UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS.

Ríad Mattos Nassiffe

ABORDAGENS PARA RECONFIGURAÇÃO DE SISTEMAS DE TEMPO REAL COM QoS E RESTRIÇÕES DE ENERGIA E TEMPERATURA.

Florianópolis(SC)
2015

Ríad Mattos Nassiffe

ABORDAGENS PARA RECONFIGURAÇÃO DE SISTEMAS DE TEMPO REAL COM QoS E RESTRIÇÕES DE ENERGIA E TEMPERATURA.

Tese submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do Grau de Doutor em Automação e Sistemas.
Orientador: Prof. Eduardo Camponogara.
Coorientador: Prof. Daniel Mossé
Coorientador: Prof. George Marconi de Araújo Lima

Florianópolis(SC)
2015

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Nassiffe, Ríad Mattos
Abordagens para Reconfiguração de Sistemas de Tempo Real com QoS e Restrições de Energia e Temperatura / Ríad Mattos Nassiffe ; orientador, Eduardo Camponogara; co-orientador, George Lima, Daniel Mossé. - Florianópolis, SC, 2015 .
145 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de Automação e Sistemas. 2. Tempo real. 3. Otimização. 4. Escalonamento de tarefas. 5. Economia de energia. I. Camponogara, Eduardo. II. Lima, George. III. Mossé, Daniel. IV Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. V. Título.

Ríad Mattos Nassiffe

"ABORDAGEM PARA RECONFIGURAÇÃO DE SISTEMAS DE TEMPO REAL COM QoS E RESTRIÇÕES DE ENERGIA E TEMPERATURA"

Esta tese foi julgada adequada para a obtenção do título de Doutor em Engenharia de Automação e Sistemas, Área de Concentração de Controle, Automação e Sistemas, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina.

Florianópolis(SC), 25 de Junho 2015.

Prof. Eduardo Camponogara, Ph.D.
Orientador

Prof. George Marconi de Araújo Lima, Ph.D.
Coorientador

Prof. Daniel Mossé, Ph.D.
Coorientador

Prof. Rômulo Silva de Oliveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Antônio Augusto Fröhlich, Dr.

Prof. Sérgio Fernando Mayerle, Dr.

Prof. Rômulo Silva de Oliveira, Dr.

Profª. Luciana Salete Buriol ,Drª.

Prof. Orlando Gomes Loques Filho, Ph.D.

Tese submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do Grau de Doutor em Automação e Sistemas.

AGRADECIMENTOS

Felizmente muitas pessoas fizeram parte da minha vida acadêmica e todas de alguma forma contribuíram para meu desenvolvimento. Sem o apoio delas 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 anos de doutorado, mas em todos os outros que o antecederam sempre me apoiaram como possível. Meus agradecimentos especiais para Maria Flavia, por me alegrar e ouvir nos momentos que necessários.

Relembrando o tempo de graduação, gostaria de agradecer ao meu antigo orientador Edeyson Gomes e aos professores Adolfo Duran e Cláudio Amorim, pois, eles acreditaram na minha capacidade e enviaram cartas de recomendação para que fosse possível minha inscrição no programa de pós-graduação do DAS. Também agradeço aos meus orientadores de mestrado, Eduardo Camponogara e George Lima, que aceitaram continuar me orientando no doutorado. Também gostaria de deixar meus agradecimentos ao Daniel Mossé, que me foi apresentado pelo professor George Lima, e se tornou meu orientador após o ano em que fiz sanduíche na universidade de Pittsburgh.

Gostaria de agradecer a todos os colegas que me apoiaram de alguma forma durante esta jornada que foi o doutorado, onde com certeza não tive apenas crescimento acadêmico mas também um crescimento pessoal. Em especial aos colegas Cassiano, Helton, Robinho, Tonho, Toscano, Bruno, Jim, Maycon, Adriano, Tiago, Maradona, Ebrahim e Vanessa que ajudaram a fazer esse momento mais fácil.

Ainda gostaria de agradecer ao apoio financeiro fornecido pela CAPES e Petrobras, o que permitiu o desenvolvimento do meu trabalho de doutorado.

RESUMO

Esta tese propõe uma infraestrutura para alocação dinâmica de recursos do processador em sistemas de tempo real com tarefas multi-modais ou não, sob restrições de escalonabilidade, consumo de energia e temperatura. Tal infraestrutura pode ser usada para sistemas de tempo real crítico, não crítico e sistemas embarcados que necessitam de garantia de economia de energia. A alocação dinâmica é modelada como um problema de otimização discreto e contínuo (convexos e lineares por parte) para os quais foram analisados algoritmos eficientes para resolução do problema. Embora o problema discreto formulado seja NP-Difícil, os outros possuem soluções eficientes conhecidas e as análises numéricas e simulações mostraram que os modelos usados alcançam bons resultados, com baixo custo computacional.

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

ABSTRACT

This thesis proposes a framework for dynamic reconfiguration, value-based processor resource allocation in multi-modal or not real-time applications, under schedulability, energy consumption and temperature constraints. The framework is suitable for critical and soft real-time adaptive embedded systems which need guarantees of energy savings. The dynamic allocation is formulated as a discrete and continuous (convex and piecewise linear) optimization problem for which efficients algorithms were tested. Although the discrete problem is NP-Hard, the others have efficient solution and numerical analysis and simulations have shown that the used algorithms and models achieves very good results, with low computational cost.

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

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AVFS	*Adaptive Voltage and Frequency Scaling.*
BASH	*Bandwidth Sharing.*
CASH	*Capacity Sharing.*
CBS	*Constant Bandwidth Server.*
CDMA	*Code division multiple access.*
CPU	Unidade de Processamento Central.
DPM	*Dynamic Power Management.*
DSS	*Dynamic Sporadic Server.*
DVFS	*Dynamic Voltage Frequency Scale.*
DVS	*Dynamic Voltage Scale.*
EDF	*Earliest Deadline First.*
GB	*Gigabytes.*
GPRS	Serviço de Rádio de Pacote.
GPS	*Global Positioning System.*
GRUB	*Greedy Reclamation of Unused Bandwidth.*
GSM	*Global System for Mobile.*
I/O	*Input and Output.*
MB	*Megabytes.*
MCVF	Modelo Convexo com Variação de Frequência.
MCVFC	Modelo Convexo Com Variação de Frequência e Carga Computacional.
MDVFC	Modelo Discreto com Variação de Frequência e Carga.
MHz	*Megahertz.*
MLPVF	Modelo Linear por Partes com Variação de Frequência.
QoS	Qualidade de Serviço.
RAM	*Random-access memory.*
SD	*Secure Digital.*
SoC	*System on a Chip.*
SQP	Sequential Quadratic Programming.
STR	Sistema de Tempo Real.

TBS *Total Bandwidth Server*.

W Watts.

LISTA DE SÍMBOLOS

C	Tempo de computação de um conjunto de tarefas.
C^{D}	Tempo de execução máximo da tarefa, referente ao tempo de acesso ao processador pela tarefa.
C^{I}	Tempo de execução máximo da tarefa, referente ao tempo de acesso a dispositivos pela tarefa.
P	Potência do sistema computacional.
P^{D}	Potência dependente da CPU.
P^{Fan}	Potência utilizada pelo *cooler*.
P^{I}	Potência independente da CPU.
P^{Leak}	Potência estática ou corrente de fuga do dispositivo.
P^{S}	Potência consumida pelo sistema computacional para ficar em modo *sleep*.
Q	Tempo de execução máximo de um servidor.
S	Servidor CBS.
T	Período de uma tarefa.
U	Utilização de um conjunto de tarefas.
τ	Tarefa de tempo real.
d	*Deadline* de uma tarefa.
f	Frequência do processador.
$f^{\max}$	Frequência máxima suportada pelo processador.
$f^{\min}$	Frequência mínima suportada.
u	Utilização de uma tarefa.

SUMÁRIO

1 INTRODUÇÃO

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.

No restante desse capítulo será apresentado o problema tratado nesse trabalho, a sua relevância, escopo da solução desejada e por fim a organização desse documento.

1.1 PROBLEMÁTICA

Existem sistemas de tempo real em que conjuntos diferentes de tarefas podem executar de acordo com as condições do ambiente. Um exemplo é um sistema aviônico que quando está taxiando executa um certo conjunto de tarefas e quando está voando executa outro conjunto, com requisitos diferentes (NELIS et al., 2011). Em alguns casos, além do conjunto de tarefas, a qualidade de execução das tarefas pode mudar e a necessidade de economizar energia pode ainda existir também (NELIS et al., 2011; RUSU; MELHEM; MOSSÉ, 2003, 2005). Sistemas robóticos e de sensores são dois exemplos de sistemas que podem apresentar variabilidade na qualidade e no conjunto de tarefas em execução e na disponibilidade de energia (LIU; WU; QIU, 2009; BINI; BUTTAZZO; LIPARI, 2005).

O uso sistemas computacionais com restrição de tempo e energia vem crescendo nas mais diversas atividades, desde o entretenimento até o suporte a trabalhos que envolvem riscos, como a exploração de lugares inóspitos para o homem. Consequentemente, a complexidade envolvendo esses sistemas aumenta de acordo com o número de funcionalidades agregadas e a precisão do sistema, fazendo com que eles necessitem de dispositivos computacionais cada vez mais poderosos.

Um exemplo de aplicação envolve um conjunto sensores distribuídos em uma região afetada por radiação, onde não é viável a troca constante de bateria ou a instalação de uma rede elétrica que alimente o sistema (LIU; WU; QIU, 2009). Nesse caso, é necessário um mecanismo que defina a qualidade de execução das tarefas e frequência do processador, enquanto segura-se as restrições de energia e escalonabilidade de tempo real. Sistemas robóticos são exemplos de aplicações cujos tempo de funcionamento depende da bateria, e executam algoritmos de alta complexidade, que possuem demanda computacional variante no tempo (RUSU; MELHEM; MOSSÉ, 2003; LIU; WU; QIU, 2009).

O tempo de funcionamento de um robô *Pioneer 3-DX* diminui em torno de 50% quando o computador embarcado é ativado (MOBILE ROBOTS, 2010). Tal computador possui um processador *Intel Dual Core* $2,26GHz$, um disco rígido com tecnologia de estado solido, placa de rede sem fio e placa gráfica. Outros exemplos de robôs com propriedades similares são *BigDog* e *Exomars* (ELLERY et al., 2005), que são capazes de tomar decisões em ambientes desestruturados, onde as condições do ambiente não podem ser previamente determinadas e a energia disponibilizada depende da luz solar.

Buscando utilizar esses dispositivos com a melhor qualidade de serviço possível, bem como obedecer as restrições de energia e escalonabilidade, mecanismos que permitem adaptação a diferentes disponibilidades de energia e demandas de

tarefas computacionais são necessários. De acordo com (KHAN; BILAVARN; BELLEUDY, 2012b, 2012a), o limite de energia disponibilizada impacta diretamente na autonomia dos sistemas embarcados. Para maximizar o tempo de autonomia desses dispositivos e permitir que eles cumpram a suas missões, mecanismos de adaptação a diferentes disponibilidades de energia e demandas de tarefas computacionais estão ssendo desenvolvidos.

1.2 OBJETIVOS

Devido às restrições de energia, temporal e de capacidade de processamento dos sistemas computacionais, faz-se necessário a utilização eficiente dos recursos disponíveis no sistema. Visando obedecer esses requisitos, e considerado que todas as tarefas devem ser executadas, a única forma de economizar energia é através do controle do qualidade de execução e da frequência do processador para cada tarefa.

Em sistemas operacionais de propósito geral são utilizadas técnicas de "melhor esforço", as quais tentam adequar a utilização do processador de acordo com a carga de trabalho. Apesar de tais técnicas gerarem economia de energia, elas não provêm garantias temporais para a execução das tarefas e nem de consumo de energia.

Como tais técnicas não funcionam em sistemas de tempo real, é necessário a criação de mecanismos que levem em consideração restrições temporais e energéticas. Para tanto, será necessário cuidar de fatores que possam influenciar o desempenho e consumo energético do processador, como a temperatura. Ou seja, além das já citadas restrições de energia e utilização, tem-se a temperatura como fator que influenciará no escalonamento das tarefas.

Portanto, tem-se o problema de desenvolver um mecanismo de reconfiguração flexível, capaz de selecionar um nível de qualidade e frequência para tarefas, que garanta uma economia de energia mínima, escalonabilidade e temperatura que não comprometa o funcionamento do sistema, garantindo o maior nível de QoS (qualidade de serviço) possível.

Buscando disponibilizar mecanismos para gerenciar a qualidade e a frequência das tarefas, serão ser criado modelos discretos e contínuos que representem o consumo de recursos do sistema e garatam a economia de energia desejada. Nesse sentido serão projetados e analisados algoritmos de gerenciamento de tarefas baseados em programação dinâmica, *branch and bound*, método de ponto-interior, programação quadrática e heurísticas. Alguns desses algoritmos podem ter complexidade exponencial (BŁAŻEWICZ; ECKER, 2007), tornando interessante a aplicação de heurísticas e a investigação sobre as condições de parada para os algoritmos usados. Apesar dos algoritmos de otimização tenderem a ter um custo maior que uma heurística, eles garantem o melhor uso dos recursos envolvidos no problema, por isso serão investigados neste trabalho.

De forma mais detalhada as contribuições desse trabalho podem ser divididas em:

1. análise experimental, para verificar a precisão da formulação utilizada;
2. construção de um modelo matemático inteiro e análise de ferramentas para resolver o problema;
3. criação de um modelo convexo considerando como variável de decisão apenas a frequência;
4. extensão do modelo convexo para considerar a qualidade de execução da tarefa como variável de decisão e análise de algoritmos para resolver o problema;

5. utilização de um modelo linear por partes para resolver os problemas associados.

Por meio dessas contribuições pretende-se fornecer um mecanismo de reconfiguração que seja flexível para atender a diferentes sistemas de tempo real que necessitem de garantia de economia de energia, controle de temperatura e otimização de QoS.

1.3 ORGANIZAÇÃO

Este documento está estruturado da seguinte forma:

1. Capítulos 1 e 2 - Fundamentação Teórica:

   Apresenta a introdução da tese, algumas ferramentas existentes para lidar com escalonamento de sistemas de tempo real e fatores que podem afetar o consumo energético. Está dividido em dois capítulos, sendo o capítulo 1 a presente introdução e o capítulo 2 a revisão bibliográfica do trabalho.

2. Capítulo 3 - Justificativa Experimental:

   Aqui serão mostrados os resultados de alguns experimentos que visam verificar a eficácia de algumas equações mostradas na parte anterior. Esta parte é composta por apenas o capítulo 3.

3. Capítulos 4 e 5 - Modelagem e Algoritmos:

   Nessa parte estão os capítulos 4 e 5, onde serão apresentados modelos discretos e contínuos para representar o mecanismo de reconfiguração desejado. Após a apresentação dos modelos, são apresentados algoritmos e uma análise numérica para mensurar a performance.

4. Capítulo 6 - Avaliações:

   Após apresentar os modelos e uma análise numérica dos algoritmos considerados, será o momento de realizar simulações para comprovar o funcionamento do mecanismo proposto. Essa parte da tese terá somente o capítulo 6, sendo seu início uma descrição dos cenários a serem simulados e, na sequência, os resultados das simulações.

5. Capítulo 7 - Conclusões:

   Por último, o capítulo 7 irá apresentar as conclusões e propostas de trabalhos futuros que foram percebidos durante o desenvolvimento desta tese.

2 REVISÃO BIBLIOGRÁFICA E NOTAÇÕES

Este capítulo é um resumo de assuntos necessários ao entendimento do trabalho realizado. Começará com uma definição sobre tarefas de tempo real, seguido por uma revisão sobre a política de escalonamento utilizada, o tratamento de tarefas não críticas, a alta variabilidade de carga computacional, reconfiguração dinâmica, o consumo de energia e influência da temperatura. Todos os problemas associados ao trabalho serão apresentados juntamente com as abordagens sugeridas pela literatura para resolvê-los. Por último, este capítulo descreve as características, o comportamento do sistema a ser modelado e uma formulação genérica de um problema de otimização.

2.1 TAREFAS DE TEMPO REAL

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 pelo monitoramento de um ambiente em um STR pode ser definida como um conjunto de instâncias que visam verificar modificações no ambiente para realizar a funcionalidade da tarefa. As tarefas que compõem o sistema podem ser utilizadas para classificá-lo.

Portanto, considerando o *deadline* (prazo para executar uma tarefa) os sistemas podem ser classificados como *hard real-time* (crítico) ou *soft real-time* (não crítico), isso vai ser definido de acordo com o tipo de tarefa que compõe o sistema (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 *hard real-time* (BUTTAZZO, 1997). No segundo tipo, existem somente tarefas cuja a perda do prazo implica na perda de qualidade do serviço. Com isso, as tarefas do tipo *hard real-time* serão consideradas periódicas. Isso implica que seu tempo de ativação, momento em que uma tarefa é disponibilizada para execução, e o período, tempo entre as ativações das instâncias, são conhecidos e fixos. As tarefas *soft real-time* são não críticas e podem perder o *deadline*. Elas podem ser: periódicas, esporádicas (não é conhecido o tempo de ativação, mas o tempo mínimo entre duas ativações consecutivas é conhecido) e aperiódicas (não se conhece o tempo de ativação ou período).

Para execução correta de um sistema de tempo real é necessário que esses diferentes tipos de tarefas sejam executadas em uma ordem que garanta o funcionamento sem perda de *deadline*. O procedimento de ordenar as tarefas é chamado de escalonamento, sendo desempenhado pelo escalonador, 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*, segundo o qual todas as tarefas:

- definidas como críticas são periódicas;
- devem executar uma instância antes da próxima chegar;
- são independentes;
- têm o tempo de execução constante;

- definidas como não periódicas no sistema serão tratadas de forma especial e não possuirão 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 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 à concepção de uma heurística, a qual não irá garantir a melhor solução mas fornecerá uma solução que obedeça as 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 subconjunto de tarefas de prioridade fixa forma uma classe, enquanto o subconjunto de tarefas com prioridade fixa e *deadline* igual ao período forma outra classe.

Um algoritmo pode ser considerado ótimo por dois motivos. Primeiro quando consegue escalonar todos os conjuntos de tarefas de uma classe do problema de escalonamento. Caso exista mais de um algoritmo que escalone qualquer conjunto de tarefas de uma determinada classe, aquele que minimiza algum custo ou métrica como, por exemplo, achar uma solução com menor tempo ocioso de *CPU* será o ótimo. No segundo caso, um algoritmo pode ser ótimo mesmo que falhe ao escalonar um ou mais conjuntos de tarefas, contudo nenhum outro algoritmo proposto para resolver a mesma situação pode ser capaz de encontrar uma solução para os conjuntos que ele falhou em resolver (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 detalhada a política *Earliest Deadline First (EDF)*, considerada nesse trabalho.

2.2.1 *Earliest Deadline First*

O*Earliest Deadline First (EDF)* é uma política de escalonamento baseada em prioridade dinâmica, para tentar aumentar a utilização do sistema. As tarefas escalonadas por essa política têm a prioridade definida de acordo com os *deadlines*. Quanto mais próxima uma tarefa estiver de alcançar seu prazo de execução, maior será a sua prioridade e ela poderá preemptar as outras que estão mais longe do *deadline*.

Portanto, a política *EDF* não faz nenhuma suposição sobre a periodicidade das tarefas para atribuir prioridade, ao contrário de outras que fazem, permitindo que seja empregado no escalonamento de tarefas aperiódicas e periódicas. A prioridade é definida de acordo com o *deadline*.

Para um sistema ser escalonável por *EDF*, considerando o melhor cenário, basta que a sua utilização de *CPU* no pior caso seja menor ou igual a 100% (LIU; LAYLAND, 1973), sendo a utilização de cada tarefa representada entre 0 e 1, conforme estabelece a equação a seguir:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1. \tag{2.2}$$

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 que qualquer tarefa pode forçar a interrupção de outra do sistema. Isso ocorre, pois a prioridade é definida com base no *deadline*, fazendo com que todas as tarefas tenham a mesma importância para o sistema. Logo, caso ocorra um erro na medição do C_i por subestimação que leve a equação (2.2) a ser desrespeitada, será difícil prever o comportamento do sistema, uma vez que todas as tarefas críticas ou não terão a mesma importância.

2.3 SERVIDORES

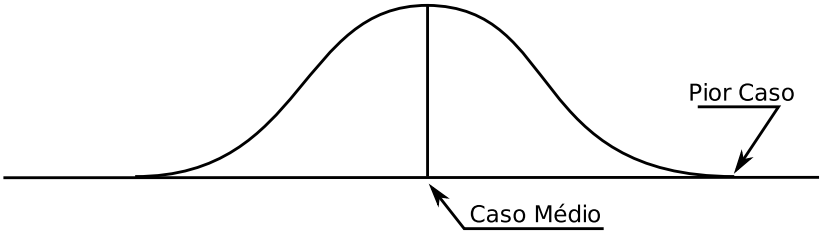
A política de escalonamento citada anteriormente não diferencia uma tarefa crítica de uma não crítica, forçando o sistema a ter somente um tipo de tarefa. Contudo, muitos dos sistemas existentes trabalham com tarefas de diferentes níveis de criticidade (BUTTAZZO, 1997). Esse tipo de sistema pode ser escalonado com o uso de servidores de tarefas aperiódicas, que na realidade são tarefas periódicas 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 pelas quais 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 1, que mostra 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 servidor pertence ao

Figura 1: Gráfico da distribuição normal, mostrando a probabilidade de ocorrência do caso médio e pior caso do uso do processador.



Fonte: Criado por (OLIVEIRA, 2009)

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) sejam 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 seu período, 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.

Tendo essas três propriedades definidas, a seguir será explicado o algoritmo do *Constant Bandwidth Server (CBS)*, considerando a política *EDF*.

2.3.1 *Constant Bandwidth Server (CBS)*

O *CBS* é inspirado nos servidores *Total Bandwidth Server* (*TBS*) (BUTTAZZO, 1997) e *Dynamic Sporadic Server* (*DSS*)(SPURI; BUTTAZZO, 1994). O diferencial desse servidor em relação aos que o inspiraram 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 $i = 1, ..., n$, 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 orçamento do servidor é 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 $d_{i,k+1} = d_{i,k} + T_i$.

Tabela 1: Tarefas do exemplo do escalonamento com *CBS*.

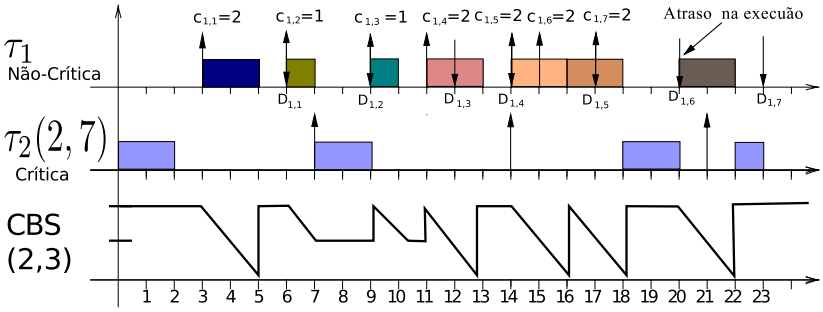
	C_i	T_i	U_i
$CBS(\tau_1)$	2	3	≈0,67
τ_2	2	7	≈0,29
Total			0,96

Fonte: Criado pelo Autor.

O funcionamento de um sistema que implementa o *CBS* é ilustrado na Figura 2, onde existem duas tarefas, uma não crítica (τ_1, com sete instâncias) e outra crítica (τ_2 com quatro instâncias). A Tabela 1 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.

Figura 2: Execução das tarefas τ_1 e τ_2 em um sistema híbrido usando servidor CBS para a tarefa não crítica.



Fonte: Criado pelo Autor.

A primeira linha da Figura 2 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 $T_i = 3$.

A segunda linha da Figura 2 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 lançada 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 o 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 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, que ao entrar em um novo ambiente precisa mapeá-lo, depois traçar uma rota e no momento que começa a se locomover fica monitorando 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 (*i.e.*, 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 ocorre de forma independente, 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 existem 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.

2.5 ENERGIA

A economia de energia em um sistema computacional pode ser realizada por meio do gerenciamento de operações de *I/O*, de acesso à memória e uso do processador, entre outras formas. Em escalonadores, a economia está restrita ao controle de energia utilizada pelo processador, pois ele é responsável por dividir o tempo de *CPU* entre as tarefas que estão executando. Sistemas operacionais modernos como *Windows*, *Mac OSX* e *Linux* modificam a frequência e a voltagem do processador de acordo com a carga de trabalho ou vontade do usuário em diminuir o consumo energético.

Contudo, tal recurso implementado não leva em consideração as necessidades de um sistema de tempo real, o qual tem restrições temporais para executar. Ajustar somente a frequência e voltagem do processador, *DVS* (*Dynamic voltage Scaling*) ou *DVFS* (*Dynamic Voltage and Frequency Scaling*), em função da carga de trabalho sem considerar os dispositivos que uma tarefa utiliza pode não gerar economia de energia. Quando usado sem auxílio de uma modelagem eficiente, o *DVS* considera o gasto energético do sistema como o gasto de energia da *CPU* e a redução da frequência e voltagem pode aumentar o consumo energético da tarefa ao invés de reduzir (ZHU; MELHEM; MOSSÉ, 2004).

Por isso, nesse trabalho as tarefas serão modeladas de acordo com (ZHU; MELHEM; MOSSÉ, 2004), no qual o tempo total de execução de uma tarefa é dado pelo tempo consumido de *CPU* e dispositivos. Como o tempo da tarefa é dividido em uma parte que depende do processador e outra que é independente, qualquer ajuste realizado com *DVS* deve afetar somente o tempo que depende da *CPU*, assim

ficando o tempo de uma tarefa no pior caso definido como:

$$C_i = {C^{\mathrm{I}}}_i + \frac{{C^{\mathrm{D}}}_i}{f_i}$$

onde C_i^{D} é o tempo efetivo que uma tarefa usa de processador, C_i^{I} é o tempo utilizado por outros dispositivos e f_i é a utilização do processador. Caso o valor da utilização da *CPU* seja pela metade, o tempo da tarefa que depende dela será o dobro e o que não depende não será alterado.

Assim como o modelo de tarefas, o modelo energético *DPM* (*Dynamic Power Management*) usado nesse trabalho considera outros dispositivos além do processador (ZHAO; AYDIN, 2009; ZHUO; CHAKRABARTI, 2008). O consumo de energia (watts) de uma tarefa i por unidade de tempo em um sistema será visto como:

$$P_i = {P^{\mathrm{I}}}_i + {P^{\mathrm{D}}}_i + P^{\mathrm{S}} + P^{\mathrm{Leak}} + P^{\mathrm{Fan}} \tag{2.3}$$

onde:

- P_i^{I} representa a potência elétrica que não depende da frequência do processador (disco rígido e outros dispositivos), determinado pelos dispositivos utilizados pela tarefa i;
- P_i^{D} indica a quantidade de energia por unidade de tempo usada pela *CPU* para executar a tarefa i;
- P^{S} é o gasto energético para manter os dispositivos do sistema em modo *sleep*, quando não estão em uso; esse é um parâmetro do modelo que não depende das tarefas e é fixo, não podendo variar, por isso é desconsiderado;
- P^{Leak} representa potência estática ou corrente de fuga dos circuitos do processador; não depende da tarefa em execução e sim da tecnologia de fabricação e temperatura da CPU.
- P^{Fan} é a potência gasta pelo *cooler* para resfriar a CPU, sendo esta considerada constante (ZAPATER et al., 2013) e não varia de acordo com a tarefa.

O valor de P_i^{D} é dado pela potência dinâmica do processador, que de acordo com (KYUNG; YOO, 2011) pode ser expressa por:

$$P_i^{\mathrm{D}} \doteq C_{\mathrm{ef}} V^2 F_i \tag{2.4}$$

onde F_i é a frequência usada pelo processador para executar a tarefa i, C_{ef} a capacitância necessária para execução de uma instrução e V a voltagem necessária para o funcionamento do sistema em uma determinada frequência.

É considerado que a voltagem decresce de forma linear à frequência f_i usada por uma tarefa, sendo essa calculada como $f_i = \frac{F_i}{F_{\max}}$. Logo, a voltagem necessária para o processador operar em uma frequência F_i é dada com base na utilização da *CPU*, $V = V_{\max} f_i$ (ZHU; MELHEM; MOSSÉ, 2004). Os parâmetros $V_{\max}$, C_{ef} e $F_{\max}$ são valores constantes, por isso podem ser substituídas por C_0. Assim, ficando o consumo energético de uma tarefa em relação ao processador expresso por:

$$P_i^{\mathrm{D}} \doteq C_0 f_i^3. \tag{2.5}$$

O P^{Leak}, de acordo com (ZAPATER et al., 2013), é determinado pela seguinte equação:

$$P^{\mathrm{Leak}} = C_1 + C_2 e^{HC_3} \tag{2.6}$$

onde: C_1, C_2 e C_3 são constantes definidas de acordo com o material e disposição do processador no sistema, podendo também serem determinadas por experimentos com a *CPU*. Ainda de acordo com (ZAPATER et al., 2013) a soma da energia consumida por P^{Leak} mais sistema de resfriamento é convexa, assim existindo um ponto de mínimo onde o sistema de resfriamento tem a melhor performance energética. Ao determinar esse ponto de mínimo para o sistema, o valor de P^{Fan} torna-se constante.

Portanto o consumo energético do sistema para uma tarefa se torna:

$$P_i(f_i) = C_0 f_i^3 + P_i^{\text{I}} + P^{\text{Leak}}. \tag{2.7}$$

Em alguns sistemas críticos de tempo real pode ser possível estimar quanto tempo uma tarefa precisará de *CPU* durante a vida de um sistema. Entretanto, estimar o tempo de processamento de uma tarefa quando o sistema permite a troca de modo de operação, devido a fatores externos e independentes, é impossível.

Isto torna interessante a análise do consumo energético de acordo com a potência do conjunto de tarefas que está em execução no sistema. A economia realizada com base na energia consumida por unidade de tempo refletirá no consumo total de energia. Ficando o consumo energético dinâmico de um conjunto de tarefas do sistema dado por:

$$\sum_{i=1}^{n} \left((C_0 f_i^3 + P_i^{\text{I}} + P^{\text{Leak}}) u_i \right) = \sum_{i=1}^{n} \left((C_0 f_i^3 + P_i^{\text{I}} + P^{\text{Leak}}) \left(\frac{C_i^{\text{I}}}{T_i} + \frac{C_i^{\text{D}}}{T_i f_i} \right) \right). \tag{2.8}$$

Como visto anteriormente, tanto o valor de P_i^{D} quanto P_i^{I} são dados em relação à energia consumida por unidade de tempo dos dispositivos e u_i representa quanto uma tarefa usa por unidade de tempo da *CPU* e por consequência do sistema. Sendo assim, ao multiplicar $P_i^{\text{D}} + P_i^{\text{I}} + P^{\text{Leak}}$ por u_i obtém-se a potência de uma tarefa.

Uma análise da equação (2.8) foi realizada para entender melhor os comportamentos que podem ocorrer em um sistema. Para a análise, a fórmula de energia de uma tarefa por unidade de tempo foi dividida em quatro termos, como pode ser visto logo abaixo:

$$\begin{aligned} P_i(f_i) &= (C_0 f_i^3 + P_i^{\text{I}} + P^{\text{Leak}}) u_i \\ &= (C_0 f_i^3 + P_i^{\text{I}} + P^{\text{Leak}}) \left(\frac{C_i^{\text{D}}}{T_i f_i} + \frac{C_i^{\text{I}}}{T_i} \right) \\ &= \left(\frac{C_0 C_i^{\text{I}}}{T_i} \right) f_i^3 + \left(\frac{C_0 C_i^{\text{D}}}{T_i} \right) f_i^2 \\ &\quad + \left(\frac{(P_i^{\text{I}} + P^{\text{Leak}}) C_i^{\text{D}}}{T_i} \right) \frac{1}{f_i} + \frac{(P_i^{\text{I}} + P^{\text{Leak}}) C_i^{\text{I}}}{T_i} \\ &= \alpha_3 f_i^3 + \alpha_2 f_i^2 + \alpha_1 \frac{1}{f_i} + \alpha_0 \end{aligned}$$

com:

$$\alpha_3 = \frac{C_0 C_i^{\mathrm{I}}}{T_i}$$
$$\alpha_2 = \frac{C_0 C_i^{\mathrm{D}}}{T_i}$$
$$\alpha_1 = \frac{(P_i^{\mathrm{I}} + P^{\mathrm{Leak}}) C_i^{\mathrm{D}}}{T_i}$$
$$\alpha_0 = \frac{(P_i^{\mathrm{I}} + P^{\mathrm{Leak}}) C_i^{\mathrm{I}}}{T_i}.$$

Para realizar a análise as hipóteses abaixo foram consideradas:

Hipótese 1. *f_i é a frequência do processador, dada pela razão da sua utilização para cada tarefa i, com domínio em* $\mathrm{dom} f_i = [f_i^{\min}, f_i^{\max}]$ *tal que* $0 < f_i^{\min} < f_i^{\max} \leq 1$.

Essa hipótese pode ser facilmente observada em um processador, pois em condições normais ele pode operar em alguma frequência acima de zero e abaixo da máxima determinada pelo fabricante.

Hipótese 2. *Os parâmetros do sistema que satisfazem as condições operacionais se:* $C_0, T_i > 0$; $P_i^{\mathrm{I}}, C_i^{\mathrm{D}}, C_i^{\mathrm{I}}, P^{\mathrm{Leak}} \geq 0$ *e* $C_i^{\mathrm{D}} > 0$ *ou* $C_i^{\mathrm{I}} > 0$.

Observação 1. $\alpha_i \geq 0$ *para todo i, com pelo menos* $\alpha_2 > 0$ *ou* $\alpha_3 > 0$.

Agora, considerando-se a primeira e segunda derivadas da função de energia, tem-se que:

$$P_i' = 3\alpha_3 f_i^2 + 2\alpha_2 f_i - \frac{\alpha_1}{f_i^2}$$
$$P_i'' = 6\alpha_3 f_i + 2\alpha_2 + 2\frac{\alpha_1}{f_i^3}.$$

Observação 2. *Tendo em vista a Observação 1,* $P_i''(f_i) > 0$ *para todo* $f_i \in \mathrm{dom} f_i$ *e, portanto $P_i(f_i)$ é estritamente convexa no domínio. Logo $P_i(f_i)$ tem um mínimo global único no dom f_i.*

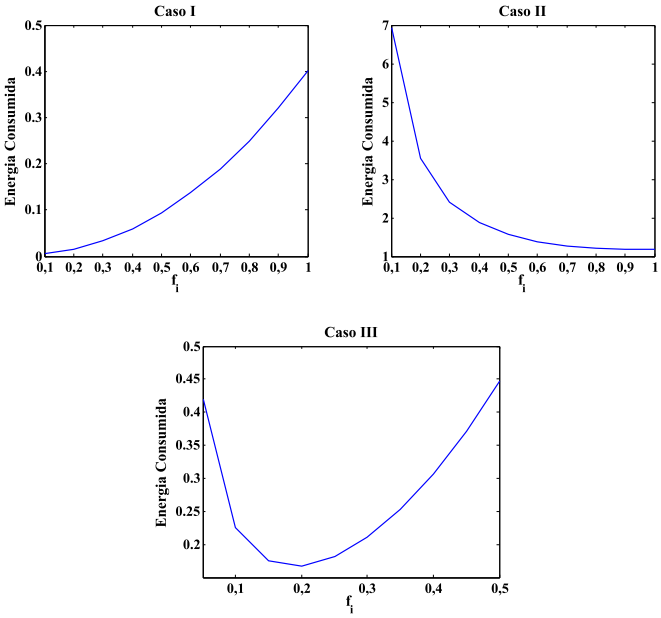
Observação 3. *Devido a* $P_i'' > 0$ *dentro de seu domínio, P_i' é uma função crescente.*

Sendo assim a função de energia P_i pode ser categorizada em três categorias, conforme a observação abaixo:

Observação 4. *A função de energia P_i pertence a uma das seguintes classes:*

- *Caso I:* $P_i'(f_i^{\min}) \geq 0$. *Então $P_i(f_i)$ é crescente em f_i, que aumenta de* $f_i^{\min}$ *até* $f_i^{\max}$.
- *Caso II:* $P_i'(f_i^{\min}) < 0$ *e* $P_i'(f_i^{\max}) < 0$. *Então $P_i(f_i)$ diminui à medida que f_i aumenta de* $f_i^{\min}$ *até* $f_i^{\max}$.
- *Caso III:* $P_i'(f_i^{\min}) < 0$ *e* $P_i'(f_i^{\max}) \geq 0$. *Então $P_i(f_i)$ alcança o valor mínimo no* $\mathrm{dom} f_i$, *sendo este dado por algum* $f_i^\star \in dom f_i$ *tal que* $P_i'(f_i^\star) = 0$.

Figura 3: No primeiro caso pode-se observar que o consumo energético cresce de acordo com aumento de f_i, no segundo ocorre o inverso e no terceiro a função tem um ponto de mínimo que é único.



Fonte: Criado pelo Autor.

Sendo assim, cada tarefa terá sua frequência mínima determinada pelo mínimo da função de energia no intervalo de $f_i = (0, 1]$ ou a menor frequência suportada pelo processador, caso ela seja maior que o mínimo dado por $f_i^{\min}$. Caso o valor do mínimo da função de energia encontrado seja maior que o suportado pelo processador, este será utilizado.

Esses três casos da função de energia são ilustrados na Figura 3. Para fazer os gráficos, as seguintes considerações sobre o processador foram feitas: frequência máxima de $2,53GHz$ e mínima de $0,253GHz$, $P^{\mathrm{S}} = 0W$ e $C_0 = 24.99W$. A tarefa em questão tinha um $P^{\mathrm{Leak}} = 0,01W$, $C_i^{\mathrm{I}} = 0,7 \times 10^{-4}s$, $C_i^{\mathrm{D}} = 0,45 \times 10^{-2}s$ e $T_i = 0,33s$. Para a simulação do Caso I a tarefa tinha um $P_i^{\mathrm{I}} = 0,2W$, no Caso II $P_i^{\mathrm{I}} = 50W$ e para o terceiro caso $P_i^{\mathrm{I}} = 3W$.

Pelo fato do consumo energético de uma tarefa não ser fixo, devido à variabilidade da carga computacional, o tempo de execução do sistema varia e a troca de modo do sistema ocorre de acordo com alguma condição do ambiente, não sendo possível estimar exatamente o quanto de energia é necessário para realizar um objetivo. Portanto, se torna necessário estimar a economia necessária no pior caso, cenário de maior consumo energético, para garantir uma economia desejada e continuamente verificar se existe a quantidade esperada na bateria.

2.6 TEMPERATURA

De acordo com (LIAO; HE; LEPAK, 2005) a temperatura do processador tem uma grande influência no consumo energético dos sistemas computacionais, uma vez que a tecnologia de resfriamento não evolui na mesma velocidade que as *CPUs*. Em (LIU et al., 2007; FLAUTNER; FLYNN; RIVES, 2003) é mostrado que através do controle da temperatura do processador é possível economizar energia por meio da diminuição do uso do sistema de resfriamento e P^{Leak}. Além do aumento do consumo de energia, o superaquecimento do processador pode trazer outros problemas como a redução do tempo de vida e de desempenho. Por isso, para evitar o aquecimento do processador o gerenciamento dinâmico de temperatura (*DTM-Dynamic Themal Management*) é necessário.

Uma das formas mais usadas para gerenciar a temperatura de um processador é o *throttling*, no qual ao atingir uma temperatura máxima estipulada a frequência é reduzida à mínima possível, só podendo aumentar quando a *CPU* resfria a uma outra temperatura mínima. Essas temperaturas são definidas de acordo com as capacidades térmicas do processador. Apesar dessa técnica ser amplamente usada e implementada em *hardware* para diminuir o *overhead*, ela tem efeitos negativos para os sistemas de tempo real, uma vez que reduz a frequência do processador a mínima possível sem comunicar ao sistema operacional e por um tempo indeterminado. Esse comportamento pode fazer com que uma tarefa de tempo real demore mais tempo para terminar e por consequência cause a perda do seu *deadline* e não garante economia energia.

Em consequência disso, trabalhos como (LIAO; HE; LEPAK, 2005) controlam a temperatura da *CPU* para que o mecanismo de *throttling* não seja ativado e o processador tenha um melhor desempenho. Esse trabalho em específico não se preocupa com tarefas de tempo real, ele apenas tenta melhorar a performance e diminuir o consumo energético. Isso é realizado pela definição de novas frequências máxima e mínima que evitam a utilização da menor frequência suportada pela *CPU*.

De acordo com (GU; QU, 2013; FU et al., 2010), a variação de temperatura da *CPU* pode ser modelada em função da potência consumida e a capacidade de resfriamento, conforme a equação a seguir:

$$\Delta H(f_i) = \frac{P_i^{\text{D}}}{C_{\text{th}}} - \frac{h - h_{\text{amb}}}{R_{\text{th}} C_{\text{th}}} \tag{2.11}$$

sendo: $P_i^{\text{D}} = C_0 f_i^3$, C_{th} a capacitância térmica do *chip*, R_{th} resistência térmica, e h e h_{amb} respectivamente a temperatura do processador e ambiente. Para simplificar a função $\Delta H(f_i)$ serão introduzidas as seguintes constantes:

$$a = \frac{C_0}{C_{\text{th}}}, \tag{2.12a}$$

$$b = \frac{1}{R_{\text{th}} C_{\text{th}}}. \tag{2.12b}$$

Com isso a temperatura atual da *CPU* é definida pela soma da atual (h) mais a variação daquele momento, como escrito na equação a seguir:

$$H(f_i) = a f_i^3 - b(h - h_{\text{amb}}) + h. \tag{2.13}$$

A temperatura do processador depende apenas da frequência f_i selecionada, por isso ela será usada para definir uma frequência máxima ($f^{\max}$) para o processador:

$$H(f_i) = a{f^{\max}}^3 - b(h - h_{\text{amb}}) + h \tag{2.14a}$$

$$f^{\max} = \min\{\sqrt[3]{\frac{H(f_i) + b(h - h_{\text{amb}}) - h}{a}}, 1\} \tag{2.14b}$$

onde $H(f_i)$ deve ser substituída pela temperatura máxima suportada pelo processador. Com isso é possível determinar um valor para $f^{\max}$ que irá impedir que o processador use o *throttling* ou superaqueça e pare de funcionar.

2.7 ABORDAGENS DE RECONFIGURAÇÃO COM ECONOMIA DE ENERGIA

Uma breve descrição sobre sistemas de tempo real e meios para economizar energia neles foi comentado nas seções acima, podendo-se agora entender algumas das abordagens encontradas na literatura. A economia de energia em sistemas de tempo real é um problema antigo que já foi estudado por muitos pesquisadores, para diversos casos distintos e algumas abordagens serão comentadas a seguir.

Em (YAO; DEMERS; SHENKER, 1995; QUAN; HU, 2001; JEJURIKAR; PEREIRA; GUPTA, 2004; XIAN; LU; LI, 2008) é realizada a economia de energia com melhor esforço, não sendo fornecida nenhuma garantia de quanto vai ser economizado, por meio de uma abordagem *off-line*. Já nos trabalhos (SHIN; CHOI; SAKURAI, 2000; KIM; KIM; MIN, 2002; ZHU; MUELLER, 2005; LIU; WU; QIU, 2009) é proposta uma solução *on-line*, capaz de aproveitar o tempo de execução não utilizado por uma tarefa.

Já os trabalhos (RUSU; MELHEM; MOSSÉ, 2003a, 2003b, 2005; XU; MOSSÉ; MELHEM, 2007; ZHAO; AYDIN; ZHU, 2010; NIU, 2011; NASSIFFE; CAMPONOGARA; LIMA, 2013; NASSIFFE et al., 2013) oferecem garantia de economia de energia, trabalham com conceito de QoS e alguns suportam tarefas com múltiplas versões. Entretanto estes trabalhos não consideram o efeito da temperatura no sistema e os modelos do consumo energético não são detalhados e eficientes para reproduzir o comportamento do sistema. A solução fornecida em (RUSU; MELHEM; MOSSÉ, 2003b) é considerada ótima.

O efeito da temperatura no consumo energético de sistemas computacionais é analisado e comprovado em (LIAO; HE; LEPAK, 2005; ZAPATER et al., 2013). Esses trabalhos mostram que a alta temperatura pode causar vários efeitos indesejáveis como: aumento do consumo de energia do processador e sistema de resfriamento, redução no desempenho devido ao *throttling*. Tentando minimizar esses problemas, os trabalhos (YUAN; LEVENTHAL; QU, 2006; GU; QU, 2013; RAO; VRUDHULA, 2007; RAJAN; YU, 2007; FU et al., 2010; WANG; BETTATI, 2008) propõem novos modelos que tentam diminuir a frequência para controlar a temperatura e melhorar a utilização do sistema, sem se preocupar com a economia de energia.

Com isso pode-se concluir que existem diversos trabalhos em economia de energia em sistemas de tempo real. Contudo, não foi encontrado nenhum trabalho que considerasse um sistema com tarefas modais, modelando seus tempos de execução em tempo de dispositivos e CPU, garantindo a economia de energia, influência da temperatura no sistema, QoS e com uso de isolamento temporal ou reserva de banda. Por isso esta tese propõe um mecanismo que trabalhe com todos esses pontos.

2.8 NOTAÇÕES

Em geral o sistema considerado nesta tese é constituído de n tarefas que compartilham um único processador, que opera com um conjunto de frequências que variam de forma discreta ou contínua. O conjunto de tarefas do sistema pode conter somente tarefas do tipo *hard*, *soft* ou uma mistura dos dois tipos. Para garantir que uma tarefa não irá fazer outra perder o *deadline*, é utilizada uma técnica de servidores aperiódicos *CBS*.

Sendo assim, uma 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, podendo se utilizar até 100% da *CPU*.

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. É considerado que o valor de Q_i^{D} é inversamente proporcional ao valor da frequência selecionada: quando se reduz 50% da frequência do processador o valor de Q_i^{D} aumenta em 100%. 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, em função dos orçamentos e de f_i da *CPU*, por:

$$Q_i = \frac{Q_i^{\mathrm{D}}}{f_i} + Q_i^{\mathrm{I}}. \tag{2.15}$$

Os modelos podem assumir que as tarefas são multi-modais ou não. A utilização de tarefas multi-modais advém da necessidade de alguns sistemas que dinamicamente têm suas necessidades computacionais modificadas, por exemplo: em sistemas de câmeras de segurança no qual o custo para processamento de imagens varia, sistemas robóticos que interagem com o meio que está constantemente mudando, *etc.*

A utilização do processador é estabelecida pelo somatório da utilização de cada servidor S_i, que é calculada por:

$$u_i \doteq \frac{Q_i^{\mathrm{I}}}{T_i} + \frac{Q_i^{\mathrm{D}}}{T_i f_i}. \tag{2.16}$$

Assim, para que as tarefas possam ser executadas em um escalonador *EDF*, tem-se a seguinte condição:

$$\sum_{i=1}^{n} u_i \leq 1 \Longleftrightarrow \sum_{i=1}^{n} \left(\frac{Q_i^{\mathrm{I}}}{T_i} + \frac{Q_i^{\mathrm{D}}}{T_i f_i} \right) \leq 1.$$

No trabalho é pressuposto que existe um custo na chamada do mecanismo de reconfiguração. Sendo assim, ele deve ser invocado só quando necessário. O mecanismo de reconfiguração deve determinar somente a frequência na qual as tarefas irão executar ou a frequência e o tempo de processador.

2.9 MODELO DE OTIMIZAÇÃO

Considerando as equações encontradas na literatura e os experimentos apresentados nesse capítulo, um modelo matemático para representar sistemas de tempo

real com limites de consumo energético, temperatura e escalonabilidade será apresentado agora de forma genérica. O sistema será modelado como um problema de otimização e induz o QoS, que será representado pela função objetivo. Poderão ser utilizados algoritmos exatos (como Branch-and-Bound) ou heurísticas para resolver o problema. Métodos exatos podem prover uma solução ótima (obedece a todas as restrições e o maior QoS possível) para uma situação. Dependendo do tamanho do problema eles podem demorar muito para fornecer uma resposta, surgindo assim a necessidade de utilizar heurísticas capazes de fornecer uma solução com QoS mais próximo da solução ótima.

O sistema modelado será composto por n tarefas τ_i de tempo real. Cada uma com múltiplos modos de operação associados a um tempo de execução, que será representado por $Q_i^{\mathrm{D,min}}$ e $Q_i^{\mathrm{D,max}}$. O sistema considerará a existência de um processador, com a frequência f_i, pertencente ao intervalo $[f^{\min}, f^{\max}]$. Sob essas condições o modelo genérico fica representado por:

$$P: \quad \max \quad \sum_{i=1}^{n} G(Q_i^{\mathrm{D}}, f_i) \tag{2.17a}$$

$$\text{s.t.}: \sum_{i=1}^{n} u_i(f_i, Q_i^{\mathrm{D}}) \leq 1 \tag{2.17b}$$

$$\sum_{i=1}^{n} P_i \leq P^{\star} \tag{2.17c}$$

$$f^{\min} \leq f_i \leq f^{\max},\ i = 1, \ldots, n \tag{2.17d}$$

$$Q_i^{\mathrm{D,min}} \leq Q_i^{\mathrm{D}} \leq Q_i^{\mathrm{D,max}},\ i = 1, \ldots, n \tag{2.17e}$$

onde:

- a função objetivo (2.17a) representa o somatório do QoS, que está relacionado com as configurações selecionadas para cada tarefa do sistema;
- Eq. (2.17b) modela a utilização da *CPU* de acordo com a política de escalonamento *EDF*;
- a restrição (2.17c) representa a soma da potência P_i gasta por cada configuração de tarefa τ_i, e $P^{\star}$ é a energia disponível por unidade de tempo para o sistema;
- a Eq. (2.17d) estabelece os valores de f_i dentro dos limites. O valor $f^{\max}$ deve ser definido de forma que não deixe o sistema violar a temperatura máxima escolhida;
- a restrição (2.17e) define o valor máximo e mínimo de execução que uma tarefa tem e está associado a um modo de execução.

Um importante ponto do modelo é a função objetivo, pois ela tem que refletir algumas propriedades do sistema. Ela deve essencialmente capturar três aspectos:

1. priorizar sempre as configurações que agreguem maior qualidade de serviço ao sistema;
2. definir a prioridade de cada tarefa de acordo com a sua importância para a aplicação;
3. considerar o peso de cada tarefa, conforme a quantidade de recurso consumido. Uma tarefa que consome 10% de processador não deve ser tratada como uma que consome 1%.

Este trabalho apresenta quatro modelos para reconfigurar sistemas de Tempo Real que podem usar qualquer política de escalonamento que consiga verificar a escalonabilidade pela soma da utilização das tarefas. Esses modelos serão discretos ou contínuos, podendo considerar somente a variação de frequência ou frequência e carga computacional. Segue abaixo uma breve descrição dos modelos propostos aqui:

- **Modelo Discreto com Variação de Frequência e Carga (MDVFC)**: Os atuais sistemas computacionais funcionam de forma discreta, podendo trocar a frequência e voltagem somente em valores pré-determinados. Nesse modelo, assume-se valores pré-determinados para Q_i^{D} e f_i, o que leva a um problema de programação inteira. No capítulo 4 será apresentada a adaptação de duas heurísticas para resolver o problema e uma reformulação compacta para o modelo, para que possa ser usada uma abordagem de desigualdades de cobertura para resolver.
- **Modelo Convexo com Variação de Frequência (MCVF)**: Apesar dos atuais sistemas computacionais funcionarem de forma discreta, existem vários processadores que implementam *AVFS* (*Adaptive Voltage Frequency Scaling*) ou suportam um grande número de frequências diferentes. Neste sentido, o modelo matemático a ser seguido assume que f_i pode variar no intervalo contínuo $[f_i^{\min}, f_i^{\max}]$. Esse modelo considera que a tarefa só tem um modo de operação. No capítulo 5 serão apresentados algoritmos eficientes para a resolução do problema com garantia de otimalidade, que é consequência da natureza convexa do modelo criado.
- **Modelo Convexo Com Variação de Frequência e Carga Computacional (MCVFC)**: Neste modelo será considerado o problema com duas variáveis contínuas de decisão Q_i^{D} e f_i. Tal modelo assume que para cada Q_i^{D} e f_i existem respectivamente os intervalos: $[Q_i^{\mathrm{D,min}}, Q_i^{\mathrm{D\,max}}]$ e $[f_i^{\min}, f_i^{\max}]$. Esse modelo permite a utilização de algoritmos eficientes para a resolução de problemas de grande porte com garantia de otimalidade a serem apresentados no capítulo 5. Estas propriedades advêm da natureza convexa do problema de otimização proposto.
- **Modelo Linear por Partes com Variação de Frequência (MLPVF)**: Pelos testes realizados nesse capítulo, foi percebido que a curva do consumo energético do *Raspberry Pi* pode ser representada por um modelo linear por partes. Isso corre pelo fato de que apesar do processador simular a frequência desejada ele não faz o mesmo com a voltagem. O modelo a ser apresentado irá considerar epenas a existência da variável de decisão f_i no intervalo $[f_i^{\min}, f_i^{\max}]$, conforme discussão no capítulo 5. Contudo, vale ressaltar que este modelo pode ser estendido para o caso multidimensional, o que permite a otimização da frequência f_i e do orçamento Q^{D} simultaneamente.

2.10 SUMÁRIO

Este capítulo apresentou os conceitos necessários para o entendimento do trabalho, como a fundamentação de escalonamento de sistemas em tempo real e a política de escalonamento utilizada. O tratamento de tarefas não críticas com ênfase no *CBS* foi detalhado pois este será utilizado nos testes computacionais. A modelagem energética do sistema procurou ser a mais fiel possível aos sistemas reais e a influência da temperatura foi mostrada e será levada em consideração. A questão de mudança de modos, necessária à reconfiguração dinâmica dos parâmetros dos servidores, foi

abordada e depois foi feita uma revisão sobre trabalhos relacionados ao tema. Por último, foi apresentada a notação a ser empregada na descrição dos modelos e uma formulação genérica de um problema de otimização.

3 AVALIAÇÃO EXPERIMENTAL

No capítulo anterior foram apresentadas informações sobre formas conhecidas na literatura para modelar e representar o consumo energético de um dispositivo *Raspberry Pi*. Apesar dos trabalhos teóricos realizados até agora, muitas dúvidas ainda persistem sobre a precisão das equações. Vários trabalhos usam tais equações somente para representar o comportamento do consumo energético do sistema e não para determinar a quantidade de energia que está sendo consumida.

Nesse capítulo serão apresentados experimentos para mensurar a diferença do consumo energético de um *Raspberry Pi* e as equações apresentadas. Para isso, a primeira seção descreve as condições gerais em que os testes foram realizados, seguida por experimentos para validar o modelo de tarefas e uma análise sobre o consumo de energia e temperatura.

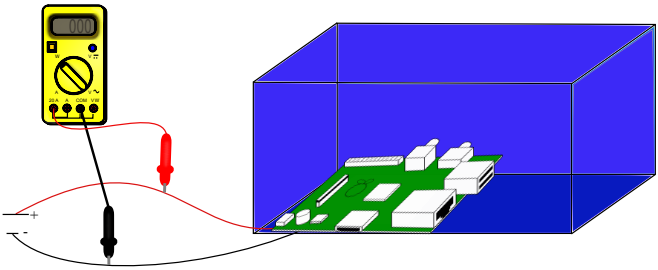
3.1 CONDIÇÕES DOS EXPERIMENTOS

Para realizar os testes foi utilizado um dispositivo *Raspberry Pi* modelo B. O *Raspberry Pi* é um computador do tamanho de um cartão de crédito com consumo energético máximo de 3,5 W. Ele é baseado em um sistema *SoC* (*System On a Chip*), possui um processador *ARM1176JZF-S* que por padrão funciona a $700MHz$, podendo variar até $1000MHz$, tem 512 MB de memória RAM e utiliza como disco rígido um cartão SD de $16GB$.

A distribuição *Arch Linux* foi instalada no *Raspberry Pi*, com *kernel Linux* 3.15. Durante os testes duas configurações do *kernel* foram alteradas para tentar diminuir a interferência do sistema operacional nos testes, uma vez que ele não é *hard real-time*. A primeira foi a troca do escalonador de tarefas, que foi substituído pelo *SCHED_FIFO*. Isso garante que uma tarefa só tem a sua execução interrompida se alguma outra de prioridade maior for inicializada. A outra modificação foi a do escalonador de disco, que foi trocado pelo *NOOP*, que executa o pedido de acesso ao disco assim que solicitado. Com essas alterações espera-se eliminar o *overhead* da troca de contexto ou chaveamento da tarefa e o tempo de espera para acessar o disco.

Para realizar os testes envolvendo temperatura e energia, o *Raspberry Pi* foi colocado em uma caixa, como mostra a Figura 4. Durante os testes o interior da caixa foi aquecido, para evitar que a placa trocasse calor com o ambiente e assim esquentasse. A medição do consumo energético foi realizada com uso de um multímetro da *Tenma 72-9380A*. O multímetro usado é capaz de coletar entre duas e três amostras do valor da corrente que está sendo usada, como ilustra a Figura 4.

Figura 4: Dispositivo *Raspberry Pi* dentro de uma caixa e conectado ao multímetro, para realização dos experimentos com temperatura e energia.



Fonte: Criado pelo Autor.

3.2 TESTE COM O MODELO DE TAREFAS

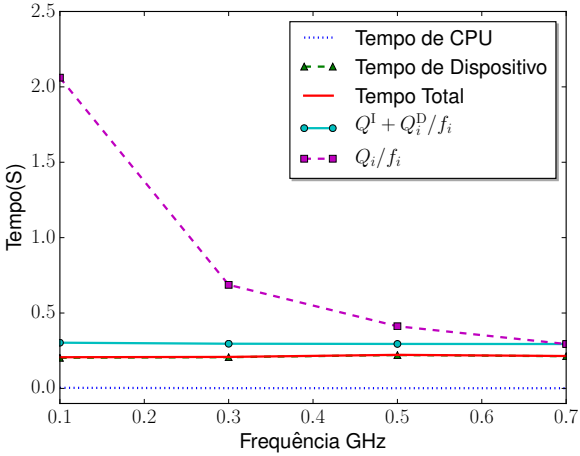
O primeiro experimento a ser apresentado será relacionado ao modelo de tarefa. O objetivo é mostrar que o modelo de tarefa utilizado é representativo para o problema abordado. Para validar a Eq. (2.16) foram realizados quatro experimentos com três tarefas diferentes e variando a frequência do processador.

No primeiro experimento, foi verificado o comportamento do modelo para uma tarefa que utiliza a maior parte do seu tempo total de execução acessando dispositivos. A tarefa consistia em abrir um arquivo na *pendrive* e enviar uma mensagem para outra máquina, em uma rede dedicada, e receber uma confirmação de recebimento. O tempo utilizado pela tarefa para acessar os dispositivos foi definido como a diferença do tempo total de execução e o tempo de uso da *CPU* pelo programa, dados fornecidos pelo *Kernel Linux*. Após executar algumas vezes o programa no *Raspberry Pi* variando a frequência da *CPU* entre 0,2GHz à 0,7GHz, foi obtida a média dos tempos de execução da tarefa. A média em função da frequência do processador é apresentada na Figura 5.

Na Figura 5 pode-se perceber que o tempo de *CPU*, representado pela reta pontilhada, fica muito próximo de zero devido à baixa utilização da *CPU* pela aplicação. Por outro lado, o tempo de execução usado pelos dispositivos é quase igual ao tempo total de execução, representado pela linha tracejada com triângulos. Para evidenciar a diferença entre os modelos, foi utilizado o pior tempo de execução de uma tarefa com 700MHz e calculado com as frequências de 0,5, 0,3 e 0,1 GHz. Como resultado é possível notar que o modelo com tempo de execução dos dispositivos, ilustrado pela reta tracejada com círculos, é menos pessimista que considera tudo como tempo de *CPU*, que é representada pela curva tracejada com quadrados.

No segundo experimento foi criada uma tarefa que executa apenas operações matemáticas. Após executá-la 100 vezes no *Raspberry Pi* para as frequências de 0,1GHz, 0,2GHz, 0,3GHz, 0,5GHz e 0,7GHz, os resultados estão apresentados na Figura 6. Foi constatado que o tempo total de execução, curva contínua, é muito próximo do tempo utilizado pela *CPU*, curva tracejada. A diferença entre o tempo previsto pelos modelos é pequena (menos de 1%), sendo o modelo que considera dispositivos menos pessimista.

Figura 5: Comportamento de uma aplicação que tem o tempo de acesso a dispositivos muito maior que o tempo uso do processador, ao aumentar a frequência do processador.



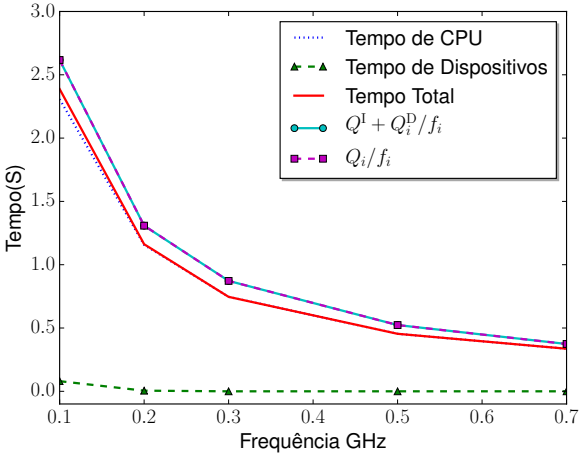
Fonte: Criado pelo Autor.

Um terceiro teste foi realizado para verificar o comportamento da aplicação *Susan* de acordo com a variação do parâmetro t (limiar do brilho). Essa aplicação faz parte do *MiBench* (GUTHAUS et al., 2001), que é uma ferramenta utilizada para fazer testes de *benchmark* em sistemas embarcados. Durante esse experimento foi utilizada uma figura com tamanho de $7KB$, o parâmetro t variou entre os números inteiros de 2 a 20, o processador funcionou com a frequência de $700MHz$ e foi utilizada a função de detectar cantos. A Figura 7 apresenta o tempo médio de utilização da *CPU* pelo *Susan*, de acordo com a variação do t. Os resultados indicam que ao mudar o valor de t entre 18 e 20 o tempo de execução é reduzido de forma linear. Assim como nos intervalos de 14 a 18 e 2 a 14. Pode-se dizer que no intervalo de 2 a 14 o tempo varia de forma linear e a pequenos passos, uma vez que do 2 a 14 o tempo de execução varia 8 ms e existem 6 possíveis parâmetros.

Ainda com *Susan*, foi realizado um último experimento executando a referida tarefa diversas vezes para o processamento gráfico de uma figura de $1MB$, com o objetivo de detectar as bordas. A tarefa foi executada no *Raspbery Pi* operando com uma frequência de $700MHz$ e $900MHz$. O pior caso de execução da tarefa com $900MHz$ foi de 3,09s (1,75s de *CPU* e 1,34s de dispositivos). O modelo de tarefa que considera todo tempo de execução da tarefa como tempo de *CPU* define o pior caso em $700MHz$ como 3,98s, enquanto o modelo usado nesse trabalho definiu 3,59s. Após executar o *Susan* com 700 MHz, o pior caso encontrado foi de 3,43s.

Com base nesses experimentos é possível concluir que o modelo de tarefas adotado nesse trabalho pode se adaptar e representar melhor o tempo de execução utilizado por diferentes tipos de tarefa, com menor pessimismo que o modelo nor-

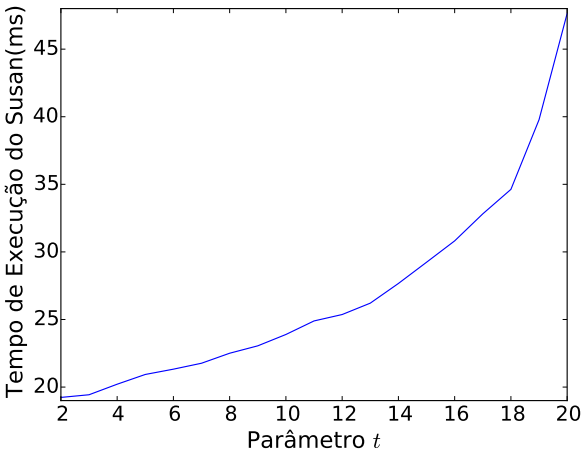
Figura 6: Comportamento de uma aplicação onde a maior parte do tempo de utilização é consumido pela *CPU*, ao aumentar a frequência do processador.



Fonte: Criado pelo Autor.

malmente utilizado.

Figura 7: Comportamento da aplicação *Susan* à medida em que parâmetro t aumenta.



Fonte: Criado pelo Autor.

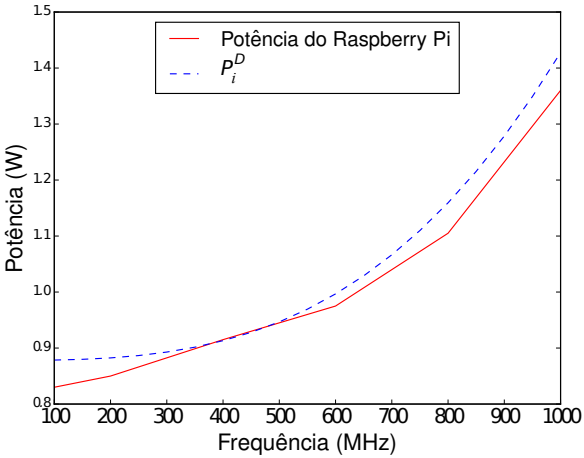
3.3 CURVA DE ENERGIA

Na literatura técnica a equação (2.5) é muito utilizada para representar a tendência do consumo energético da *CPU*, de acordo com a troca de voltagem e frequência. Para verificar a diferença entre o consumo apontado pela equação e um dispositivo real, foi realizado um experimento no qual a potência do *Raspberry Pi* foi medida com diferentes frequências e voltagens, para isso foi utilizado um multímetro. O sistema executou uma tarefa que consistia de um laço infinito, com 100% de utilização do processador. Para reduzir a interferência do consumo energético dos outros dispositivos, como a saída de vídeo e a placa de rede, foram desabilitados.

O consumo energético do *Raspberry Pi* foi medido de acordo com a variação da frequência e voltagem entre $100MHz$ e $1GHz$, em intervalos de $100MHz$. Como não era possível medir somente a potência da *CPU* separada do P^{I} (consumo energético da memória, leitor de cartão SD e placa de som), além da falta de informações necessárias para determinar C_0, foi utilizado o processo de interpolação polinomial para defini-los. Uma vez que o modelo é linear nos parâmetros a síntese deles pode ser realizada resolvendo um problema de mínimos quadrados linear, que determinou o valor de P^{I}=0,878W e C_0=0,5498. Com isso foi traçada uma linha pontilhada que representa o consumo previsto pela equação e uma linha contínua com o valor real, conforme ilustra a Figura 8. O erro máximo entre a formulação proposta e o consumo verificado foi de 5,48%, quando o processador está funcionando com $100MHz$.

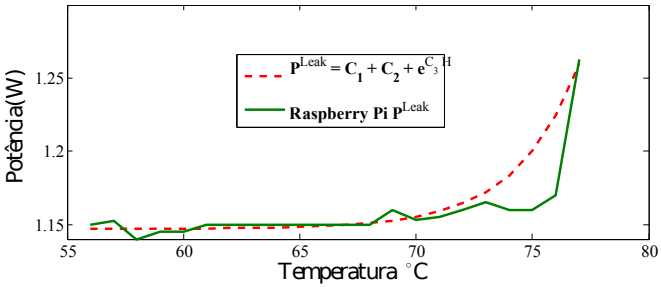
Ao analisar o consumo de energia de acordo com a troca de voltagem e frequência, é possível notar na Figura 8 que nos intervalos de $1GHz$ para $800MHz$, de

Figura 8: Consumo energético da *CPU* do *Raspberry Pi* (linha contínua) e o consumo P_i^{D} previsto pela formulação usada nesse trabalho (linha pontilhada).



Fonte: Criado pelo Autor.

Figura 9: Comportamento do consumo energético quando a temperatura do processador aumenta. Tal variação ocorre devido ao crescimento da corrente de fuga.



Fonte: Criado pelo Autor.

$800MHz$ para $600MHz$ e de $600MHz$ a $200MHz$ que a redução do consumo é linear. Isso acontece pois a troca de voltagem é discreta, fazendo com que a execução de uma tarefa com frequências entre $700MHz$ e $800MHz$ não altere o consumo energético, como mostrado na seção 2.5.

3.4 INFLUÊNCIA DA TEMPERATURA

A temperatura é um fator que pode impactar tanto no desempenho quanto no consumo energético do sistema. O aumento excessivo da temperatura danifica de forma permanente o processador, por isso muitos processadores têm dois mecanismos de proteção que reduzem a sua performance. O mecanismo mais empregado é o *throttling*, seguido do mecanismo que desliga o processador caso o *throttling* não consiga diminuir a temperatura. O aumento da temperatura também aumenta o consumo energético devido ao aumento da corrente de fuga. De acordo com (FLAUTNER; FLYNN; RIVES, 2003), P^{Leak} pode aumentar em até 14% quando a temperatura do processador atinge $85°C$. Para verificar a influência da temperatura e do *throttling* em um *Raspberry Pi*, dois testes foram realizados.

O primeiro teste visa verificar o comportamento do consumo de energia com a variação da temperatura. Para realizar os testes, uma tarefa que utiliza 100% de CPU foi executada. Para aumentar a temperatura do processador foi aplicada indiretamente uma fonte de calor para aquecer a caixa dentro da qual se encontrava o processador.

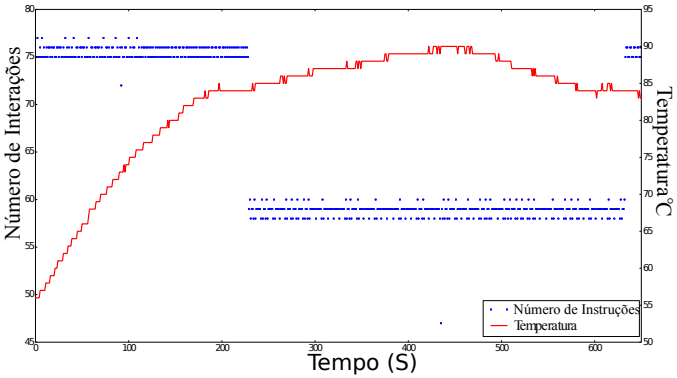
O resultado do teste foi colocado no gráfico da Figura 9 e comparado a Eq. (2.6). Os valores de C_1, C_2 e C_3 foram determinados da mesma forma que os valores de C_0 e P^{I} do teste anterior. Com o resultado é possível verificar que a Eq. (2.6) descreve o comportamento de P^{Leak} de forma satisfatória. Entretanto, por não ser calculado diretamente com nenhuma das variáveis de decisão do problema (frequência e tempo de execução da tarefa) e devido ao erro existente entre a equação sugerida e o valor verificado, a melhor solução para evitar erros na definição do P^{Leak} é medi-lo *off-line* e criar uma tabela com seus valores.

Para avaliar o efeito da temperatura sobre o desempenho do *Raspberry Pi*, foi feito um teste usando a mesma estrutura do experimento anterior que evita a troca de temperatura entre a *CPU* e o ambiente. Um programa executou um laço que a cada 1.000.000 iterações salva o tempo e a temperatura do processador. Com as informações do programa e do multímetro foi possível fazer dois gráficos. Um relacionando desempenho com temperatura e outro energia com temperatura.

Na Figura 10 está o gráfico que relaciona o desempenho do processador com a temperatura. É possível observar que aproximadamente no instante $230s$ a temperatura da *CPU* atinge $85°C$ e o número de interações por segundo, representado pelos círculos, cai drasticamente do intervalo de 75 a 77 para o de 56 a 60 interações por segundo. Isso ocorre pois aos $85°C$ o processador ativa o *throttling* para resfriar o processador. Depois do instante $600s$, quando a temperatura fica estabilizada abaixo de $85°C$, o número de interações volta a ficar acima de 75.

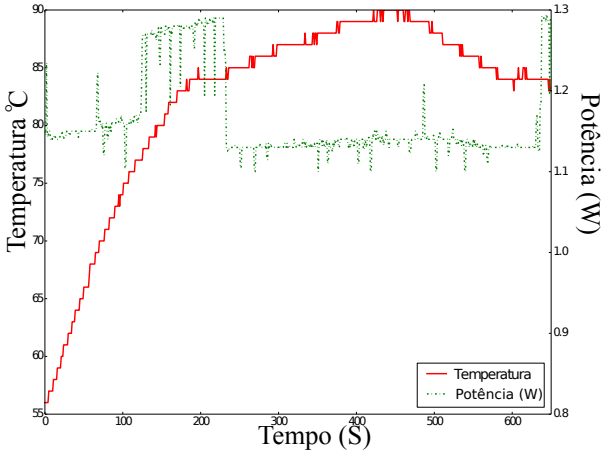
O segundo gráfico gerado pelo experimento está na Figura 11. Nela é possível perceber que do instante $0s$ até aproximadamente $120s$ o consumo energético cresce de forma linear, a uma taxa pequena. Entretanto, ao atingir a temperatura de $78°C$, a potência utilizada pelo sistema passa de um valor abaixo de $1{,}2W$ para um valor próximo de $1{,}3W$ devido ao P^{Leak}, já evidenciado no experimento da Figura 9. Ao atingir $85°C$ o consumo energético cai para um valor próximo de $1{,}1W$, enquanto o *throttling* está ativado. Fica então evidenciado que, durante o *throttling*, o número de interações por segundo resolvidas pelo processador diminui

Figura 10: Comportamento do números de interações de acordo com a temperatura, os pontos representam o número de interações realizadas por segundo pelo processador e a linha sólida ilustra a temperatura processador.



Fonte: Criado pelo Autor.

Figura 11: Comportamento do consumo energético de acordo com a temperatura, a linha pontilhada mostra a potência consumida pelo processador e a linha sólida ilustra a temperatura processador.



Fonte: Criado pelo Autor.

e a potência consumida é reduzida, uma vez que o processador está trabalhando na menor frequência possível.

Como resultado desses experimentos fica comprovado que apesar do *throttling* entrar em funcionamento aos $85°C$, o consumo energético apresenta um aumento significativo para manter o processador funcionando sem aumentar a performance ao atingir a temperatura de $78°C$. Com isso fica estabelecido que a nova temperatura máxima desejada para o *Raspberry Pi* é de $76°C$, um pouco antes do P^{Leak} subir drasticamente.

3.5 SUMÁRIO

Este capítulo apresentou experimentos para verificar a precisão das formulações existentes na literatura técnica, o impacto da temperatura no consumo energético e performance no *Raspberry Pi*. Com o resultado é possível afirmar que as equações existentes são capazes de fornecer um limite superior para o consumo de energia, com um erro pequeno e conhecido. Também foi constatado que além dos modelos inteiros e convexos encontrados na literatura, é possível usar um modelo linear por partes para representar o problema.

4 MODELO INTEIRO

Este capítulo apresenta uma abordagem de programação inteira para um modelo discreto de reconfiguração dinâmica de tarefas com restrição de energia e utilização. O modelo pode ser visto como um problema da mochila com duas dimensões, uma dimensão seria a restrição de utilização de processador e a outra a energia disponível. Uma instância exemplo foi criada para ilustrar a abordagem de programação inteira.

4.1 FORMULAÇÃO DO PROBLEMA DE RECONFIGURAÇÃO DINÂMICA

O modelo discreto de reconfiguração de tarefas sujeito a restrições de escalonabilidade e disponibilidade de energia a ser usado nesse trabalho é baseado no modelo proposto em (NASSIFFE; CAMPONOGARA; LIMA, 2012). Foi assumido que o processador pode operar em uma frequência f de um conjunto finito de frequências $\mathcal{F} = \{f_1, \ldots, f_m\}$, com m sendo o número de frequências possíveis do processador. Há um conjunto $\mathcal{S} = \{S_1, \ldots, S_n\}$ 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 $(Q_i^{k,\mathrm{I}}, Q_i^{k,\mathrm{D}}, P_i^{k,\mathrm{I}}, T_i^k)$, sendo o problema expresso como:

$$
\begin{aligned}
P^{\mathrm{INT}}: \quad \max \;\; & g = \sum_{S_i \in \mathcal{S}} \sum_{(k,j) \in \Omega_i} G_i^{k,j} x_i^{k,j} && \text{(4.1a)} \\
\text{s.a}: \;\; & \sum_{(k,j) \in \Omega_i} x_i^{k,j} = 1, \; \forall S_i \in \mathcal{S} && \text{(4.1b)} \\
& \sum_{S_i \in \mathcal{S}} \sum_{(k,j) \in \Omega_i} u_i^{k,j} x_i^{k,j} \leq 1 && \text{(4.1c)} \\
& \sum_{S_i \in \mathcal{S}} \sum_{(k,j) \in \Omega_i} P_i^{k,j} x_i^{k,j} \leq \beta P^{\star} && \text{(4.1d)} \\
& x_i^{k,j} \in \{0;1\}, S_i \in \mathcal{S}, (k,j) \in \Omega_i && \text{(4.1e)}
\end{aligned}
$$

onde:

- $\Omega_i = \{(k,j) : k \in K_i, j \in F\}$ é o conjunto de configurações para o servidor S_i;
- $x_i^{k,j}$ assume o valor 1 se o servidor S_i for operar no modo k e com frequência de processador f_j;
- a equação (4.1b) garante que cada servidor S_i irá executar em exatamente uma configuração (k,j);
- $F = \{1, 2, 3, \ldots, m\}$ é o conjunto de índices das frequências do processador, sendo que $j \in F$ se e somente se a frequência $f_j \in \mathcal{F}$ que é o conjunto de frequências do processador;
- $u_i^{k,j}$ é a utilização do processador pelo servidor S_i quando no modo k e frequência f_j, sendo definida por:

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

- a equação (4.1c) limita o uso da *CPU* de forma a garantir escalonabilidade segundo a política de escalonamento *EDF*;
- a equação (4.1d) limita a energia consumida por cada servidor S_i, quando no modo k e frequência f_j, usando uma modelagem *DPM* definida por:

$$P_i^{k,j} = (P_i^{k,\mathrm{I}} + C_0 f_j + P^{\mathrm{Sleep}} + P^{\mathrm{Leak+P^{Fan}}})u_i^{k,j}$$

onde $P_i^{k,\mathrm{I}}$ é a energia consumida por dispositivos que não dependem da frequência do processador e $C_0 f_j$ é a energia consumida pelo processador. Sendo $P^\star = \sum_{S_i \in \mathcal{S}} \max\{P_i^{k,j} : (k,j) \in \Omega_i\}$ e $\beta \in (0,1)$ o parâmetro que define a energia disponível por unidade de tempo.

Como discutido na seção 2.9, foi desenvolvida uma função objetivo para o modelo inteiro, que prioriza as configurações de maior qualidade, considera o peso da tarefa e a sua importância para o sistema. Assim sendo, foi criada a função benefício $G_i^{k,j}(W_i, f_j, u_i^{k,j}, U_i)$, onde W_i é a importância da tarefa para o sistema, f_j é a frequência usada, $u_i^{k,j}$ é a utilização da configuração em análise e U_i é a utilização do modo de maior qualidade e frequência. 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:

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

Essa forma de calcular o benefício implica que o modo de maior qualidade sempre terá mais benefício e que ao diminuir a frequência e manter o modo, há uma redução do benefício da configuração. Por isso, para encontrar a melhor configuração para o sistema, deve-se maximizar $\sum_{S_i \in \mathcal{S}} G(W_i, f_j, u_i^{k,j}, U_i)$, sendo $\mathcal{S}$ um conjunto de servidores.

Motivado pelo alto custo computacional do algoritmo de programação dinâmica, que pode chegar a $10s$, e *solver* o *CPLEX*, que consome de $100ms - 230ms$, duas heurísticas foram adaptadas para resolver esse tipo de problema: ***HDG*** (NASSIFFE; CAMPONOGARA; LIMA, 2012) e ***MV-Pack*** (RUSU; MELHEM; MOSSÉ, 2003b). A primeira heurística segue uma estratégia gulosa aplicada a um problema *surrogate*, o que combina as restrições usando multiplicadores de Lagrange baseada em (MARTELLO; TOTH, 2003). A segunda também tem um comportamento guloso, que escalona as tarefas com a pior configuração possível e depois vai aumentando o modo e a frequência das tarefas de acordo com os recursos disponíveis.

A heurística apresentada em (NASSIFFE; CAMPONOGARA; LIMA, 2012) é baseada na densidade das configurações das tarefas e usa as relaxações *surrogate* e Lagrangeana do problema P^{INT}. A relaxação é

um novo modelo, onde todas as soluções do modelo inicial são válidas, como detalhado no Anexo A.6.

A relaxação *surrogate* é obtida pela combinação de duas restrições (energia e utilização) em uma única, por meio dos multiplicadores *surrogate* (GLOVER, 1968). Então, dado o vetor $\boldsymbol{\tau} = (\boldsymbol{\tau_u}, \boldsymbol{\tau_p}) \geq 0$, a função *surrogate* dual f_{S} do problema inicial é:

$$SP^{\mathrm{INT}}(\boldsymbol{\tau}) : g_{\mathrm{S}}(\boldsymbol{\tau}) = \max \sum_{(k,j)\in\Omega_i} G_i^{k,j} x_i^{k,j} \ \forall S_i \in \mathcal{S} \tag{4.2a}$$

$$\text{s.a} : \sum_{(k,j)\in\Omega_i} x_i^{k,j} = 1, \ \forall S_i \in \mathcal{S} \tag{4.2b}$$

$$\sum_{(k,j)\in\Omega_i} (\boldsymbol{\tau}_{\mathrm{u}} u_i^{k,j} + \boldsymbol{\tau}_{\mathrm{p}} P_i^{k,j}) x_i^{k,j} \leq \boldsymbol{\tau}_{\mathrm{u}} + \boldsymbol{\tau}_{\mathrm{p}} \beta P^{\star} \ \forall S_i \in \mathcal{S} \tag{4.2c}$$

$$x_i^{k,j} \in \{0,1\}, (i,k,j) \in \Omega \tag{4.2d}$$

Uma vez que $g_{\mathrm{S}}(\boldsymbol{\tau}) \geq g$ para qualquer $\boldsymbol{\tau} \geq \mathbf{0}$, a função dual do *surrogate* gera um limite superior para o problema de reconfiguração. O dual *surrogate* expressa a vontade de minimizar o limite superior do *surrogate*, sendo definida como:

$$SD : \ g_{\mathrm{S}}(\boldsymbol{\tau}^{\star}) = \min_{\boldsymbol{\tau}\geq\mathbf{0}} f_{\mathrm{S}}(\boldsymbol{\tau}) \tag{4.3}$$

Resolver a função dual do *surrogate* $g_{\mathrm{S}}(\boldsymbol{\lambda})$ pode ser computacionalmente pesado, visto que ela generaliza o problema da mochila. Por isso foi utilizada uma relaxação Lagrangeana, a qual pode computar a solução de forma analítica.

A relaxação Lagrangeana do problema inicial pode ser definida com dois multiplicadores Lagrangeanos, um para a restrição de escalonabilidade e outro para a de consumo de energia, $\boldsymbol{\lambda}_{\mathrm{u}}$ e $\boldsymbol{\lambda}_{\mathrm{p}}$ respectivamente. A função dual Lagrangeana g_{L} para o problema de reconfiguração é definida como segue:

$$LP^{\mathrm{INT}}(\boldsymbol{\lambda}) : \quad g_{\mathrm{L}}(\boldsymbol{\lambda}) = \max \sum_{(i,k,j)\in\Omega} (G_i^{k,j} - \boldsymbol{\lambda}_{\mathrm{u}} u_i^{k,j} - \boldsymbol{\lambda}_{\mathrm{p}} P_i^{k,j}) x_i^{k,j} + \boldsymbol{\lambda}_{\mathrm{u}} + \boldsymbol{\lambda}_{\mathrm{p}} \beta P^{\star} \tag{4.4a}$$

$$\text{s.a} : \sum_{(k,j)\in\Omega_i} x_i^{k,j} = 1, \ S_i \in \mathcal{S} \tag{4.4b}$$

$$x_i^{k,j} \in \{0,1\}, \ (i,k,j) \in \Omega \tag{4.4c}$$

onde $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_{\mathrm{u}}, \boldsymbol{\lambda}_{\mathrm{p}})$. Para qualquer $\boldsymbol{\lambda} \geq \mathbf{0}$, a função Lagrangeana dual induz um limite superior $g_{\mathrm{L}}(\boldsymbol{\lambda}) \geq g^{\star}$. Esta função dual pode ser computada de forma analítica, como mostram os dois passos a seguir:

1. define-se $(\hat{k}, \hat{j})(i) = \arg\max\{(A_i^{k,j} - \boldsymbol{\lambda}_{\mathrm{u}} u_i^{k,j} - \boldsymbol{\lambda}_{\mathrm{p}} P_i^{k,j}) : (k,j) \in \Omega_i\}$.

2. a solução $\mathbf{x}(\boldsymbol{\lambda})$ de $LP^{\mathrm{INT}}(\boldsymbol{\lambda})$ é obtida definindo, para todo $S_i \in \mathcal{S}$, $x_i^{k,j}(\boldsymbol{\lambda}) = 1$ se $(k,j) = (\hat{k}, \hat{j})(i)$ e caso contrário $x_i^{k,j}(\boldsymbol{\lambda}) = 0$ para todo $(k,j) \in \Omega_i$.

Ao resolver o problema Lagrangeano, procura-se encontrar um valor para o vetor $\boldsymbol{\lambda}^{\star}$ que minimize o limite superior f_{L} (FISHER, 2004). Formalmente, o Lagrangeano dual é escrito como:

$$LD : g_{\mathrm{L}}(\boldsymbol{\lambda}^{\star}) = \min_{\boldsymbol{\lambda} \geq \mathbf{0}} f_{\mathrm{L}}(\boldsymbol{\lambda}) \tag{4.5}$$

Com isso, a heurística adaptada em (NASSIFFE; CAMPONOGARA; LIMA, 2012) resolve o problema Lagrangeano, definido os valores para o vetor $\boldsymbol{\lambda}$ e então resolve o problema $SP^{\mathrm{INT}}(\boldsymbol{\tau})$ com $\boldsymbol{\tau} = \boldsymbol{\lambda}$ segundo uma estratégia gulosa.

O primeiro passo para aplicar heurística é reescrever o problema em uma forma equivalente, usando a ideia de densidade, como segue:

$$P^{\mathrm{INT}}(\mathcal{X}) : \quad g = \max \sum_{(i,k,j) \in \Omega(\mathcal{X})} \Delta G_i^{k,j} x_i^{k,j} + \sum_{S_i \in \mathcal{S}} G_i^{(k,j)_{\mathcal{X},i}} \tag{4.6a}$$

$$\text{s.a} : \sum_{(k,j) \in \Omega_i(\mathcal{X})} x_i^{k,j} \leq 1, \; S_i \in \mathcal{S} \tag{4.6b}$$

$$\sum_{(i,k,j) \in \Omega(\mathcal{X})} \Delta u_i^{k,j} x_i^{k,j} \leq 1 - \sum_{S_i \in \mathcal{S}} u_i^{(k,j)_{X,i}} \tag{4.6c}$$

$$\sum_{(i,k,j) \in \Omega(\mathcal{X})} \Delta P_i^{k,j} x_i^{k,j} \leq \beta P^{\star} - \sum_{S_i \in \mathcal{S}} P_i^{(k,j)_{X,i}} \tag{4.6d}$$

$$x_i^{k,j} \in \{0,1\}, (i,k,j) \in \Omega(\mathcal{X}) \tag{4.6e}$$

onde:

- $\mathcal{X}$ é um conjunto com configurações de servidores factível, considerando a menor quantidade de energia possível;
- $\Omega = \{(i,k,j) : S_i \in \mathcal{S}, (k,j) \in \Omega_i\}$;

- $\Omega(\mathcal{X}) = \Omega - \mathcal{X}$;
- $\Delta G_i^{k,j} = G_i^{k,j} - G_i^{(k',j')\mathcal{X}_i}$, sendo k' e j' a configuração inicialmente selecionada em $\mathcal{X}$ para a tarefa i;
- $\Delta u_i^{k,j} = u_i^{k,j} - u_i^{(k',j')\mathcal{X}_i}$;
- $\Delta P_i^{k,j} = P_i^{k,j} - P_i^{(k',j')\mathcal{X}_i}$.

Dado então um conjunto de configurações factível $\mathcal{X}$, a heurística adaptada em (NASSIFFE; CAMPONOGARA; LIMA, 2012) resolve problema $P^{\mathrm{INT}}(\mathcal{X})$ com os seguintes passos:

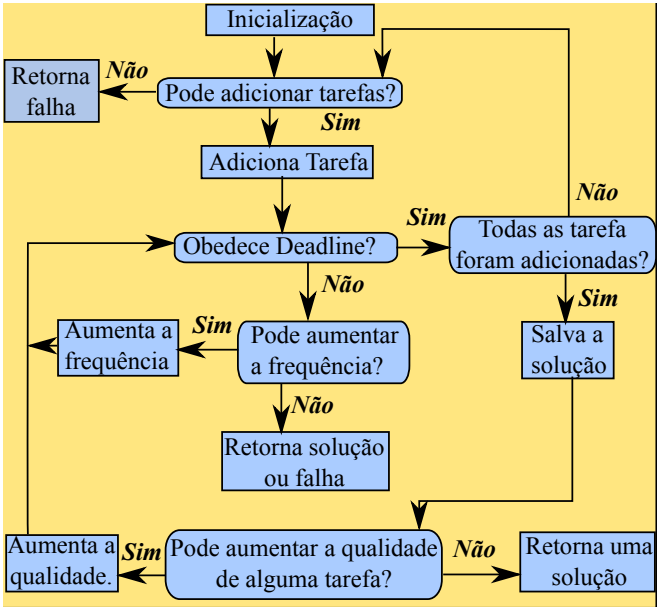
1. resolve o Lagrangeano Dual (LD) do problema P^{INT} utilizando um algoritmo subgradiente para encontrar valores aproximados para os multiplicadores Lagrangeanos $\widetilde{\boldsymbol{\lambda}} = \boldsymbol{\lambda}^{\mathrm{best}}$ e uma solução factível $\mathbf{x}^{\mathrm{best}}$; definir uma configuração $\mathcal{X}$ viável usando $\mathbf{x}^{\mathrm{best}}$ se a solução candidata for factível, caso contrário usar $\mathcal{X}$ factível previamente definida;
2. obtém $P^{\mathrm{INT}}(\mathcal{X})$ usando $\mathcal{X}$;
3. resolve de forma aproximada o problema *surrogate* $SP^{\mathrm{INT}}(\mathcal{X}, \boldsymbol{\tau})$ associado com $P^{\mathrm{INT}}(\mathcal{X})$ usando uma estratégia gulosa. Tal estratégia pode ser dividida nos seguintes passos:
   (a) $\boldsymbol{\tau} = \widetilde{\boldsymbol{\lambda}}$;
   (b) ordena as configurações de forma decrescente pela razão do benefício pela utilização da CPU e energia consumida ponderada pelos multiplicadores de Lagrange; em notação matemática esta razão é dada por:

$$\Delta \widetilde{G}_i^{k,j} = \frac{\Delta G_i^{k,j}}{\widetilde{\boldsymbol{\lambda}}_{\mathrm{u}} \Delta u_i^{k,j} + \widetilde{\boldsymbol{\lambda}}_{\mathrm{p}} \Delta P_i^{k,j}}$$

   (c) examina as configurações segundo a ordem, mudando caso alguma outra configuração melhore o benefício global e continue obedecendo às restrições.

Essa heurística apresentada acima é baseada em densidade e tem uma complexidade polinomial(NASSIFFE; CAMPONOGARA; LIMA, 2012) e é chamada de ***HDG***.

Figura 12: Fluxograma do ***MV-Pack***.



Fonte: Criado por Rusu; Melhem; Mossé (2003)

Outra heurística foi desenvolvida em (RUSU; MELHEM; MOSSÉ, 2003b), é chamada de ***MV-Pack***. Suas principais funções são: adicionar tarefa, aumentar frequência e aumentar qualidade. O fluxograma dessa heurística é ilustrado na Figura 12.

O sistema começa sem nenhuma tarefa escalonada. O primeiro passo é adicionar uma tarefa usando a frequência mais baixa e a pior qualidade possível. Assume-se que $G_i^{k,j} < G_i^{k+1,j}$, para todo $k < k(i)$ e j. Inicialmente a frequência só deve aumentar quando nenhuma tarefa pode ser escalonada. Ao modificar a frequência deve-se verificar se a restrição de energia continua sendo obedecida.

Após escalonar todas as tarefas, a solução com qualidade mínima é determinada, caso contrário não é possível escalonar o sistema e é retornado falha. Então, a partir da solução encontrada o sistema começa a aumentar a qualidade das tarefas obedecendo os seguintes critérios:

- se a configuração selecionada para $x_i^{k,j}$ não é a qualidade máxima;
- trocar a configuração atual por uma de maior qualidade com a

frequência mínima se ela não violar a restrição de energia com a frequência mínima;

- a tarefa que terá a qualidade de execução aumentada é a que apresentar a maior recompensa calculada da seguinte forma:

$$\frac{G_i^{k,j}}{u_i^{k,j} P_i^{k,j}}$$

Embora o problema formulado seja NP-Difícil, as análises experimentais a serem realizadas no final deste capítulo mostram que as heurísticas alcançam soluções de boa qualidade e com baixo custo computacional em relação à programação dinâmica e ao solver *CPLEX*. Entretanto a heurísticas não garantem soluções ótimas. Para facilitar a aplicação de uma algoritmo de plano de corte, no Apêndice B é apresentada uma reformulação do problema. A aplicação dos planos de corte a essa formulação poderá diminuir o custo computacional do *CPLEX*.

4.2 ANÁLISE NUMÉRICA DAS HEURÍSTICAS

Com intuito de comparar e verificar o comportamento das heurísticas citadas na resolução do modelo proposto, começando de uma solução infactível, foram realizadas duas análises numéricas. A primeira procura mensurar a performance, pela medição do tempo utilizado de *CPU* para resolver instâncias do problema. A segunda avalia qualidade da solução obtida, comparando com o resultado do CPLEX.

4.2.1 Condições dos Experimentos

Para avaliar as heurísticas citadas foram usadas quinhentas instâncias de conjuntos com 10, 20 e 100 tarefas. Elas foram geradas com o algoritmo *UUniFast* (BINI; BUTTAZZO, 2005). Cada instância tem utilização do sistema igual a 80%, sendo 20% tempo de execução utilizado por outros dispositivos e 60% utilizado pela *CPU*.

Os experimentos foram executados em um dispositivo *Raspberry Pi*, modelo B, com a frequência de $1GHz$. Os períodos das tarefas consideradas eram múltiplos de cinco e o consumo energético máximo encontrado de uma instância foi de aproximadamente $0,999W$. Os parâmetros de energia utilizados pelas instâncias foram: $C_0 = 0,5498$, $P^{\text{Leak}} = 0,34$, $\mathcal{F} = \{1,0; 0,8; 0,6; 0,5; 0,4; 0,3\}$, P^{I} variando de $0,2W$ até $0,01W$, $P^{\star} = 1W$ e β variando entre 1 e 0,7.

Foi considerado que ambas heurísticas partiam de uma solução infactível. Por isso, o conjunto inicial $\mathcal{X}$ do ***HDG*** foi definido pela resolução da relaxação Lagrangeana Dual, que também fornece os multiplicadores $\boldsymbol{\lambda}$. Com isso, para a análise da performance foi considerado o tempo necessário para encontrar a solução inicial e melhorá-la.

Os resultados dos experimentos serão representados por gráficos do tipo *boxplot*. Tais gráficos permitem verificar o comportamento médio, valores mínimos, máximos, *outliers* (valores atípicos) e os quartis. A linha dentro da caixa representa a média dos valores. As partes abaixo e acima da caixa representam respectivamente o primeiro e o terceiro quartil, com 25% das menores amostras abaixo da média e 25% dos maiores valores acima da média, respectivamente. O traço acima da caixa é o valor máximo encontrado, o abaixo é o valor mínimo e os valores atípicos são representados pelos círculos.

4.2.2 Performance

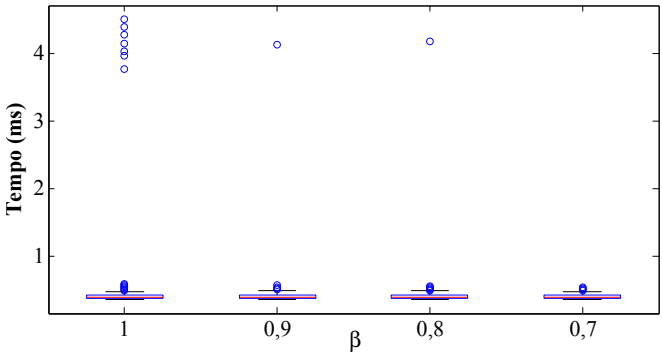
As Figuras 13, 14 e 15 mostram o tempo utilizado pela heurística baseada em densidade, para resolver o problema com diferentes números de tarefas e variações no recurso de energia disponível. Para calcular os multiplicadores de Lagrange foi utilizado um algoritmo do subgradiente. Pode-se perceber que as caixas dos gráficos estão bem achatadas, com o valor máximo próximo delas e com poucos *outliers*. Tal comportamento mostra que a variação do tempo de execução para as diferentes instâncias é baixa.

A heurística baseada em densidade resolveu as instâncias dos conjuntos de 10, 20 e 100 tarefas com $\beta = 1$ em um tempo médio de $0,46ms$, $0,93ms$ e $4,76ms$ respectivamente. Isto mostra que, quando o conjunto de tarefas passa de 10 para 20, o tempo de execução cresce em $202,17\%$ e de 10 para 100 cresce $1034,78\%$, ou seja, o tempo de execução cresceu aproximadamente na mesma proporção que o conjunto de tarefas.

Ainda nas Figuras 13, 14 e 15 não fica evidente se a disponibilidade de energia influencia no tempo de execução da heurística. Entretanto existe pequena variação, que não é contínua, como mostra a Tabela 2, demonstrando que não existe uma relação direta com a quantidade de energia disponível e o tempo de execução.

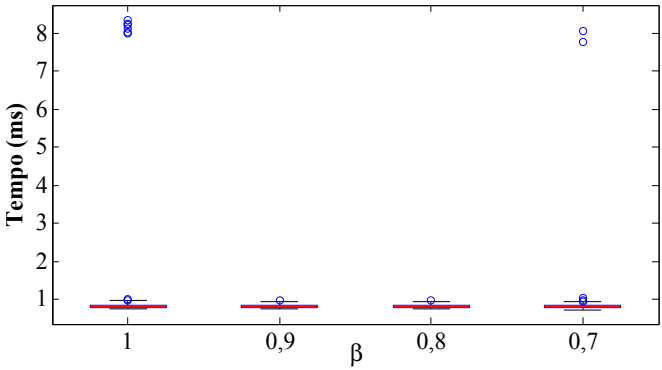
Nas Figuras 16, 17 e 18 é mostrada a performance do ***MV-Pack*** para resolver os mesmos problemas que a heurística baseada em densidade resolveu. Pode ser observado que no ***MV-Pack*** as caixas dos gráficos estão bem achatadas, com o valor máximo próximo delas e com poucos *outliers*, o que implica em uma baixa variação de tempo

Figura 13: Tempo de execução da heurística baseada em densidade para os conjuntos com 10 tarefas.



Fonte: Criado pelo Autor.

Figura 14: Tempo de execução da heurística baseada em densidade para os conjuntos com 20 tarefas.

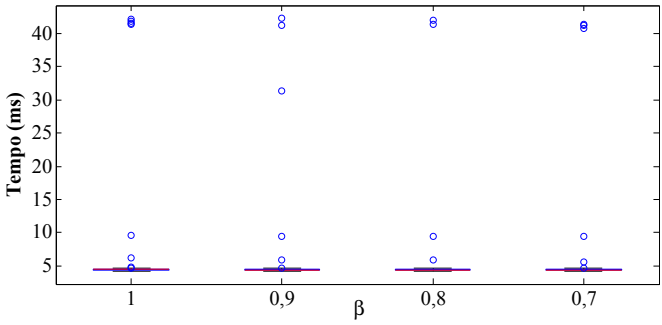


Fonte: Criado pelo Autor.

para resolver os conjuntos com mesmo número de tarefas. O tempo médio para resolver o problema com conjuntos diminui de acordo com a disponibilidade de energia, como mostra a Tabela 3

A partir dos resultados apresentados nesse experimento é possível perceber que a heurística baseada em densidade tem um custo menor de execução para o sistema. A disponibilidade de energia não influencia muito o tempo de execução, como mostrado na Tabela 2, onde a

Figura 15: Tempo de execução da heurística baseada em densidade para os conjuntos com 100 tarefas.



Fonte: Criado pelo Autor.

Tabela 2: Tempo médio de execução da heurística baseada em densidade, de acordo com β.

	β			
Tarefas	1	0,9	0,8	0,7
10	0,46ms	0,42ms	0,42ms	0,41ms
20	0,93ms	0,81ms	0,81ms	0,84ms
100	4,76ms	4,65ms	4,60ms	4,74ms

Fonte: Criado pelo Autor.

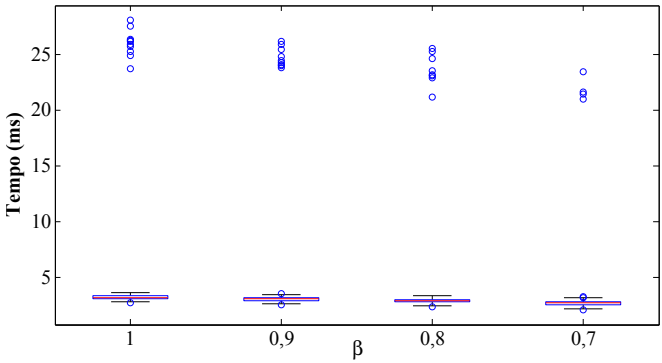
Tabela 3: Tempo médio da execução do ***MV-Pack*** de acordo com β.

	β			
Tarefas	1	0,9	0,8	0,7
10	3,65ms	3,44ms	3,23ms	2,91ms
20	6,39ms	6,10ms	5,45ms	4,76ms
100	36,98ms	32,24ms	29,14ms	25,17ms

Fonte: Criado pelo Autor.

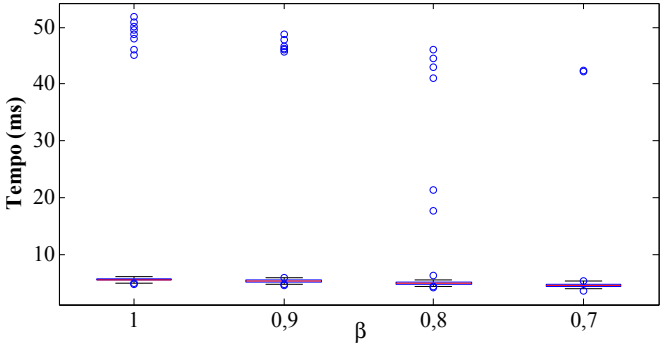
diminuição do β não necessariamente implicou em um menor tempo de execução. O ***MV-Pack***, apesar do maior custo para o sistema, diminui o seu tempo de execução de acordo com a disponibilidade de energia e pode melhorar sua performance se começar com a configuração mínima já definida.

Figura 16: Tempo de execução do *MV-Pack* para os conjuntos com 10 tarefas



Fonte: Criado pelo Autor.

Figura 17: Tempo de execução do ***MV-Pack*** para os conjuntos com 20 tarefas.

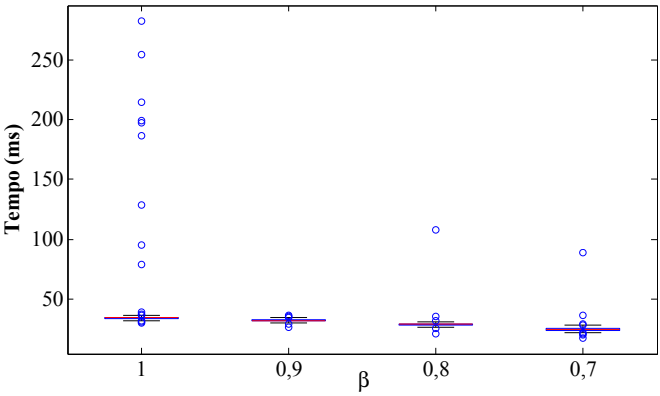


Fonte: Criado pelo Autor.

4.2.3 Qualidade da Solução

A qualidade da solução é definida de acordo com a diferença entre o *QoS* (valor da função objetivo) encontrado pelas heurísticas e o CPLEX. Essa diferença dividida pelo *QoS* encontrado pelo CPLEX foi considerada como erro e mostrada nas figuras a seguir. Com isso pretende-se descobrir qual heurística consegue fornecer a melhor solução.

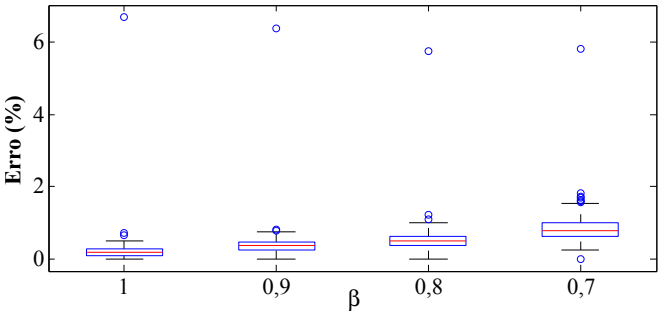
Figura 18: Tempo de execução do ***MV-Pack*** para os conjuntos com 100 tarefas.



Fonte: Criado pelo Autor.

A heurística baseada em densidade apresentou o mesmo comportamento para o conjunto de 10 e 100 tarefas. Por isso foi colocado na Figura 19 o erro referente ao conjunto com 100 tarefas. Pode-se perceber que o erro médio cresce de acordo com a diminuição da energia disponível.

Figura 19: Erro do QoS obtido pela ***HDG*** de acordo com as respostas do CPLEX, em função de β, para um conjunto com 100 tarefas.

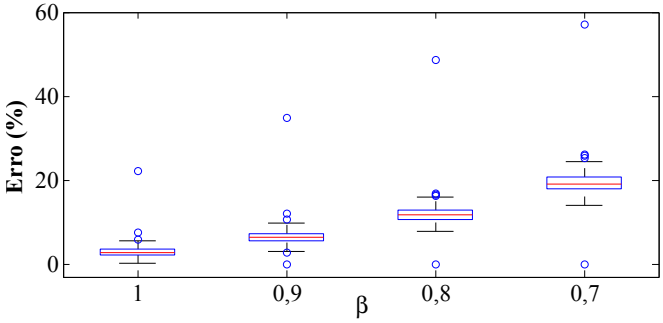


Fonte: Criado pelo Autor.

O ***MV-Pack*** apresentou um comportamento semelhante à outra heurística. Entretanto, o erro encontrado foi mais alto, chegando a

apresentar erro médio de 19,45% quando β =0,7, como mostra a Figura 20.

Figura 20: Erro do *QoS* obtido pela ***MV-Pack*** de acordo com as respostas do CPLEX, em função de β, para um conjunto com 100 tarefas.



Fonte: Criado pelo Autor.

Como resultado dos testes pode-se concluir que a heurística baseada em densidade obteve melhor *QoS* e performance ao ser comparada com o ***MV-Pack***. Era esperado que o ***MV-Pack*** apresentasse soluções melhores ou no mesmo nível de qualidade que a ***HDG***, entretanto não ocorreu. Isso pode ter sido causado pelo fato que o ***MV-Pack*** foi proposto para um modelo em que a função objetivo era baseada apenas no modo da tarefa, a frequência não importava.

4.3 SUMÁRIO

Esse capítulo mostrou um modelo inteiro para representar o problema apresentado nessa tese, acompanhado por duas heurísticas já existentes na literatura. Elas foram implementadas e tiveram o desempenho e qualidade da solução analisados em um dispositivo embarcado.

5 MODELOS CONTÍNUOS

Este capítulo apresenta três modelos contínuos para o problema de escalonamento com restrição de energia e utilização. O primeiro modelo apresentado é convexo e possui uma variável de decisão, a frequência. O segundo modelo é uma extensão do primeiro, no qual é adicionado uma segunda variável, o tempo de execução. O último modelo é linear por partes com uma variável de decisão, a frequência. Métodos para resolução dos modelos também serão mostrados e analisados.

5.1 MODELO CONVEXO COM VARIAÇÃO DE FREQUÊNCIA

Assume-se agora um sistema monoprocessado com frequência variando de forma contínua no intervalo $[f^{\min}, f^{\max}]$, que representam respectivamente a menor e maior utilização da *CPU*. Tem-se um problema contínuo, o qual consiste em definir frequências de operação para um conjunto de tarefas, garantindo que essas tarefas executem sem perder o *deadline* e economizem uma quantidade definida de energia. Esse problema matematicamente é formulado como:

$$
\begin{aligned}
P^{\text{CONT}}: \quad \min g = & \sum_{S_i \in \mathcal{S}} G_i(u_i, w_i) && \text{(5.1a)}\\
\text{s.a}: & \sum_{S_i \in \mathcal{S}} u_i \leq 1 && \text{(5.1b)}\\
& \sum_{S_i \in \mathcal{S}} P_i(f_i) u_i \leq P^{\star} && \text{(5.1c)}\\
& f_i^{\min} \leq f_i, S_i \in \mathcal{S} && \text{(5.1d)}\\
& f_i \leq f_i^{\max}, S_i \in \mathcal{S} && \text{(5.1e)}
\end{aligned}
$$

onde:

- a equação (5.1a) deve ser formulada de acordo com os aspectos já citados: degradação de qualidade, importância da tarefa para o sistema e utilização dos recursos;
- a primeira restrição (5.1b) limita a utilização do processador de acordo com a política de escalonamento *EDF*;
- a equação (5.1c) é responsável por calcular a energia consumida por cada servidor S_i, de acordo com a frequência f_i selecionada. $P^{\star}$ é a energia disponível por unidade de tempo no momento;

- por último as restrições (5.1d) e (5.1e) limitam o valor mínimo e máximo de f_i.

A equação (5.1a) é responsável por determinar o benefício atingido pelo sistema de acordo com a frequência escolhida para cada tarefa. A redução da frequência do processador implica em um tempo maior para executar uma tarefa, como mostra a equação (2.4).

Em sistemas de tempo real *soft* é considerado o tempo médio de execução de uma tarefa. Como resultado, algumas instâncias de uma tarefa podem perder o *deadline*. Sendo assim, a redução da frequência em sistemas não críticos aumenta o atraso na execução das instâncias de uma tarefa quando o tempo de execução é maior que o médio. Já no caso de um sistema de tarefas *hard*, é sempre considerado o pior caso de execução. Logo, nesse tipo de sistema não há perda de *deadline*.

O atraso na execução da instância de uma tarefa é considerado como degradação, pois não invalida o funcionamento do sistema. Como no caso de uma aplicação de vídeo que pode congelar a imagem por alguns segundos. Assim, faz-se necessário penalizar o valor do benefício agregado de acordo com a diminuição da frequência. Ainda algumas tarefas podem ser consideradas de maior importância devido às necessidades específicas do sistema, por isso só devem ser degradadas em último caso.

Para a construção de uma função objetivo que tenha o comportamento desejado, os parâmetros W_i^1 e W_i^2 foram criados, onde esses valores são constantes que representam respectivamente como a variação de f_i deve influenciar o benefício agregado e a importância semântica da tarefa para a aplicação. Ou seja, um valor negativo para W_i^1 implica que reduzir a frequência diminui o valor do QoS agregado àquela tarefa. Sendo assim, a função objetivo fica expressa como:

$$G_i(Q_i^{\mathrm{D}}, Q_i^{\mathrm{I}}, f_i, W_i^1, W_i^2) = -\sum_{S_i \in \mathcal{S}} \Big(\frac{Q_i^{\mathrm{I}} f_i W_i^1}{T_i} + \frac{Q_i^{\mathrm{D}} W_i^2}{T_i} \Big). \tag{5.2}$$

Essa função é linear e decrescente pois o termo f_i da expressão $f_i t_i$ é positivo, sendo t_i o termo constante da expressão. Como toda função linear é convexa e o somatório de funções convexas é uma outra função convexa, a função benefício é, portanto, convexa. Sendo assim, se o restante do problema for composto por equações convexas será possível aplicar métodos conhecidos e eficientes da otimização, com garantia de ótimo global.

A primeira restrição a ser verificada quanto à convexidade é a de

utilização, equação (5.1b), que pode ser vista como:

$$\sum_{S_i \in \mathcal{S}} \Big(\frac{Q_i^{\mathrm{I}}}{T_i} + \frac{Q_i^{\mathrm{D}}}{T_i f_i} \Big) \leq 1 \iff \sum_{S_i \in \mathcal{S}} \frac{Q_i^{\mathrm{D}}}{T_i f_i} \leq 1 - \sum_{S_i \in \mathcal{S}} \frac{Q_i^{\mathrm{I}}}{T_i}. \tag{5.3}$$

Para verificar a convexidade dessa restrição primeiro será analisada se a função de utilização é convexa. Depois, pela propriedade que garante que o somatório de funções convexas é uma função convexa, será provada a convexidade da restrição, que é o somatório de funções de utilização. Calculando a segunda derivada da função de utilização temos:

$$\frac{d^2 u_i}{d f_i^2} = 2 \frac{Q_i^{\mathrm{D}}}{T_i f_i^3}.$$

Logo, considerando que f_i varia no domínio $(0, 1]$ a segunda derivada é sempre positiva e a restrição (5.3) define um conjunto convexo.

Agora a mesma ideia será aplicada para verificar a convexidade da Restrição (5.1c), que pode ser vista como:

$$\begin{aligned}
&\sum_{S_i \in \mathcal{S}} \Big(C_0 f_i^3 + P_i^{\mathrm{I}} \Big) u_i + P^{\mathrm{S}} + P^{\mathrm{Leak}} \leq P^{\star} \iff \\
&\sum_{S_i \in \mathcal{S}} \Big(C_0 f_i^3 + P_i^{\mathrm{I}} \Big) \Big(\frac{Q_i^{\mathrm{D}}}{T_i f_i} + \frac{Q_i^{\mathrm{I}}}{T_i} \Big) \leq P^{\star} - P^{\mathrm{S}} - P^{\mathrm{Leak}} \iff \\
&\sum_{S_i \in \mathcal{S}} \Big(\frac{C_0 f_i^3 Q_i^{\mathrm{I}}}{T_i} + \frac{C_0 f_i^2 Q_i^{\mathrm{D}}}{T_i} + \frac{P_i^{\mathrm{I}} Q_i^{\mathrm{D}}}{T_i f_i} \Big) \\
&\qquad \leq P^{\star} - P^{\mathrm{S}} - P^{\mathrm{Leak}} - \sum_{S_i \in \mathcal{S}} \frac{P_i^{\mathrm{I}} Q_i^{\mathrm{I}}}{T_i}.
\end{aligned}$$

Calculando a segunda derivada da função de energia obtém-se:

$$\frac{d^2 u_i}{d f_i^2} = 6 C_0 f_i \frac{Q_i^{\mathrm{I}}}{T_i} + 2 C_0 \frac{Q_i^{\mathrm{D}}}{T_i} + 2 P_i^{\mathrm{I}} \frac{Q_i^{\mathrm{D}}}{T_i f_i^3}.$$

Novamente considerando o domínio de f_i entre $(0, 1]$ a segunda derivada sempre será positiva, permitindo concluir que a restrição define um conjunto convexo.

As restrições (5.1d) e (5.1e) são obviamente lineares e portanto convexas. Tendo essas informações pode-se definir uma versão equivalente de P^{CONT} em termos apenas das frequências das tarefas que é convexo:

$$P^{\text{CONT1}}: \quad \min\ g = -\sum_{S_i \in \mathcal{S}} \left(\frac{Q_i^{\text{I}} f_i W_i^1}{T_i} + \frac{Q_i^{\text{D}} W_i^2}{T_i} \right) \tag{5.5a}$$

$$\text{s.a}: \sum_{S_i \in \mathcal{S}} \frac{Q_i^{\text{D}}}{T_i f_i} \leq 1 - \sum_{S_i \in \mathcal{S}} \frac{Q_i^{\text{I}}}{T_i} \tag{5.5b}$$

$$\sum_{S_i \in \mathcal{S}} \left(C_0 f_i^2 \frac{Q_i^{\text{D}}}{T_i} + C_0 f_i^3 \frac{Q_i^{\text{I}}}{T_i} + P_i^{\text{I}} \frac{Q_i^{\text{D}}}{T_i f_i} \right) \leq P^\star - P^{\text{S}} - P^{\text{Leak}} - \sum_{S_i \in \mathcal{S}} P_i^{\text{I}} \frac{Q_i^{\text{I}}}{T_i} \tag{5.5c}$$

$$f_i^{\min} - f_i \leq 0, S_i \in \mathcal{S} \tag{5.5d}$$

$$f_i - f_i^{\max} \leq 0, S_i \in \mathcal{S} \tag{5.5e}$$

O modelo apresentado acima é capaz de selecionar uma configuração de frequência para um sistema de tempo real, sem invalidar o seu funcionamento e garantindo economia de energia. Tal modelo define o QoS pela função objetivo e aceita qualquer política de escalonamento que possa provar a escalonabilidade pela soma da utilização das tarefas. E por ser convexo é possível resolvê-lo de forma eficiente e achar a configuração maximize o QoS.

5.2 MODELO CONVEXO COM VARIAÇÃO DE FREQUÊNCIA E CARGA

Para estender o modelo anterior para um sistema multi-modal, foi assumido que cada tarefa tem um Q^{D} variando entre $Q^{\text{D,min}}$ e $Q^{\text{D,max}}$. Este intervalo representa respectivamente o menor e maior tempo de computação que uma tarefa pode ter, ficando o problema expresso

como:

$$P^{\text{CONT2}}: \quad \min f = -\sum_{S_i \in \mathcal{S}} \Big(\frac{Q_i^{\text{I}} f_i W_i^1}{T_i} + \frac{Q_i^{\text{D}} W_i^2}{T_i} \Big) \tag{5.6a}$$

$$\text{s.a}: \sum_{S_i \in \mathcal{S}} \frac{Q_i^{\text{D}}}{T_i f_i} \leq 1 - \sum_{S_i \in \mathcal{S}} \frac{Q_i^{\text{I}}}{T_i} \tag{5.6b}$$

$$\sum_{S_i \in \mathcal{S}} \Big(C_{\text{ef}} V_{\max}^2 f_i^2 F_{\max} \frac{Q_i^{\text{D}}}{T_i} + C_{\text{ef}} V_{\max}^2 f_i^3 F_{\max} \frac{Q_i^{\text{I}}}{T_i} + P_i^{\text{I}} \frac{Q_i^{\text{D}}}{T_i f_i} \Big) \leq \frac{E^\star}{T_{\text{est}}} - P^{\text{S}} - \sum_{S_i \in \mathcal{S}} P_i^{\text{I}} \frac{Q_i^{\text{I}}}{T_i} \tag{5.6c}$$

$$f_i^{\min} \leq f_i \leq f_i^{\max}, S_i \in \mathcal{S} \tag{5.6d}$$

$$Q_i^{\text{D,min}} \leq Q_i^{\text{D}} \leq Q_i^{\text{D,max}}, S_i \in \mathcal{S} \tag{5.6e}$$

Em relação ao problema anterior, somente uma nova restrição foi adicionada. A equação (5.6e), que garante a variação do valor de Q_i^{D} entre $Q_i^{\text{D,min}}$ e $Q_i^{\text{D max}}$.

Ao considerar Q_i^{D} como uma variável de decisão, o problema acima deixa de ser convexo. Por isso ele foi matematicamente reformulado, colocando as variáveis de decisão como funções exponenciais:[1]

$$\begin{cases} f_i = e^{\overline{f_i}} \\ Q_i^{\text{D}} = e^{\overline{Q_i^{\text{D}}}} \end{cases} \iff \begin{cases} \overline{f_i} = \log(f_i) \\ \overline{Q_i^{\text{D}}} = \log(Q_i^{\text{D}}) \end{cases}$$

[1] A função *log*() tem base natural, a menos que explicitamente redefinida

Ficando o problema reescrito como:

$$P^{\text{CONT2}}: \quad \min f = \sum_{S_i \in \mathcal{S}} \Big(-\log(\frac{Q_i^{\text{I}} e^{\overline{f_i}} W_i^1}{T_i}) - \log(\frac{e^{\overline{Q_i^{\text{D}}}} W_i^2}{T_i}) \Big) \tag{5.7a}$$

$$\text{s.a}: \sum_{S_i \in \mathcal{S}} \frac{e^{\overline{Q_i^{\text{D}}}}}{T_i e^{\overline{f_i}}} \leq 1 - \sum_{S_i \in \mathcal{S}} \frac{Q_i^{\text{I}}}{T_i} \tag{5.7b}$$

$$\sum_{S_i \in \mathcal{S}} \Big(C_0 e^{2\overline{f_i}} \frac{e^{\overline{Q_i^{\text{D}}}}}{T_i} + C_0 e^{3\overline{f_i}} \frac{Q_i^{\text{I}}}{T_i} + P_i^{\text{I}} \frac{e^{\overline{Q_i^{\text{D}}}}}{T_i e^{\overline{f_i}}} \Big)$$

$$\leq P^{\star} - P^{\text{S}} - P^{\text{Leak}} - \sum_{S_i \in \mathcal{S}} P_i^{\text{I}} \frac{Q_i^{\text{I}}}{T_i} \tag{5.7c}$$

$$\log(f_i^{\min}) \leq \overline{f_i} \leq \log(f_i^{\max}), S_i \in \mathcal{S} \tag{5.7d}$$

$$\log(Q_i^{\text{D,min}}) \leq \overline{Q_i^{\text{D}}} \leq \log(Q_i^{\text{D,max}}), S_i \in \mathcal{S} \tag{5.7e}$$

A função objetivo, (5.7a), do problema proposto acima pode ser vista como um somatório de funções convexas tanto em $e^{\overline{f_i}}$ e $e^{\overline{Q_i^{\text{D}}}}$. e^x é obviamente uma função convexa. Logo, a função a ser minimizada é convexa. Analisando a restrição de utilização, tem-se que:

$$\sum_{S_i \in \mathcal{S}} \Big(\frac{e^{\overline{Q_i^{\text{D}}}}}{T_i e^{\overline{f_i}}} + \frac{Q_i^{\text{I}}}{T_i} \Big) \leq 1 \tag{5.8a}$$

onde:

$$u_i = \frac{e^{\overline{Q_i^{\text{D}}} - \overline{f_i}}}{T_i} + \frac{Q_i^{\text{I}}}{T_i} \tag{5.9a}$$

logo:

$$\sum_{S_i \in \mathcal{S}} u_i \leq 1.$$

Seja $p(\mathbf{z}) = \mathbf{A}\mathbf{z}$ uma função linear em $\mathbf{z}^T = [e^{\overline{Q_i^{\text{D}}}}; e^{\overline{f_i}}]^T$. Seja $q(w) = e^w$ para um escalar w. Considere a função de composição $h = q \circ p$ onde $h(\mathbf{z}) = q(p(\mathbf{z}))$. Conforme (BOYD; VANDENBERGHE, 2004) $h(\mathbf{z})$ é uma função convexa pois $q(w)$ é uma função convexa e $p(\mathbf{z})$ é uma função afim. Logo, a função u_i é convexa e a restrição de escalonabilidade

que é formada pelo somatório de u_i também é convexa, pois a soma de funções convexas é uma função convexa.

Agora analisando restrição de energia, tem-se:

$$\sum_{S_i \in \mathcal{S}} \left(C_0 e^{2\overline{f_i}} \frac{e^{\overline{Q_i^{\mathrm{D}}}}}{T_i} + C_0 e^{3\overline{f_i}} \frac{Q_i^{\mathrm{I}}}{T_i} + P_i^{\mathrm{I}} \frac{e^{\overline{Q_i^{\mathrm{D}}}}}{T_i e^{\overline{f_i}}} \right) \leq P^{\star} - P^{\mathrm{S}} - P^{\mathrm{Leak}} - \sum_{S_i \in \mathcal{S}} P_i^{\mathrm{I}} \frac{Q_i^{\mathrm{I}}}{T_i}.$$

Removendo os termos da equação, que só contém constantes, temos que a energia gasta por cada tarefa é dada por:

$$P_i = C_0 e^{2\overline{f_i}} \frac{e^{\overline{Q_i^{\mathrm{D}}}}}{T_i} + C_0 e^{3\overline{f_i}} \frac{Q_i^{\mathrm{I}}}{T_i} + P_i^{\mathrm{I}} \frac{e^{\overline{Q_i^{\mathrm{D}}}}}{T_i e^{\overline{f_i}}}.$$

Logo:

$$\sum_{S_i \in \mathcal{S}} P_i \leq P^{\star} - P^{\mathrm{S}} - P^{\mathrm{Leak}} - \sum_{S_i \in \mathcal{S}} P_i^{\mathrm{I}} \frac{Q_i^{\mathrm{I}}}{T_i}.$$

O raciocínio utilizado para provar que a função de escalonamento é convexa pode ser aplicado para provar que P_i é convexa e, por consequência, a restrição de energia. As duas últimas restrições (5.7d) e (5.7e) são funções lineares nas variáveis de decisão $\overline{f_i}$ e $\overline{Q_i^D}$ (BOYD; VANDENBERGHE, 2004).

5.3 ALGORITMOS

Esta seção começa com a apresentação de uma instância exemplo, a qual será usada posteriormente para exemplificar dois algoritmos que podem ser aplicados ao modelo contínuo com variação de frequência. O primeiro algoritmo utiliza uma estratégia de programação quadrática sequencial, enquanto o segundo é do tipo ponto-interior. Os algoritmos são válidos para qualquer tipo de problema convexo, ou seja, eles podem resolver os dois modelos convexos propostos nesse trabalho.

5.3.1 Instância Exemplo

A instância exemplo usada para ilustrar esse trabalho foi construída com base no modelo convexo com variação de carga. Ela pressupõe uma *CPU* com as seguintes configurações: $C_0 = 24,99$, frequência máxima

de $2,53Ghz$ e mínima de $0,8Ghz$, $P^{\mathrm{S}} = 0$, $P^{\mathrm{Leak}} = 0$ e $P^{\star} = 18,26W$. Os valores de Q_i^{I} e Q_i^{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 quatro servidores, cada um responsável por uma tarefa cujos parâmetros são fornecidos na Tabela 4. Os dados apresentados a seguir consideram a frequência de 100%. Os valores das demandas computacionais são dados na Tabela 5, calculados conforme a Eq. (2.16). A energia consumida é dada na Tabela 6, obtida com base na Eq. (2.3). E por último, os maiores valores dos benefícios associados a cada tarefa aparecem na Tabela 7, calculados com o uso da Eq. (5.2).

Tabela 4: Parâmetros de configuração dos servidores.

i	Q_i^{D}	Q_i^{I}	P_i^{I}	T_i	W_i^1	W_i^2
1	$0,01350$	$0,00150$	$0,800$	$0,05$	70	20
2	$0,02025$	$0,00225$	$0,017$	$0,15$	60	30
3	$0,00540$	$0,00060$	$0,462$	$0,10$	50	10
4	$0,01800$	$0,00200$	$0,182$	$0,10$	30	10

Fonte: Criado pelo Autor.

Tabela 5: Utilização da CPU.

u_i	%
u_1	30%
u_2	15%
u_3	6%
u_4	20%

Fonte: Criado pelo Autor.

Nas subseções a seguir serão apresentadas duas soluções para o problema modelado.

Tabela 6: Energia Consumida.

Tarefas	W
1	$7,83$
2	$3,80$
3	$0,16$
4	$5,10$

Fonte: Criado pelo Autor.

Tabela 7: Benefício associado a cada tarefa.

Tarefa	Benefício
1	$7,50$
2	$5,40$
3	$0,84$
4	$2,40$

Fonte: Criado pelo Autor.

5.3.2 *Sequential Quadractic Programming(SQP)*

SQP é um método interativo para resolver problemas não lineares que possuam a função objetivo e restrições continuamente diferenciáveis. Foi utilizado o *CFSQP* (*C Code of Feasible Sequential Quadratic Programming*) para aplicação do método *SQP*. O *CFSQP* é um *framework* projetado com algoritmos eficientes para resolução de problemas com restrições não lineares, que consiste na implementação de dois algoritmos baseados em programação quadrática sequencial (*Sequential Quadratic Programming - SQP*). Um dos algoritmos, de busca linear monotônica, é uma busca em arco do tipo Armijo e o segundo faz uma busca não monotônica ao longo de uma reta. O problema é resolvido por uma série de subproblemas de aproximações quadráticas, que para problemas não convexos garante o ótimo local e para problemas convexos garante ótimo global. Os problemas resolvidos pelo *CFSQP* tem a seguinte forma:

$$\begin{aligned} P: \quad & \min\ g(\mathbf{x}) \\ & \text{s.a} : \mathbf{x} \in \mathcal{X} \end{aligned}$$

com $g : \mathbb{R}^n \to \mathbb{R}$ e a variável de decisão $\mathbf{x} \in \mathbb{R}^n$ pertence ao conjunto $\mathcal{X}$, satisfazendo as seguintes condições:

$$\begin{aligned} &\mathbf{bl} \leq \mathbf{x} \leq \mathbf{bu} \\ &c_j(\mathbf{x}) = 0, j \in \mathcal{E} \\ &c_j(\mathbf{x}) \geq 0, j \in \mathcal{I} \end{aligned}$$

onde $\mathbf{bl} \in \mathbb{R}^n$, $\mathbf{bu} \in \mathbb{R}^n$ e $\mathcal{I}$ e $\mathcal{E}$ são respectivamente dois conjuntos de índices para desigualdades e igualdades.

O problema foi implementado com uso do *CFSQP* e a instância exemplo 5.3.1 foi resolvida. O resultado obtido foi: $f_1 = 1; f_2 = 1; f_3 = 1; f_4 = 0,82$. O problema foi resolvido para economizar 10% de energia, consumindo $16,27W$. A configuração obtida realiza a economia desejada e o tempo computacional gasto foi de $0,1ms$.

5.3.3 Método de Ponto-Interior

O segundo algoritmo para resolver o problema de otimização (5.5) é o método de ponto-interior que garante convergência global para problemas convexos. Tal método pode ser visto como um *framework* para resolução de problemas convexos, onde o método de Newton é aplicado a uma sequência de aproximações quadráticas ou uma sequência modificada das restrições *KKT*(*Karush Kuhn Tucker*). O método do ponto-interior pode ser executado com mais de um algoritmo. Nesse trabalho será usado com o método de barreira. O problema a ser resolvido pode ser visto de uma forma genérica como:

$$\min\ g_0(\mathbf{x}) \tag{5.16a}$$
$$\text{s.a}:\ g_i(\mathbf{x}) \leq 0, \forall i = 1, ..., m \tag{5.16b}$$
$$A\mathbf{x} = \mathbf{b} \tag{5.16c}$$

onde a equação (5.16a) representa a função objetivo e as restrições de desigualdade são postas como (5.16b), sendo m o número de restrições do problema. As igualdades representadas por (5.16c) serão desconsideradas, pois não existem igualdades no problema proposto.

O primeiro passo para o uso do *framework* é a reescrita do problema removendo as restrições de desigualdades e colocando-as de forma implícita na função objetivo, ficando da seguinte forma:

$$\min\ g_0(\mathbf{x}) + \sum_{i=1}^{m} I\_(g_i(\mathbf{x})).$$

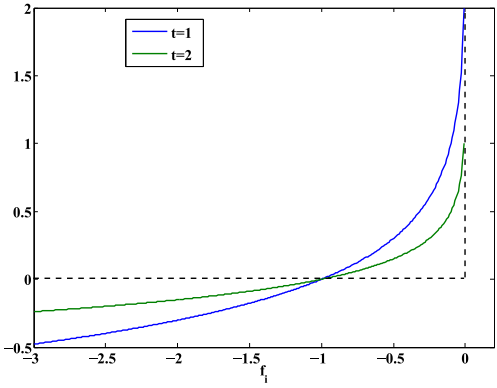
Como resultado do passo citado anteriormente, a função objetivo se torna não diferenciável, pois $I\_ : \mathbb{R} \to \mathbb{R}$ é uma função indicadora para números reais não positivos, assumindo os seguintes valores:

$$I\_(g_i(\mathbf{x})) = \begin{cases} 0 & g_i(\mathbf{x}) \leq 0 \\ \infty & g_i(\mathbf{x}) > 0 \end{cases}$$

Para tornar as equações adicionadas à função objetivo diferenciáveis, elas são reescritas de forma aproximada por uma função barreira logarítmica. Representando as restrições de forma aproximada na função objetivo com uma função barreira logarítmica, o problema P^{CONT1} é aproximado por um problema $P^{\text{CONT1}}(t)$ como segue:

$$P^{\text{CONT1}}(t): \quad \min \; g_0(\mathbf{x}) + \sum_{i=1}^{m} -\frac{1}{t}\log(-g_i(\mathbf{x})).$$

Figura 21: A linha pontilhada mostra a função $I\_(g(\mathbf{x}))$, enquanto as linhas sólidas mostram a curva da função aproximada $(-1/t)\log(-g(\mathbf{x}))$, para $t = 1, 2$. Quando $t = 2$ a função logarítmica tem a melhor aproximação.



Fonte: Criado pelo Autor.

Como pode ser visto na Figura 21, o valor de $t > 0$ influencia na qualidade da aproximação $P^{\text{CONT1}}(t)$ em relação a P^{CONT1}, quanto maior, mais precisa é a função aproximada. Seja $\mathbf{x}(t)$ a solução ótima de $P^{\text{CONT1}}(t)$. Então é possível mostrar que $\mathbf{x}(t)$ converge para o ótimo global à medida que $t \to \infty$.

Para simplificar a notação acima, toda função objetivo será multiplicada por t e o somatório dos $\log(-g_i)$, $i = 1, ..., m$, substituído pela função:

$$\phi(\mathbf{x}) = \sum_{i=1}^{m} \log(-g_i(\mathbf{x}))$$

ficando o problema reescrito como:

$$P^{\text{CONT1}}(t): \quad \min \Theta(\mathbf{x}) = tg_0(\mathbf{x}) - \phi(\mathbf{x})$$
$$\mathbf{x} \in \operatorname{dom} \Theta$$

onde $\operatorname{dom} \Theta = \{\mathbf{x} : g_i(\mathbf{x}) < 0, i = 1, ..., m\}$.

Como resultado tem-se um problema convexo e sem restrições que pode ser resolvido com o método de *Newton*. Um valor maior de t implica em uma precisão melhor da função de aproximação, entretanto a eficiência do método de *Newton* diminui por causa da Hessiana que varia rapidamente próxima da fronteira. Por isso o problema deve ser resolvido de forma sequencial, aumentando o valor de t a cada passo da minimização e reaproveitando o valor de $\mathbf{x}(t)$ encontrado no passo anterior para ser ponto inicial do próximo passo.

Cada solução $\mathbf{x}(t)$ com $t > 0$ encontrada é única e faz parte do caminho central associado ao problema, sendo definido por $\mathcal{C} = \{\mathbf{x}(t) : t > 0\}$. Cada solução $\mathbf{x}(t)$ é chamada de ponto central, sendo caracterizada por obedecer a condição $\nabla\Theta(\mathbf{x}) = \mathbf{0}$ o que, além de ser estritamente factível é necessária e suficiente para x ser um ponto ótimo, para $P^{\text{CONT1}}(t)$.

A formulação mostrada acima parte do pressuposto que se conhece um ponto inicial factível e é desejado chegar em um ponto ótimo. Para encontrar um ponto inicial factível o problema é reescrito como:

$$\min \ s \tag{5.18a}$$
$$\text{s.a} : g_i(\mathbf{x}) \leq s, i = 1, ..., m \tag{5.18b}$$

O Problema (5.18) troca a função objetivo do problema inicial por s, que pode ser visto como um limite máximo de infactibilidade das desigualdades do problema. Portanto, ao minimizar s a um valor abaixo de zero chega-se a uma solução $\mathbf{x}$ que obedece a todas as restrições do problema de forma estrita.

O problema (5.18) é sempre factível para qualquer valor para $\mathbf{x}$, bastando definir s como: $s > \max_i\{g_i(\mathbf{x}) : i = 1, ..., m\}$. Para resolver esse problema e encontrar o ponto inicial, pode-se aplicar o método da barreira descrito nessa seção, uma vez que qualquer valor de $\mathbf{x}$ pode ser utilizado como ponto inicial.

Para aplicar o método do ponto-interior ao problema P^{CONT1}, apresentado em (5.5), a função objetivo e as restrições foram reescritas como:

$$g_0 = -\sum_{S_i \in \mathcal{S}} \Big(\frac{Q_i^{\mathrm{I}} f_i W_i^1}{T_i} + \frac{Q_i^{\mathrm{D}} W_i^2}{T_i} \Big) \tag{5.19a}$$

$$g_1 = \sum_{S_i \in \mathcal{S}} \frac{Q_i^{\mathrm{D}}}{T_i f_i} + \sum_{S_i \in \mathcal{S}} \frac{Q_i^{\mathrm{I}}}{T_i} + 1 \leq 0 \tag{5.19b}$$

$$g_2 = \sum_{S_i \in \mathcal{S}} \Big(C_0 f_i^2 \frac{Q_i^{\mathrm{I}}}{T_i} + C_0 f_i^3 \frac{Q_i^{\mathrm{D}}}{T_i} + P_i^{\mathrm{I}} \frac{Q_i^{\mathrm{I}}}{T_i f_i} + P_i^{\mathrm{I}} \frac{Q_i^{\mathrm{D}}}{T_i} \Big) + P^{\mathrm{S}} + P^{\mathrm{Leak}} - P^{\star} \leq 0 \tag{5.19c}$$

$$g_{3,\ldots,3+n} = f_i^{\min} - f_i \leq 0, S_i \in \mathcal{S} \tag{5.19d}$$

$$g_{4+n,\ldots,4+2n} = f_i - f_i^{\max} \leq 0, S_i \in \mathcal{S} \tag{5.19e}$$

Então a solução para o modelo proposto irá ser dividida em duas fases. A primeira usa o algoritmo detalhado em Algoritmo 1, o qual consiste em resolver o problema (5.18), encontrando uma solução factível. A segunda fase é mostrada em Algoritmo 2, o qual consiste em alcançar a resposta ótima do problema.

Uma implementação do método de ponto-interior com as duas fases foi realizada em linguagem C para testar o algoritmo sugerido. O conjunto de frequências encontrado foi: $f_1 = 1; f_2 = 1; f_3 = 1; f_4 = 0,82$, consumindo $16,43W$ e o tempo computacional gasto foi de $58ms$, sendo $4ms$ para achar o ponto inicial factível e $52ms$ para chegar à solução ótima. Como esperado, a solução encontrada foi igual para os dois algoritmos.

Algorithm 1: Método da Barreira, Fase 1

Entrada: Conjunto de servidores
$\mathcal{S} = \{Q_i^{\mathrm{D}}, Q_i^{\mathrm{I}}, T_i, \alpha_i, f_i^{\min}, f_i^{\max} : i = 1, ..., n\}$,
$\alpha \in (0; 0,5)$, $t = t^{(0)} > 0$, $\beta \in (0, 1)$ é a taxa de decrescimento do passo de *backtracking*, tolerância $\epsilon > 0$, μ taxa de crescimento, m número de restrições

Dado: Um ponto inicial $\mathbf{x}$, calcula-se o valor de s, para que $\mathbf{x}$ seja estritamente factível

repita

  Resolve Caminho central

  Computa: $\mathbf{x}(t) = \min f(\mathbf{x}, t) = ts - \phi(\mathbf{x})$, usando método de *Newton* a partir de um $\mathbf{x} \in$ **dom** f

  repita

    Calcular *o passo de Newton e o decremento*

$$\Delta \mathbf{x_{nt}} := -\nabla^2 f(\mathbf{x}, t)^{-1} \nabla f(\mathbf{x}, t);$$
$$\lambda^2 := \nabla f(\mathbf{x}, t)^T \nabla^2 f(\mathbf{x}, t)^{-1} \nabla f(\mathbf{x}, t);$$

    Busca linear *para escolher o tamanho do passo p com backtracking;*

    $p := 1$;

    enquanto $(f(\mathbf{x} + p\Delta \mathbf{x_{nt}}, t) > f(\mathbf{x}, t) + \alpha p \nabla f(\mathbf{x}, t)^T \Delta \mathbf{x_{nt}})$ **faça**

      $p := p\beta$;

    fim

    Atualiza: $\mathbf{x} := \mathbf{x} + p\Delta \mathbf{x_{nt}}$;

  até $\frac{\lambda^2}{2} < \epsilon$;

  Incrementa $t = \mu t$;

até $\frac{m}{t} < \epsilon$;

Resultado: $\mathbf{x}$

Chama *Método da Barreira Fase 2*

Algorithm 2: Método da Barreira, Fase 2

Entrada: Conjunto de servidores
$\mathcal{S} = \{Q_i^{\mathrm{D}}, Q_i^{\mathrm{I}}, T_i, \alpha_i, f_i^{\min}, f_i^{\max} : i = 1, ..., n\}$,
$t = t^{(0)} > 0$, β é a taxa de decrescimento do passo de *backtracking*, tolerância $\epsilon > 0$, μ taxa de crescimento, m número de restrições, $\mathbf{x}$ estritamente factível

repita
- **Resolve** Caminho central
- **Computa** $\mathbf{x}(t) = \min tf_0(\mathbf{x}) - \phi(\mathbf{x})$, *usando método de Newton a partir de um* $\mathbf{x} \in$ **dom** f
- **Atualiza** $\mathbf{x} := \mathbf{x}(t)$
- **Incrementa** $t = \mu t$

até $\frac{m}{t} < \epsilon$;

Resultado: x

5.4 ANÁLISE NUMÉRICA DOS MODELOS CONVEXOS

Com intuito de comparar e verificar o comportamento dos algoritmos sugeridos na resolução do modelo proposto, foi realizada uma análise numérica para mensurar a performance no *Raspberry Pi*. Neste equipamento foi realizada a medição do tempo utilizado de *CPU* para resolver instâncias do problema com o algoritmo do método do ponto-interior e o SQP. Essa análise irá separar o tempo usado para encontrar uma solução inicial e a ótima, respectivamente chamadas de Fase 1 e Fase 2.

5.4.1 Condições dos Experimentos

Para avaliar os algoritmos citados foram usadas quinhentas instâncias de conjuntos com 10, 20, 30 e 50 tarefas. Elas foram geradas com o algoritmo *UUniFast* (BINI; BUTTAZZO, 2005). Cada instância tem utilização do sistema igual a 80%, sendo 20% tempo de execução utilizado por outros dispositivos e 60% pela *CPU*.

Os experimentos foram executados em um dispositivo *Raspberry Pi*, modelo B, com a frequência de $1GHz$. Os períodos das tarefas consideradas eram múltiplos de cinco. Os parâmetros de energia utilizados pelas instâncias foram: $C_0 = 0,5498$, $P^{\text{Leak}} = 0,34$, $f^{\max} = 1$, $f^{\min} = 0,3$, P^{I} variando de $0,023W$ até $0,2W$ e $P^{\star}$ foi definido de acordo com a configuração que traz maior consumo energético para o sistema. Nos experimentos com o modelo que suporta variação de frequência e carga o valor de Q_i^{D} pode ser reduzido em até 20%.

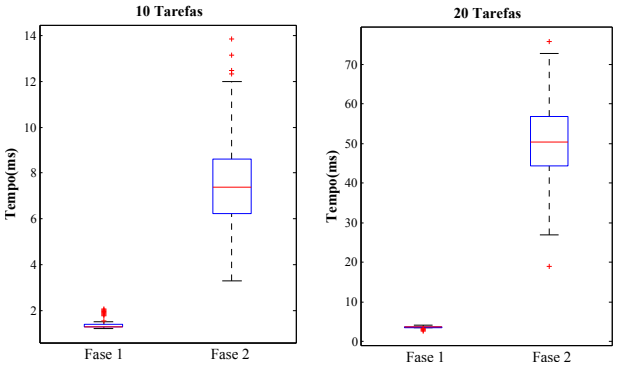
Os algoritmos usados consideram uma tolerância $\epsilon = 10^{-3}$, para as condições de otimalidade. Todas as instâncias tem como configuração inicial os menores valores para a frequência e Q_i^{D}. Nessa análise a performance é medida em duas fases. A Fase 1 é o tempo necessário para encontrar uma solução factível e a Fase 2 representa o tempo usado para achar a solução ótima. A diferença entre a solução ótima e factível está explicada no Anexo A.1. Assim como os resultados do capítulo anterior, os experimentos serão representados por gráficos do tipo *boxplot*.

5.4.2 Análise do Modelo com Variação de Frequência

As Figuras 22 e 23 mostram o tempo utilizado pelo *CFSQP* para resolver as instâncias do problema com 10, 20, 30 e 50 tarefas. O cenário com 10 tarefas teve o tempo médio da Fase 1 de $1,35ms$ e pior caso $2,7ms$, e na Fase 2 $8,8ms$ e $15,11ms$. Esses resultados implicam

que o tempo para achar a solução ótima não é muito maior que o de achar uma solução factível. Os conjuntos com 20 tarefas tiveram um comportamento similar ao de 10 tarefas, com o caso médio da Fase 1 igual a $3,5ms$ e no pior caso $4,03ms$ e na Fase 2 $54,16ms$ e $78,79ms$.

Figura 22: Tempo em segundos utilizados pelo *CFSQP* para achar a solução das instâncias com 10 e 20 tarefas.



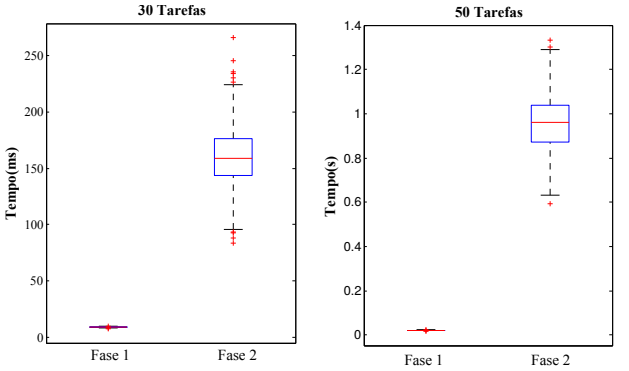
Fonte: Criado pelo Autor.

Como pode se observar na Figura 23, o tempo utilizado pela Fase 2 é muito maior que a Fase 1. Sendo o pior caso da Fase 1 $9,58ms$ e o tempo médio de $168,82ms$ e $274,7ms$ o pior caso. Já o cenário com 50 tarefas possui um caso médio de $0,98s$, o que é muito demorado, por isso foi aplicada uma técnica de *warm start*. Para isso, todas as instâncias foram resolvidas *off-line* com 10% a menos de energia e essa solução foi utilizada como solução inicial. Tal técnica diminuiu substancialmente o tempo de resolução do problema como mostra a Figura 24.

Os tempos de execução do algoritmo do método de ponto-interior para os casos de 10 e 20 tarefas, foram colocados na Figura 25. Pode-se notar que a Fase 1 é mais rápida que a do *CFSQP*, enquanto a Fase 2 é mais demorada. Nos cenários com 10 tarefas o tempo médio e pior caso para a Fase 1 foram respectivamente de: $1,7ms$ e $2ms$ e na Fase 2 foram: $50,3ms$ e $55,5ms$. E nos cenários com 20 tarefas o mesmo comportamento foi apresentado, sendo o tempo médio e o pior caso da Fase 1 $1,8ms$ e $2ms$ e da Fase 2 $266,5ms$ e $284,8ms$.

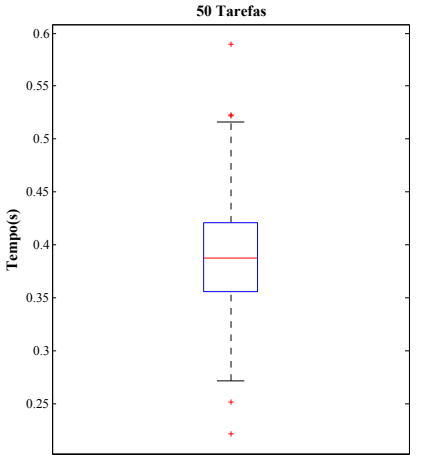
Para os cenários com 30 e 50 tarefas os resultados não foram bons se comparados ao *CFSQP*, o tempo médio total para o problema com 30 tarefas foi de $822,1ms$ e para 50 tarefas foi de $4s$. O uso da técnica

Figura 23: Tempo em segundos utilizados pelo *CFSQP* para achar a solução das instâncias com 30 e 50 tarefas.



Fonte: Criado pelo Autor.

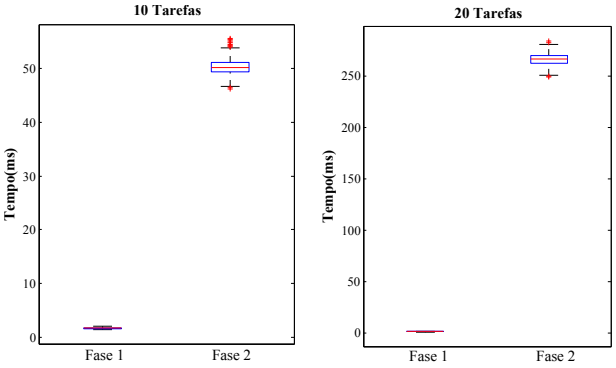
Figura 24: Tempo em segundos utilizados pelo *CFSQP* para achar a solução das instâncias com 50 tarefas, usando a técnica de *warm start*.



Fonte: Criado pelo Autor.

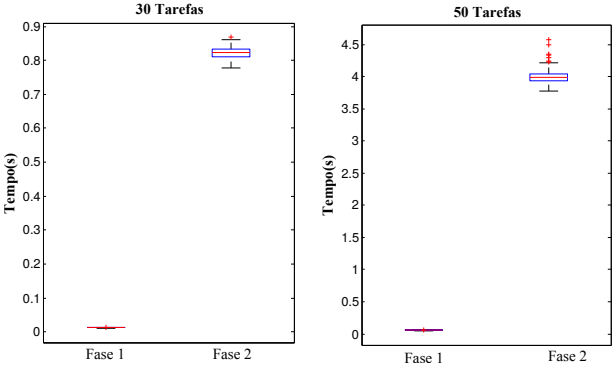
de começo quente não atingiu a melhora desejada. Para melhorar o desempenho do algoritmo do ponto-interior seria necessário uma forma mais eficiente de calcular o passo do algoritmo de Newton.

Figura 25: Tempo em segundos utilizados pelo algoritmo do método do ponto-interior para achar a solução do problema com 30 e 50 tarefas.



Fonte: Criado pelo Autor.

Figura 26: Tempo em segundos utilizados pelo algoritmo do método do ponto-interior para achar a solução do problema com 30 e 50 tarefas.



Fonte: Criado pelo Autor.

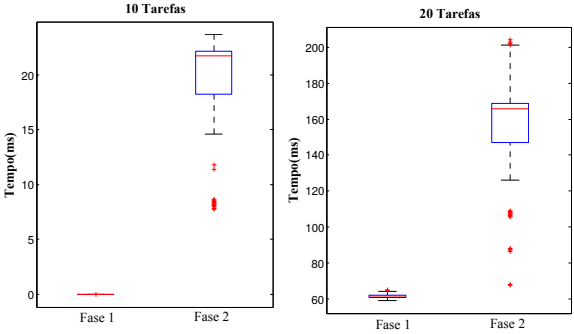
Como conclusão, pode ser observado que ambos os algoritmos tem a distribuição do tempo de execução concentrada perto do tempo médio, o que indica uma pequena variação de tempo de execução. Pode-se perceber que o *CFSQP* apresentou uma performance claramente melhor. O algoritmo do método de ponto-interior pode melhorar a performance caso seus parâmetros de entrada, mostrados nos Algoritmos 1 e 2, sejam

definidos para cada instância, as matrizes sejam armazenadas em uma estrutura mais sofisticada como *CCS* (*Compressed Column Storage*), e uma forma melhor para definir o passo do método de *Newton*.

5.4.3 Análise do Modelo com Variação de Frequência e Carga

A Figura 27 mostra o tempo utilizado por *CFSQP* para resolver as instâncias do problema com 10 e 20 tarefas. No cenário com 10 tarefas o tempo médio utilizado pela Fase 1 foi de $13ms$ e no pior caso $14,47ms$ e na Fase 2 $19ms$ $23,68ms$, respectivamente. Os conjuntos com 20 tarefas tiveram um comportamento similar, o caso médio da Fase 1 foi de $61,65ms$ e pior caso de $65,30ms$, e na Fase 2 $156,07ms$ e $204ms$.

Figura 27: Tempo em segundos utilizados pelo *CFSQP* para achar a solução das instâncias com 10 e 20 tarefas.



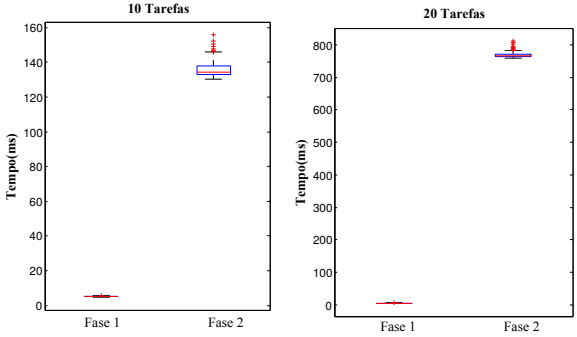
Fonte: Criado pelo Autor.

Os cenários com 30 e 50 tarefas tiveram um comportamento similar, sendo o tempo médio para o problema com 30 tarefas igual a $421ms$ e com 50 tarefas foi de $2,25s$. A utilização da técnica de começo quente, considerando a solução ótima de um cenário com 10% a menos de energia, obteve um bom desempenho, resolvendo o problema com 50 tarefas em um tempo médio de $0,4s$.

O tempo utilizado pelo algoritmo do método de ponto-interior para os casos de 10 e 20 tarefas, foi colocado na Figura 27. Novamente, pode-se notar que a Fase 1 é muito mais rápida que a do CFSQP. Entretanto, a Fase 2 é mais demorada. Considerando o cenário de 10 tarefas tem-se respectivamente para a Fase 1 e Fase 2 o tempo médio

de $5,20ms$ e $135,84ms$. Já no cenário com 20 tarefas a Fase 1 tem o tempo médio de $5,81ms$ e a Fase 2 de $811,20ms$.

Figura 28: Tempo em segundos utilizados pelo algoritmo do método do ponto-interior para achar a solução do problema com 20 tarefas.



Fonte: Criado pelo Autor.

Nos casos com 30 e 50 tarefas a implementação do método de ponto-interior apresentou tempo médio de $105ms$ e $537ms$ para resolver a Fase 1 e $5,09s$ e $26,04s$ para a Fase 2. Logo, a implementação não é suficientemente eficiente para resolver o problema, ela precisa ser aprimorada.

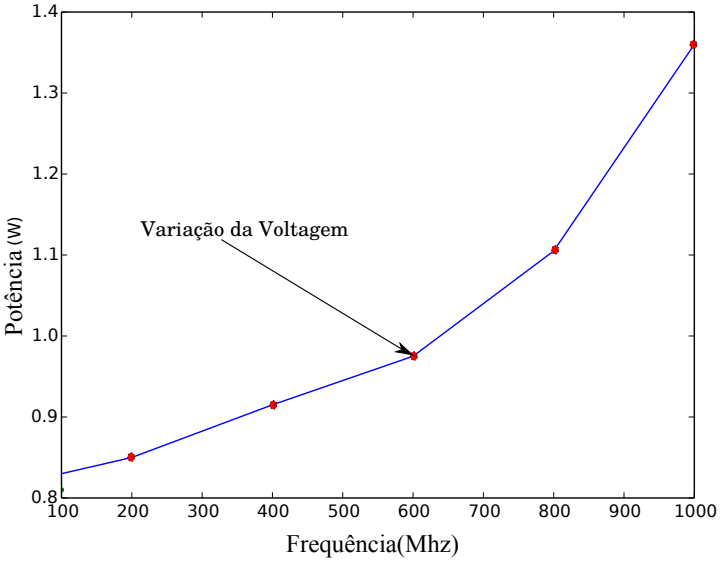
Assim como na análise anterior, pode ser observado que ambos os algoritmos tem a distribuição do tempo de execução concentrada perto do tempo médio, o que indica uma pequena variação de tempo de execução. As mesmas melhorias podem ser aplicadas aqui também.

5.5 MODELO LINEAR POR PARTES COM VARIAÇÃO DE FREQUÊNCIA

Os experimentos do capítulo 3 mostram que as equações (2.5) e (2.16), utilizadas na literatura para descrever o consumo energético e utilização podem ser descritas por equações lineares por partes. Isso ocorre pois, apesar do processador conseguir simular a troca de frequência, ele não é capaz de fazer o mesmo para a voltagem. Ao trocar a frequência do processador outras frequências como da memória, podem ser alteradas. Assim, o consumo energético do processador, muda de forma linear entre os pontos de troca de voltagem, como mostra a Figura 29. Além disso, a utilização também varia de forma linear, de

acordo com a redução da frequência da memória.

Figura 29: A linha contínua indica o consumo energético do Raspberry Pi em diferentes frequências e os círculos indicam a troca de voltagem.



Fonte: Criado pelo Autor.

Na linearização por partes, funções complexas são divididas em intervalos, que são segmentos de retas: uma função conhecida $f : \mathcal{R} -> \mathcal{R}$. Isso permite transformar uma função de $\mathcal{R}^n$ para $\mathcal{R}$, discretizando ela em um conjunto de K pares $(x, f(x))$, sendo estes combinados para se obter uma aproximação da função original. Três modelos são propostos em (PADBERG, 2000) e (SHERALI, 2001). Devido à natureza das equações desse problema será usada a representação de Combinação Convexa Agregada.

Considerando os experimentos, assume-se um sistema monoprocessado com a frequência variando de forma contínua no intervalo $[f^{\min}, f^{\max}]$, que representa respectivamente a menor e maior utilização da *CPU*. Com isso tem-se o problema de selecionar uma frequência f_i que economize uma quantidade de energia desejada e obedeça às restrições

temporais da aplicação.

Para transformar a equação de energia (2.5) em uma equação linear por partes, iremos selecionar as frequências $\tilde{f}_i^k$ sendo i a tarefa que usará a configuração e k a frequência onde a voltagem varia. Usando o modelo de combinação convexa, esse problema fica matematicamente representado por:

$$P^{\mathrm{L}}: \quad \max \sum_{i=1}^{n} \tilde{G}_i(Q_i^{\mathrm{D}}, Q_i^{\mathrm{I}}, f_i, W_i^1, W_i^2) \tag{5.20a}$$

$$\text{s.a}: \sum_{i=1}^{n} \tilde{P}_i \leq P^{\star} \tag{5.20b}$$

$$\sum_{i=1}^{n} \tilde{U}_i \leq 1 - \sum_{i=1}^{n} \frac{Q_i^I}{T_i} \tag{5.20c}$$

$$\tilde{P}_i = \sum_{k=0}^{K_i} \lambda_i^k P_i(\tilde{f}_i^k), i = 1, \ldots, n \tag{5.20d}$$

$$\tilde{U}_i = \sum_{k=0}^{K_i} \lambda_i^k U_i(\tilde{f}_i^k), i = 1, \ldots, n \tag{5.20e}$$

$$\tilde{G}_i = \sum_{k=0}^{K_i} \lambda_i^k G_i(\tilde{f}_i^k, Q_i^{\mathrm{D}}, Q_i^{\mathrm{I}}, W_i^1, W_i^2), i = 1, \ldots, n \tag{5.20f}$$

$$f_i = \sum_{k=0}^{K_i} \lambda_i^k \tilde{f}_i^k \tag{5.20g}$$

$$\sum_{k=0}^{K_i} \lambda_i^k = 1, i = 1, \ldots, n \tag{5.20h}$$

$$0 \leq \lambda_i^k \leq 1, i = 0, \ldots, n \text{ e } k = 0, \ldots, K_i \tag{5.20i}$$

sendo:

- as equações (5.20a) representa o QoS a ser maximizado pelo modelo. A equação (5.20f) define o QoS de cada tarefa i como uma função linear por partes que depende da frequência $\tilde{f}_i^k$ do processador, sendo este definido por:

$$G_i(\tilde{f}_i^k, Q_i^{\mathrm{D}}, Q_i^{\mathrm{I}}, W_i^1, W_i^2) = \sum_{k=0}^{K_i} \lambda_i^k \Big(\frac{Q_i^{\mathrm{I}} \tilde{f}_i^k W_i^1}{T_i} + \frac{Q_i^{\mathrm{D}} W_i^2}{T_i} \Big) \tag{5.21}$$

- as equações (5.20c) e (5.20e) representam a restrição de escalonabilidade;
- as equações (5.20b) e (5.20d) representam a restrição energética;
- a Eq. (5.20g) define a frequência que será usada;
- as Eq. (5.20h) e (5.20i) garantem que a frequência selecionada f_i é uma combinação convexa dos pontos $\tilde{f}_i^k$;
- λ_i^k é uma variável contínua, entre 0 e 1, que define a combinação convexa.

De acordo com os experimentos realizados no Capítulo 3, esse modelo consegue representar o sistema do Raspberry Pi com mais precisão. Apesar da frequência da CPU se comportar de forma contínua, outras variáveis que influenciam o funcionamento do sistema são discretas.

5.6 ANÁLISE NUMÉRICA

Agora será utilizado o *ECOS* (*Embedded Conic Solver*) para resolver o problema linear por partes. Com isso poderá ser verificado o tempo necessário para resolver o problema no *Raspberry Pi* e comparar o QoS com o do modelo convexo variando a frequência conforme o Problema 5.1, uma vez que ambos possuem a mesma função objetivo.

5.6.1 Condições dos Experimentos

Para avaliar o desempenho do modelo linear por partes foram usados os cenários utilizados nos experimentos do modelo convexo variando a frequência. Logo, nessa análise será verificado o comportamento do algoritmo linear por partes para resolver os conjuntos de 10, 20, 30 e 50 tarefas. Os experimentos foram executados em um dispositivo *Raspberry Pi*, modelo B, com a frequência de $1GHz$. Foi considerado que a voltagem mudava nas seguintes frequências: $100MHz$, $300MHz$, $400MHz$, $700MHz$, $900MHz$ e $1GHz$.

Os parâmetros de energia utilizados pelas instâncias foram: $C_0 = 0,5498W$, $P^{\text{Leak}} = 0,34W$, $f^{\max} = 1$, $f^{\min} = 0,3$, P^{I} variando de $0.023W$ até $0.2W$ e $P^{\star}$ foi definido de acordo com a configuração que traz maior consumo energético para o sistema. Esses parâmetros são os mesmos utilizados para análise do modelo convexo com troca de frequência, para comparar os resultados.

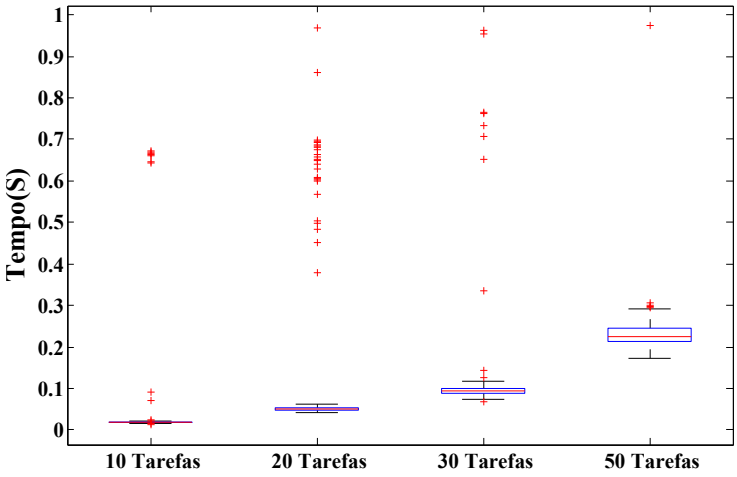
O algoritmo usado considera uma tolerância $\epsilon = 10^{-13}$ da solução ótima. Todas as instâncias tem como configuração inicial igual a zero.

Nessa análise a performance é dada pelo tempo total para resolver a instância do problema, pois não foi possível separar o tempo necessário para encontrar uma solução qualquer e a ótima. Assim como os resultados do capítulo anterior, os experimentos serão apresentados por gráficos do tipo *boxplot*.

5.6.2 Análise Numérica do Modelo Linear por Partes

A Figura 30 mostra o tempo utilizado pela *ECOS* para resolver as instâncias do problema com 10, 20, 30 e 50 tarefas. O tempo médio para achar a resposta para os cenários com 10, 20, 30 e 50 tarefas foram respectivamente: $28ms$, $80ms$, $106ms$ e $231ms$. Apesar do tempo de resposta do *ECOS* ter sido maior que a do *CFSQP* em média para os cenários de 10 e 20 tarefas, é importante ressaltar que ele teve um valor de $\epsilon = 10^{-10}$, muito menor que o *CFSQP*.

Figura 30: Tempo em segundos do *(ECOS)* utilizado para resolver as instâncias dos cenários.



Fonte: Criado pelo Autor.

Analisando 20 cenários com 50 tarefas foi constatado que o modelo linear consegue atingir um QoS maior que o modelo convexo. Essa diferença aumenta de acordo com a diminuição da energia disponível. Considerando o cenário com 100% de energia, a diferença obtida do QoS foi de $0,6\%$ devido ao menor valor de ϵ. Crescendo em alguns

cenários até 5%, quando o consumo de energia diminui até o valor mínimo aceitado pelo modelo convexo. O modelo linear por partes conseguiu escalonar energia com até 3% a menos que o modelo convexo.

Com esses resultados, pode-se perceber que a solução obtida pelo modelo linear consegue um QoS mais alto que o modelo convexo à medida que diminui-se a energia disponível. Isso ocorre pois o modelo linear por partes representa o consumo energético de forma exata e o tempo para resolver o problema não cresce de forma linear ao número de tarefas.

5.7 SUMÁRIO

Nesse capítulo foram apresentados modelos contínuos para resolução do problema de escalonamento com restrição de energia. Iniciou-se o capítulo com um modelo contínuo variando a frequência. Depois foi apresentada uma extensão que permite a variação de frequência e carga, o qual foi provado ser convexo também. Em seguida foram apresentados alguns algoritmos para resolução desses problemas e uma análise numérica. Por último foi apresentado um modelo linear por partes e uma análise numérica para verificar a sua performance.

6 SIMULAÇÃO

Os experimentos do capítulo 3 indicam que as equações utilizadas, nessa tese descrevem o comportamento de um sistema real com pequeno erro. As análises numéricas, nos capítulos 4 e 5, por sua vez apontam que os algoritmos sugeridos apresentam um baixo *overhead* e pouca variação na performance, considerando um conjunto com número fixo de tarefas. Agora, para validar o funcionamento dos modelos propostos, serão executadas simulações. Essas simulações devem revelar se o emprego do *framework* de reconfiguração será capaz de garantir a economia de energia desejada e evitar o *throttling*.

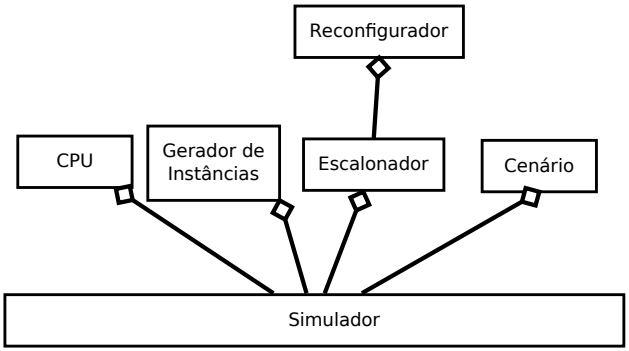
6.1 SIMULADOR

A simulação deve levar em consideração as características abordadas no problema aqui discutido. No caso do trabalho proposto, o simulador deverá considerar questões como: temperatura, energia, modelo de tarefas com tempo de dispositivos e reconfiguração dinâmica. Os simuladores RTMultiSim (HANGAN; SEBESTYEN, 2012), STORM (URUNUELA; DEPLANCHE; TRINQUET, 2010), Fortas (COURBIN; GEORGE, 2011) e Yartiss (CHANDARLI et al., 2012) foram analisados para simular o modelo proposto. Apesar de facilitarem a simulação do prolema de escalonamento de tempo real, eles não consideram todas as características aqui abordadas. Assim sendo, foi decidido implementar um simulador que levasse todas essas características abordadas pelo trabalho.

Então, para implementar o simulador que levasse em consideração as características do problema abordado, foi usada a linguagem de programação *Python*. Essa linguagem de programação foi escolhida devido a sua portabilidade, facilidade de comunicação com códigos em *C* e o grande suporte a bibliotecas matemáticas eficientes, como: *numpy, OpenOpt, CVXOPT* e *CVXPY*.

A simulação tem seu ambiente definido pela especificação da carga de trabalho, política de escalonamento selecionada, ambiente de execução e plataforma em questão. Essas informações são definidas pelo usuário e geram os componentes independentes, ilustrados na Figura 31. O simulador irá executar as tarefas, guardando informações como número de instâncias, os seus modos e tempos de execução. Com isso será possível verificar o quanto de energia foi realmente consumida e a qualidade na qual as tarefas foram executadas. O simulador é consti-

Figura 31: Componentes do Simulador.



Fonte: Criado pelo Autor.

tuído pelos seguintes componentes:

- simulador - responsável por inicializar todos os componentes existentes no cenário, controlar o tempo e conjunto de tarefas que está executando.
- escalonador - define a política de escalonamento do sistema e realiza a troca de modo de execução das tarefas;
- reconfigurador - encontra a configuração que garantirá a economia de energia e evitará o super aquecimento do processador. Deve ser chamado pelo escalonador;
- gerador de instâncias - cria as instâncias das tarefas, respeitando as regras definidas, como utilização e modos selecionados pelo reconfigurador;
- *CPU* - simula o comportamento do processador como temperatura, energia e troca de frequência;
- cenário - descreve os componentes utilizados, as condições em que a simulação ocorrerá e os momentos em que deverão ocorrer as trocas de modos.

Com isso, o modelo energético do simulador é independente do modelo usado pelo mecanismo de reconfiguração. Logo, será possível usar no simulador um modelo energético extraído pelos experimentos do capítulo 3 e mensurar a diferença com relação às equações utilizadas pelos modelos e o dispositivo real. O simulador pode ser visto como uma coleção de componentes como mostra a Figura 31. Através desse simulador será possível simular um sistema de tempo real, considerando todas as variáveis estudadas nessa tese.

6.2 DEFINIÇÃO DO SISTEMA SIMULADO

Os sistemas de tempo real executam tarefas com características distintas em diferentes *hardwares*. Logo, a simulação deve levar essas situações em conta. Um exemplo seria o processador, que pode implementar *AVFS* ou *DVFS* discreto. Outro caso seriam as tarefas que podem ter vários modos de execução, alguns com uma pequena ou grande diferença do tempo de execução entre os modos disponibilizados.

Com isso, os dois tipos de processadores citados acima serão usados. O processador com *AVFS* terá a curva de energia variando de forma linear entre os pontos de troca de voltagem, como o *Raspberry Pi*. O processador usando *DVFS* discreto terá suas frequências definidas nos pontos de troca de voltagem. Quando o modelo contínuo definir uma frequência que não pode ser selecionada, será usada a primeira frequência maior. Tal comportamento assegurará o cumprimento do *deadline*, uma vez que uma frequência maior diminui o tempo de execução esperado e a restrição de energia não deve sofrer impacto, pois não irá alterar a voltagem.

As tarefas terão modos discretos, como a aplicação *SUSAN* apresentada no capítulo 3. Entretanto o número de modos considerado nos experimentos pode variar, assim como de granularidade do tempo de execução suportado. Essa variação permitirá avaliar como o modelo contínuo funciona em cenários onde o tempo de execução muda em pequenos e grandes intervalos. O mecanismo de reconfiguração deverá selecionar o modo com tempo de execução igual ou inferior ao selecionado pelos modelos contínuos. Novamente esse comportamento assegura o cumprimento do *deadline* e da restrição de energia.

A solução do modelo contínuo não necessariamente reflete a configuração que o sistema está usando, pois ele é discreto. Tornando-se necessário recalcular o valor do QoS do sistema após a aplicação da configuração. Dessa forma, será possível verificar a diferença de QoS previsto pelo modelo contínuo e o que realmente é aplicado, assim como

a eficiência dos modelos contínuos em relação aos discretos.

Logo, inspirado na simulação do trabalho de mestrado do Augusto Born de Oliveira (OLIVEIRA, 2009), uma simulação considerando um robô será realizada. Essa simulação deverá levar em consideração um processador com *AVFS* e outro com *DVFS* discreto, e um conjunto de tarefas com uma variação entre os modos pequena ou alta. Esses diferentes tipos de processadores e conjunto de tarefas permitirão verificar o comportamento dos modelos em diferentes sistemas de tempo real.

6.3 CONJUNTO DE TAREFAS

O sistema simulado é composto por um conjunto com sete tarefas, que podem variar o estado de execução de acordo com as atividades e condições do meio que circunda o robô. Assim sendo, os estados podem mudar de forma arbitrária e independente.

Para a realização dos experimentos foram consideradas a seguintes tarefas, listadas em ordem de prioridade:

1. Mapeamento (T_{map}): é responsável por gerar uma representação do ambiente onde o robô está inserido, para que um caminho até o objetivo seja traçado; geralmente é executada quando o robô está parado, podendo ser executada quando está em movimento com objetivo de gerar simplificações no mapa;

2. Monitoramento (T_{mon}): consiste na utilização dos sensores do robô para identificar as condições e movimentações no ambiente;

3. Monitoramento Remoto (T_{rem}): realiza trocas de informações com uma máquina informando dados como posição, ações que estão sendo executadas e transmissão de imagens;

4. Localização (T_{loc}): adquire as coordenadas de localização do robô;

5. Reconhecimento (T_{rec}): é responsável por identificar sons e objetos;

6. Visão (T_{vis}): é responsável por processar as imagens adquiridas pela câmera e gerar informações sobre as mesmas;

7. Manipulação (T_{man}): está associada com a realização de algum trabalho manual no ambiente, como mover um objeto qualquer.

As tarefas citadas acima foram levadas em consideração pois, de acordo com o fabricante (ROBOTS, 2014), elas são as possíveis tarefas executadas por um robô de vigilância. O consumo energético dos dispositivos

foi estipulado com especificações encontradas no sítio especializado em vendas de peças de robô (SHOP, 2012). Com base nessas informações foi realizada uma associação entre dispositivos e tarefas executadas pelo robô. Na Tabela 8 estão representados o gastos energéticos independente do processador de cada tarefa. Nela existe uma associação das tarefas a dispositivos.

Tabela 8: Tabela com associação de tarefas a dispositivos.

Tarefa	Dispositivos	Energia (P^{I})W
T_{map}	Disco rígido	$(0,8)$
T_{mon}	Câmeras, sensores e sonar	$(0,0165)$
T_{rem}	Placa de rede sem fio	$(0,462)$
T_{loc}	GPS	$(0,17)$
T_{rec}	Disco rígido e câmeras	$(0,9815)$
T_{vis}	Câmeras	$(0,1815)$
T_{man}	Braço mecânico	(15)

Fonte: Criado pelo Autor.

Algumas tarefas podem ter utilizações diferentes devido aos estados que o robô pode assumir. Um exemplo é a tarefa de mapeamento, que enquanto o robô está parado modifica ou cria o mapa e quando ele está em movimento simplifica o mapa. Na Tabela 9 foram estipuladas diferentes utilizações para as tarefas em Watts, considerando que o robô poderia assumir dois estados: parado ou em movimento.

Considerando as tarefas e suas utilizações, foi criada uma associação entre os estados e o conjunto de respectivas tarefas na Tabela 10. Tais conjuntos de tarefas associados a um estado foram chamados de: S1, S2, S3, S4 e S5.

Para realizar a simulação, todas as tarefas foram consideradas como *hard*. A utilização u_i das tarefas estão Tabela 9. As instâncias sempre terão o tempo de execução igual ao máximo do modo selecionado.

Além das sete tarefas aqui citadas, o sistema terá mais uma tarefa, a $T_{reconfig}$ que terá a utilização igual a %8. Essa tarefa será responsável por executar o reconfigurador do sistema. Seu tempo de execução é considerado igual a 2 ms, com período de 25ms.

Tabela 9: Utilização de cada tarefa de acordo com o estado do robô.

Estado	Tarefas	Utilização
Parado	Mapeando (T_{map})	30%
Parado	Monitoramento (T_{mon})	15%
Parado	Monitoramento Remoto (T_{rem})	6%
Parado	Localização (T_{loc})	3%
Parado	Reconhecimento (rec)	20%
Parado	Visão (T_{vis})	20%
Parado	Manipulação (T_{man})	25%
Movimento	Mapeando (T_{map})	10%
Movimento	Monitoramento (T_{mon})	20%
Movimento	Monitoramento Remoto (T_{rem})	10%
Movimento	Localização (T_{loc})	6%
Movimento	Reconhecimento (T_{rec})	14%
Movimento	Visão (T_{vis})	25%

Fonte: Criado pelo Autor.

Tabela 10: Estados do robô e tarefas associadas.

Estado	Tarefas	*CPU%*
Parado (S1)	T_{map}, T_{mon}, T_{rem}, T_{vis}	71%
Parado (S2)	T_{mon}, T_{rem}, T_{loc}, T_{rec}, T_{vis}	64%
Parado (S3)	T_{mon}, T_{rem}, T_{loc}, T_{vis}, T_{man}	69%
Movimento(S4)	T_{map}, T_{mon}, T_{rem}, T_{loc}, T_{rec}, T_{vis}	85%
Movimento(S5)	T_{mon}, T_{rem}, T_{loc}, T_{rec}, T_{vis}	75%

Fonte: Criado pelo Autor.

6.4 CENÁRIO DE SIMULAÇÃO

No modelo discreto que considera frequência e carga computacional e nos modelos que consideram somente frequência, existirão dois cenários. O primeiro usando um processador AVFS e o segundo com um processador DVFS discreto. Já os modelos com frequência e carga de tarefa contarão com quatro cenários. Os cenários 1 e 2 terão um processador com *AVFS*, enquanto cenários 3 e 4 um processador com *DVFS* discreto. Sendo o conjunto de tarefas, nos cenários 1 e 3 com a diferença entre os tempos de execução dos modos pequenas e os cenários 2 e 4 grandes.

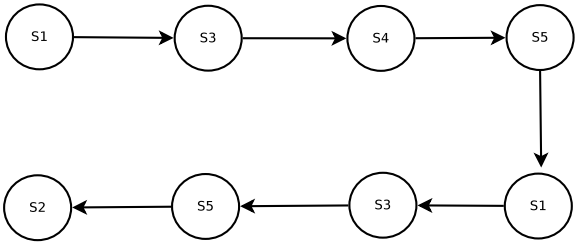
Para todos os cenários será considerada a sequência de estados mos-

trada na Figura 32. No início do funcionamento do robô, a partir do instante zero, ele estará parado executando o conjunto de tarefas S1. Após 2 minutos o sistema passa para o estado S3 e executa por mais 1 minuto, quando começa a andar e muda para S4.

Após 2 minutos troca-se novamente de estado, passando mais 9 minutos no estado S5. Troca-se para S1 onde permanece por 1 minuto, quando vai para o estado S3 e fica por mais 3 minutos. Sai de S3 e começa a andar no estado S5 por 3 minutos e começa a executar em S2 por 9 minutos e o sistema termina a execução, totalizando 30 minutos de execução.

Esse cenário aqui descrito ao ser executado sem o reconfigurador, sempre usando 100% da frequência e máxima qualidade das tarefas, resultou no aquecimento da *CPU*. Tal fato ocasionou *throttling*, que implicou em perdas de *deadline*. A CPU atingiu a temperatura de $85C^{\circ}$, ficou $211S$ executando na frequência mais baixa, teve um tempo de ocioso de $755S$, algumas tarefas chegaram a perder 66% dos *deadlines* e o consumo energético foi de $0,855Watts$. Sendo assim, os experimentos a seguir utilizarão o reconfigurador para economizarem 3% de energia e evitar que o processador chegue a $78C^{\circ}$, pois nessa temperatura o P^{Leak} atinge seu maior valor.

Figura 32: Ilustração de um fluxo de execução.



Fonte: Criado pelo Autor.

O peso W_i^1, W_i^2 e W_i de cada tarefa foi definido como mostra a Tabela 11. É importante relembrar que W_i^1, W_i^2 representam respectivamente a importância da frequência selecionada e tarefa para o sistema no modelo contínuo. Enquanto W_i é a variável que define a importância de uma tarefa para o sistema discreto, logo ela é igual a W_i^2. Devido aos valores mostrados na Tabela 11, a frequência e qualidade de T_{rem} sempre será a primeira a ser reduzida, uma vez que ela tem menor importância.

Tabela 11: Peso dos servidores.

T_i	W_i^1	W_i^2/W_i
T_{map}	70	45
T_{mon}	60	35
T_{rem}	5	5
T_{loc}	70	40
T_{rec}	60	35
T_{vis}	5	5
T_{man}	30	40

Fonte: Criado pelo Autor.

6.4.1 Parâmetros do Sistema

Após definir as propriedades, comportamento e conjunto de tarefas do sistema a ser simulado, deve-se determinar os valores dos parâmetros usados pelas equações: (2.14), (2.3) e (2.6). Tais informações não foram encontradas na documentação fornecida pelo site oficial do projeto *Raspberry Pi* ou fabricante do processador.

Dessa forma, os parâmetros energéticos das funções de energia foram definidos de acordo com os testes apresentados no capítulo 3. Enquanto os parâmetros da equação de temperatura foram definidos para garantir que o *Raspberry Pi* não esquente ao funcionar a $700MHz$, em uma temperatura igual a $23°C$. Tal condição foi observada durante os experimentos.

Assim sendo, para o modelo contínuo os valores definidos para os parâmetros necessários para a simulação estão da Tabela 12 e os do modelo discreto estão na Tabela 13. O P^{Leak} foi definido de acordo com experimentos no *Raspberry Pi*, sendo esses valores associados as temperaturas apresentados na Tabela 14.

Tabela 12: Parâmetros para Simulação Contínua.

Parâmetro	Valor
C_0	$0,866$
a	$713,89 \times 10^{-5}$
b	$20242,91 \times 10^{-8}$
P^{Sleep}	$0,85W$

Fonte: Criado pelo Autor.

Tabela 13: Parâmetros para Simulação Discreta.

Parâmetro	Valor
$\mathcal{F}$	$\{0.1, 0.2, 0.4, 0.6, 0.8, 1\}$
P^{D}	$\{0,81, 0,83, 0,9, 1,0, 1,1, 1,5\}$
P^{Sleep}	$0,85W$

Fonte: Criado pelo Autor.

Tabela 14: P^{Leak}

Temperatura C°	P^{Leak}(W)
$1 - 50$	$0,01$
$50 - 58$	$0,015$
$58 - 60$	$0,02$
$60 - 65$	$0,03$
$65 - 70$	$0,04$
$70 - 75$	$0,1$
$75 - 85$	$0,12$

Fonte: Criado pelo Autor.

6.5 MODELOS COM VARIAÇÃO DE CARGA E FREQUÊNCIA

O resultados das simulações dos cenários ilustrados acima, considerando os modelos que modificam somente a frequência serão apresentado aqui. Para resolver o modelo MDVFC [1] foi utilizada a heurística *HDG*, considerando somente um modo de operação para as tarefas. O modelo MCVF[2] foi resolvido com *CFSQP*. O último modelo, MLPVF[3] foi resolvido com *ECOS*. Os resultados da simulação estão dispostos nas Tabelas 15 e 16.

Na Tabela 15 estão os resultados das simulações para um processador com AVFS. Com relação aos modelos é possível notar que, apesar de todos eles terem garantido uma economia de 3%, o MDVF gastou menos energia que o restante. Isso ocorre devido à natureza discreta do modelo, que não consegue economizar exatamente o que foi pedido, por isso acaba economizando energia a mais. O mesmo ocorre com o modelo convexo que trabalha com um limite superior um pouco pessimista, que implica em salvar um pouco a mais do que o desejado. Uma

[1] MDVFC - Modelo Discreto Variando Frequência e Carga.
[2] MCVF - Modelo Convexo Variando Frequência.
[3] MLPVF - Modelo Linear por Partes Variando Frequência.

vez que o consumo energético do *Raspberry Pi* pode ser definido de forma exata por uma função linear por partes, o modelo MLPVF representa o consumo energético do processador sem erros. Outro efeito da utilização do reconfigurador foi o controle da temperatura, que evitou o aumento do gasto com P^{Leak} e o *throtling*, resultando no aumento do tempo ocioso da *CPU*. A temperatura passou do valor delimitado em $0.25°C$ pois o reconfigurador funciona como uma tarefa e a nova configuração tem que esperar *jobs* existentes terminarem.

Como última observação dessa simulação pode-se perceber que o número de reconfigurações foi pequeno, e além disso cada execução do reconfigurador utilizou menos de $1ms$ de *CPU*. No caso do modelo discreto só ocorreram reconfigurações quando houve troca de estado do robô. Já nos modelos contínuos que executavam com frequências maiores, foi necessário fazer uma reconfiguração devido a temperatura.

Tabela 15: Simulação com Processador *AVFS*, variando frequência.

Modelos	MDVFC	MCVF	MLPVF
Temperatura Max ($°C$)	$70,16$	$72,25$	$72,25$
Energia Consumida (W)	$0,729$	$0,738$	$0,759$
Tempo ocioso (s)	972	1042	999
Reconfigurações	8	9	9

Fonte: Criado pelo Autor.

A Tabela 16 contém os resultados obtidos usando um processador com *DVFS* discreto. Pode-se perceber que os resultados com modelo MDVFC não foram alterados, uma vez que ele tem um conjunto fixo de frequências que ele pode mudar. Por outro lado, os modelos contínuos controlaram a temperatura máxima desejada, consumiram menos energia que o modelo MDVFC, com tempo ocioso e número de reconfigurações superiores. Apesar da *CPU* usar uma frequência superior à definida pelo reconfigurador, foi percebido que a quantidade mínima desejada de energia economizada foi obedecida.

Os resultados das simulações indicaram que o mecanismo de reconfiguração não gera um grande *overhead* para o sistema. Considerando um sistema que usa um processador com *AVFS*, os modelos contínuos conseguem um melhor resultado, utilizando frequências mais altas e conseguindo uma economia de energia mais próxima do valor desejado. Enquanto o MDVFC economiza mais energia que o desejado e usa frequências menores. No caso da temperatura, todos os modelos se demonstraram capazes de gerenciar. Sendo que o modelo contínuo é

Tabela 16: Simulação com Processador *DVFS* discreto, variando frequência.

Modelos	MDVFC	MCVF	MLPVF
Temperatura Max ($^{\circ}C$)	$70,16$	$72,15$	$72,15$
Energia Consumida (W)	$0,729$	$0,712$	$0,712$
Tempo ocioso (s)	972	1039	1039
Reconfigurações	8	9	9

Fonte: Criado pelo Autor.

melhor nesse ponto, pois permite escalonar o sistema com qualquer valor de frequência, já o MDVFC só escalona as tarefas com as frequências suportadas.

6.6 MODELOS COM VARIAÇÃO DE FREQUÊNCIA E CARGA COMPUTACIONAL

Nessa seção serão mostrados os resultados das simulações dos cenários ilustrados acima, considerando os modelos que modificam a frequência e carga computacional das tarefas. Para resolver o modelo MDVFC foi utilizada a heurística *HDG*, considerando três modos de operação das tarefas. O modelo MCVFC[4] foi resolvido com *CFSQP*, considerando diferentes granularidades para as tarefas.

Quando o modelo MCVFC possui tarefas com alta granularidade do tempo de execução, é chamado de MCVFC1, sendo a variação do tempo de execução da tarefa a cada milissegundo. Já considerando o conjunto de tarefas com baixa granularidade, o MCVFC2, o tempo de execução varia 10% entre os modos suportados, tendo três modos como o modelo MDVFC. Os resultados da simulação estão dispostos nas Tabelas 17 e 18.

Foi considerado que o modo de menor qualidade de uma tarefa teria o tempo de execução igual 80% do tempo da qualidade máxima. Por isso, o modelo MCVFC1 pode modificar sem tempo de execução em passos de $1ms$ nesse intervalo. Enquanto o modelo MCVFC2 e MDVFC possuem três modos de execução, com 100%, 90% e 80% do tempo máximo de execução das tarefas.

Os resultados estão na Tabela 17 e indicam que houve um consumo energético maior com essa modelagem. De acordo com o histórico das configurações, isso ocorreu pois a maioria das tarefas diminuíram a

[4]MCVFC - Modelo Convexo Variando Frequência e Carga

qualidade ao invés de reduzir a frequência, em acordo com a Tabela 11. Isso implicou em uma temperatura média de aproximadamente $10C^{\circ}$ superior para os modelos que consideravam troca de carga das tarefas, o que também implicou em um número maior de chamadas ao reconfigurador para controlar a temperatura.

Ainda de acordo com os dados apresentados na Tabela 17, os modelos contínuos foram capazes de fornecer configurações de escalonamento que utilizassem mais o processador.

Tabela 17: Simulação com Processador *AVFS*, variando frequência e carga computacional.

Modelos	MDVFC	MCVFC1	MCVFC2
Temperatura Max ($^{\circ}C$)	$72,8$	$72,54$	$72,52$
Energia Consumida (W)	$0,759$	$0,778$	$0,768$
Tempo ocioso (s)	984	941	967
Reconfigurações	12	14	14

Fonte: Criado pelo Autor.

Os resultados da simulação considerando o modelo com variação de carga e frequência em um processador com DVFS discreto estão na Tabela 18. Novamente, nota-se que o modelo *MDVFC* obtém o mesmo resultado nos dois processadores, sendo que o modelo MDVFC usou menos reconfigurações para garantir as restrições desejadas e novamente economizou mais energia que o desejado.

Tabela 18: Simulação com Processador *DVFS* discreto, variando frequência e carga computacional.

Modelos	MDVFC	MCVFC1	MCVFC2
Temperatura Max ($^{\circ}C$)	$72,8$	$72,54$	$72,47$
Energia Consumida (W)	$0,759$	$0,780$	$0,770$
Tempo ocioso (s)	984	948	938
Reconfigurações	12	75	74

Fonte: Criado pelo Autor.

De uma forma geral os mecanismos de reconfiguração testados nessas simulações tiveram um baixo custo computacional e conseguiram prover as garantias para os dispositivos que eles foram desenvolvidos. Assim como a primeira simulação, os modelos contínuos conseguem garantir o consumo energético e a temperatura mínima definida pelo

usuário para processadores com suporte para *AVFS*. Apesar de terem um número de chamadas muito superior, o *overhead* foi de aproximadamente $61ms$, em $30min$ de execução.

6.7 SUMÁRIO

Nesse capítulo foram realizadas simulações para verificar o comportamento do mecanismo de reconfiguração apresentado nessa tese em diferentes tipos de dispositivos e conjuntos de tarefas. Iniciou-se o capítulo apresentando diferentes sistemas encontrados no mundo real e como eles seriam representados na simulação. Os resultados aqui apresentados mostraram que os modelos contínuos conseguem assegurar a economia de energia e podem ter um *overhead* baixo com o uso do *warm start*. O modelo inteiro, mesmo que resolvido por uma heurística, é também capaz de cumprir as garantias desejadas nos dois tipos de processadores simulados.

7 CONCLUSÕES E PERSPECTIVAS

O número de sistemas de tempo real que necessitam interagir com o ambiente de forma inteligente, tendo uma alta variabilidade de carga computacional e operando com bateria, cresce a cada dia da mesma forma que a complexidade das tarefas computacionais que auxiliam seu funcionamento. Visando dar suporte a esses sistemas, novos componentes eletrônicos estão sendo desenvolvidos de forma a agregar novas funcionalidades. Sendo assim, nesta tese foi abordado o problema de reconfiguração dinâmica, visando prover um mecanismo que garantisse economia de energia em sistemas de tempo real.

Inicialmente foram apresentadas ferramentas existentes para lidar com o problema de escalonamento em sistemas de tempo real e fatores que afetam o consumo energético foram apresentados no capítulo 2. Na sequência, foi verificada a eficácia das equações usadas na literatura para modelar o consumo energético de um sistema computacional. Somente depois disso foram propostos modelos contínuos e discretos, para representar o problema.

O modelo discreto foi desenvolvido, uma vez que o problema aqui tratado é de natureza discreta. Contudo, o custo computacional para encontrar a solução ótima era elevado, mesmo usando pacotes de otimização inteira-mista eficientes. Ainda foi percebido que os solvers mais utilizados só davam suporte a arquitetura $X86$, não sendo possível utilizá-los em outras arquiteturas, como *ARM*. Então, diante das dificuldades citadas, foram analisadas algumas heurísticas (HDG e ***MV-Pack***) para resolver o problema. Essas conseguiram um bom desempenho e uma resposta próxima da ótima.

Devido às abordagens contínuas encontradas na literatura e a existência de processadores com suporte a *AVFS* foram criados modelos contínuos também. Inicialmente foram propostos dois modelos convexos, um variando somente a frequência e outro, que variava a frequência e carga computacional da tarefa simultaneamente. Tais modelos utilizam uma equação aproximada para medir o consumo energético. Esses modelos apresentaram um custo computacional elevado para sistemas com 50 tarefas, entretanto isso pode ser reduzido com uso do *warm start*.

Ainda devido aos resultados dos experimentos, foi possível verificar que um modelo linear por partes seria capaz de descrever o consumo energético de forma exata. Sendo assim, foi criado um modelo linear por partes, variando somente a frequência. Tal modelo foi resolvido com

um pacote de otimização linear e seu custo computacional foi inferior em comparação ao modelo convexo, onde somente a frequência variava.

Por último, no capítulo 6, foram executadas simulações que verificaram o comportamento dos modelos desenvolvidos em processadores com *AVFS* e *DVFS* discreto. Os resultados apontaram que os modelos contínuos são ótimos para processadores com *AVFS*, e funcionam em uma *CPU* com *DVFS* discreto. Também foi demonstrado que o modelo discreto funciona de forma ótima em sistemas com *DVFS* discreto, onde os modelos contínuos não apresentaram resultado melhor que ele ao considerar uma *CPU* com *DVFS* discreto.

7.1 TRABALHOS FUTUROS

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:

- uma extensão do modelo linear por partes para tratar sistemas com variação de frequência e carga computacional;
- concepção do modelo energético de outros dispositivos, como por exemplo a memória;
- extensão dos modelos aqui citados para sistemas multiprocessados com uso de algoritmos de escalonamento globais;
- realizar uma implementação para testar o desempenho da reformulação compacta feita para investigar a melhora de desempenho de métodos exatos com o uso de cortes especializados.

Estudos aprofundados e a implementação destes tópicos certamente gerariam resultados inovadores, que agregariam muito conhecimento ao tema discutido neste trabalho.

REFERÊNCIAS

BALAS, E. Facets of the Knapsack Polytope. *Mathematical Programming*, v. 8, p. 146 – 164, 1975.

BALAS, E.; ZEMEL, E. Facets of the Knapsack Polytope from Minimal Covers. *SIAM Journal on Applied Mathematics*, v. 34, p. 119 – 148, 1978.

BINI, E.; BUTTAZZO, G. Measuring the Performance of Schedulability Tests. *Real-Time Systems*, v. 30, n. 1, p. 129–154, 2005.

BINI, E.; BUTTAZZO, G.; LIPARI, G. Speed Modulation in Energy-Aware Real-Time Systems. In: *17th Euromicro Conference on Real-Time Systems*. [S.l.: s.n.], 2005. p. 3–10.

BŁAŻEWICZ, J.; ECKER, K. *Handbook on Scheduling: From Theory to Applications*. Berlin, Heidelberg: Springer, 2007.

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

BUTTAZZO, G. *Hard Real-Time Computing Systems*. [S.l.]: Springer, 1997.

BUTTAZZO, G.; LIPARI, G.; CACCAMO, M.; ABENI, L. Elastic Scheduling for Flexible Workload Management. *IEEE Transactions on Computers*, IEEE Computer Society, v. 51, p. 289–302, 2002.

CACCAMO, M.; BUTTAZZO, G.; SHA, L. Capacity Sharing for Overrun Control. In: IEEE REAL-TIME SYSTEMS SYMPOSIUM. Walt Disney World Orlando, Florida, USA, 2000. p. 295 –304.

CACCAMO, M.; BUTTAZZO, G.; THOMAS, D. Efficient Reclaiming in Reservation-Based Real-Time Systems with Variable Execution Times. *IEEE Transactions on Computers*, v. 54, n. 2, p. 198–213, 2005.

CHANDARLI, Y.; FAUBERTEAU, F.; MASSON, D.; MIDONNET, S.; QAMHIEH, M. et al. Yartiss: A Tool to Visualize, Test, Compare and Evaluate Real-Time Scheduling Algorithms. In: *Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*. [S.l.: s.n.], 2012. p. 21–26.

COURBIN, P.; GEORGE, L. Fortas: Framework for real-time analysis and simulation. *Proceedings of Workshop on Analysis Tools and Methodologies for Embedded and Real-time*, p. 21–26, 2011.

CROWDER, H.; JOHNSON, E.; PADBERG, M. Solving Large-Scale Zero-One Linear Programming Problems. *Operations Research*, v. 5, p. 803–834, 1983.

ELLERY, A.; PATEL, N.; RICHTER, L.; BERTRAND, R.; DALCOMO, J. Exomars Rover Chassis Analysis and Design. In: *Proceedings of the 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*. [S.l.: s.n.], 2005.

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

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

FLAUTNER, K.; FLYNN, D.; RIVES, M. A Combined Hardware-Software Approach for Low-Power SoCs: Applying Adaptive Voltage Scaling and Intelligent Energy Management Software. In: *Proceedings of the 29th European Solid-State Circuits Conference*. [S.l.: s.n.], 2003.

FU, Y.; KOTTENSTETTE, N.; CHEN, Y.; LU, C.; KOUTSOUKOS, X.; WANG, H. Feedback Thermal Control for Real-Time Systems. In: *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. [S.l.: s.n.], 2010. p. 111–120.

GAREY, M.; JOHNSON, D. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. [S.l.]: W.H. Freeman and Company, 1979.

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

GU, J.; QU, G. Incorporating Temperature-Leakage Interdependency Into Dynamic Voltage Scaling for Real-Time Systems. In: *Proceedings of the 24th IEEE International Conference on Application-Specific Systems, Architectures and Processors*. [S.l.: s.n.], 2013. p. 289–296.

GUTHAUS, M. R.; RINGENBERG, J. S.; ERNST, D.; AUSTIN, T. M.; MUDGE, T.; BROWN, R. B. Mibench: A Free, Commercially Representative Embedded Benchmark Suite. In: *IEEE International Workshop on Workload Characterization*. [S.l.: s.n.], 2001. p. 3–14.

HAMMER, P. L.; JOHNSON, E. L.; PELED, U. N. Facet of regular 0–1 polytopes. *Mathematical Programming*, Springer, v. 8, n. 1, p. 179–206, 1975.

HANGAN, A.; SEBESTYEN, G. RTMultiSim: A Versatile Simulator for Multiprocessor Real-Time Systems. In: *Proceedings of The 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, Pisa, Italy*. [S.l.: s.n.], 2012. p. 15.

JEJURIKAR, R.; PEREIRA, C.; GUPTA, R. Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems. In: ACM. *Proceedings of the 41st annual Design Automation Conference*. [S.l.], 2004. p. 275–280.

KHAN, J.; BILAVARN, S.; BELLEUDY, C. Energy Analysis of a DVFS Based Power Strategy on ARM Platforms. In: *Faible Tension Faible Consommation*. [S.l.: s.n.], 2012. p. 1 – 4.

KHAN, J.; BILAVARN, S.; BELLEUDY, C. Impact of Operating Points on DVFS Power Management. In: *Design Technology of Integrated Systems in Nanoscale Era*. [S.l.: s.n.], 2012. p. 1 – 6.

KIM, W.; KIM, J.; MIN, S. A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. [S.l.: s.n.], 2002.

KYUNG, C.; YOO, S. *Energy-Aware System Design*. [S.l.]: Springer, 2011.

LIAO, W.; HE, L.; LEPAK, K. Temperature and Supply Voltage Aware Performance and Power Modeling at Microarchitecture Level. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 24, n. 7, p. 1042–1053, 2005.

LIN, C.; BRANDT, S. A. Improving Soft Real-Time Performance Through Better Slack Reclaiming. In: *In Proceedings 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: *Proceedings of the 12th Euromicro Conference on Real-Time Systems*. [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. 47–61, 1973.

LIU, J. *Real-Time Systems*. [S.l.]: Prentice Hall, 2000.

LIU, S.; WU, Q.; QIU, Q. An Adaptive Scheduling and Voltage/Frequency Selection Algorithm for Real-Time Energy Harvesting Systems. In: *Proceedings of the ACM/IEEE 46th Conference on Design Automation*. [S.l.: s.n.], 2009. p. 782–787.

LIU, Y.; DICK, R.; SHANG, L.; YANG, H. Accurate Temperature-Dependent Integrated Circuit Leakage Power Estimation is Easy. In: *Proceedings of the Conference on Design Automation and Test in Europe*. [S.l.: s.n.], 2007. p. 1526–1531.

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.

MOBILE ROBOTS. *Pioneer 3 Operations Manual*. [S.l.], 2010.

NASSIFFE, R.; CAMPONOGARA, E.; LIMA, G. Optimizing Quality of Service in Real-time Systems Under Energy Constraints. *ACM SIGOPS Operating Systems Review*, v. 46, n. 1, p. 82 – 92, 2012.

NASSIFFE, R.; CAMPONOGARA, E.; LIMA, G. Optimizing QoS in Energy-Aware Real-Time Systems. *ACM SIGBED Review*, v. 10, n. 2, p. 25–25, 2013.

NASSIFFE, R.; CAMPONOGARA, E.; LIMA, G.; MOSSÉ, D. Optimizing QoS in Adaptive Real-Time Systems With Energy Constraint Varying CPU Frequency. In: BRAZILIAN SYMPOSIUM ON COMPUTING SYSTEMS ENGINEERING, V. Manaus, 2013.

NELIS, V.; ANDERSSON, B.; MARINHO, J.; PETTERS, S. M. Global-EDF Scheduling of Multimode Real-Time Systems Considering Mode Independent Tasks. In: *23rd Euromicro Conference on Real-Time Systems*. [S.l.: s.n.], 2011. p. 205–214.

NIU, L. Energy Efficient Scheduling for Real-Time Embedded Systems with QoS Guarantee. *Real-Time Systems*, Springer, v. 47, n. 2, p. 75–108, 2011.

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.

PADBERG, M. Approximating Separable Nonlinear Functions via Mixed Zero-one Programs. *Operations Research Letters*, v. 27, n. 1, p. 1 – 5, 2000.

PADBERG, M. W. On the facial structure of set packing polyhedra. *Mathematical programming*, Springer, v. 5, n. 1, p. 199–215, 1973.

QUAN, G.; HU, X. Energy Efficient Fixed-priority Scheduling for Real-time Systems on Variable Voltage Processors. In: *Proceedings of the Design Automation Conference*. [S.l.: s.n.], 2001. p. 828–833.

RAJAN, D.; YU, P. On Temperature-Aware Scheduling for Single-Processor Systems. In: ALURU, S.; PARASHAR, M.; BADRINATH, R.; PRASANNA, V. (Ed.). *High Performance Computing*. [S.l.]: Springer, 2007. p. 342–355.

RAO, R.; VRUDHULA, S. Performance Optimal Processor Throttling Under Thermal Constraints. In: *Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. [S.l.: s.n.], 2007. p. 257–266.

REAL, J.; CRESPO, A. Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. *Real-Time Systems*, Springer, v. 26, p. 161–197, 2004.

ROBOTS, M. *Mobile Robots*. 2014. Disponível em: <http://www.mobilerobots.com>.

RUSU, C.; MELHEM, R.; MOSSÉ, D. Maximizing Rewards for Real-Time Applications with Energy Constraints. *ACM Transactions on Embedded Computing Systems*, v. 2, n. 4, p. 537–559, 2003.

RUSU, C.; MELHEM, R.; MOSSÉ, D. Maximizing the System Value While Satisfying Time and Energy Constraints. *IBM journal of research and development*, v. 47, n. 13, p. 689–702, 2003.

RUSU, C.; MELHEM, R.; MOSSÉ, D. Multi-Version Scheduling in Rechargeable Energy-Aware Real-Time Systems. *Journal of Embedded Computing*, IOS Press, v. 1, n. 2, p. 271–283, 2005.

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.

SHERALI, H. D. On Mixed-Integer Zero-One Representations for Separable Lower-Semicontinuous Piecewise-Linear Functions. *Operations Research Letters*, v. 28, n. 4, p. 155 – 160, 2001.

SHIN, Y.; CHOI, K.; SAKURAI, T. Power Optimization of Real-time Embedded Systems on Variable Speed Processors. In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. [S.l.: s.n.], 2000. p. 365–368.

SHOP, R. 2012. Disponível em: <http://www.robotshop.com>.

SOUSA, J. P.; WOLSEY, L. A. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical programming*, Springer, v. 54, n. 1-3, p. 353–367, 1992.

SPURI, M.; BUTTAZZO, G. Efficient Aperiodic Service Under Earliest Deadline Scheduling. In: *Proceedings of Real-Time Systems Symposium*. [S.l.: s.n.], 1994. p. 2 – 11.

URUNUELA, R.; DEPLANCHE, A.; TRINQUET, Y. Storm a Simulation Tool for Real-Time Multiprocessor Scheduling Evaluation. In: *IEEE Conference on Emerging Technologies and Factory Automation*. [S.l.: s.n.], 2010. p. 1–8.

WANG, S.; BETTATI, R. Reactive Speed Control in Temperature-Constrained Real-Time Systems. *Real-Time Systems*, Springer, v. 39, n. 1-3, p. 73–95, 2008.

WOLSEY, L. *Integer Programming*. New York: Wiley, 1998. v. 42.

WOLSEY, L. A. Faces for a linear inequality in 0–1 variables. *Mathematical Programming*, Springer, v. 8, n. 1, p. 165–178, 1975.

XIAN, C.; LU, Y.-H.; LI, Z. Dynamic Voltage Scaling for Multitasking Real-Time Systems with Uncertain Execution Time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 27, n. 8, p. 1467 –1478, aug 2008.

XU, R.; MOSSÉ, D.; MELHEM, R. Minimizing Expected Energy Consumption in Real-Time Systems Through Dynamic Voltage Scaling. *ACM Transactions on Computer Systems*, v. 25, n. 4, dez. 2007.

YAO, F.; DEMERS, A.; SHENKER, S. A Scheduling Model for Reduced CPU Energy. In: *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*. [S.l.: s.n.], 1995. p. 374–382.

YUAN, L.; LEVENTHAL, S.; QU, G. Temperature-Aware Leakage Minimization Technique for Real-Time Systems. In: *Proceedings of the IEEE/ACM International Conference on Computer-aided Design*. [S.l.: s.n.], 2006. p. 761–764.

ZAPATER, M.; AYALA, J.; MOYA, J.; VAIDYANATHAN, K.; GROSS, K.; COSKUN, A. Leakage and Temperature Aware Server Control for Improving Energy Efficiency in Data Centers. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. [S.l.: s.n.], 2013. p. 266–269.

ZHAO, B.; AYDIN, H. Minimizing Expected Energy Consumption Through Optimal Integration of DVS and DPM. In: *Proceedings of the 2009 International Conference on Computer-Aided Design*. [S.l.: s.n.], 2009. p. 449 –456.

ZHAO, B.; AYDIN, H.; ZHU, D. On Maximizing Reliability of Real-Time Embedded Applications Under Hard Energy Constraint. *IEEE Transactions on Industrial Informatics*, v. 6, n. 3, p. 316–328, 2010.

ZHU, D.; MELHEM, R.; MOSSÉ, D. The Effects of Energy Management on Reliability in Real-Time Embedded Systems. *International Conference on Computer-Aided Design*, IEEE Computer Society, v. 0, p. 35–40, 2004.

ZHU, D.; MELHEM, R.; MOSSÉ, D. The Effects of Energy Management on Reliability in Real-Time Embedded Systems. In: *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*. [S.l.: s.n.], 2004. p. 35–40.

ZHU, Y.; MUELLER, F. Feedback EDF Scheduling of Real-Time Tasks Exploiting Dynamic Voltage Scaling. *Real-time Systems*, Springer, v. 31, n. 1-3, p. 33–63, 2005.

ZHUO, J.; CHAKRABARTI, C. Energy-Efficient Dynamic Task Scheduling Algorithms for DVS Systems. *ACM Trans. Embed. Comput. Syst.*, ACM, v. 7, n. 2, p. 17–42, 2008.

APÊNDICE A CONCEITOS DE OTIMIZAÇÃO

A.1 OTIMIZAÇÃO

Um problema de otimização matemática pode ser expresso como:

$$\begin{aligned} P: \quad & \min\ f = f_0(\mathbf{x}) \\ & \text{s.a}: \mathbf{x} \in \mathcal{X} \end{aligned}$$

sendo o vetor $\mathbf{x} = (x_1, ..., x_n)$ a variável a ser otimizada ou de decisão, $f_0 : \mathbb{R}^n \to \mathbb{R}$ é a função que se deseja otimizar e $\mathcal{X}$ o conjunto de soluções possíveis, sendo este conjunto delimitado por funções chamadas de restrições. Tais restrições podem ser funções $f_i : \mathbb{R}^n \to \mathbb{R}, i = 1, ..., m$ de igualdade ou desigualdade. Diz-se que o vetor $\mathbf{x}^\star$ é ótimo para o problema se obedece todas as restrições, ou seja, se está no conjunto $\mathcal{X}$, e para qualquer outro vetor $\mathbf{z}$ que satisfaça as restrições tem-se $f(\mathbf{z}) \geq f_0(\mathbf{x}^\star)$.

Existem diferentes classes de problemas de otimização, de acordo com as funções que compõem o problema. Por exemplo, uma importante família de problemas da otimização é a dos problemas lineares, que é composta por uma função objetivo e restrições que são lineares. Uma função f_i é dita linear se satisfaz:

$$f_i(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha f_i(\mathbf{x}) + \beta f_i(\mathbf{y}), \forall x, y \in dom f_i \text{ e } \forall \alpha, \beta \in \mathcal{R} \qquad \text{(A.2)}$$

A.2 CONJUNTOS CONVEXOS

Considere quaisquer dois vetores $\mathbf{x}_1$, $\mathbf{x}_2 \in \mathbb{R}^n$. O vetor $\mathbf{x}$ é dito combinação linear de $\mathbf{x}_1$ e $\mathbf{x}_2$ se existem escalares θ_1 e $\theta_2 \in \mathbb{R}$ tal que:

$$\mathbf{x} = \theta_1\mathbf{x}_1 + \theta_2\mathbf{x}_2 \qquad \text{(A.3)}$$

isso pode ser visto na Figura 33.

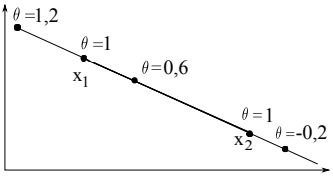
O vetor $\mathbf{x}$ é dito combinação afim de $\mathbf{x}_1$ e $\mathbf{x}_2$ se existem os escalares θ_1, $\theta_2 \in \mathbb{R}$ tais que:

$$\mathbf{x} = \theta_1\mathbf{x}_1 + \theta_2\mathbf{x}_2 \qquad \text{(A.4)}$$

$$\theta_1 + \theta_2 = 1 \qquad \text{(A.5)}$$

O vetor $\mathbf{x}$ é dito combinação convexa de x_1 e x_2, se existem escalares

Figura 33: Na figura acima é ilustrada uma reta que passa pelos vetores $\mathbf{x}_1$ e $\mathbf{x}_2$. Considera-se que $\theta_1 = \theta$ e $\theta_2 = 1 - \theta$ sendo a reta descrita por $\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2$, com $\theta \in \mathbb{R}$. O segmento de reta, a parte mais escura, entre os vetores $\mathbf{x}_1$ e $\mathbf{x}_2$ corresponde ao valor de θ entre 1 e 0.



Fonte: Criado pelo Autor.

θ_1 e θ_2 tal que:

$$\mathbf{x} = \theta_1\mathbf{x}_1 + \theta_2\mathbf{x}_2 \tag{A.6}$$

$$\theta_1 + \theta_2 = 1 \tag{A.7}$$

$$\theta_1, \theta_2 \geq 0 \tag{A.8}$$

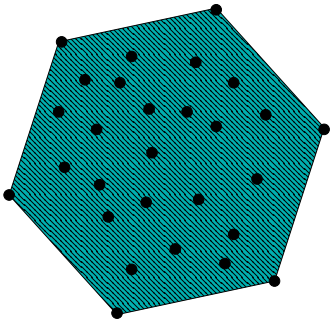
Os conceitos mostrados acima podem ser estendidos para mais de dois vetores, ficando o vetor $\mathbf{x} = \theta_1\mathbf{x_1} + ... + \theta_k\mathbf{x}_k$, com $\theta_1 + ... + \theta_k = 1$ e $\theta_1, ..., \theta_k \leq 0$. De forma que $\mathbf{x}$ é uma combinação convexa de $\mathbf{x}_1, ..., \mathbf{x}_k$.

O invólucro convexo de um conjunto $\mathcal{C}$ é denotado por conv $\mathcal{C}$ e representa o menor conjunto convexos que contém $\mathcal{C}$:

$$\text{conv } \mathcal{C} = \{\theta_1\mathbf{x}_1 + ... + \theta_k\mathbf{x}_k | \mathbf{x}_i \in \mathcal{C}, \theta_i \geq 0, i = 1, ..., k,\ \theta_1 + ... + \theta_k = 1\} \tag{A.9}$$

O invólucro convexo de um conjunto sempre será convexo e se $\mathcal{B}$ for um conjunto convexo que contém $\mathcal{C}$ então conv $\mathcal{C} \subseteq \mathcal{B}$. Na Figura 34 tem-se o exemplo de um conjunto que é um invólucro convexo.

Figura 34: Esta figura é o exemplo de um invólucro convexo em $\mathbb{R}^2$.



Fonte: Criado pelo Autor.

A.3 FUNÇÕES CONVEXAS

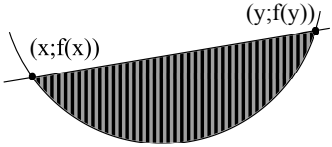
Definição 1. $\mathcal{C} \subseteq \mathbb{R}^n$ é um conjunto convexo se, e se somente se, para todo $\mathbf{x}_1$ e $\mathbf{x}_2 \in \mathcal{C}$ e $\theta \in [0,1]$, $\mathbf{x} = \theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2 \in \mathcal{C}$. Ou seja, $\mathcal{C}$ é convexo se o segmento de reta que une quaisquer dois pontos de $\mathcal{C}$ está totalmente dentro de $\mathcal{C}$. □

Definição 2. $f : \mathbb{R}^n \to \mathbb{R}$ é uma função convexa se, e somente se, o domínio de f denotado por dom$f$ é um conjunto convexo e para todo $x, y \in$ dom$f$ e $\theta \in [0,1]$, tem-se:

$$f(\theta x + (1-\theta)y \leq \theta f(x) + (1-\theta)f(y)) \tag{A.10}$$

Geometricamente esta desigualdade significa que o segmento de reta entre $(\mathbf{x}, f(\mathbf{x}))$ e $(\mathbf{y}, f(\mathbf{y}))$ situa-se acima do gráfico de f, como mostra a Figura 35. □

Figura 35: Na figura acima é ilustrado o gráfico de uma função convexa, onde o segmento de reta entre $f(\mathbf{x})$ e $f(\mathbf{y})$ fica acima do gráfico.



Fonte: Criado pelo Autor.

Teorema 1. *Se f é uma função derivável, então f é convexa se e somente:*

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \tag{A.11}$$

para todo $x, y \in \mathrm{dom} f$. □

Teorema 2. *Se f é duas vezes derivável, então f é convexa se, e somente se:*

$$\nabla^2 f(x) \geq 0, \forall\ x \in\ dom\ f. \tag{A.12}$$

□

A.4 OTIMIZAÇÃO CONVEXA

O problema de otimização definido como segue:

$$P: \quad \min\ f_0(\mathbf{x}) \tag{A.13a}$$
$$\text{s.a}: \mathbf{x} \in \mathcal{X} \tag{A.13b}$$

é convexo se f_0 é uma função convexa e o conjunto de soluções é convexo.

Um problema de maximização pode ser resolvido com métodos de otimização convexa, desde que sua função objetivo seja côncava e as restrições induzam conjuntos convexos. Uma função côncava pode ser facilmente transformada em convexa, se multiplicada por menos um. Logo o problema de maximizar $f_0(\mathbf{x})$ é igual ao de minimizar $-f_0(\mathbf{x})$.

A condição de otimalidade de um problema convexo é estabelecida no seguinte teorema:

Teorema 3. *$\mathbf{x}^\star \in \mathcal{X}$ é um ótimo global para o problema A.13 se e somente se:*

$$\nabla f(\mathbf{x}^\star)^T (\mathbf{y} - \mathbf{x}^\star) \geq 0 \text{ para todo } \mathbf{y} \in \mathcal{X}. \tag{A.14}$$

□

Em um problema convexo, todo o mínimo local é um mínimo global (BOYD; VANDENBERGHE, 2004).

A.5 OTIMALIDADE

Em um problema de minimização pode existir mais de uma solução que obedeça às restrições, entretanto a função objetivo tem um valor mínimo. É dito que geralmente existe uma solução que minimiza ao máximo o problema, mas dependendo do problema pode existir mais de uma solução que induz o valor mínimo da função objetivo. Tal solução que chega ao ponto de mínimo do problema é chamada de solução ótima global.

Devido às características do problema de otimização, pode ser muito custoso garantir que a solução encontrada seja o ótimo global. Por isso em alguns casos são empregadas heurísticas. Se a solução é ótima local, pode-se afirmar que para sua vizinha ela é a melhor solução existente. Matematicamente uma solução $x^\star$ é definida como ótimo global em $f(x)$ se para todos os outros x do conjunto solução a afirmação $f(x) \geq f(x^\star)$ for verdadeira. Quando essa condição só for provada para um $\varepsilon > 0$, tal que $||x - x^\star|| < \varepsilon$ tem-se um ótimo local.

Se não for possível garantir o ótimo global para o problema, o ponto inicial influencia no resultado encontrado. Por isso às vezes, mesmo que um problema tenha uma solução, ela pode não ser encontrada. Por outro lado, quando se trabalha com problemas onde é possível encontrar o ponto ótimo global, é garantido que independentemente do ponto inicial o problema pode ser resolvido até sua otimalidade.

A.6 RELAXAÇÕES

Seja $\mathbf{x} = (x_i^{k,j} : (i,k,j) \in \Omega)$ um vetor com todas as variáveis do problema P e seja $\mathcal{P} = \{\mathbf{x} : \mathbf{x}$ satisfaz as equações (4.1b) até (4.1e)$\}$ o espaço de soluções factíveis. Então, P é representado de forma compacta por $f^\star = \max\{f(\mathbf{x}) : \mathbf{x} \in \mathcal{P}\}$. Um problema R, definido por $r^\star = \max\{r(\mathbf{x}) : \mathbf{x} \in \mathcal{R}\}$, é uma relaxação de P se (WOLSEY, 1998):

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

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

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 P. Isto significa que $r^\star \geq f^\star$. Relaxações são elementos chave para se estabelecer certificados de qualidade para uma solução factível[1],

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

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 relaxações.

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

Considere o problema geral de programação inteira dado por:

$$P_I: \quad f^\star = \max \mathbf{c}^T\mathbf{x} \tag{A.15a}$$
$$\text{s.a}: A\mathbf{x} \leq \mathbf{b} \tag{A.15b}$$
$$\mathbf{x} \in \mathbb{Z}^n \tag{A.15c}$$

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

$$R: \quad r^\star = \max \mathbf{c}^T\mathbf{x} \tag{A.16a}$$
$$\text{s.a}: A\mathbf{x} \leq \mathbf{b} \tag{A.16b}$$
$$\mathbf{x} \in \mathbb{R}^n \tag{A.16c}$$

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:

$$R_L: \quad r^\star = \min_{\boldsymbol{\lambda}\geq\mathbf{0}} \max_{\mathbf{x}\in\mathbb{Z}^n} \mathbf{c}^T\mathbf{x} - \lambda^T(A\mathbf{x} - \mathbf{b}) \tag{A.17}$$

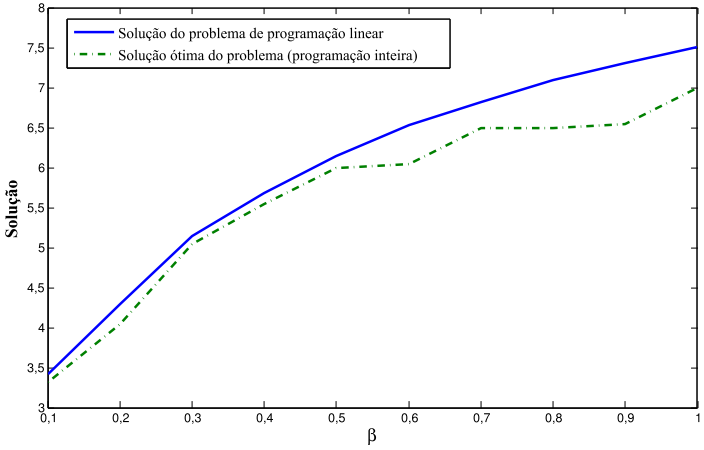
Por fim, a relaxação *surrogate* é induzida com um vetor de pesos $\mathbf{w} \geq \mathbf{0}$,

$$R_S: \quad r^\star = \max \mathbf{c}^T\mathbf{x} \tag{A.18a}$$
$$\text{s.a}: \mathbf{w}^TA\mathbf{x} \leq \mathbf{w}^T\mathbf{b} \tag{A.18b}$$
$$\mathbf{x} \in \mathbb{Z}^n \tag{A.18c}$$

Considere o problema de reconfiguração dado pelas equações (4.1a)-(4.1e). 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, $\mathcal{R} = \{\mathbf{x} : \mathbf{x}$ satisfaz (4.1b) a (4.1d), $\mathbf{0} \leq \mathbf{x} \leq \mathbf{1}\}$. Claramente $\mathcal{P} \subseteq \mathcal{R}$. Se definirmos $r(\mathbf{x}) = f(\mathbf{x})$, então a segunda condição é também satisfeita. Portanto, o problema R resultante, definido por $r^\star = \max\{r(\mathbf{x}) : \mathbf{x} \in \mathcal{R}\}$, é uma relaxação válida de P.

Resolvendo a relaxação do problema 5.5 com uma instância qualquer com um algoritmo de programação linear, encontraremos o valor $r^\star \geq f(\mathbf{x}^\star) \geq f(\mathbf{x}), \forall \mathbf{x} \in \mathcal{P}$, onde $\mathbf{x}^\star$ é a solução ótima de P. Como pode ser visto na Figura 36, a solução encontrada com a solução da relaxação contínua R gerou um limite superior, sempre maior o valor da solução ótima do problema original.

Figura 36: Gráfico com resultados das soluções do problema de reconfiguração dinâmica e da sua relaxação linear, considerando a instância exemplo com variação de β.



Fonte: Criado pelo Autor.

A.7 DESIGUALDADES DE COBERTURA

Uma cobertura é um conjunto solução para um problema que excede as restrições. Para ilustrar o caso, considere o problema da mochila, com uma dimensão onde tem-se o conjunto solução delimitado por $\mathcal{X} = \{x \in \mathbb{B}^n : \sum_{j=1}^{n} a_j x_j \leq b\}$ e $\mathcal{N} = \{1, ..., n\}$.

Definição 3. Um conjunto qualquer $\mathcal{C} \subseteq \mathcal{N}$ tal que $\sum_{j=1} a_j > b$, é dito cobertura. A cobertura será mínima se para todo $j \in C$, $C \backslash \{j\}$ não for uma cobertura. □

Proposição 1. *Dada uma cobertura* $\mathcal{C} \subset \mathcal{N}$*, a desigualdade:* $\sum_{j \in \mathcal{C}} x_j \leq |\mathcal{C}| - 1$ *é válida para* $\text{conv}(\mathcal{X})$.

Com intuito de exemplificar uma cobertura com o problema da mochila, considere o conjunto de soluções $X = \{x \in \mathbb{B}^4 : 2x_1 + 3x_2 + 4x_3 + 2x_4 \leq 7\}$. Para tal conjunto $\mathcal{C} = \{1; 2; 3\}$ é uma cobertura, pois $2 + 3 + 4 = 9$ e o valor desse somatório é maior que 7. A cobertura $\mathcal{C}$ também é mínima, uma vez que os subconjuntos formados por $2 + 3 \leq 7$, $2 + 4 \leq 7$ e $3 + 4 \leq 7$, ou seja, obedecem a restrição da mochila. Com essa cobertura ainda pode-se inferir a desigualdade válida $x_1 + x_2 + x_3 \leq 2$.

O uso de coberturas é uma ferramenta importante para identificar famílias de desigualdades válidas ou a até mesmo o conjunto $\text{conv}(\mathcal{X})$. As coberturas podem ser utilizadas para explorar características específicas de um problema na construção de heurísticas ou algoritmos exatos.

APÊNDICE B ABORDAGEM INTEIRA

B.1 REFORMULAÇÃO COMPACTA

Como dito anteriormente, as heurísticas não garantem encontrar a resposta ótima de um problema, mas o uso de um solver garante. Entretanto ele aumenta muito o tempo de resolução do problema, como mostrado em (NASSIFFE; CAMPONOGARA; LIMA, 2012). Esse fato motivou uma investigação teórica, com a proposição de cortes especializados para diminuir o tempo da solução. Contudo, a falta de softwares para a plataforma embarcada usada no trabalho tornou inviável a implementação para verificar a performance, assim ficando a investigação computacional dos cortes para um trabalho futuro.

Objetivando o desenvolvimento de algoritmos especializados, o problema reconfiguração definido nas equações (4.1) pode ser expresso em uma forma compacta, por:

$$P: \quad \max \; f = \sum_{i=1}^{n} \sum_{j=0}^{m(i)} c_{i,j} x_{i,j} \tag{B.1a}$$

$$\text{s.a}: \; \sum_{j=0}^{m(i)} x_{i,j} = 1, \; i \in N \tag{B.1b}$$

$$\sum_{i=1}^{n} \sum_{j=0}^{m(i)} a_{i,j}^{l} x_{i,j} \leq b^{l}, \; l = 1, 2 \tag{B.1c}$$

$$x_{i,j} \in \{0, 1\}, \; i \in N, \; j = 0, \ldots, m(i) \tag{B.1d}$$

onde:

- i é o índice de uma tarefa e $N = \{1, \ldots, n\}$ é o conjunto de índices das tarefas do sistema;
- j é o índice que representa um par modo-frequência;
- $m(i) = |\Omega_i| - 1$ define o número de configurações (pares modo e frequência) da tarefa i;
- $a_{i,j}^{1}$ representa a utilização de CPU da configuração j da tarefa i;
- $a_{i,j}^{2}$ é a energia consumida por uma tarefa i quando operando na configuração j;

- $x_{i,j}$ representa a escolha de uma configuração para a tarefa i.

B.1.1 Instância Exemplo

Uma instância exemplo será definida agora com o propósito de ilustrar os conceitos, modelo e algoritmo que serão propostos a seguir. As tarefas utilizadas foram geradas pelo algoritmo *UUniFast* (BINI; BUTTAZZO, 2005), o qual tende a gerar um cenário equilibrado evitando otimismo ou pessimismo.

O exemplo conta com 3 tarefas, podendo elas serem executadas em três modos e frequências distintas. O conjunto de frequências é $\mathcal{F} = \{1,0; 0,75; 0,5\}$, onde cada valor representa a utilização da *CPU*. Supondo um processador de $1,0Ghz$, $f_i = 0,5$ implica que a tarefa está executando com a frequência de $500MHz$. Na Tabela 19 estão os dados sobre o tempos das tarefas e seus k modos.

Tabela 19: Valores estipulados de $(Q^{\mathrm{I}}_{i,k}, Q^{\mathrm{D}}_{i,k}, T_{i,k})$, para cada modo de um servidor (ms).

	Servidores S_i		
	$i = 1$	$i = 2$	$i = 3$
k	$(Q^{\mathrm{D}}_{1,k}, Q^{\mathrm{I}}_{1,k}, T_{1,k})$	$(Q^{\mathrm{D}}_{2,k}, Q^{\mathrm{I}}_{2,k}, T_{2,k})$	$(Q^{\mathrm{D}}_{3,k}, Q^{\mathrm{I}}_{3,k}, T_{3,k})$
1	(0,8548; 3,4877; 33)	(3,4918; 4,1901; 33)	(2,2534; 11,1222; 33)
2	(3,8693; 3,7696; 66)	(2,6438; 4,7122; 66)	(0,0869; 11,3182; 66)
3	(5,1818; 9,3806; 200)	(0,2504; 9,4074; 200)	(1,1678; 1,0121; 200)

Fonte: Criado pelo Autor.

Com os dados da Tabela 19 foi criada a Tabela 20, que contém os dados referentes à utilização de *CPU* por cada configuração possível de uma tarefa.

Os seguintes parâmetros foram usados para calcular o consumo energético: $P^{\mathrm{Leak}} = 0$, $C_0 = 24,99$, $P^I = 0,0043W$ para todas as tarefas e um processador com $2,56Ghz$, com o conjunto de frequências suportadas $F = \{1,0; 0,75; 0,5\}$. Usando esses parâmetros e os dados da Tabela 20, foram calculados os dados referentes ao consumo energético, mostrados na Tabela 21.

A Eq. (4.2a) foi utilizada para calcular o valor do benefício de cada configuração. O valor de W_i foi considerado 1 para todas as tarefas e o resultado está na Tabela 22.

Utilizando o modelo inicial, a instância exemplo com um limite de $b^2 = 20$ Watts foi resolvida com uso do CPLEX. O resultado obtido

Tabela 20: Utilização de cada configuração (k, i) de S_i.

j	$a^1_{1,j}$	$a^1_{2,j}$	$a^1_{3,j}$
$a^1_{i,0}$	$7,3\%$	$4,8\%$	$1,1\%$
$a^1_{i,1}$	$16,7\%$	$27,5\%$	$51,8\%$
$a^1_{i,2}$	$26,1\%$	$38,8\%$	$81,7\%$
$a^1_{i,3}$	$11,6\%$	$11,2\%$	$17,3\%$
$a^1_{i,4}$	$13,5\%$	$13,5\%$	$23,0\%$
$a^1_{i,5}$	$18,6\%$	$19,9\%$	$38,2\%$
$a^1_{i,6}$	$13,2\%$	$23,3\%$	$40,5\%$
$a^1_{i,7}$	$8,8\%$	$6,4\%$	$1,3\%$
$a^1_{i,8}$	$13,0\%$	$10,6\%$	$1,7\%$

Fonte: Criado pelo Autor.

Tabela 21: Energia consumida por unidade de tempo de cada configuração (k, i) de S_i.

	$a^2_{i,j}$ (Watts)								
i	$a^2_{i,0}$	$a^2_{i,1}$	$a^2_{i,2}$	$a^2_{i,3}$	$a^2_{i,4}$	$a^2_{i,5}$	$a^2_{i,6}$	$a^2_{i,7}$	$a^2_{i,8}$
1	$1,84$	$1,78$	$0,60$	$2,93$	$1,44$	$0,43$	$3,33$	$0,94$	$0,30$
2	$1,22$	$2,94$	$0,90$	$2,82$	$1,44$	$0,46$	$5,89$	$0,68$	$0,24$
3	$0,28$	$5,96$	$2,04$	$4,37$	$2,46$	$0,88$	$11,02$	$0,13$	$0,04$

Fonte: Criado pelo Autor.

Tabela 22: Benefício de cada configuração.

	$c_{i,j}$								
i	$c_{i,0}$	$c_{i,1}$	$c_{i,2}$	$c_{i,3}$	$c_{i,4}$	$c_{i,5}$	$c_{i,6}$	$c_{i,7}$	$c_{i,8}$
1	$0,88$	$2,50$	$1,90$	$1,00$	$0,97$	$0,93$	3	$0,80$	$0,70$
2	$0,48$	$2,50$	$1,90$	$0,99$	$0,87$	$0,71$	3	$0,41$	$0,33$
3	$0,40$	$2,50$	$1,90$	$0,99$	$0,85$	$0,67$	3	$0,30$	$0,19$

Fonte: Criado pelo Autor.

mostra que a configuração ótima é $x_{1,1} = 1$, $x_{2,6} = 1$ e $x_{3,6} = 1$. Essa configuração gera um benefício agregado de $8,5$ unidades, utilização de *CPU* $80,50\%$ e consumo energético previsto de $18,69$ Watts.

B.2 ABORDAGEM DE PROGRAMAÇÃO INTEIRA

Aqui desenvolve-se uma reformulação do problema de reconfiguração dinâmica a partir da remoção das dependências lineares, o que facilita a análise da dimensionalidade do poliedro de formulação e a síntese de desigualdades válidas que podem ser empregadas em um algoritmo de planos de corte.

Para simplificar a análise, assume-se conhecida uma configuração inicial factível para a instância de reconfiguração dinâmica. Tal hipótese é estabelecida a seguir e pode ser assegurada pelo projetista do sistema de tempo real.

Hipótese 1. $\mathbf{x} = ((i,0) : i \in N)$ *é uma* configuração factível mínima *para qualquer sistema tempo-real. Por factível entende-se que:*

$$\sum_{i=1}^{n} a_{i,0}^{l} \leq b^{l},\ l = 1,2 \tag{B.2a}$$

Por mínima entende-se que para todo (i,j), $i \in N$ *e* $j \in M(i) = \{1,\ldots,m(i)\}$, *temos que:*

$$\sum_{t \in N \setminus \{i\}} a_{t,0}^{l} + a_{i,j}^{l} \leq b^{l},\ l = 1,2 \tag{B.2b}$$

Em outras palavras, a configuração é mínima se a configuração inicial $(i,0)$ de qualquer tarefa i pode ser substituída por qualquer outra configuração (i,j) sem tornar infactível a configuração resultante. No caso da instância exemplo dada nas Tabelas 20 e 21, o conjunto $I = \{(1,0),(2,0),(3,0)\}$ leva a um consumo de *CPU* de 13,2 % e um consumo de energia de 3,34 Watts, abaixo do limite de 20 Watts. A condição (B.2b) de configuração mínima pode ser verificada como válida.

Podemos eliminar a equação (B.1b) substituindo $x_{i,0}$ pela expressão abaixo:

$$x_{i,0} = 1 - \sum_{j \in M(i)} x_{i,j}$$

o que nos leva ao seguinte problema de reconfiguração equivalente:

$$\tilde{P}: \quad \max \ f = \sum_{i \in N} \sum_{j \in M(i)} \tilde{c}_{i,j} x_{i,j} + \sum_{i \in N} c_{i,0} \tag{B.3a}$$

$$\text{s.a}: \sum_{i \in N} \sum_{j \in M(i)} \tilde{a}^l_{i,j} x_{i,j} \leq \tilde{b}^l, \ l = 1, 2 \tag{B.3b}$$

$$\sum_{j \in M(i)} x_{i,j} \leq 1, \ i \in N \tag{B.3c}$$

$$x_{i,j} \in \{0; 1\}, \ i \in N, \ j \in M(i) \tag{B.3d}$$

onde:

- $\tilde{c}_{i,j} = c_{i,j} - c_{i,0}$ é o acréscimo de benefício induzido pela configuração j em relação à configuração inicial da tarefa i;
- $\tilde{a}^l_{i,j} = a^l_{i,j} - a^l_{i,0}$ é o consumo de recurso l (CPU ou energia) em relação à configuração inicial;
- $\tilde{b}^l = b^l - \sum_{i \in N} a^l_{i,0}$ é a quantidade de recurso l residual após a seleção da configuração mínima das tarefas.

Note que o problema $\tilde{P}$ pode ser colocado como um problema de programação inteira em uma forma compacta:

$$\tilde{P}: \quad \max \ f = \mathbf{c}^T \mathbf{x} \tag{B.4a}$$

$$\text{s.a}: \mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{B.4b}$$

$$\mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \tag{B.4c}$$

$$\mathbf{x} \in \mathbb{Z}^p \tag{B.4d}$$

onde $p = \sum_{i \in N} m(i)$ é a dimensão do espaço de decisão. Note que o poliedro $\mathcal{P} = \{\mathbf{x} \in \Re^p : A\mathbf{x} \leq \mathbf{b}, \mathbf{0} \leq \mathbf{x} \leq \mathbf{1}\}$ define uma formulação do problema de reconfiguração pois o conjunto de soluções $\mathcal{X}$ é precisamente $\mathcal{X} = \mathcal{P} \cap \mathbb{Z}^p$.

B.2.1 Adaptação da Instância Exemplo

A partir da configuração inicial, os parâmetros da instância exemplo foram recalculados conforme apresentado na Tabela 23 para utilização de CPU, na Tabela 24 para consumo de energia e na Tabela 25 para benefício. A configuração inicial demanda uma utilização de 13,2% de CPU e um consumo de 3,3397 Watts, portanto $\tilde{b}^1 = 86,8\%$ e $\tilde{b}^2 = 16,6603$.

Tabela 23: Utilização de cada modo dos servidores S_i.

	$\tilde{a}^1_{i,j}$							
i	$\tilde{a}^1_{i,1}$	$\tilde{a}^1_{i,2}$	$\tilde{a}^1_{i,3}$	$\tilde{a}^1_{i,4}$	$\tilde{a}^1_{i,5}$	$\tilde{a}^1_{i,6}$	$\tilde{a}^1_{i,7}$	$\tilde{a}^1_{i,8}$
1	$8,88$	$18,28$	$3,77$	$5,68$	$10,75$	$5,36$	$1,04$	$5,21$
2	$22,68$	$33,97$	$6,32$	$8,70$	$15,04$	$18,45$	$1,57$	$5,75$
3	$50,68$	$80,64$	$16,19$	$21,91$	$37,15$	$39,44$	$0,17$	$0,62$

Fonte: Criado pelo Autor.

Tabela 24: Energia consumida por unidade de tempo de cada configuração S_i.

	$\tilde{a}^2_{i,j}(\%)$							
i	$\tilde{a}^2_{i,1}$	$\tilde{a}^2_{i,2}$	$\tilde{a}^2_{i,3}$	$\tilde{a}^2_{i,4}$	$\tilde{a}^2_{i,5}$	$\tilde{a}^2_{i,6}$	$\tilde{a}^2_{i,7}$	$\tilde{a}^2_{i,8}$
1	$-0,06$	$-1,24$	$1,09$	$-0,40$	$-1,41$	$1,49$	$-0,90$	$-1,54$
2	$1,72$	$-0,33$	$1,60$	$0,22$	$-0,76$	$4,67$	$-0,54$	$-0,98$
3	$5,68$	$1,77$	$4,10$	$2,18$	$0,61$	$10,75$	$-0,14$	$-0,24$

Fonte: Criado pelo Autor.

Tabela 25: Benefício agregado a cada configuração.

	$\tilde{c}_{i,j}$							
i	$\tilde{c}_{i,1}$	$\tilde{c}_{i,2}$	$\tilde{c}_{i,3}$	$\tilde{c}_{i,4}$	$\tilde{c}_{i,5}$	$\tilde{c}_{i,6}$	$\tilde{c}_{i,7}$	$\tilde{c}_{i,8}$
1	$1,62$	$1,02$	$0,12$	$0,09$	$0,05$	$2,12$	$-0,08$	$-0,18$
2	$2,02$	$1,42$	$0,52$	$0,39$	$0,23$	$2,52$	$-0,07$	$-0,15$
3	$2,10$	$1,50$	$0,60$	$0,45$	$0,27$	$2,60$	$-0,09$	$-0,20$

Fonte: Criado pelo Autor.

B.2.2 Dimensão do Poliedro da Formulação

Seja o $\mathcal{P} = \{x \in \Re^p : Ax \leq b, 0 \leq x \leq 1\}$ o poliedro correspondente à formulação do problema $\tilde{\mathcal{P}}$ dado em (B.3a)-(B.3d). Isto significa que $\mathcal{P} \cap \mathcal{Z}^p = \mathcal{X}$ onde $\mathcal{X}$ é o conjunto de soluções factíveis de $\tilde{\mathcal{P}}$. Assim sendo, conv($\mathcal{X}$) é o fecho convexo do conjunto de soluções factíveis[1]. Da teoria de programação inteira linear, sabe-se que conv($\mathcal{X}$) = $\{\mathbf{x} \in \Re^p : \tilde{\mathbf{A}}\mathbf{x} \leq \tilde{\mathbf{b}}\}$ é um poliedro que, em geral, não se conhece o sistema de desigualdades que o define. Se conhecêssemos $\tilde{\mathbf{A}}$ e $\tilde{\mathbf{b}}$ o problema de

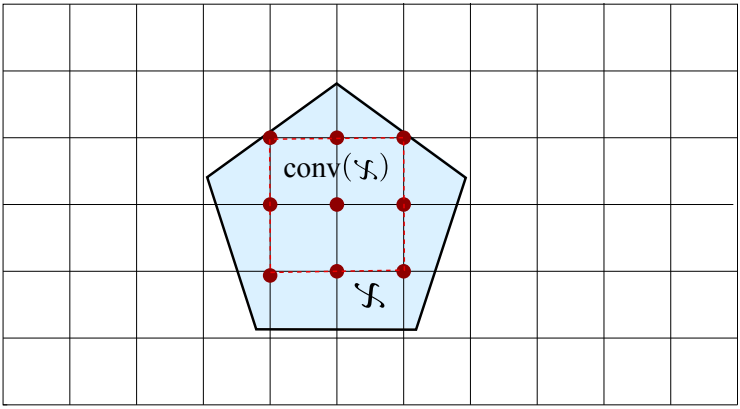
[1] O fecho convexo de $\mathcal{X}$ é o conjunto de todos os vetores obtidos por combinação convexa de um número finito de pontos de $\mathcal{X}$.

reconfiguração se tornaria um programa linear, visto que os vértices de conv($\mathcal{X}$) seriam todos inteiros. Esta propriedade é ilustrada na Figura 37.

Proposição 2. *O poliedro* conv($\mathcal{X}$) *tem dimensão cheia.*

Demonstração. Seja $\widehat{X} \in \Re^{p \times p}$ uma matriz com colunas correspondendo às variáveis de decisão $x_{1,1}$, $x_{1,2}$, ..., $x_{1,m(1)}$, ..., $x_{n,1}$, $x_{n,2}$, ..., $x_{n,m(n)}$ segundo esta ordem. Para cada $i \in N$ e $j \in M(i)$, seja $\widehat{\mathbf{x}}(i,j) \in \Re^p$ o vetor com todas as entradas nulas, exceto a entrada correspondente à variável $x_{i,j}$. Em função da Hipótese 1, $\widehat{\mathbf{x}}(i,j) \in$ conv($\mathcal{X}$). As linhas de $\widehat{X}$ são precisamente os vetores $\widehat{\mathbf{x}}(1,1)^{\mathrm{T}}$, $\widehat{\mathbf{x}}(1,2)^{\mathrm{T}}$, ..., $\widehat{\mathbf{x}}(1,m(1))^{\mathrm{T}}$, ..., $\widehat{\mathbf{x}}(n,1)^{\mathrm{T}}$, $\widehat{\mathbf{x}}(n,2)^{\mathrm{T}}$, ..., $\widehat{\mathbf{x}}(n,m(n))^{\mathrm{T}}$. Obviamente $\widehat{X}$ é uma matriz diagonal e seu posto é p. Uma vez que $\mathbf{0} \in$ conv($\mathcal{X}$) que corresponde à configuração inicial, concluímos que existem $p+1$ vetores afim independentes em conv($\mathcal{X}$), demostrando que dim(conv($\mathcal{X}$)) = p. □

Figura 37: Ilustração do poliedro de formulação $\mathcal{X}$ e o conv($\mathcal{X}$).



Fonte: Criado pelo Autor.

B.2.3 Desigualdades de Cobertura

Tendo em vista que não se conhece a formulação ideal podemos identificar famílias de desigualdades válidas com alta dimensionalidade, ou mesmo facetas de conv($\mathcal{X}$), as quais poderão ser empregadas em um

algoritmo de planos de corte. Aqui desenvolvemos uma família de desigualdades válidas para conv($\mathcal{X}$) tomando como base as desigualdades de cobertura para o problema da mochila (BALAS, 1975; BALAS; ZEMEL, 1978).

Seja $\Omega = \{(i,j) : i \in N,\, j \in M(i)\}$ o conjunto com todos os pares de tarefas e configurações. Seja $A^l(C) = \sum_{(i,j)\in C} \tilde{a}^l_{i,j}$ a quantidade de recursos $l \in \{1,2\}$ consumida pelas tarefas e configurações que fazem parte de um subconjunto $C \subseteq \Omega$. Seja $N(C) = \{i : (i,j) \in C\}$ o conjunto das tarefas que aparecem em C.

Definição 4. $C \subseteq \Omega$ é uma cobertura se:

1. $|N(C)| = |C|$, ou seja, se existir no máximo uma configuração para cada tarefa em C;
2. $A^1(C) > \tilde{b}^1$ ou $A^2(C) > \tilde{b}^2$.

□

O conjunto $C = \{(2,3),(3,2)\}$ define uma cobertura para o problema ilustrativo visto que $|N(C)| = 2 = |C|$ e $A^1(C) = 86,96 > 86,8 = \tilde{b}^1$. O conjunto $C' = \{(1,2),(2,2),(3,1)\}$ também define uma cobertura para o exemplo sendo $|N(C')| = 3$ e $A^1(C') = 102,93$.

Proposição 3. *Uma cobertura C induz a seguinte desigualdade válida para* conv($\mathcal{X}$)*:*

$$\sum_{(i,j)\in C} x_{i,j} \leq |C| - 1 \tag{B.5}$$

Dada uma cobertura C, as tarefas que aparecem em C não podem ser processadas nos seus respectivos modos e frequências sem violar a restrição de escalonabilidade ou de consumo de energia. Seja ainda $C(i,j) = C \setminus \{(i,j)\}$.

Defina $\mathbf{A}(C) = (A^1(C), A^2(C))$ como o vetor com recursos consumidos por C, $\tilde{\mathbf{b}} = (\tilde{b}^1, \tilde{b}^2)$ o vetor com disponibilidade de recursos e $\tilde{\mathbf{a}}_{i,j} = (\tilde{a}^1_{i,j}, \tilde{a}^2_{i,j})$ o vetor com os valores de recursos consumidos por cada configuração das tarefas.

Definição 5. Uma cobertura C^l é dita mínima com respeito ao recurso $l \in L = \{1,2\}$ se:

1. $C^l(i,j)$ não é uma cobertura para todo $(i,j) \in C^l$;
2. $A^l(C^l) > \tilde{b}^l$, porém $A^{\bar{l}}(C^l) \leq \tilde{b}^{\bar{l}}$; e

3. para todo $(i,j) \in C^l$ e todo $(i,k) \in \Omega \setminus C^l$ tal que $\tilde{a}^l_{i,k} < \tilde{a}^l_{i,j}$, $A^l(C') \leq \tilde{b}^l$ para $C' = C^l(i,j) \cup \{(i,k)\}$ quando $A^{\bar{l}}(C') \leq \tilde{b}^{\bar{l}}$.

□

No contexto acima $\bar{l} = 1$ se $l = 2$, caso contrário $\bar{l} = 2$ se $l = 1$. Note que para uma cobertura mínima com respeito ao recurso l tem-se $\tilde{a}^l_{i,j} > 0$ para todo $(i,j) \in C^l$, de outra forma $A^l(C^l(i,j)) \geq A^l(C^l) > \tilde{b}^l$ se $\tilde{a}^l_{i,j} \leq 0$ nos levando a concluir que $C^l(i,j)$ é uma cobertura.

Definição 6. Uma cobertura C é mínima com respeito a todos os recursos se:

1. $\mathbf{A}(C) > \tilde{\mathbf{b}}$;
2. $C(i,j)$ não é uma cobertura para todo $(i,j) \in C$; e
3. para todo $(i,j) \in C$ e todo $(i,k) \in \Omega \setminus C$ tal que $\tilde{\mathbf{a}}_{i,k} < \tilde{\mathbf{a}}_{i,j}$, $C(i,j) \cup \{(i,k))\}$ não é uma cobertura.

□

Em uma cobertura com respeito a todos os recursos, note que $\tilde{\mathbf{a}}_{i,j} \not< 0$ para todo $(i,j) \in C$, pois de outra forma $C(i,j)$ seria uma cobertura.

Considerando o exemplo, note que para a cobertura $C^1 = \{(2,3), (3,2)\}$ tem-se $A^2(C^1) = 1,5982 + 1,7671 = 3,3653 < 16,6603 = \tilde{b}^2$, porém $A^1(C^1) = 6,32 + 80,64 = 86,96 > 86,8 = \tilde{b}^1$. Além disso, para cada elemento $(i,k) \in \{(2,7),(2,8)\}$ são tais que $\tilde{a}^1_{2,k} < \tilde{a}^1_{2,3}$, tem-se $A^2(C') \leq \tilde{b}^2$ e $A^1(C^1) \leq \tilde{b}^1$ com $C = \{(2,k)(3,2)\}$. Tendo em vista que as condições da Definição 5 são satisfeitas, concluímos que C^1 é uma cobertura mínima com respeito ao recurso $l = 1$.

De forma similar nota-se que para $D^1 = \{(1,2),(2,2),(3,1)\}$ tem-se $A^2(D^1) = -1,24 - 0,3257 + 5,6923 = 4,1266 \leq 16.6603 = \tilde{b}^2$ e$A^1(D^1) = 18,28 + 33,97 + 50,68 = 102,93 > \tilde{b}^1$ o que nos leva a concluir que D^1 é uma cobertura. Porém não é mínima com relação a $l = 1$.

Seja $\mathcal{P}(C) = \text{conv}\{\mathbf{x} \in \mathcal{X} : x_{i,j} = 0 \text{ para todo } (i,j) \in \Omega \setminus C\}$ o poliedro correspondente à projeção de $\mathcal{P} = \text{conv}(\mathcal{X})$ sobre o espaço de variáveis de uma cobertura C.

Proposição 4. *A face* $\mathcal{F}(C) = \{\mathbf{x} \in \mathcal{P}(C) : \sum_{(i,j)\in C} x_{i,j} = |C| - 1\}$ *induzida por uma cobertura mínima* C*, com respeito a um ou ambos os recursos, tem dimensão* $|C| - 1$.

Demonstração. Seja $\widehat{\mathbf{x}}(i,j)$ a solução obtida ao definir $x_{l,t} = 1$ para todo $(l,t) \in C(i,j)$, porém $x_{l,t} = 0$ para todo $(l,t) \in \Omega \setminus C(i,j)$. Uma vez que C é uma cobertura mínima, $\widehat{\mathbf{x}}(i,j) \in \mathcal{P}(C)$. O conjunto de soluções $\{\widehat{\mathbf{x}}(i,j) : (i,j) \in C\}$ é afim independente, portanto $\dim(\mathcal{F}(C)) = |C| - 1$. □

Uma vez que a origem é um elemento de $\mathcal{P}(C)$, concluímos que $\mathcal{F}(C) \subset \mathcal{P}(C)$ e portanto $\mathcal{F}(C)$ é uma faceta de $\mathcal{P}(C)$.

B.2.4 *Lifting* de Desigualdades de Cobertura

Uma cobertura mínima C induz uma faceta do poliedro $\mathcal{P}(C)$ que pode ser estendida a uma faceta de $\text{conv}(\mathcal{X})$ por meio do procedimento de *lifting*. Tal procedimento foi introduzido por Padberg em (PADBERG, 1973). Desde então, tem sido muito empregado na síntese de desigualdades para problemas da mochila como em (SOUSA; WOLSEY, 1992; HAMMER; JOHNSON; PELED, 1975; WOLSEY, 1975). Para aplicação do *lifting* é gerada a seguinte desigualdade:

$$\sum_{(i,j)\in C} x_{i,j} + \sum_{(i,j)\in\Omega\setminus C} \alpha_{i,j} x_{i,j} \leq |C| - 1 \tag{B.6}$$

onde $\alpha_{i,j}$ são conhecidos como fatores de *lifting*. A partir de um conjunto ordenado $\bar{C} = \langle (i_1,j_1), \ldots, (i_{p-|C|}, j_{p-|C|}) \rangle$ dos índices $(i,j) \in \Omega \setminus C$, o fator de *lifting* α_{i_t,j_t} é calculado resolvendo o problema:

$$L_t : \ \xi_t = \max \sum_{(i,j)\in C} x_{i,j} + \sum_{l=1}^{t-1} \alpha_{i_l,j_l} x_{i_l,j_l} \tag{B.7a}$$

$$s.t. : \sum_{(i,j)\in C\cup\bar{C}(t)} x_{i,j} \leq 1, \ i \in N(C \cup \bar{C}(t)) \tag{B.7b}$$

$$\sum_{(i,j)\in C\cup\bar{C}(t)} \tilde{\mathbf{a}}_{i,j} x_{i,j} \leq \tilde{\mathbf{b}} - \tilde{\mathbf{a}}_{i_t,j_t} \tag{B.7c}$$

$$x_{i,j} \in \{0,1\}, \ (i,j) \in C \cup \bar{C}(t) \tag{B.7d}$$

onde $\bar{C}(t) = \{(i_1,j_1), \ldots, (i_{t-1}, j_{t-1})\}$ e $\alpha_{i_t,j_t} = |C| - \xi_t - 1$. Para um par $(i,j) = (i_j, j_t) \in \bar{C}$, note que o menor valor ξ_t ocorre quando $t = 1$, ou seja, o fator de *lifting* α_{i_t,j_t} não cresce à medida que t aumenta segundo uma sequência de *lifting*. Assumiremos que *lifting* será realizado apenas nos pares (i,j) com fator não negativo, portanto $\alpha_{i,j} = \max\{|C| - \xi_t - 1, 0\}$ para todo $(i,j) \in \bar{C}$.

Aplicando o procedimento de *lifting* à cobertura mínima $C^1 = \{(2,3),\ (3,2)\}$ com respeito ao recurso 1 segundo a sequência $\bar{C}^1 = \langle(2,1),(2,2),(2,4),\ \ldots,\ (2,8),(3,1),(3,3),\ \ldots,\ (3,8),(1,1),\ldots,\ (1,8)\rangle$, obtemos os seguintes fatores não nulos: $\alpha_{2,1} = 1$, $\alpha_{2,2} = 1$, $\alpha_{2,4} = 1$, $\alpha_{2,5} = 1$ e $\alpha_{2,6} = 1$. Isto resulta na seguinte desigualdade de cobertura:

$$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} + x_{2,5} + x_{2,6} + x_{3,2} \leq 1 \tag{B.8}$$

Para uma cobertura C^l mínima com respeito ao recurso $l \in L$, seja $\bar{E}^l(C^l) = \{(i,k) \in \Omega \backslash C^l : (i,j) \in C^l,\ \tilde{a}^l_{i,k} \leq \tilde{a}^l_{i,j},\ A^{\bar{l}}(C(i,j) \cup \{(i,k)\}) \leq \tilde{b}^{\bar{l}}\}$. Para uma cobertura C mínima com respeito a todos os recursos, seja $\bar{E}(C) = \{(i,k) \in \Omega \setminus C : (i,j) \in C,\ \tilde{\mathbf{a}}_{i,k} < \tilde{\mathbf{a}}_{i,j}\}$. Seja ainda $\widetilde{E}(C) = \{(i,j) \in \Omega : i \notin N(C),\ \mathbf{A}(C') \leq \tilde{\mathbf{b}}$ onde $C' = C(l,t) \cup \{(i,j)\}$ para algum $(l,t) \in C\}$.

Proposição 5. *Para uma cobertura mínima C^l com respeito ao recurso l, $\alpha_{i,j} = 0$ para todo $(i,j) \in \bar{E}^l(C^l) \cup \widetilde{E}(C^l)$ qualquer que seja a sequência $\bar{C}^l$ para* lifting.

Demonstração. Para todo $(i,j) \in \bar{E}^l(C^l)$ tem-se $A^{\bar{l}}(C') \leq \tilde{b}^{\bar{l}}$ onde $C' = C^l(i,j) \cup \{(i,k)\}$ e $A^l(C') \leq \tilde{b}^l$ pois C^l é mínima. Logo $\xi_1 = |C^l| - 1$ para $(i_1, j_1) = (i,j)$ e daí concluímos que $\alpha_{i,j} = 0$ qualquer que seja a ordem de (i,j) em $\bar{C}^l$. Claramente $\alpha_{i,j} = 0$ para todo $(i,j) \in \widetilde{E}(C^l)$ a partir de sua definição. □

Para o exemplo ilustrativo, $\bar{E}^1(C^1) = \{(2,7),\ (2,8),\ (3,1),\ (3,3),\ \ldots,\ (3,8)\}$ e $\widetilde{E}(C^1) = \{(1,1),\ldots,(1,8)\}$ o que confere com os fatores de *lifting* nulos calculados com a aplicação do procedimento.

Proposição 6. *Para uma cobertura mínima C com respeito a todos os recursos, $\alpha_{i,j} = 0$ para todo $(i,j) \in \bar{E}(C) \cup \widetilde{E}(C)$ qualquer que seja a sequência $\bar{C}$ para* lifting.

Demonstração. Para uma cobertura C mínima com respeito a todos os recursos, o procedimento de *lifting* aplicado a $(i_1, j_1) = (i,j) \in \bar{E}(C)$ leva a $\xi_1 = |C| - 1$ e daí concluímos que $\alpha_{i,j} = 0$ qualquer que seja a ordem de (i,j) na sequência $\bar{C}$. Claramente $\alpha_{i,j} = 0$ para todo $(i,j) \in \widetilde{E}(C)$ a partir de sua definição. □

B.2.5 *Lifting* Aproximado de Coberturas Mínimas com Respeito a um Recurso

Dada uma cobertura mínima C^l com respeito a um recurso $l \in L$, podemos desenvolver uma aproximação $\tilde{\alpha}_{i,j}$ para o fator de *lifting* $\alpha_{i,j}$

de certos elementos de $\Omega \setminus C^l$. Para tanto seja $E^l(C^l) = E^l_{\mathrm{A}}(C^l) \cup E^l_{\mathrm{B}}(C^l)$ a extensão da cobertura C^l sendo:

- $E^l_{\mathrm{A}}(C^l) = \{(i,k) \in \Omega \setminus C^l : (i,j) \in C, \tilde{a}^l_{i,k} \geq \tilde{a}^l_{i,j}\}$; e
- $E^l_{\mathrm{B}}(C^l) = \{(i,j) \in \Omega : i \in N \setminus N(C^l),\ \tilde{a}^l_{i,j} \geq \max\{\tilde{a}^l_{p,q} : (p,q) \in C^l\}\}$.

Defina ainda[2]:

- para todo $(i,j) \in C^l$ e $0 \leq k < |C^l|$, seja $C^l_k(i,j) \subseteq C^l(i,j)$ tal que $|C^l_k(i,j)| = k$ e $\min\{\tilde{a}^l_{p,q} : (p,q) \in C^l_k(i,j)\} \geq \max\{\tilde{a}^l_{p,q} : (p,q) \in C^l(i,j) \setminus C^l_k(i,j)\}$;
- para todo $0 \leq k \leq |C^l|$, $C^l_k \subseteq C^l$ tal que $|C^l_k| = k$ e $\min\{\tilde{a}^l_{p,q} : (p,q) \in C^l_k\} \geq \max\{\tilde{a}^l_{p,q} : (p,q) \in C^l \setminus C^l_k\}$.

Pode-se definir fatores de *lifting* aproximados para $(i,j) \in E^l(C^l)$ como segue:

- $\tilde{\alpha}_{i,j} = 1 + \max\{k : A^l(C^l_k(i,k)) \leq \tilde{a}^l_{i,j} - \tilde{a}^l_{i,k}, (i,k) \in C^l\}$ se $(i,j) \in E^l_{\mathrm{A}}(C^l)$; e
- $\tilde{\alpha}_{i,j} = \max\{k : A^l(C_k) \leq \tilde{a}^l_{i,j}\}$ se $(i,j) \in E^l_{\mathrm{B}}(C^l)$.

Assim, chega-se a desigualdade de cobertura com *lifting* aproximado:

$$\sum_{(i,j) \in C^l} x_{i,j} + \sum_{(i,j) \in E^l(C^l)} \tilde{\alpha}_{i,j} x_{i,j} \leq |C^l| - 1 \tag{B.9}$$

Para o exemplo ilustrativo, a cobertura $C^l = \{(2,3),(3,2)\}$ relativa ao recurso $l = 1$ induz $E^l_{\mathrm{A}}(C^l) = \{(2,1),\ (2,2),\ (2,4),\ (2,5),\ (2,6)\}$ e $E^l_{\mathrm{B}}(C^l) = \emptyset$. O cálculo dos fatores de *lifting* aproximados nos levam a obter $\tilde{\alpha}_{2,1} = \tilde{\alpha}_{2,2} = \tilde{\alpha}_{2,4} = \tilde{\alpha}_{2,5} = \tilde{\alpha}_{2,6} = 1$, produzindo a mesma expressão da desigualdade válida (B.8) obtida com *lifting* exato.

Proposição 7. *A desigualdade de cobertura com lifting aproximado dada por* (B.9) *é válida para* $\mathrm{conv}(\mathcal{X})$.

Demonstração. Suponha o contrário e seja $\mathbf{x} \in \mathcal{X}$ uma solução que não satisfaz a desigualdade (B.9). Note que se um elemento de $\mathrm{conv}(\mathcal{X})$ não satisfaz esta desigualdade então deve existir pelo menos um elemento de $\mathcal{X}$ que não a satisfaz. Defina $\widehat{C}^{l,q} = \{(i,j) \in C^l : x_{i,j} = q\}$, $\widehat{E}^{l,q}_{\mathrm{A}}(C^l) =$

[2] O operador $\min\{S\}$ assume valor $-\infty$ quando o conjunto $S = \emptyset$, enquanto $\max\{S\}$ assume valor $+\infty$ quando $S = \emptyset$

$\{(i,j) \in E^l_{\rm A}(C^l) : x_{i,j} = q\}$, e $\widehat{E}^{l,q}_{\rm B}(C^l) = \{(i,j) \in E^l_{\rm B}(C^l) : x_{i,j} = q\}$ para $q = 0, 1$. Segue destas definições que:

$$\sum_{(i,j)\in\Omega} \tilde{a}^l_{i,j} x_{i,j} \geq$$

$$\sum_{(i,j)\in\widehat{C}^{l,1}\cup\widehat{E}^{l,1}_{\rm A}(C^l)\cup\widehat{E}^{l,1}_{\rm B}(C^l)} \tilde{a}^l_{i,j} = A^l(\widehat{C}^{l,1}) + A^l(\widehat{E}^{l,1}_{\rm A}(C^l)) + A^l(\widehat{E}^{l,1}_{\rm B}(C^l))$$

$$\geq A^l(\widehat{C}^{l,1}) +$$

$$\sum_{(i,j)\in\widehat{E}^{l,1}_{\rm A}(C^l):(i,k)\in C^l} \left(A^l(C^l_{\tilde{\alpha}_{i,j}-1}(i,k)) + \tilde{a}^l_{i,k}\right) + \sum_{(i,j)\in\widehat{E}^{l,1}_{\rm B}(C^l)} A^l(C^l_{\tilde{\alpha}_{i,j}})$$

$$\geq A^l(\widehat{C}^{l,1,+}) + A^l(\widehat{C}^{l,0,+}) + A^l(C^l_s) > A^l(C^l)$$

onde $\widehat{C}^{l,1,+} = \widehat{C}^{l,1} \cup \{(i,k) \in C^l : (i,j) \in \widehat{E}^{l,1}_{\rm A}(C^l)\}$, $\widehat{C}^{l,0,+} \subseteq \{(i,j) \in C \setminus \widehat{C}^{l,1,+}\}$ com $|\widehat{C}^{l,0,+}| = r$, $r = \sum_{(i,j)\in\widehat{E}^{l,1}_{\rm A}(C^l)}(\tilde{\alpha}_{i,j} - 1)$, e $s = \sum_{(i,j)\in\widehat{E}^{l,1}_{\rm B}(C^l)} \tilde{\alpha}_{i,j}$. A última desigualdade segue do fato que $\mathbf{x}$ viola a restrição (B.9), o que acaba por implicar em $|\widehat{C}^{l,1,+}| + r + s = |\widehat{C}^{l,1}| + \sum_{(i,j)\in\widehat{E}^{l,1}_{\rm A}(C^l)\cup\widehat{E}^{l,1}_{\rm B}(C^l)} \tilde{\alpha}_{i,j} \geq |C^l|$. Portanto, chega-se a uma contradição que invalida a hipótese. □

B.2.6 *Lifting* Aproximado de Coberturas Mínimas com Respeito a Todos os Recursos

Dada uma cobertura mínima C com respeito a todos os recursos, defina a extensão desta cobertura como $E(C) = E_{\rm A}(C) \cup E_{\rm B}(C)$ onde:

- $E_{\rm A}(C) = \{(i,j) \in \Omega : (i,k) \in C, \tilde{\mathbf{a}}_{i,j} > \tilde{\mathbf{a}}_{i,k}\}$; e
- $E_{\rm B}(C) = \{(i,j) \in \Omega : i \notin N(C), \tilde{\mathbf{a}}_{i,j} > \max\{\tilde{\mathbf{a}}_{p,q} : (p,q) \in C\}\}$.

Com base nas definições acima pode-se definir fatores de *lifting* aproximados para $(i,j) \in E(C)$ como segue:

- $\tilde{\alpha}_{i,j} = 1 + \max\{q : A^l(C^l_q(i,k)) \leq \tilde{a}^l_{i,j} - \tilde{a}^l_{i,k}, (i,k) \in C^l, C^l = C, \text{para todo } l \in L\}$ se $(i,j) \in E_{\rm A}(C)$; e
- $\tilde{\alpha}_{i,j} = \max\{k : A^l(C^l_k) \leq \tilde{a}^l_{i,j}, C^l = C, \text{para todo } l \in L\}$ se $(i,j) \in E_{\rm B}(C)$.

Estes fatores levam à desigualdade de cobertura com *lifting* aproximado:

$$\sum_{(i,j)\in C} x_{i,j} + \sum_{(i,j)\in E(C)} \tilde{\alpha}_{i,j} x_{i,j} \leq |C| - 1 \qquad \text{(B.10)}$$

Proposição 8. *A desigualdade de cobertura com lifting aproximado dada por* (B.10) *é válida para* $\mathrm{conv}(\mathcal{X})$.

Demonstração. Suponha que existe $\mathbf{x} \in \mathcal{X}$ que não satisfaz (B.10). Defina $\widehat{C}^q = \{(i,j) \in C : x_{i,j} = q\}$, $\widehat{E}^q_{\mathrm{A}}(C) = \{(i,j) \in E_{\mathrm{A}}(C) : x_{i,j} = q\}$, e $\widehat{E}^q_{\mathrm{B}}(C) = \{(i,j) \in E_{\mathrm{B}}(C) : x_{i,j} = q\}$ para $q = 0, 1$. Para qualquer $l \in L$, note que

$$\begin{aligned}
\sum_{(i,j)\in\Omega} \tilde{a}^l_{i,j} x_{i,j} &\geq \sum_{(i,j)\in\widehat{C}^1\cup\widehat{E}^1_{\mathrm{A}}(C)\cup\widehat{E}^1_{\mathrm{B}}(C)} \tilde{a}^l_{i,j} x_{i,j} \\
&= A^l(\widehat{C}^1) + A^l(\widehat{E}^1_{\mathrm{A}}(C)) + A^l(\widehat{E}^1_{\mathrm{B}}(C)) \\
&\geq A^l(\widehat{C}^1) + \sum_{(i,j)\in\widehat{E}^1_{\mathrm{A}}(C):(i,k)\in C} \left(A^l(C_{\tilde{\alpha}_{i,j}-1}(i,k)) + \tilde{a}^l_{i,k}\right) + \\
&\qquad \sum_{(i,j)\in\widehat{E}^1_{\mathrm{B}}(C)} A^l(C_{\tilde{\alpha}_{i,j}}) \\
&\geq A^l(\widehat{C}^{1,+}) + A^l(\widehat{C}^{0,+}) + A^l(C_s) > A^l(C)
\end{aligned}$$

onde $\widehat{C}^{1,+} = \widehat{C}^1 \cup \{(i,k) \in C : (i,j) \in \widehat{E}^1_{\mathrm{A}}(C)\}$, $\widehat{C}^{0,+} \subseteq \{(i,j) \in C \setminus \widehat{C}^{1,+}\}$ com $|\widehat{C}^{0,+}| = r$, $r = \sum_{(i,j)\in\widehat{E}^1_{\mathrm{A}}(C)} (\tilde{\alpha}_{i,j} - 1)$, e $s = \sum_{(i,j)\in\widehat{E}^1_{\mathrm{B}}(C)} \tilde{\alpha}_{i,j}$. A última desigualdade segue do fato que $\mathbf{x}$ viola a restrição (B.10), implicando $|\widehat{C}^{1,+}| + r + s = |\widehat{C}^1| + \sum_{(i,j)\in\widehat{E}^1_{\mathrm{A}}(C)\cup\widehat{E}^1_{\mathrm{B}}(C)} \tilde{\alpha}_{i,j} \geq |C|$. Portanto, chega-se a uma contradição que invalida a hipótese. □

B.2.7 Separação de Desigualdades de Cobertura

Dada uma família $\mathcal{F}$ de desigualdades válidas como as desigualdades de cobertura (B.5), surge o problema de encontrar uma desigualdade $f \in \mathcal{F}$ que não seja satisfeita por uma solução fracionária $\mathbf{x} \in \mathcal{P} \setminus \mathcal{X}$ caso exista. Tal problema é conhecido como problema de separação e em geral é NP-Difícil. Não surpreendentemente, pode-se colocar o problema de separação como um problema em programação inteira.

Para o caso da família de coberturas com respeito a um recurso l,

o problema de separação pode ser colocado na forma:

$$S^l(V,\mathbf{x}) : s^l(V,\mathbf{x}) = \max \sum_{(i,j)\in\Omega(V)} (x_{i,j}-1)\chi_{i,j} + 1 \tag{B.11a}$$

$$\text{s.t.} : \sum_{j\in M(i)} \chi_{i,j} \leq 1,\ i \in V \tag{B.11b}$$

$$\sum_{j\in M(i)} \tilde{a}^l_{i,j}\chi_{i,j} > \tilde{b}^l \tag{B.11c}$$

$$\chi_{i,j} \in \{0,1\},\ (i,j) \in \Omega(V) \tag{B.11d}$$

onde $V \subseteq N$ e $\Omega(V) = \{(i,j) \in \Omega : i \in V\}$. A desigualdade de cobertura será $C^l = \{(i,j) : \chi_{i,j} = 1, (i,j) \in \Omega(V)\}$ e esta irá eliminar a solução fracionária se $s^l(V,\mathbf{x}) > 0$, mas por outro lado se $s^l(V,\mathbf{x}) \leq 0$ a desigualdade resultante será a menos satisfeita dentre todas as coberturas que envolvam as tarefas em V. O problema de separação $S^l(V,\mathbf{x})$ foi inspirado no procedimento geral de separação para desigualdade de cobertura proposto em (CROWDER; JOHNSON; PADBERG, 1983). Visto que a solução de $S^l(V,\mathbf{x})$ pode demandar muito recurso computacional, tipicamente heurísticas são empregadas na síntese de coberturas e coberturas estendidas.

O conjunto V pode ser definido como o conjunto das tarefas que foram reconfiguradas, ou seja, $V = \{i : x_{i,j} = 1 \text{ para algum } j \in M(i)\}$. Esta escolha torna o problema $S^l(V,\mathbf{x})$ menor do que quando $V = N$. Caso a cobertura C^l induzida pelo problema de separação não seja violada, pode-se aplicar o procedimento de *lifting* exato ou aproximado.