4 ALGORITMO DE PROGRAMAÇÃO DINÂMICA

Programação dinâmica é um método para a síntese de algoritmos de resolução de problemas computacionais, como por exemplo, os de otimização combinatória. Esse tipo de algoritmo consiste em quebrar o problema principal em instâncias menores, que podem ser resolvidas recursivamente para calcular o resultado ótimo.

Para aplicar programação dinâmica ao problema de reconfiguração P, se faz necessário a obtenção de um problema equivalente envol-

vendo apenas inteiros nas constantes de utilização e energia do sistema. Para isso é definido $\Omega = \{(i,k,j) : S_i \in \mathscr{S}, (k,j) \in \Omega_i\}$ e duas constantes positivas Λ e Γ tal que $\Lambda u_i^{k,j}$ e $\Gamma P_i^{k,j}$ são inteiros para toda a tripla $(i,k,j) \in \Omega$. Então P pode ser colocado na forma equivalente,

$$P: \quad f = \max \sum_{(i,k,j)\in\Omega} A_i^{k,j} x_i^{k,j} \tag{3.7a}$$

$$\text{s.a}: \sum_{(k,j)\in\Omega_i} x_i^{k,j} = 1,\ S_i \in \mathscr{S} \tag{3.7b}$$

$$\sum_{(i,k,j)\in\Omega} \Lambda u_i^{k,j} x_i^{k,j} \leq \Lambda \tag{3.7c}$$

$$\sum_{(i,k,j)\in\Omega} \Gamma P_i^{k,j} x_i^{k,j} \leq \Gamma\beta P^{\star} \tag{3.7d}$$

$$x_i^{k,j} \in \{0,1\}, (i,k,j) \in \Omega \tag{3.7e}$$

Seja $\mathscr{S}_i = \{S_1; \ldots; S_i\}$ o conjunto com os i primeiros servidores. Define-se então o subproblema de reconfigurar os l primeiros servidores com uma disponibilidade δ de recurso computacional e um consumo máximo γ de energia, conforme segue:

$$P_l(\delta,\gamma): \quad f_l(\delta,\gamma) = \max \sum_{S_i\in\mathscr{S}_l} \sum_{(k,j)\in\Omega_i} A_i^{k,j} x_i^{k,j} \tag{3.8a}$$

$$\text{s.a}: \sum_{(k,j)\in\Omega_i} x_i^{k,j} = 1,\ S_i \in \mathscr{S}_l \tag{3.8b}$$

$$\sum_{S_i\in\mathscr{S}_l} \sum_{(k,j)\in\Omega_i} \Lambda u_i^{k,j} x_i^{k,j} \leq \delta \tag{3.8c}$$

$$\sum_{S_i\in\mathscr{S}_l} \sum_{(k,j)\in\Omega_i} \Gamma P_i^{k,j} x_i^{k,j} \leq \gamma \tag{3.8d}$$

$$x_i^{k,j} \in \{0,1\}, S_i \in \mathscr{S}_l, (k,j) \in \Omega_i \tag{3.8e}$$

que, por sua vez pode ser escrito de forma recursiva:

$$P_l(\delta,\gamma): \quad f_l(\delta,\gamma) = \max \sum_{(k,j)\in\Omega_l} A_l^{k,j} x_l^{k,j} + f_{l-1}(\delta-\delta_l, \gamma-\gamma_l) \tag{3.9a}$$

$$\text{s.a}: \sum_{(k,j)\in\Omega_l} x_l^{k,j} = 1 \tag{3.9b}$$

$$\delta_l = \sum_{(k,j)\in\Omega_l} \Lambda u_l^{k,j} x_l^{k,j} \leq \delta \tag{3.9c}$$

$$\gamma_l = \sum_{(k,j)\in\Omega_l} \Gamma P_l^{k,j} x_l^{k,j} \leq \gamma \tag{3.9d}$$

$$x_l^{k,j} \in \{0,1\}, (k,j) \in \Omega_l \tag{3.9e}$$

onde $f_l(\delta,\gamma) = -\infty$ se $[\Lambda u_l^{k,j}\ \Gamma P_l^{k,j}] \not\leq [\delta\ \gamma]$ para todo $(k,j) \in \Omega_l$. E ainda $f_0(\delta,\gamma) = 0$ para qualquer δ e γ.

Algoritmo 1: Algoritmo de Programação Dinâmica

```
   input: 𝒮, A_i^{k,j}, u_i^{k,j}, P_i^{k,j}, β, P*, Λ e Γ
 1 for δ := 0 to Λ do
 2     for γ = 0 to ΓβP* do
 3         f_0(δ,γ) := 0
 4         w_0(δ,γ) := (0,0)
 5 for l := 1 to n do
 6     for δ_l := 0 to Λ do
 7         for γ_l := 0 to ΓβP* do
 8             f_l^max := −∞
 9             (k_l^max, j_l^max) := (0,0)
10             for all (k,j) ∈ Ω_l do
11                 if Λu_l^{k,j} ≤ δ_l and ΓP_l^{k,j} ≤ γ_l then
12                     f_l := A_l^{k,j} + f_{l−1}(δ_l − Λu_l^{k,j}, γ_l − ΓP_l^{k,j})
13                     if f_l > f_l^max then
14                         f_l^max := f_l
15                         (k_l^max, j_l^max) := (k,j)
16             f_l(δ_l,γ_l) := f_l^max
17             w_l(δ_l.γ_l) := (k_l^max, j_l^max)
18 return(f,w)
```

Em essência, o algoritmo PD resolve a família de problemas $P_l(\delta,\gamma)$, resolvendo a sequência $P_0(\delta,\gamma)$, $P_1(\delta,\gamma)$, ..., $P_n(\delta,\gamma)$ para todo $\delta \in \{0;\ldots;\Lambda\}$ e $\gamma \in \{0;\ldots;\Gamma\beta P^{\star}\}$. O problema P é equivalente ao problema $P_n(\Lambda,\Gamma\beta P^{\star})$. Com ele pode-se chegar ao Algoritmo 1, baseado em programação dinâmica que é construído de acordo com a recursão (3.9a)–(3.9e) do problema (3.9).

No Algoritmo 1, $f_l^{\max}$ armazena o maior valor de benefício que pode ser obtido com uma quantidade determinada de γ_l e δ_l, com as frequências e modos disponibilizados pelo processador, enquanto a configuração fica registrada em ($k_l^{\max}$, $j_l^{\max}$). O primeiro laço aninhado de *for* corresponde ao caso base do problema onde nenhuma configuração ainda foi selecionada. Após isso existem quatro laços aninhados que constituem o laço principal do programa, responsável por realizar o procedimento de escolha das melhores configurações do problema.

A partir do pseudo-código do algoritmo de programação dinâmica, pode-se facilmente determinar que o algoritmo tem um tempo de execução dado por $\Theta(n\Lambda\Gamma\beta P^{\star}|\Omega|)$, que tem uma complexidade pseudo-polinomial. Sendo a memória utilizada pelo algoritmo no armazenamento das tabelas f e w da ordem de $\Theta(n\Lambda\Gamma\beta P^{\star})$, que também cresce em uma taxa pseudo-polinomial. A memória utilizada deve crescer linearmente de acordo com o aumento dos valores de Λ, Γ e β.

O algoritmo de PD foi aplicado ao problema P dado pela instância exemplo da Seção 3.3, com $\Lambda = 100$, $\Gamma = 100$ e β variando entre 1 e 0,1. Com isso foi gerado o gráfico mostrado na Figura 6, onde é possível perceber que com o aumento da energia do sistema, o benefício também aumenta. Devido à natureza do algoritmo é possível afirmar que a qualidade da resposta vai depender dos valores dos multiplicadores: quanto mais próximos dos números que transformam qualquer $u_i^{k,j}$ e $P_i^{k,j}$ em valores inteiros, melhor a precisão da resposta.

3.5 HEURÍSTICA PARA RECONFIGURAÇÃO DINÂMICA

O desenvolvimento de uma heurística é motivado pela alta complexidade do algoritmo de programação dinâmica. A heurística a ser desenvolvida segue uma estratégia gulosa aplicada a um problema *surrogate*, o que combina as restrições usando multiplicadores de Lagrange. Abaixo apresenta-se as relaxações Lagrangeanas e *surrogate*, seguidas pela heurística gulosa.

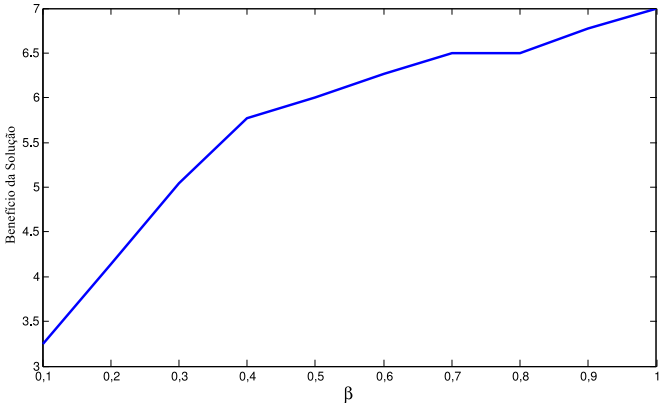

Figura 6: Variação do benefício obtido com o algoritmo de programação dinâmica de acordo com a variação de energia causada pelo parâmetro β.

3.5.1 Relaxações

Seja $\mathbf{x} = (x_i^{k,j} : (i,k,j) \in \Omega)$ um vetor com todas as variáveis do problema P e seja $\mathscr{P} = \{\mathbf{x} : \mathbf{x}$ satisfaz as Equações (3.4b) até (3.4e)$\}$ o espaço de soluções. Então, P é representado de forma compacta por $f^\star = \max\{f(\mathbf{x}) : \mathbf{x} \in \mathscr{P}\}$. Um problema R, definido por $r^\star = \max\{r(\mathbf{x}) : \mathbf{x} \in \mathscr{R}\}$, é uma relaxação de P se (WOLSEY, 1998):

i) $\mathscr{P} \subseteq \mathscr{R}$;

ii) $r(\mathbf{x}) \geq f(\mathbf{x})$ para todo $\mathbf{x} \in \mathscr{P}$.

Um problema de relaxação deve ser resolvido até sua otimalidade, obtendo $r^\star$, para que seja induzido um limite superior do valor ótimo do problema de reconfiguração P. Isto significa que $r^\star \geq f^\star$. Relaxações são elementos chaves para se estabelecer certificados de qualidade para uma solução factível[1], sendo também importantes no projeto de soluções algorítmicas voltadas à resolução de problemas de otimização, tais como algoritmos de enumeração implícita e de planos de corte que dependem dos limites obtidos com as relações.

Neste trabalho foram desenvolvidas relaxações Lagrangeana e *surrogate* que foram empregadas em uma heurística gulosa proposta para resolução do problema de reconfiguração.

[1] $\mathbf{x}$ é uma solução factível para P se $\mathbf{x} \in \mathscr{P}$

Considere o problema geral de programação inteira dado por:

$$P_I: \quad f^\star = \max\ \mathbf{c}^T\mathbf{x} \tag{3.10a}$$
$$\text{s.a}: A\mathbf{x} \leq \mathbf{b} \tag{3.10b}$$
$$\mathbf{x} \in \mathbb{Z}^n \tag{3.10c}$$

Uma relaxação R de P_I é obtida através da relaxação contínua de P_I, levando a um problema de programação linear:

$$R: \quad r^\star = \max\ \mathbf{c}^T\mathbf{x} \tag{3.11a}$$
$$\text{s.a}: A\mathbf{x} \leq \mathbf{b} \tag{3.11b}$$
$$\mathbf{x} \in \mathbb{R}^n \tag{3.11c}$$

Outra relaxação pode ser obtida através da relaxação Lagrangeana. Para um vetor $\boldsymbol{\lambda} \geq 0$ de multiplicadores de Lagrange, a relaxação Lagrangeana é dada por:

$$R_L: \quad r^\star = \min_{\boldsymbol{\lambda}\geq\mathbf{0}}\ \max_{\mathbf{x}\in\mathbb{Z}^n}\ \mathbf{c}^T\mathbf{x} - \boldsymbol{\lambda}^T(A\mathbf{x}-\mathbf{b}) \tag{3.12a}$$

Por fim, a relaxação *surrogate* é induzida com um vetor de pesos $\mathbf{w} \geq \mathbf{0}$,

$$R_S: \quad r^\star = \max\ \mathbf{c}^T\mathbf{x} \tag{3.13a}$$
$$\text{s.a}: \mathbf{w}^TA\mathbf{x} \leq \mathbf{w}^T\mathbf{b} \tag{3.13b}$$
$$\mathbf{x} \in \mathbb{Z}^n \tag{3.13c}$$

Considere o problema de reconfiguração dado pelas Equações (3.4a)-(3.4e). Uma relaxação R é facilmente obtida permitindo que as variáveis de decisão assumam valores reais. Isto significa que $x_i^{k,j} \in [0;1]$ para todo $(i,k,j) \in \Omega$ e, portanto, $\mathscr{R} = \{\mathbf{x} : \mathbf{x}$ satisfaz (3.4b) a (3.4d), $\mathbf{0} \leq \mathbf{x} \leq \mathbf{1}\}$. Claramente $\mathscr{P} \subseteq \mathscr{R}$. Se definirmos $r(\mathbf{x}) = f(\mathbf{x})$, então a segunda condição é também satisfeita. Portanto o problema R resultante, definido por $r^\star = \max\{r(\mathbf{x}) : \mathbf{x} \in \mathscr{R}\}$, é uma relaxação válida de P.

Resolvendo a relaxação para a instância exemplo definida na Seção 3.3 com um algoritmo de programação linear, encontraremos o valor $r^\star \geq f(\mathbf{x}^\star) \geq f(\mathbf{x})$, $\forall \mathbf{x} \in \mathscr{P}$, onde $\mathbf{x}^\star$ é a solução ótima de P. Como pode ser visto na Figura 7, a solução encontrada com a solução da relaxação continua R gerou um limite superior, sempre maior o valor da solução ótima do problema original.

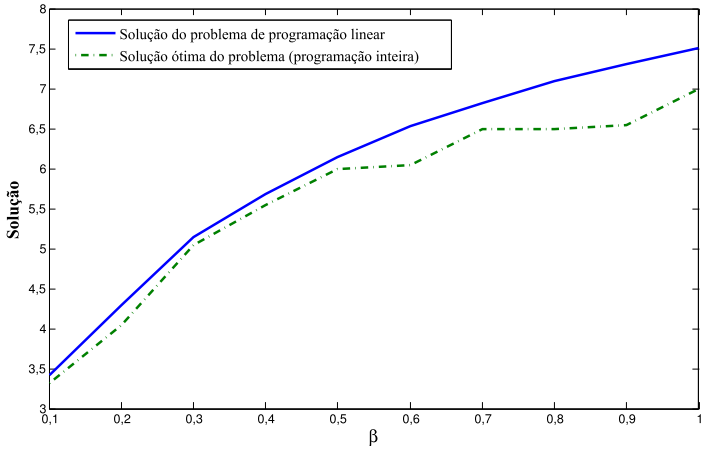

Figura 7: Gráfico com resultados das soluções do problema de reconfiguração dinâmica e da sua relaxação linear, considerando a instância exemplo com variação de β.

3.5.2 Relaxação Lagrangeana

Dado os multiplicadores de Lagrange para a restrição de escalonabilidade e o consumo de energia, λ_u e λ_p respectivamente, a função dual Lagrangeana f_L para o problema de reconfiguração é obtida por:

$$LP(\lambda): \quad f_L(\lambda) = \max \sum_{(i,k,j)\in\Omega} (A_i^{k,j} - \lambda_u u_i^{k,j} - \lambda_p P_i^{k,j}) x_i^{k,j} + \lambda_u + \lambda_p \beta P^\star \tag{3.14a}$$

$$\text{s.a}: \sum_{(k,j)\in\Omega_i} x_i^{k,j} = 1, \; S_i \in \mathscr{S} \tag{3.14b}$$

$$x_i^{k,j} \in \{0,1\}, (i,k,j) \in \Omega \tag{3.14c}$$

onde $\lambda = (\lambda_u, \lambda_p)$. Para qualquer $\lambda \geq \mathbf{0}$, a função Lagrangeana dual induz um limite superior $f_L(\lambda) \geq f^\star$. Está função dual tem uma propriedade que permite que ela seja computada de forma analítica, como mostrado abaixo:

1. Define-se $(\hat{k}, \hat{j})(i) = \arg\max\{(A_i^{k,j} - \lambda_u u_i^{k,j} - \lambda_p P_i^{k,j}) : (k,j) \in \Omega_i\}$.

2. a solução $\mathbf{x}(\lambda)$ de $LP(\lambda)$ é obtida definindo, para todo $S_i \in \mathscr{S}$,

$x_i^{k,j}(\lambda) = 1$ se $(k,j) = (\hat{k},\hat{j})(i)$ e caso contrário $x_i^{k,j}(\lambda) = 0$ para todo $(k,j) \in \Omega_i$.

O problema Lagrangeano dual consiste em encontrar um vetor $\lambda^\star$ o qual minimiza o limite superior f_L (FISHER, 2004). Formalmente, o Lagrangeano dual é escrito como:

$$LD: f_\mathrm{L}(\lambda^\star) = \min_{\lambda \geq \mathbf{0}} f_\mathrm{L}(\lambda) \tag{3.15}$$

De acordo com a teoria da dualidade, o problema Lagrangeano dual é convexo, pois $f_\mathrm{L}(\lambda)$ é uma função convexa e não diferenciável. Contudo a propriedade de f_L não ser diferenciável impede o uso de algoritmos eficientes baseados em gradientes como o método de Newton amortecido (BOYD; VANDENBERGHE, 2004). Um algoritmo do tipo subgradiente pode ser facilmente obtido para minimizar f_L pois subgradientes são facilmente computados.

O vetor $\xi(\lambda) \in \mathbb{R}^2$ é um subgradiente para f_L em λ se $f_\mathrm{L}(\bar{\lambda}) \geq f_\mathrm{L}(\lambda) + \xi(\lambda)'(\bar{\lambda} - \lambda)$ para todo $\bar{\lambda} \geq \mathbf{0}$, factível. Dada a solução $\mathbf{x}(\lambda) = (\mathbf{x}_i^{k,j}(\lambda) : (i,k,j) \in \Omega)$ para $LP(\lambda)$, um subgradiente para f_L em λ é obtido da seguinte forma:

$$\xi(\lambda) \quad = \quad \begin{bmatrix} \xi_\mathrm{u}(\lambda) \\ \xi_\mathrm{p}(\lambda) \end{bmatrix} \quad = \quad \begin{bmatrix} 1 - \sum\limits_{(i,k,j)\in\Omega} u_i^{k,j} x_i^{k,j}(\lambda) \\ \beta P^\star - \sum\limits_{(i,k,j)\in\Omega} P_i^{k,j} x_i^{k,j}(\lambda) \end{bmatrix} \tag{3.16}$$

Com a disponibilidade de subgradientes, pode-se utilizar algoritmo de subgradiente para resolver, de forma aproximada, o problema dual Lagrangeano. O algoritmo subgradiente para resolver o Lagrangeano dual é detalhado no Algoritmo 2. Sob certas condições, este algoritmo gera uma série de multiplicadores que convergem para um vetor ótimo Lagrangeano $\lambda^\star$. Em particular, o Teorema 10.4 em (WOLSEY, 1998) garante que a série $\{f_\mathrm{L}\ (\lambda^{(k)})\}_{k=0}^{\infty}$ irá convergir para $f_\mathrm{L}(\lambda^\star)$ se $\mu^{(0)}$ e $\rho < 1$ forem suficientemente grandes.

A performance do algoritmo subgradiente depende muito da escolha da sequência $\{\mu^{(k)}\}$ que será responsável pelo tamanho do passo na direção do subgradiente. Isto significa que os parâmetros $\mu^{(0)}$ e ρ devem ser ajustados para cada problema.

O tempo de execução do Algoritmo Subgradiente (ASG) é controlado pela variável $k^{\max}$, que indica o número máximo de interações. O passo da linha 5 requer que todos os parâmetros definidos pelo problema P sejam analisados, ficando o custo computacional em $\Theta(|\Omega|)$.

Algoritmo 2: Algoritmo Subgradiente (ASG)

input: servers $\mathscr{S}$, $\{(A_i^{k,j}, u_i^{k,j}, P_i^{k,j}) : (i,k,j) \in \Omega\}$, β, Multiplicadores Lagrangeanos iniciais $\lambda^{(0)}$, passo inicial $\mu^{(0)}$, taxa de decrescimento $\rho < 1$, tolerância η e limite de interações $k^{\max}$

$k := 0$ {*contador de interações*}
$f_{\mathrm{L}}^{\mathrm{best}} := \infty$ {*melhor limite superior*}
$f^{\mathrm{best}} := -\infty$ {*melhor solução factível*}
repeat
  resolve $LP(\lambda^{(k)})$, obtendo $\mathbf{x}(\lambda^{(k)})$ e $f_{\mathrm{L}}(\lambda^{(k)})$
  if $f_{\mathrm{L}}(\lambda^{(k)}) < f_{\mathrm{L}}^{\mathrm{best}}$ **then**
    $\lambda^{\mathrm{best}} := \lambda^{(k)}$
    $f_{\mathrm{L}}^{\mathrm{best}} := f_{\mathrm{L}}(\lambda^{(k)})$
  if $\mathbf{x}(\lambda^{(k)})$ *é factível para* P *e* $f(\mathbf{x}(\lambda^{(k)})) > f^{\mathrm{best}}$ **then**
    $\mathbf{x}^{\mathrm{best}} := \mathbf{x}(\lambda^{(k)})$
    $f^{\mathrm{best}} := f(\mathbf{x}^{\mathrm{best}})$
  computar *o subgradiente* $\xi(\lambda^{(k)})$ *de acordo com a Eq.* (3.16) *usando* $\mathbf{x}(\lambda^{(k)})$
  gera *os próximos multiplicadores:*

$$\lambda_{\mathrm{u}}^{(k+1)} := \max\{0, \lambda_{\mathrm{u}}^{(k)} - \mu^{(k)} \xi_{\mathrm{u}}(\lambda^{(k)})\}$$
$$\lambda_{\mathrm{p}}^{(k+1)} := \max\{0, \lambda_{\mathrm{p}}^{(k)} - \mu^{(k)} \xi_{\mathrm{p}}(\lambda^{(k)})\}$$

  $\mu^{(k+1)} := \rho \mu^{(k)}$ {*reduz o passo do subgradiente*}
  $k := k + 1$
until $(k > k^{\max}$ *ou* $\frac{\|\lambda^{(k+1)} - \lambda^{(k)}\|}{\|\lambda^{(k)}\|} \leq \eta)$;
return λ^{best}, $f_{\mathrm{L}}^{\mathrm{best}}$, $\mathbf{x}^{\mathrm{best}}$ *e* f^{best}

Uma vez que todos os outros passos tem um tempo proporcional a n ou menor, o algoritmo apresentado executa em tempo $\Theta(|\Omega| k^{\max})$.

O algoritmo subgradiente foi aplicado à instância exemplo, com os parâmetros de entradas iguais a $\lambda^{(0)} = \mathbf{1}$, $\mu^{(0)} = 1$, $\rho = 0{,}95$, $\eta = 10^{-3}$ e $k^{\max} = 200$:

- com valor $\beta = 1$, foi alcançado $f_{\mathrm{L}}^{\mathrm{best}} = 7{,}6131$, $\lambda^{\mathrm{best}} = (0{,}3484; 0{,}1707)$ e $f^{\mathrm{best}} = 5{,}7724$, induzido pela solução $x_1^{3,2} = 1$, $x_2^{2,3} = 1$ e

$x_3^{1,1} = 1$.

- com $\beta = 0,5$, o algoritmo produziu $f_{\mathrm{L}}^{\mathrm{best}} = 6,2437$, com $\boldsymbol{\lambda}^{\mathrm{best}} = (0;\ 0,4335)$, e $f^{\mathrm{best}} = 6$ induzido pela solução $x_1^{3,1} = 1$, $x_2^{2,3} = 1$ e $x_3^{1,1} = 1$. Pode ser observado que a solução encontrada coincide com a solução ótima obtida pelo algoritmo de programação dinâmica.

Duas outras relaxações duais podem ser obtidas com a restrição da escalonabilidade (3.4c) e de consumo de energia (3.4d), chamadas respectivamente de LD_{u} e LD_{p}. A computação das funções duais dessas relaxações é mais custosa do ponto de vista computacional que $f_L(\boldsymbol{\lambda})$ visto que elas implicam em resolver uma generalização do problema da mochila. Todavia, $LP(\boldsymbol{\lambda})$ tem a propriedade de ser inteira, significando que a solução da relaxação contínua é uma solução inteira (GEOFFRION, 1974). Isto implica que o limite inferior $f_{\mathrm{L}}(\boldsymbol{\lambda}^{\star})$ tem o mesmo valor que o limite obtido com a resolução da relaxação de programação linear do problema P. Por outro lado a relaxação LD_{u} ou LD_{p} não tem a propriedade de ser inteira, portanto o limite superior obtido por estes problemas deve ser mais baixo que o limite alcançado com a programação linear.

A função Lagrangeana dual f_{L}, dualizando as restrições de escalonabilidade e consumo de energia, foi escolhida pela sua simplicidade, por exigir menos custo computacional e por combinar os recursos de *CPU* e energia, o que será importante na concepção da heurística a ser apresentada.

3.5.3 Relaxação *Surrogate*

A relaxação *surrogate* é obtida pela combinação das restrições (3.4c) e (3.4d) por meio dos multiplicadores *surrogate* $\boldsymbol{\tau} = (\tau_{\mathrm{u}},\ \tau_{\mathrm{p}}) \geq 0$ em uma única restrição (GLOVER, 1968). Para ser mais preciso, as desigualdades (3.4c) e (3.4d) são substituídas pela desigualdade do *surrogate*:

$$\sum_{(i,k,j)\in\Omega} (\tau_{\mathrm{u}} u_i^{k,j} + \tau_{\mathrm{p}} P_i^{k,j}) x_i^{k,j} \leq \tau_{\mathrm{u}} + \tau_{\mathrm{p}} \beta P^{\star} \tag{3.17}$$

Dado um vetor $\tau \geq \mathbf{0}$, a função *surrogate* dual f_S é definida pela resolução do problema:

$$SP(\tau): \quad f_S(\tau) = \max \sum_{(i,k,j)\in\Omega} A_i^{k,j} x_i^{k,j} \tag{3.18a}$$

$$\text{s.a}: \sum_{(k,j)\in\Omega_i} x_i^{k,j} = 1,\ S_i \in \mathscr{S} \tag{3.18b}$$

$$\sum_{(i,k,j)\in\Omega} (\tau_u u_i^{k,j} + \tau_p P_i^{k,j}) x_i^{k,j} \leq \tau_u + \tau_p \beta P^\star \tag{3.18c}$$

$$x_i^{k,j} \in \{0,1\}, (i,k,j) \in \Omega \tag{3.18d}$$

Uma vez que $f_S(\tau) \geq f$ para qualquer $\tau \geq \mathbf{0}$, a função dual do *surrogate* gera um limite superior para o problema de reconfiguração. Diferentemente da função dual Lagrangeana $f_L(\lambda)$, a qual é computada analiticamente, a função dual do *surrogate* $f_S(\lambda)$ é computacionalmente mais pesada, visto que ela generaliza o problema da mochila. O dual *surrogate* expressa a vontade de minimizar o limite superior do *surrogate*, sendo definida como:

$$SD: \ f_S(\tau^\star) = \min_{\tau\geq\mathbf{0}} f_S(\tau) \tag{3.19}$$

O presente trabalho não procura resolver a função dual do *surrogate*, muito menos resolver o dual *surrogate*. Em vez disso, a função dual *surrogate* e o problema serão úteis no projeto de uma heurística para resolver o problema de reconfiguração de forma aproximada.

3.5.4 Heurística Gulosa Baseada em Densidade

O procedimento heurístico desenvolvido em (MARTELLO; TOTH, 2003) para um problema da mochila com duas dimensões foi adaptado para resolver o problema de reconfiguração com duas restrições. A heurística consiste em usar o algoritmo subgradiente para obter uma aproximação $\widetilde{\lambda}$ dos multiplicadores Lagrangeanos ótimos, $\lambda^\star$, e então resolver $SP(\tau)$ com $\tau = \widetilde{\lambda}$ seguindo uma estratégia gulosa.

Definição 1. *$X \subseteq \Omega$ é um conjunto de servidores factível se:*

1. *para cada $S_i \in \mathscr{S}$ existe exatamente uma configuração $(i,k,j) \in X$, ou seja, $|\Omega_i \cap X| = 1$ para todo S_i;*

2. $\sum_{(i,k,j)\in X} u_i^{k,j} \leq 1$ *e* $\sum_{(i,k,j)\in X} P_i^{k,j} \leq \beta P^\star$.

Hipótese 1. *Existe um conjunto factível* $X(\beta)$ *das configurações dos servidores para o limite de consumo de energia estabelecido por* β.

Hipótese 2. *Para todo* $S_i \in \mathscr{S}$ *e* $(k,j),(k',j') \in \Omega_i$, *se* $u_i^{k,j} < u_i^{k',j'}$ *e* $P_i^{k,j} < P_i^{k',j'}$ *então tem-se* $A_i^{k,j} < A_i^{k',j'}$. *Caso contrário, a configuração* (k',j') *pode ser descartada.*

Na instância exemplo, o conjunto $X(\beta) = \{(1,3,1);\ (2,3,1);(3,\ 3,\ 1)\}$ é factível quando $\beta = 0,5$, pois $\sum_{(i,k,j)\in X(\beta)} P_i^{k,j} = 2,5565 < 5,25 = \beta P^\star$ e $\sum_{(i,k,j)\in X(\beta)}\ u_i^{k,j} = 0,1005 < 1$. Note que $X(\beta)$ foi obtido por meio da seleção dos modos de operação com baixa qualidade para as tarefas gerenciadas pelos servidores.

$\Omega(X) = \Omega - X$ e $\Omega_i(X) = \{(k,j) \in \Omega_i : (i,k,j) \notin X\}$ são os conjuntos de configurações sem as configurações do conjunto X. Para qualquer S_i, seja $(k,j)_{X,i}$ a configuração de S_i aparece em X, *i.e.*, $(i,(k,j)_{X,i}) \in X$. Com essa notação, X é usado para montar o problema $P(X)$ que é equivalente ao problema de reconfiguração P. Substituindo $x_i^{(k,j)_{X,i}}$ por $(1 - \sum_{(k,j)\in\Omega_i(X)} x_i^{k,j})$, chegada-se ao seguinte problema equivalente:

$$P(X): \qquad f = \max \sum_{(i,k,j)\in\Omega(X)} \Delta A_i^{k,j} x_i^{k,j} + \sum_{S_i\in\mathscr{S}} A_i^{(k,j)_{X,i}} \tag{3.20a}$$

$$\text{s.a}: \sum_{(k,j)\in\Omega_i(X)} x_i^{k,j} \leq 1,\ S_i \in \mathscr{S} \tag{3.20b}$$

$$\sum_{(i,k,j)\in\Omega(X)} \Delta u_i^{k,j} x_i^{k,j} \leq 1 - \sum_{S_i\in\mathscr{S}} u_i^{(k,j)_{X,i}} \tag{3.20c}$$

$$\sum_{(i,k,j)\in\Omega(X)} \Delta P_i^{k,j} x_i^{k,j} \leq \beta P^\star - \sum_{S_i\in\mathscr{S}} P_i^{(k,j)_{X,i}} \tag{3.20d}$$

$$x_i^{k,j} \in \{0,1\}, (i,k,j) \in \Omega(X) \tag{3.20e}$$

onde $\Delta u_i^{k,j} = u_i^{k,j} - u_i^{(k',j')_{X,i}}$, $\Delta P_i^{k,j} = P_i^{k,j} - P_i^{(k',j')_{X,i}}$ e $\Delta A_i^{k,j} = A_i^{k,j} - A_i^{(k',j')_{X,i}}$ para todo $(i,k,j) \in \Omega(X)$.

A heurística desenvolvida é inspirada na abordagem gulosa por densidade proposta em (MARTELLO; TOTH, 2003), a qual foi desenvolvida para o problema da mochila $0-1$ com duas restrições. Dado o conjunto de configurações de servidores factível X, a heurística proposta resolve $P(X)$ aproximadamente com os seguintes passos:

1. resolva o Lagrangeano dual *LD* com o algoritmo subgradiente e

encontre valores aproximados para os multiplicadores Lagrangeanos $\widetilde{\lambda} = \lambda^{\text{best}}$ e uma solução factível $\mathbf{x}^{\text{best}}$; definir uma configuração, X, viável usando $\mathbf{x}^{\text{best}}$ se a solução candidata for factível, caso contrário $X = \{(i,1,1) : S_i \in \mathscr{S}\}$;

2. obtenha $P(X)$ usando X;

3. obtenha o problema *surrogate* $SP(X,\tau)$ associado com $P(X)$ usando $\tau = \widetilde{\lambda}$, semelhante ao problema *surrogate* $SP(\tau)$;

4. siga a estratégia de densidade gulosa, ordenando as configurações de forma decrescente pela razão do benefício pela utilização da CPU e energia consumida ponderada pelos multiplicadores de Lagrange; em notação matemática está razão é dada por:

$$\Delta\widetilde{A}_i^{k,j} = \frac{\Delta A_i^{k,j}}{\widetilde{\lambda}_{\text{u}}\Delta u_i^{k,j} + \widetilde{\lambda}_{\text{p}}\Delta P_i^{k,j}}$$

5. examine as configurações segundo a ordem, mudando as configurações escolhidas se uma outra configuração melhorar o benefício global e continuar obedecendo às restrições.

O pseudo-código da heurística de densidade gulosa é mostrado no Algoritmo 3. A complexidade de HDG é determinada da seguinte maneira. O passo corresponde à execução de ASG que como já foi visto tem tempo de execução $\Theta(|\Omega|k^{\max})$. Passo dois ao seis são executados em tempo $\Theta(n)$. O sétimo passo leva tempo $\Theta(|\Omega|\log|\Omega|)$ para ordenar as tarefas de acordo com $\Delta\widetilde{A}_i^{k,j}$. O passo nove executa em tempo $\Theta(\log n)$, com o emprego de dicionários baseados em estrutura de árvores balanceadas, como árvores vermelhas e pretas(CORMEN, 2001). Entre os passos dez a quatorze a execução ocorre em um tempo $O(\log n)$ pois termos que excluir e incluir na árvore, dentro de um laço que roda em um tempo $\Theta(|\Omega|\log n)$. Portanto, o tempo total de execução de HDG é de $\Theta(|\Omega|k^{\max} + |\Omega|\log|\Omega|)$ sendo influenciado por ASG.

A heurística produz uma solução factível X que não é inferior à solução inicial $X(\beta)$, tendo qualidade estimada pelo limite superior $f_{\text{L}}^{\text{best}}$, *i.e*, $f^\star - f(X) \leq f_{\text{L}}^{\text{best}} - f(X)$ onde $(f_{\text{L}}^{\text{best}} - f(X))$ é o erro máximo.

A heurística de densidade gulosa foi aplicada na instância exemplo e com isso algumas observações foram feitas. O ASG foi usado com os mesmos parâmetros já citados anteriormente, quando ele foi testado na instância exemplo.

Algoritmo 3: Heurística Gulosa Baseada em Densidade (HDG)

input: servers $\mathscr{S}$, $\{(A_i^{k,j}, u_i^{k,j}, P_i^{k,j}) : (i,k,j) \in \Omega\}$, β, conjunto factível inicial $X(\beta)$, valores iniciais para os multiplicadores Lagrangeanos $\lambda^{(0)}$, passo inicial $\mu^{(0)}$, taxa de decrescimento $\rho < 1$, tolerância η, e limite de interações $k^{\max}$

1 ***executa*** *o algoritmo subgradiente com os parâmetros* ($\lambda^{(0)}$, $\mu^{(0)}$,ρ,η,$k^{\max}$)*, para obter os valores de* $\widetilde{\lambda} = \lambda^{\text{best}}$ *que aproxima o valor de* $\lambda^\star$, $f_{\text{L}}^{\text{best}}$ *e* $\mathbf{x}^{\text{best}}$

2 **if** x^{best} *é factível para* P **then**

3 $\quad X := \{(i,k,j) : x_i^{k,j} = 1 \text{ in } \mathbf{x}^{\text{best}}\}$

4 **else**

5 $\quad X := X(\beta)$ solução factível inicial

6 $u(X) := \sum_{(i,k,j)\in X} u_i^{k,j}$, $P(X) := \sum_{(i,k,j)\in X} P_i^{k,j}$, $f(X) := \sum_{(i,k,j)\in X} A_i^{k,j}$

7 **obtém** $\widehat{\Omega}(X) := \langle (i,k,j)(1), (i,j,k)(2), \ldots, (i,k,j)(T) \rangle$ *ordenando* $\Omega(X)$ *em ordem não crescente de* $\Delta\widetilde{A}_i^{k,j}$*, onde* $T := |\Omega(X)|$

8 **for** $t = 1$ to T **do**

9 $\quad$ **encontrar** $(i,k,j) \in X$ tal que $i = i(t)$

10 $\quad$ **if** $(A_i^{k(t),j(t)} \geq A_i^{k,j}) \wedge (u(X) - u_i^{k,j} + u_i^{k(t),j(t)} \leq 1) \wedge (P(X) - P_i^{k,j} + P_i^{k(t),j(t)} \leq \beta P^\star)$ **then**

11 $\quad\quad X := (X - \{(i,k,j)\}) \cup \{(i,k(t),j(t))\}$

12 $\quad\quad u(X) := u(X) - u_i^{k,j} + u_i^{k(t),j(t)}$

13 $\quad\quad P(X) := P(X) - P_i^{k,j} + P_i^{k(t),j(t)}$

14 $\quad\quad f(X) := f(X) - A_i^{k,j} + A_i^{k(t),j(t)}$

15 **return(** X, $f(X)$, $u(X)$, $P(X)$ *e* $f_{\text{L}}^{\text{best}}$**)**

Para $\beta = 0{,}5$, ASG alcançou $\lambda^{\text{best}} = (0;0{,}4336)$ e encontrou uma solução ótima $X^\star = \{(1,3,1);(2,2,3);(3,1,1)\}$ com $f(X^\star) = 6$, $U(X^\star) = 0{,}635$ e $P(X^\star) = 4{,}6878$. Usando a solução factível $X(\beta) = \{(1,3,1);(2,3,1);(3,3,\ 1)\}$ com $f(X(\beta)) = 2{,}4621$, HDG conseguiu encontrar a solução ótima $X^\star$.

Para $\beta = 1$, ASG encontrou $\lambda^{\text{best}} = (0{,}3485;0{,}1709)$ e uma solu-

ção factível $X = \{(1,3,2);(2,2,3);(3,1,1)\}$ com $f(X) = 5,7724$, $U(X) = 0,6489$ e $P(X) = 4,2165$. Usando a solução factível X e λ^{best} encontrados por ASG, HDG encontrou novamente a solução $X^\star$ que é ótima com $f(X^\star) = 7$, $U(X^\star) = 0,3880$ e $P(X^\star) = 9,8683$.

3.6 SUMÁRIO

Este capítulo apresentou a notação utilizada ao longo deste trabalho. Depois apresentou-se um modelo de sistema que assume a existência de valores discretos, conhecidos para a utilização de cada um dos modos $\kappa(i)$ dos servidores S_i, com as j frequências suportadas pela *CPU*. Foram também propostos um algoritmo de programação dinâmica e uma heurística gulosa baseada em densidade. Ambos os algoritmos além de apresentados foram testados em uma instância exemplo.

4 AVALIAÇÃO

Neste capítulo serão mostrados resultados experimentais da avaliação dos algoritmos descritos no capítulo anterior. Inicialmente, uma análise numérica será realizada. Em seguida, a heurística proposta será avaliada por simulação. Para tanto, um sistema de monitoramento, que recebe dados de câmeras de vídeo, será usado como aplicação-alvo para se avaliar o comportamento da heurística proposta em situações mais realistas.

4.1 AVALIAÇÃO NUMÉRICA

Para execução da avaliação numérica foram usados conjuntos de tarefas gerados pelo algoritmo UUniFast, apresentado em (BINI; BUTTAZZO, 2005).

Quatro cenários foram criados para realização dos testes. Todos eles com dez instâncias diferentes sendo que o primeiro tem 10 tarefas, o segundo 20, o terceiro 100 e o último 200. A média dos resultados obtidos com a execução das instâncias de cada cenário foi usada para gerar os gráficos que serão mostrados a seguir. O objetivo em usar a média de 10 instâncias de cada cenário foi evitar que um comportamento específico de uma instância, de um cenário, fosse considerado algo comum.

Cada um desses cenários representa um sistema de tempo real, que contém $\kappa(i) = 3$ modos de operação e $|\mathscr{F}| = 4$ possíveis frequências. Em cada cenário, um *CBS* é dedicado à execução de cada tarefa. O conjunto de frequências suportadas pelo processador está restrito ao conjunto $\mathscr{F} = \{1,0; 0,75; 0,5; 0,25\}$ e a frequência máxima disponível é de $2,53\,GHz$. Os valores de $Q_i^{k,\mathrm{D}}$ e $Q_i^{k,\mathrm{I}}$ são obtidos pela execução do algoritmo UUniFast, de forma que o somatório da utilização de todas as tarefas de um mesmo modo é sempre igual considerando a frequência máxima da *CPU*. Entretanto, o modo de maior benefício de uma tarefa não é necessariamente igual ao de outra tarefa. Os parâmetros energéticos do sistema foram: $C_{ef} = 6,324110671936759 \times 10^{-9}$, $V = 1,25$, $P_i^{k,I} = 0,7538$. O valor estabelecido para $Q_i^{k,\mathrm{I}}$ representa a quantidade de energia para manter uma memória de $4\,GB$ em funcionamento, para todo $S_i \in S$ e $k \in K_i$.

Os experimentos foram realizados em um computador, *MacBookPro5,5* equipado com um processador $2,53\,Ghz$ *Intel Core 2 Duo*,

$4GB$ de memória RAM, sistema operacional Debian (com *kernel* Linux versão 2.6.32-5-amd64) e o compilador Gcc versão 4.4.5. O CPLEX versão 12.2.0, que é um *solver* de programação inteira mista, a qual foi utilizado para computar a melhor configuração de cada cenário. O CPLEX é um dos melhores *solvers* encontrados no mercado para resolver problemas de otimização linear inteira mista e consegue garantir a qualidade da solução. O problema, P, resolvido pelo CPLEX é dado pelas expressões (3.4a)-(3.4e).

Os testes relacionados à memória foram feitos com a ferramenta *valgrind* (VALGRIND, 2011), junto ao *massif* que faz parte dele. Com esse aplicativo, é possível obter informações sobre alocação de memória por um aplicativo em tempo de execução.

4.1.1 Avaliação Numérica do Algoritmo de Programação Dinâmica

Um conjunto com dez tarefas foi utilizado para a análise do algoritmo de programação dinâmica. Devido ao elevado tempo de execução e consumo de memória não foram utilizados conjuntos maiores. O cenário foi testado para três possíveis combinações dos multiplicadores, uma com $\Lambda = 100$ e $\Gamma = 100$, a outra com $\Lambda = 1000$ e $\Gamma = 100$ e por último $\Lambda = 100$ e $\Gamma = 1000$.

Os valores da função objetivo alcançados de acordo com a variação do parâmetro β entre $0,1$ e 1, podem ser vistos na Figura 8. Como é esperado, à medida que a energia disponibilizada para o sistema aumenta é possível obter um benefício maior.

Pode ser observado no gráfico da Figura 8 que a combinação de valores ($\Lambda = 100$, $\Gamma = 100$) e ($\Lambda = 1000$, $\Gamma = 100$) apresentam resultados iguais. Por outro lado, os multiplicadores ($\Lambda = 100$, $\Gamma = 1000$) apresentam resultados melhores, mas eles ainda são inferiores aos obtidos pelo CPLEX. Com esse gráfico é possível perceber que dependendo dos valores dos parâmetros Λ e Γ a qualidade da resposta não irá melhorar.

O gráfico da Figura 9 evidencia a diferença entre o benefício obtido pelo CPLEX e aquele alcançado pelo algoritmo de programação dinâmica proposto. As linhas que representam o erro de ($\Lambda = 100$, $\Gamma = 100$) e ($\Lambda = 1000$, $\Gamma = 100$) estão se sobrepondo, pois o benefício alcançado por ambos foi o mesmo.

Outro ponto de análise da solução por programação dinâmica está ilustrado na Figura 10, a diferença entre tempos de solução. Neste caso pode ser observado que o tempo para encontrar a solução com os

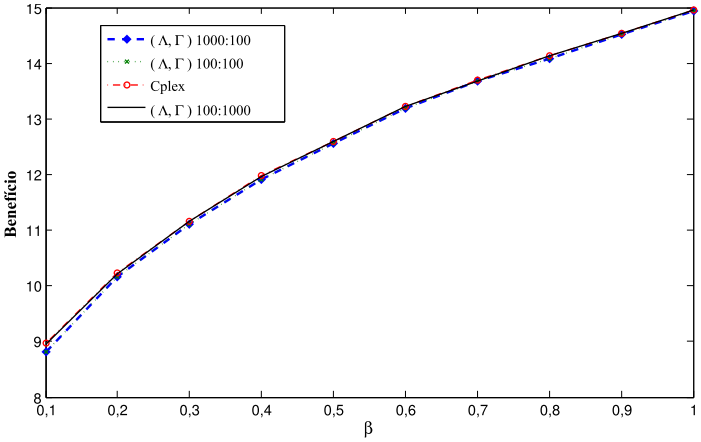

Figura 8: Valor do benefício obtido pelo sistema de acordo com a energia disponível.

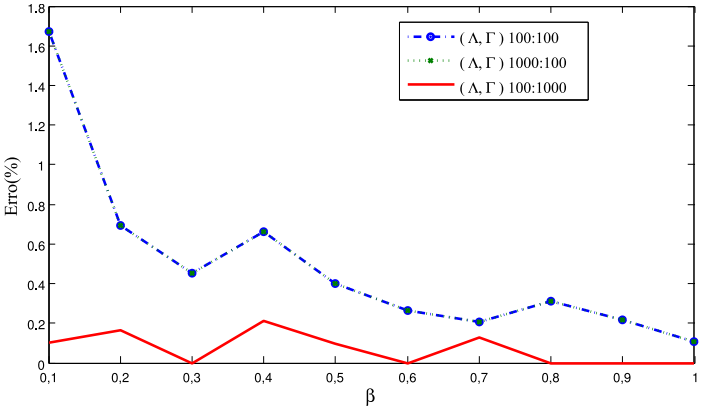

Figura 9: Erro do benefício obtido pela programação dinâmica de acordo com as respostas do CPLEX, em função da variação de β.

multiplicadores $\Lambda = 100$ e $\Gamma = 100$ é bem menor que aqueles obtidos quando se usa maiores valores para os multiplicadores, para os quais tempos de execução similares são observados.

A quantidade de memória consumida pelo algoritmo foi o último quesito da análise. O algoritmo de programação dinâmica com os parâmetros ($\Lambda = 100$, $\Gamma = 100$) consumiu entre $17MB$ e $27MB$, de

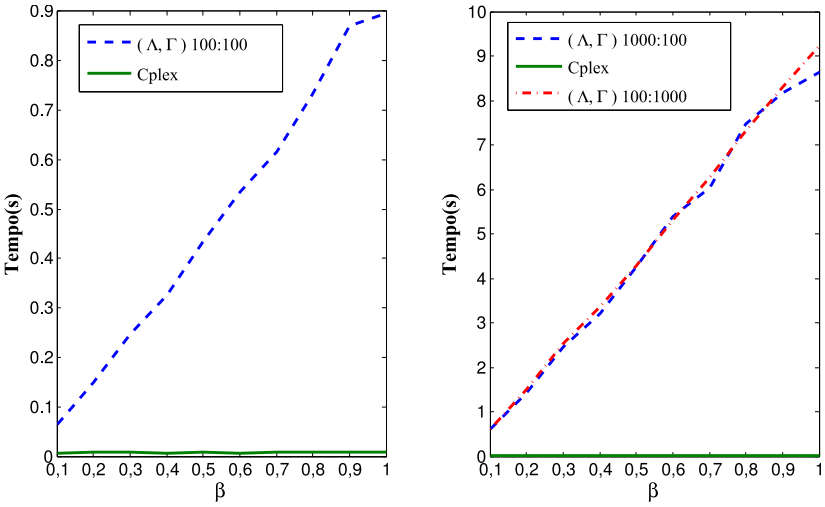

Figura 10: Comparação entre o tempo de execução do algoritmo de programação dinâmica e o CPLEX.

acordo com a variação da energia disponibilizada para o sistema. Já com ($\Lambda = 1000$, $\Gamma = 100$) usou de $27MB$ a $126MB$ e, por último, com os multiplicadores ($\Lambda = 100$, $\Gamma = 1000$) que alocaram entre $27,4MB$ e $127MB$.

De acordo com os resultados obtidos pela execução do algoritmo de programação dinâmica, em certos casos pode ser bastante lento e consumir bastante memória. Resultado que o torna ruim para problemas grandes ou com multiplicadores elevados que necessitem de uma resposta rápida. Ainda para os sistemas embarcados com capacidade de memória limitada o algoritmo de PD não seria a melhor escolha, pois ele tende a ter um elevado consumo de memória mesmo com poucas tarefas.

4.1.2 Avaliação Numérica da Heurística

Para realizar a avaliação numérica da heurística, foram usados conjuntos com vinte, cem e duzentas tarefas. Os parâmetros de entrada do algoritmo subgradiente foram devidamente escolhidos, com o propósito de gerar uma resposta que minimizasse o tempo de uso de *CPU* e erro.

A Figura 11 mostra os valores do benefício alcançados com o uso do CPLEX e a heurística para todos cenários, de acordo com a variação do parâmetro de energia β. Os resultados do gráfico mostram que a heurística produz resultados próximos do ótimo em um tempo satisfatório. A proximidade da solução encontrada pela heurística em relação à obtida pelo CPLEX é evidenciada na Figura 12.

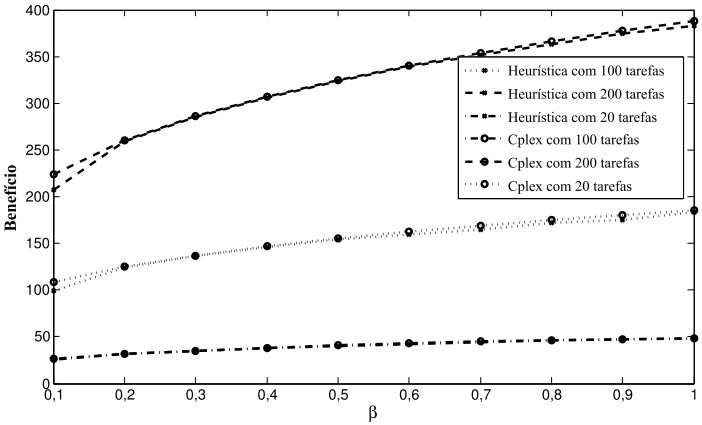

Figura 11: Valores de benefício obtidos com CPLEX e heurística, com variação da energia.

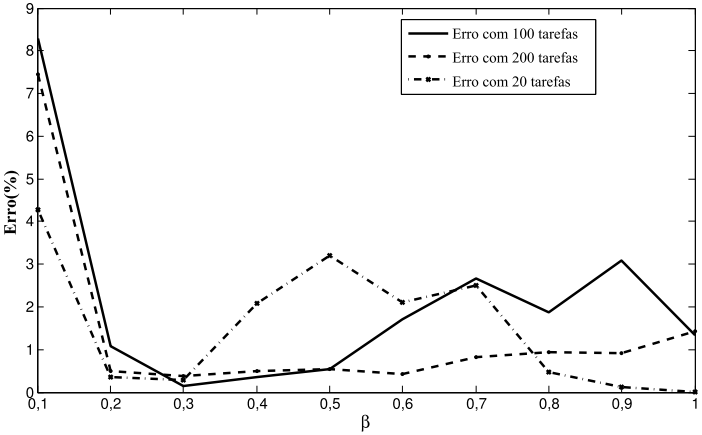

Figura 12: Diferença entre os resultados alcançados pelo CPLEX e a heurística.

Ainda pelas Figuras 11 e 12, é possível perceber que a qualidade dos resultados da heurística em relação a do CPLEX varia em função do valor de β. A essa diferença de resposta entre o CPLEX e a solução é considerada como erro, pois o resultado do CPLEX é considerado ótimo para os testes realizados. Como pode ser observado no gráfico da Figura 12, para alguns pontos chega-se a um erro nulo. O maior erro observado está em torno de 8,43%. É importante frisar que a qualidade da solução da heurística depende dos valores escolhidos para os multiplicadores Lagrangeanos, que são usados como multiplicadores de *Surrogate*. Para isso, o algoritmo subgradiente deve ter seus parâmetros iniciais devidamente ajustados, para apresentar um bom desempenho e resultado de acordo com as necessidades do problema em questão.

Na Figura 11, o benefício alcançado pelo sistema varia de acordo a com disponibilidade de energia. Isto é esperado pois quando a energia disponibilizada é reduzida, mais servidores são forçados a executar em configurações que consomem menos energia, o que implica a escolha de frequências de processador mais baixas.

Uma comparação importante considerando a otimização, que deve ser considerado na análise, é o tempo de resposta. A Figura 13 mostra o tempo utilizado pelo CPLEX para resolver o problema P, enquanto na Figura 14 o tempo do algoritmo proposto é apresentado.

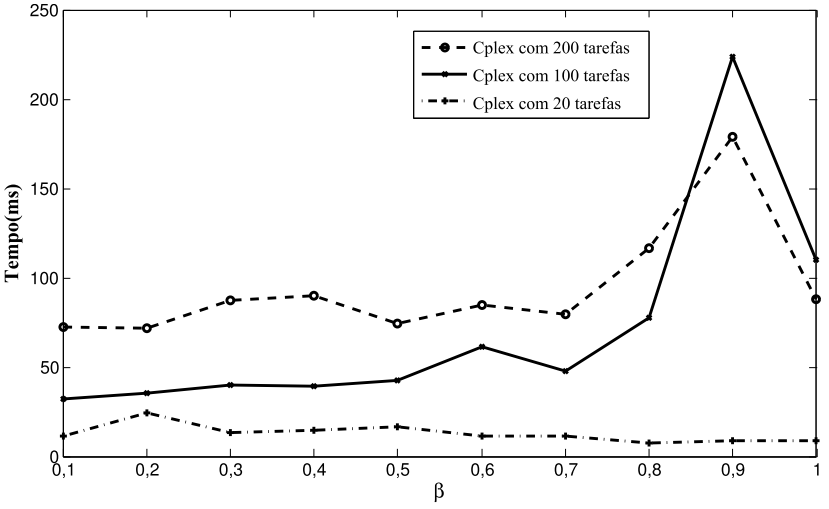

Figura 13: Tempo de execução em CPU pelo CPLEX para resolver o problema de acordo com a variação de β.

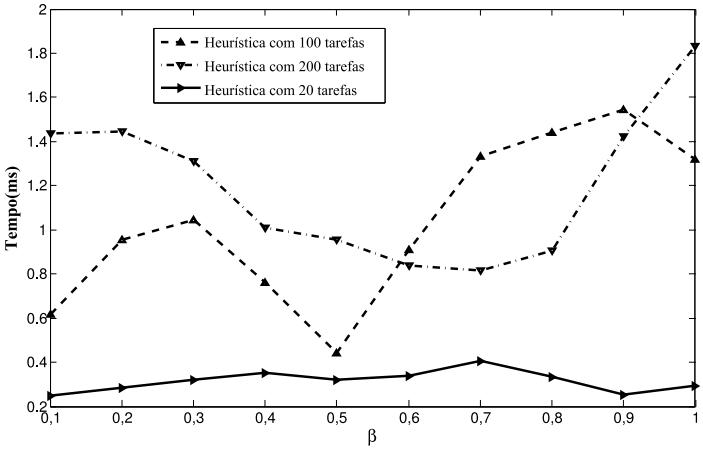

Figura 14: Tempo de uso da CPU pela heurística para resolver as instâncias de acordo com a variação de β.

O tempo de *CPU* usado pelo CPLEX não cresce necessariamente com o tamanho do problema, o que é um comportamento normal em ferramentas de problemas de programação inteira. Por exemplo, quando $\beta = 0,9$ na Figura 13, o CPLEX gasta mais tempo para resolver a instância de cem tarefas que a de duzentas. A heurística também tem um comportamento similar, como mostrado na Figura 14. Quando $\beta = 0,7$, é gasto mais tempo para encontrar uma solução para um conjunto com cem do que com duzentas tarefas. Uma causa deste tipo de comportamento pode ter origem nos parâmetros passados para o algoritmo do subgradiente.

O CPLEX é um *solver* de programação inteira mista que utiliza várias técnicas avançadas para conseguir os resultados e certificar a qualidade. Isto justifica a diferença de tempo entre o CPLEX e a heurística proposta, que em alguns pontos chega a ser quase 100 vezes mais rápida. O código do CPLEX compilado é de aproximadamente $14MB$ e de acordo com *valgrind* aloca durante a execução cerca de $2,9MB$ para resolver uma instância com duzentas tarefas. Para efeito de ilustração o algoritmo proposto compilado tem $40KB$ e aloca dinamicamente $195KB$.

As Figuras 15, 16 e 17 mostram os modos selecionados de acordo com a variação de β. A diferença entre a seleção de modos feita pelo CPLEX não é muito diferente daquela encontrada pela heurística. Em alguns casos ambos encontram a mesma seleção. Estes casos não foram

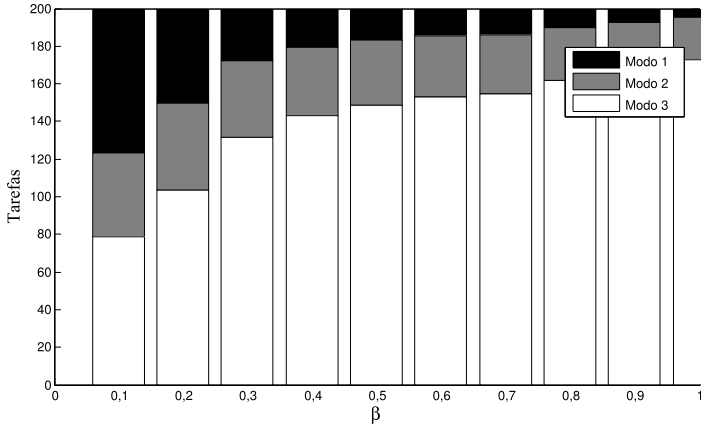

Figura 15: Comportamento da heurística em relação à escolha de modos em função da disponibilidade de energia, para um cenário de 200 tarefas.

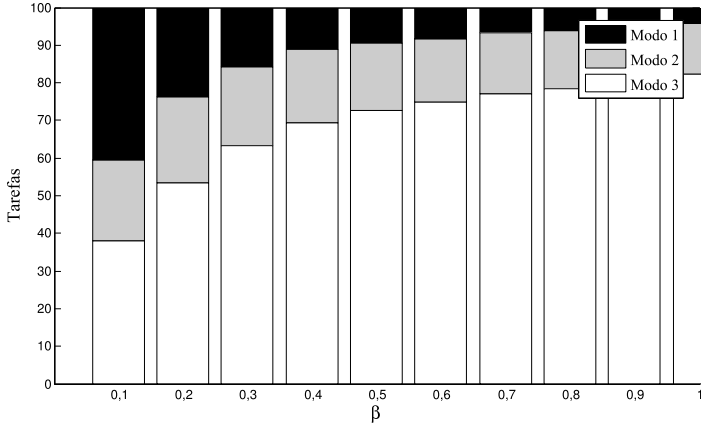

Figura 16: Comportamento da heurística em relação à escolha de modos em função da disponibilidade de energia, para um cenário de 100 tarefas.

mostrados na figura. Como pode ser notado pelos gráficos, o aumento da disponibilidade de energia no sistema implica que mais tarefas são executadas no terceiro modo. Este comportamento é esperado e ocorre porque o terceiro modo fornece maior benefício, porém requer maior consumo de energia quando comparado aos demais.

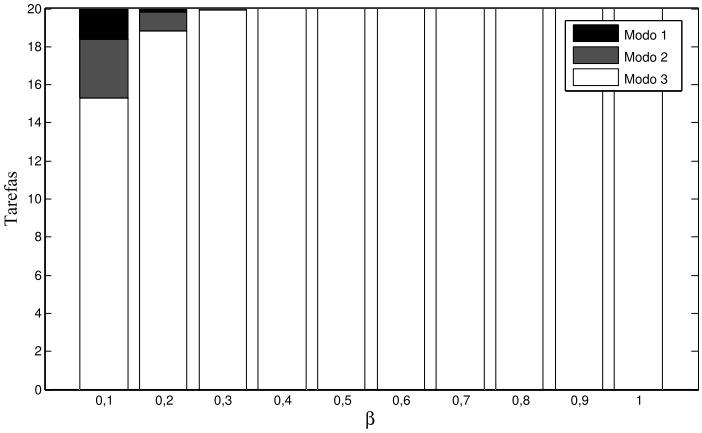

Figura 17: Comportamento da heurística em relação à escolha de modos em função da disponibilidade de energia, para um cenário de 20 tarefas.

Nos cenários escolhidos para análise, as tarefas têm seu consumo de energia muito influenciado pela variação da frequência do processador. Fato que pode ser confirmado nas Figuras 18, 19 e 20.

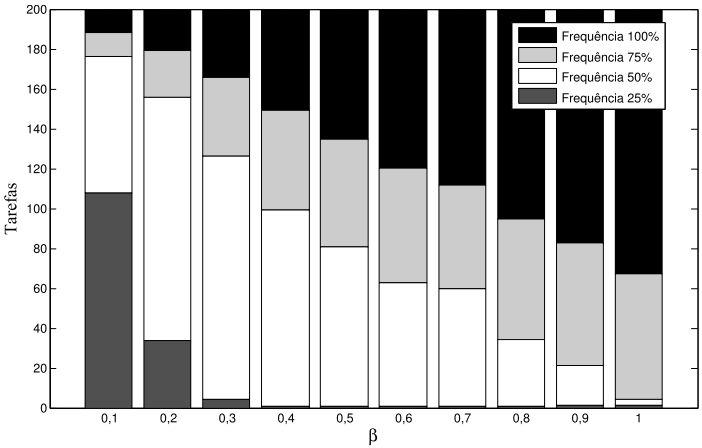

Figura 18: Comportamento da heurística em relação à escolha de frequências em função da disponibilidade de energia, para um cenário de 200 tarefas.

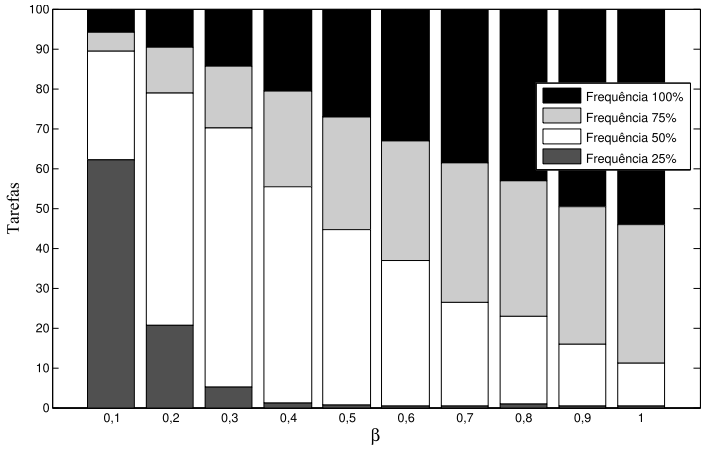

Figura 19: Comportamento da heurística em relação à escolha de frequências em função do parâmetro β, para um cenário de 100 tarefas.

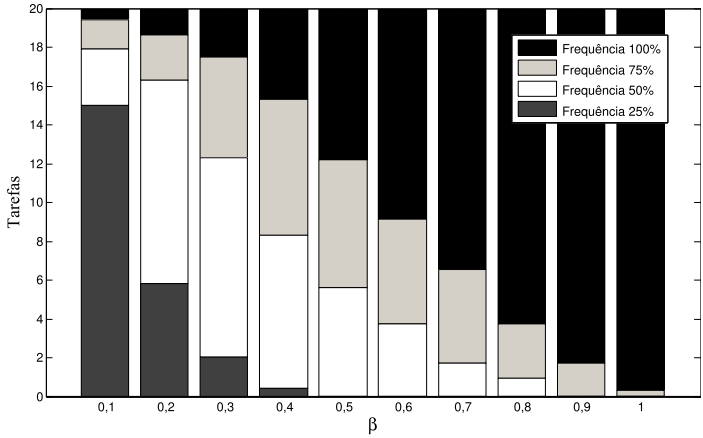

Figura 20: Comportamento da heurística em relação à escolha de frequências em função da disponibilidade de energia, para um cenário de 20 tarefas.

4.2 SIMULAÇÃO

Visando avaliar o desempenho da heurística proposta associada a uma aplicação, realizou-se uma simulação de forma a capturar o com-

portamento de um sistema de monitoramento. Tal sistema hipotético usa câmeras para monitorar o ambiente. Estas transmitem dados de vídeo para um computador central, que executa o servidor de vídeo. O codificador de vídeo usado foi o *mencoder* (MPLAYER, 2011), que foi alterado para coletar dados de tempo execução e de codificação de quadros de vídeo. O simulador usado para o teste foi desenvolvido em (OLIVEIRA, 2009).

O servidor de vídeo conectado a diferentes câmeras pode receber vídeos em diferentes formatos. Apesar de receber dados em vários formatos para realizar a retransmissão e armazenamento, precisa-se que os vídeos estejam codificados em um padrão comum. Esse tipo de servidor é responsável por sistemas que podem ser usados para prover segurança em ambientes controlados. Em caso de falta de energia, assume-se que o sistema é mantido por baterias. Um exemplo de uso é em uma biblioteca. Quando ocorre a falta de energia pessoas podem tentar sair com livros sem registrar o empréstimo, e as câmeras de segurança podem ser usadas para identificar quem tentou pegar livros sem registrar.

Foram definidos três modos de execução para o *mencoder*. Estes modos estão relacionados ao número de quadros por segundo a serem processados, 20*fps*, 25*fps* e 30*fps*. O processamento destes quadros, para dados de cada câmera, é feito por tarefas associadas a um servidor *CBS*. O total de energia consumida por execução da instância em cada um dos modos é de $0,0875$J, $0,1463$J e $0,1828$J, respectivamente. O consumo de energia destes modos de operação é diretamente proporcional ao benefício agregado e à frequência de processamento de quadros de vídeo requerida. Por exemplo, o benefício associado ao processamento de vídeo a 30a*fps* é maior e gasta mais energia que para 25*fps*, que por sua vez induz maior benefício e requer mais energia que ao processar um vídeo a uma taxa de 20*fps*.

Gerou-se dados de vídeo para cada modo de operação a partir da coleta de vídeo de uma câmera. Estes dados foram então replicados quatro vezes de forma a simular dados de quatro câmeras que transmitem as mesmas imagens. Desta forma, o servidor de vídeo é responsável por executar quatro tarefas de codificação. As Figuras 21 e 22 mostram respectivamente o tempo necessário para a computação de cada quadrado e a utilização do processador para codificar cada quadro. Como pode ser observado, há bastante variação. O menor tempo de execução medido foi igual a $1,4$ms, o maior $39,8$ms, e a média $7,2$ms, e um período de $33,33$ms, considerando uma qualidade de 30*fps*.

Para processar os dados de vídeo vindo de cada câmera, quatro

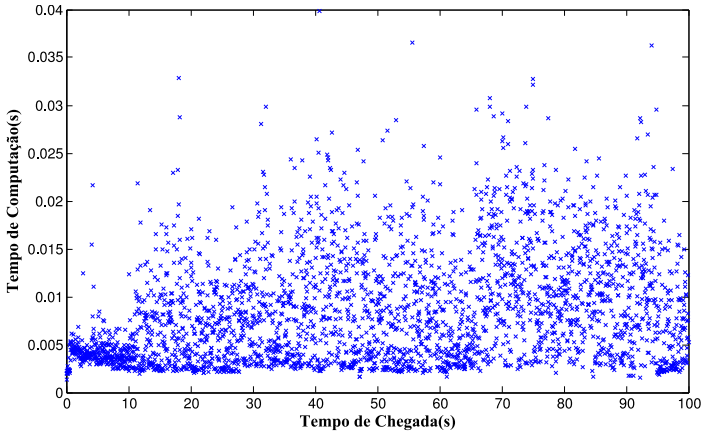

Figura 21: Gráfico com tempo de codificação do vídeo.

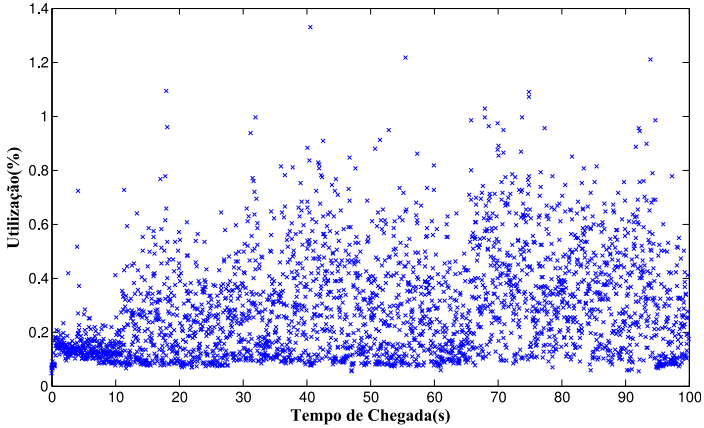

Figura 22: Utilização da *CPU* para codificação do vídeo.

servidores *CBS* foram usados, considerando os parâmetros $Q_i = 8,2$ms e $T_i = 33,3$ms, para $i = 1,2,3,4$. Isto significa que cada servidor precisa de $24,62\%$ de banda de processamento. É portanto atribuído ao reconfigurador menos que 1% de banda de processamento. Para a definição dos valores de Q_i usado pelo reconfigurador para cada modo de execução, foi simulado o sistema sendo executado no modo desejado e a média dos tempos registrados, usada como valor de Q_i. Os períodos foram escolhidos de acordo com a qualidade, ficando o primeiro modo

com período de 33,3ms, o segundo modo com 40,0ms e o terceiro com 50,0ms.

Três cenários foram pensados para simulação. O primeiro consiste em executar as tarefas sem acionar o reconfigurador, selecionando os modos de maior benefício para cada tarefa. O segundo, quando o sistema detecta que está ocorrendo atraso de forma constante aciona o reconfigurador, sem restrição de energia. E por último é pensando em um momento em que ocorre uma falha na rede elétrica, fazendo com que o reconfigurador seja acionado para economizar energia.

Os modos de melhor benefício foram escolhidos para serem usados no primeiro cenário. Eles têm a utilização variando entre 4% e 119%, com utilização média igual a 21,8%. Fato que leva o sistema a operar em sobrecarga constantemente, fato que o torna este cenário interessante para ilustrar o pior caso de consumo de energia. A Figura 23 mostra, para este cenário, os excessivos atrasos causados pela sobrecarga, o que é esperado devido à sobrecarga. O maior atraso observado para processar um quadro foi de 13,8927 segundos.

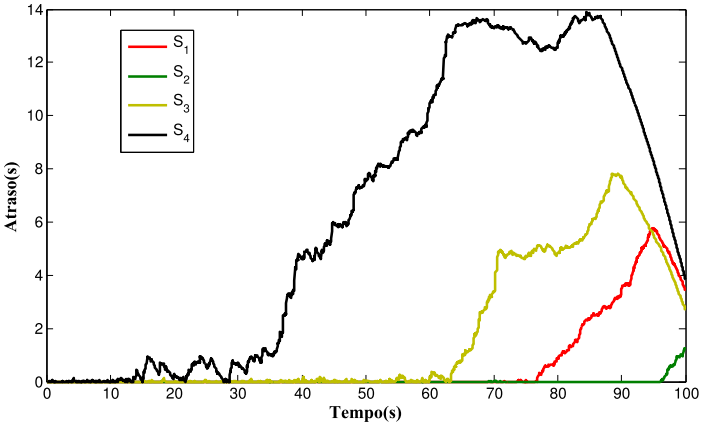

Figura 23: Resultado da simulação sem aplicação do reconfigurador.

No segundo cenário o reconfigurador é acionado no instante 27 segundos, quando o sistema identifica um nível de atraso além do desejável. A execução do reconfigurador neste instante levou a ocorrência de um atraso máximo igual a 7,1568 segundos, e a diminuição de 55,11% no pior caso, quando comparado com o primeiro cenário, da primeira simulação executada. Esta informação é comprovada na Figura 24.

Para o terceiro cenário, devido à autonomia prevista da bateria

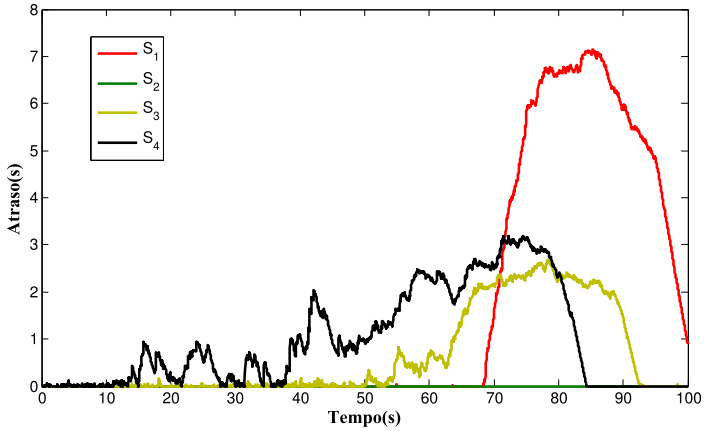

Figura 24: Resultado da simulação com aplicação do reconfigurador, com $\beta = 1$.

após falha na rede elétrica, acionou-se o reconfigurador com $\beta = 0,7$. O resultado da reconfiguração pode ser observado na Figura 25. A diminuição drástica no valor do atraso ocorre devido à diminuição da energia disponibilizada, que força o sistema a escolher modos de menor qualidade, com benefício menor, ficando o maior atraso $0,9445$S.

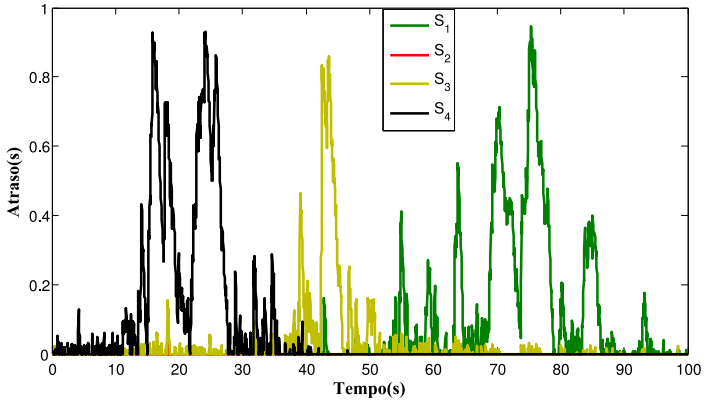

Figura 25: Resultado da simulação com aplicação do reconfigurador, com $\beta = 0,7$.

O comportamento da aplicação foi avaliado considerando o atraso

de término das tarefas, o benefício agregado a suas execuções e a economia de energia obtida. Com relação às duas primeiras métricas, foi observada a taxa de atraso (T.A.), dada pela razão do somatório do tempo de atraso pela quantidade de quadros em atrasados. O atraso médio (A.M.), calculado pela razão entre a quantidade de quadros executados com atraso pelo total de quadrados, também foi observada. O benefício total (A), dado pela soma do benefício de cada servidor (a_i) segundo o modo e frequência escolhidos, foi medido durante a simulação. A utilização u_i prevista para cada configuração foi calculada com base na média de um histórico de execução para uma tarefa operando em um modo e frequência. A Tabela 8 resume estes dados e evidencia a diferença dos resultados em função dos modos selecionados por cada cenário.

Tabela 8: Resultados da Simulação.

		CBS 1	CBS 2	CBS 3	CBS 4
Sem Reconf,	a_i	2,99	2,99	2,99	2,99
	u_i	24,65%	23,81%	24,36%	25,04%
	T.A.	0,244585	0,0563146	0,53149	0,916694
	A.M.	0,6985	0,0246	1,7562	6,4045
	A	11,96			
Com Reconf, e $\beta = 1$	a_i	2,99	2,99	2,99	0,99
	u_i	25,37%	24,51%	25,07%	22,62%
	T.A.	0,322452	0,00632911	0,513991	0,748201
	A.M.	1,5977	0,0000983344	0,6365	0,9213
	A	9,96			
Com Reconf, e $\beta = 0,7$	a_i	2,99	2,99	2,47	0,72
	u_i	24,65%	23,82%	24,36%	25,05%
	T.A.	0,324124	0,005996	0,201865	0,315213
	A.M.	0,0711	0,000068	0,0275	0,0896
	A	9,17			

As Tabelas 9, 10 e 11 fornecem os dados relativos à energia consumida. A energia consumida por cada servidor foi medida durante a simulação a partir do momento em que uma nova configuração do sistema foi ativada. A Tabela 9 fornece os resultados relativos à execução do sistema sem executar o reconfigurador. A Tabelas 10 exibe aqueles obtidos aplicando a configuração encontrada pelo reconfigurador com $\beta = 1$. A Tabela 11 apresenta os resultados obtidos com o reconfigurador executando com $\beta = 0,7$.

Comparando as três tabelas e considerando que o consumo máximo é igual a $1875J$, que seria o consumo caso a *CPU* executasse sempre 100%, é possível observar que quando $\beta = 1$ a energia consumida foi de 85,39% da disponibilizada e quando o reconfigurador operando com

$\beta = 0,7$ a economia de energia ficou em torno de $32,52\%$, economizando um pouco a acima dos 30% desejados.

Tabela 9: Resultados da Simulação sem Reconfiguração.

	CBS 1	CBS 2	CBS 3	CBS 4	Total
Tempo(s)	20,9685	19,2458	18,7313	19,2576	
$P_i^{k,I}$(W)	0,4380	0,4380	0,4380	0,4380	
$P_i^{k,D}$(W)	25	25	25	25	
Potência(W)	25,4380	25,4380	25,4380	25,4380	
Energia(J)	533,3967	489,5747	476,4868	489,8748	1914,3

Tabela 10: Resultados da Simulação com $\beta = 1$.

	S_1	S_2	S_3	S_4	Total
Tempo(s)	20,9685	19,2458	18,7313	15,055	
$P_i^{k,I}$(W)	0,4380	0,4380	0,4380	0,4380	
$P_i^{k,D}$(W)	25	25	25	6,3232	
Potência(W)	25,4380	25,4380	25,4380	6,7612	
Energia(J)	533,3967	489,5747	476.4868	101,7899	1601,2481

Tabela 11: Resultados da Simulação com $\beta = 0,7$.

	CBS 1	CBS 2	CBS 3	CBS 4	Total
Tempo(s)	20,9685	19,2458	18,3132	7,7122	
$P_i^{k,I}$(W)	0,4380	0,4380	0,4380	0,4380	
$P_i^{k,D}$(W)	25	25	9,9338	6,3232	
Potência(W)	25,4380	25,4380	10,3718	6,7612	
Energia(J)	533,3967	489,5747	189,9408	48,7658	1265,1

4.3 SUMÁRIO

Análises numéricas foram realizadas para avaliar os algoritmos desenvolvidos, neste capítulo. Foi percebido que o algoritmo de programação dinâmica pode ser aplicado em problemas de tempo real onde as variáveis envolvidas sejam pequenas ou não necessite de uma resposta imediata. Por outro lado, com a heurística obteve-se resultados importantes, pois o benefício obtido foi próximo do ótimo e o tempo de execução do algoritmo foi relativamente baixo, o que torna a heurística proposta adequada para ser aplicada em tempo de execução.

5 CONCLUSÃO

O número de dispositivos móveis que necessitam interagir com o ambiente de forma inteligente, tendo uma alta variabilidade de carga computacional e operando com bateria, cresce a cada dia, assim como a complexidade dos sistemas de tempo real que auxiliam seu funcionamento. Visando dar suporte a esses sistemas, novos componentes eletrônicos estão sendo desenvolvidos de forma a agregar novas funcionalidades.

O tempo de execução de um aplicativo fica cada vez mais difícil de se estimar. Muitas vezes essa dificuldade em medir o tempo acaba por gerar estimativas muito pessimistas. Logo, torna-se importante a construção de mecanismos que auxiliem o escalonador a lidar com a variabilidade do tempo de execução das tarefas.

Modelos de servidores de tarefas aperiódicas têm alcançado bom desempenho ao lidar com tarefas de alta variabilidade na utilização do processador. Entretanto, a definição estática de valores como Q_i e T_i pode gerar desperdícios quando o comportamento da aplicação com o tempo.

Em um sistema de câmeras de segurança, configuradas para agir de forma diferenciada dependendo da presença ou não de pessoas, podem ocorrer variações significantes na utilização média ou falhas na rede de energia elétrica que poderão acarretar o desligamento se o sistema operar com alimentação de baterias por um tempo prolongado. Essas questões não são tratadas de forma adequada por modelos de servidores de tarefas aperiódicas, necessitando de um mecanismo de reconfiguração que ajuste os parâmetros no escalonador de acordo com as necessidades.

Nesta dissertação foi apresentada a modelagem de um sistema de tempo real discreto, envolvendo tanto as questões de utilização da *CPU*, quanto de economia de energia. A função objetivo foi definida de forma a priorizar a utilização das tarefas em maior frequência e usando modos com maior benefício. E a restrição de energia foi construída de forma a penalizar os modos que consomem mais energia por unidade de tempo.

No modelo apresentado, o sistema possui um conjunto de servidores onde cada tarefa tem discretizado possíveis modos e cada um dos modos, por sua vez, pode ser executado usando um conjunto de frequências finito, que influenciam no consumo de energia. Assim, o problema de reconfigurações dinâmica consiste em selecionar um modo

e frequência para executar cada instância das tarefas de acordo com a energia disponibilizada, garantindo economia.

O modelo discreto proposto consiste em um problema de otimização NP-Difícil, que para sua resolução dois algoritmos foram propostos, um de programação dinâmica e outro heurístico. Com a implementação do primeiro algoritmo e com a realização de testes, observou-se que apesar de conseguir alcançar a solução ótima, ele tende a demorar cada vez mais a medida que o número de servidores ou parâmetros aumentam. Sua aplicação em sistemas embarcados com pouca memória talvez não seja o ideal devido aos recursos limitados de *hardware*.

O algoritmo heurístico, ao contrário do primeiro, demostrou um bom desempenho quando aplicado a sistemas com duzentas tarefas, chegando a encontrar resultados próximos do ótimo em um tempo menor que o CPLEX. De acordo com a análise computacional realizada, o algoritmo heurístico necessita de pouco recurso de memória e processador para executar, o que o torna adequado para sistemas embarcados com poucos recursos.

Na simulação realizada foi verificado o comportamento da heurística em um cenário mais realista. Os resultados dela indicaram que as restrições impostas foram cumpridas. Mesmo havendo sobrecarga no sistema o consumo de energia não ultrapassou o permitido, o que a torna uma ferramenta útil para escalonar sistemas que tem tempo e objetivos bem definidos, mas variáveis, como: robô móvel, sensores, satélites e sistemas de segurança baseados em imagens de vídeo.

No decorrer deste trabalho algumas questões foram observadas, mas não tratadas devido ao escopo proposto inicialmente. Essas questões indicam temas para trabalhos futuros. São elas:

- Desenvolvimento de um modelo contínuo que permita o emprego de técnicas de otimização contínua, o que poderia dar soluções ainda mais rápidas e mais próximas da ótima.
- Extensão do modelo monoprocessado para multiprocessado, o que implicaria o emprego de novas abordagens de escalonamento de tarefas e energia para o sistema.
- Concepção de modelo energético para outros dispositivos como, por exemplo, a memória e outros dispositivos que poderiam ter seu consumo de energia ajustado através da frequência e voltagem.
- Refinamento do modelo para tratar questões como memória *cache* e *pipeline*, que podem reduzir o pessimismo do modelo.

REFERÊNCIAS BIBLIOGRÁFICAS

ABENI, L.; BUTTAZZO, G. Resource reservation in dynamic real-time systems. *Real-Time Systems*, Springer, v. 27, n. 2, p. 123–167, 2004.

AYDIN, H.; DEVADAS, V.; ZHU, D. System-level energy management for periodic real-time tasks. In: *Proceedings of the 27th IEEE Real-Time Systems Symposium*. [S.l.: s.n.], 2006. p. 313 –322.

BINI, E.; BUTTAZZO, G.; LIPARI, G. Minimizing cpu energy in real-time systems with discrete speed management. *ACM Trans. Embed. Comput. Syst.*, ACM, New York, NY, USA, v. 8, p. 31:1–31:23, July 2009.

BINI, E.; BUTTAZZO, G. C. Measuring the performance of schedulability tests. *Real-Time Systems*, Springer Netherlands, v. 30, p. 129–154, 2005.

BOYD, S.; VANDENBERGHE, L. *Convex Optimization*. [S.l.]: Cambridge University Press, 2004.

BURD, T.; BRODERSEN, R. Energy efficient cmos microprocessor design. In: . [S.l.]: IEEE Computer Society, 1995. v. 0, p. 288.

BUTTAZZO, G. *Hard real-time computing systems*. [S.l.]: Springer, 1997.

BUTTAZZO, G. Research trends in real-time computing for embedded systems. *SIGBED Rev.*, ACM, v. 3, n. 3, p. 1–10, 2006.

BUTTAZZO, G. et al. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, IEEE Computer Society, Los Alamitos, CA, USA, v. 51, p. 289–302, 2002.

CACCAMO, M.; BUTTAZZO, G.; SHA, L. Capacity sharing for overrun control. In: *IEEE Real-Time Systems Symposium, 2000. Proceedings.* [S.l.: s.n.], 2000. p. 295 –304.

CACCAMO, M.; BUTTAZZO, G.; THOMAS, D. Efficient reclaiming in reservation-based real-time systems with variable execution times. *Computers, IEEE Transactions on*, v. 54, n. 2, p. 198 – 213, 2005.

CORMEN, T. H. *Introduction to algorithms*. [S.l.]: The MIT press, 2001.

ELNOZAHY, E.; KISTLER, M.; RAJAMONY, R. Energy-efficient server clusters. In: FALSAFI, B.; VIJAYKUMAR, T. (Ed.). *Power-Aware Computer Systems*. [S.l.]: Springer Berlin / Heidelberg, 2003, (Lecture Notes in Computer Science, v. 2325). p. 179–197.

FARINES, J.-M.; FRAGA, J. da S.; OLIVEIRA, R. S. de. *Sistemas de Tempo Real*. [S.l.: s.n.], 2000. (12ªEscola de Computação).

FISHER, M. L. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, v. 50, n. 12, p. 1861–1871, 2004.

GAREY, M.; JOHNSON, D. *Computers and intractability. A guide to the theory of NP-completeness.* [S.l.]: W.H. Freeman and Company, 1979.

GEOFFRION, A. M. Lagrangian relaxation an its uses in integer programming. *Mathematical Programming Study*, v. 2, p. 82–114, 1974.

GHAZALIE, T.; BAKER, T. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, Springer, v. 9, n. 1, p. 31–67, 1995.

GLOVER, F. Surrogate constraints. *Operations Research*, v. 16, n. 4, p. 741–749, July-August 1968.

LIMA, G.; CAMPONOGARA, E.; SOKOLONSKI, A. C. Dynamic reconfiguration for adaptive multiversion real-time systems. In: *Proc. of the 20th IEEE Euromicro Conf. on Real-Time Systems*. [S.l.: s.n.], 2008. p. 115–124.

LIN, C.; BRANDT, S. A. Improving soft real-time performance through better slack reclaiming. In: *In proc. of real-time systems symposium*. [S.l.: s.n.], 2005. p. 314.

LIPARI, G.; BARUAH, S. Greedy reclamation of unused bandwidth in constant-bandwidth servers. In: *Euromicro RTS 2000. 12th Euromicro Conference on*. [S.l.: s.n.], 2000. p. 193 –200.

LIU, C.; LAYLAND, J. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, v. 20, n. 1, p. 61, 1973.

LIU, J. *Real-time systems*. [S.l.]: Prentice Hall, 2000.

MARTELLO, S.; TOTH, P. *Knapsack Problems: Algorithms and Computer Implementations*. [S.l.]: John Wiley & Sons, 1990.

MARTELLO, S.; TOTH, P. An exact algorithm for the two-constraint 0–1 knapsack problem. *Operations Research*, v. 51, n. 5, p. 826–835, 2003.

MPLAYER. *Mplayer*. Fev 2011. <http://www.mplayerhq.hu>.

OLIVEIRA, A. B. de. *Uma Infraestrutura para Reconfiguração Dinâmica de Escalonadores Tempo Real: Modelos, Algoritmos e Aplicações*. Dissertação (Mestrado) — Programa de pós-graduação em engenharia de automação e sistema centro tecnológico universidade federal de santa catarina, 2009.

OLIVEIRA, A. B. de; CAMPONOGARA, E.; LIMA, G. Dynamic reconfiguration in reservation-based scheduling: An optimization approach. In: *Proc. of the 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*. [S.l.: s.n.], 2009. p. 173–182.

PILLAI, P.; SHIN, K. G. Real-time dynamic voltage scaling for low-power embedded operating systems. *SIGOPS Oper. Syst. Rev.*, ACM, New York, NY, USA, v. 35, p. 89–102, October 2001.

REAL, J.; CRESPO, A. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, Springer Netherlands, v. 26, p. 161–197, 2004.

RUSU, C. A.; MELHEM, R.; MOSSÉ, D. Maximizing the system value while satisfying time and energy constraints. *IBM Journal of Research and Development*, v. 47, n. 5-6, p. 689–702, Sept 2003.

SPRUNT, B.; SHA, L.; LEHOCZKY, J. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, Springer Netherlands, v. 1, p. 27–60, 1989.

SPURI, M.; BUTTAZZO, G. Efficient aperiodic service under earliest deadline scheduling. In: *Real-Time Systems Symposium, 1994., Proceedings.* [S.l.: s.n.], 1994. p. 2 –11.

SPURI, M.; BUTTAZZO, G. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, Springer, v. 10, p. 179–210, 1996.

VALGRIND. *valgrind*. Fev 2011. <http://www.valgrind.org/>.

WOLSEY, L. *Integer programming*. [S.l.]: Wiley New York, 1998.

ZHAO, B.; AYDIN, H. Minimizing expected energy consumption through optimal integration of DVS and DPM. In: *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*. [S.l.: s.n.], 2009. p. 449 –456.

ZHU, D.; MELHEM, R.; MOSSÉ, D. The effects of energy management on reliability in real-time embedded systems. *Computer-Aided Design, International Conference on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 35–40, 2004.

ZHU, Y.; MUELLER, F. Feedback edf scheduling of real-time tasks exploiting dynamic voltage scaling. *Real-Time Systems*, v. 31, n. 1, p. 33–63, 2005.