Augusto Born de Oliveira

Uma Infraestrutura para Reconfiguração Dinâmica de Escalonadores Tempo Real: Modelos, Algoritmos e Aplicações

Florianópolis – SC

2009

Augusto Born de Oliveira

Uma Infraestrutura para Reconfiguração Dinâmica de Escalonadores Tempo Real: Modelos, Algoritmos e Aplicações

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina como requisito parcial para obtenção do grau de Mestre em Engenharia de Automação e Sistemas.

Orientador:
Prof. Eduardo Camponogara, Ph.D.

Co-orientador:
Prof. George Marconi de Araújo Lima, Ph.D.

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DE SANTA CATARINA

Florianópolis – SC

2009

“UMA INFRAESTRUTURA PARA RECONFIGURAÇÃO DINÂMICA DE ESCALONADORES TEMPO REAL: MODELOS, ALGORITMOS E APLICAÇÕES”

Augusto Born de Oliveira

Esta dissertação foi julgada adequada para a obtenção do título de Mestre 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.

Prof. Eduardo Camponogara, Ph.D. – Orientador
Universidade Federal de Santa Catarina

Prof. George Marconi de Araújo Lima, Ph.D. – Co-orientador
Universidade Federal da Bahia

Prof. Dr. Eugênio de Bona Castelan Neto
Universidade Federal de Santa Catarina
Coordenador do Programa de Pós-Graduação em Engenharia de Automação e Sistemas

Banca Examinadora:

Prof. Eduardo Camponogara, Ph.D.
Universidade Federal de Santa Catarina

Prof. Arnaldo Vieira Moura , Ph.D.
Universidade Estadual de Campinas

Prof. Dr. Joni da Silva Fraga
Universidade Federal de Santa Catarina

Prof. Dr. Carlos Barros Montez
Universidade Federal de Santa Catarina

Resumo

Esta dissertação propõe uma infraestrutura para alocação dinâmica de tempo de processador em aplicações tempo real multi-modais. A infraestrutura distribui reservas de banda usando modelos de otimização que podem ser combinados e estendidos para contemplar as necessidades do sistema. Uma vez que a natureza das aplicações varia, diversos modelos são propostos de forma a suportar modos de operação discretos, contínuos ou híbridos e otimizar um dado critério de desempenho global do sistema ou a justiça entre tarefas. Os modelos são generalizações de variada complexidade para o problema da mochila, para os quais algoritmos exatos, heurísticas com garantias de desempenho e esquemas de aproximação em tempo polinomial são propostos. Para fins de avaliação, a infraestrutura foi aplicada no sistema de visão, controle e navegação de um robô móvel, e também num decodificador multimídia real. Ambas são aplicações que capturam a natureza das tarefas que esta infraestrutura busca suportar.

Abstract

This dissertation proposes a framework for dynamic, value-based processor time allocation in multi-modal real-time applications. The framework distributes time reservations using optimization models that can be combined and extended to meet the system's needs. Because the nature of applications varies, several models were proposed to handle discrete, continuous or hybrid modes of operation and optimize task fairness or a given criterion of optimal system performance. The models range from simple to complex generalizations of the knapsack problem for which exact algorithms, heuristics with performance guarantees, and polynomial-time approximation schemes were proposed. As a means of evaluation, the framework was applied to the vision, control and guidance systems on a mobile robot and to a real multimedia decoder, applications that capture the nature of the tasks this framework aims to support.

Sumário

Lista de Figuras

Lista de Tabelas

Parte I

Fundamentação e Problemática

1 Introdução

Sistemas tempo real são aqueles que devem responder a estímulos do seu ambiente dentro de prazos específicos [15]. Um exemplo seria um sistema digital de controle realimentado, onde a tarefa que calcula a lei de controle deve ser executada periodicamente, e onde cada execução deve ser feita sem atrasos para garantir o funcionamento correto do sistema controlado. O escalonamento periódico de múltiplas tarefas com diferentes requisitos temporais requer o conhecimento prévio do tempo de execução das tarefas e uma série de cuidados. Possibilitar o escalonamento correto destes sistemas é um dos objetivos da área de sistemas tempo real.

O restante deste capítulo visa apresentar a motivação e a contribuição deste trabalho, terminando com a apresentação da estrutura desta dissertação.

1.1 Motivação

O espectro de aplicações de sistemas tempo real tem se ampliado consideravelmente nas últimas décadas; isto tem tornado simplística demais sua caracterização clássica onde tarefas têm comportamentos periódicos simples e previsíveis [26]. As arquiteturas modernas de hardware e software tornam difícil ou até impossível a estimativa precisa do tempo de execução de pior caso de tarefas de tempo real. A alta variabilidade do tempo de decodificação de quadros de áudio ou vídeo é um exemplo disso. Além disso, o ambiente no qual as tarefas estão inseridas pode ser altamente dinâmico, como num robô móvel que deve responder a estímulos do ambiente.

Escalonamento baseado em reserva de banda [35] é uma boa abordagem quando se precisa lidar com sistemas que apresentam alta variabilidade nos tempos de chegada ou de computação de suas tarefas [29]. Usualmente, esta abordagem é implementada estruturando-se o sistema como um conjunto de servidores, definidos em termos de parâmetros que limitam a parcela do processador que eles utilizarão em tempo de execução. Normalmente estes parâmetros são configurados estaticamente durante o tempo de projeto dos sistemas. Entretanto, em um sis-

tema com diversas tarefas, podem ocorrer eventos que causam a necessidade da redistribuição de reservas de processador entre as tarefas durante o tempo de execução. Esta necessidade de realizar ajustes dinâmicos aos parâmetros do escalonador devido a dados externos ou propriedades da arquitetura na qual a aplicação executa tem sido alvo de pesquisa recente [9]. Estes sistemas reconfiguráveis podem ser estruturados em diversos modos de operação.

Por exemplo, um sistema de segurança que faz captura digital de imagens: ao detectar movimento em um cômodo monitorado, ele pode passar a requerer mais poder computacional para realizar cálculos de reconhecimento facial na imagem de um possível intruso. Um sistema de visão de um robô móvel pode passar por diferentes modos de operação de acordo com mudanças no ambiente, como condições de iluminação, obstáculos, ângulos de visão ou mudanças no objetivo do robô durante seu tempo de execução. Estas e outras mudanças ambientais imprevisíveis podem ser modeladas em diferentes modos de operação. Neste contexto, suporte para a migração entre modos de operação é necessário em nível de escalonador.

1.2 Contribuição

Este trabalho busca resolver o problema da reconfiguração dinâmica de escalonadores baseados em reserva durante o tempo de execução. Mais especificamente, assume-se que a aplicação de diferentes parâmetros nos servidores pode trazer diferentes níveis de benefício para o sistema como um todo. Desta forma, o problema de reconfiguração é visto como um problema de otimização de acordo com o qual um critério de desempenho global total do sistema deve ser maximizado, obedecendo restrições de escalonabilidade.

Diversos modelos do problema de reconfiguração são definidos, e para cada um deles são apresentados algoritmos exatos ou de aproximação rápidos o suficiente para execução em tempo de execução. Entre os modelos estão o Modelo Discreto, que assume a presença de valores discretos de utilização nas tarefas, o Modelo Contínuo, que assume um intervalo de possíveis utilizações e o Modelo Híbrido, que assume valores discretos de utilização com um grau de liberdade. Pode ser demonstrado que este problema, para o modelo de sistema com modos discretos de operação, é uma generalização do problema da mochila e é, portanto, NP-Difícil.

1.3 Estrutura

Esta dissertação é estruturada da seguinte forma: o restante da Parte I apresenta a fundamentação do escalonamento tempo real, o problema do tratamento de tarefas não-críticas e

a motivação por trás da reconfiguração dinâmica de escalonadores tempo real no Capítulo 2. Além disso, no Capítulo 3, é apresentada a notação utilizada ao longo do texto, e é dado um exemplo da tarefa que a infraestrutura apresentada pretende suportar.

A Parte II apresenta os diferentes modelos desenvolvidos para o suporte de diferentes tipos de aplicação. Algoritmos exatos e de aproximação para os diferentes problemas de otimização que emergem destes modelos são descritos. No Capítulo 4 é apresentado o Modelo Discreto, no Capítulo 5 o Contínuo e no Capítulo 6 o Híbrido. Os algoritmos são avaliados numericamente para confirmar a possibilidade de seu uso em tempo de execução. Finalmente, no Capítulo 7 são apresentados algoritmos para o tratamento de funções objetivo não-aditivas nestes modelos.

A Parte III apresenta, no Capítulo 8, um estudo de caso detalhado, aplicando a infraestrutura no sistema de visão, controle e navegação de um robô móvel. Este estudo de caso tem o objetivo de demonstrar o uso da infraestrutura numa aplicação já existente. Além disso, no Capítulo 9, a infraestrutura é utilizada em uma aplicação multimídia com dados reais de execução, para avaliar a sua habilidade de tratar sobrecargas no escalonador. No Capítulo 10 é feita a integração da infraestrutura proposta com o escalonamento baseado na teoria de controle realimentado (*Feedback Scheduling*), uma abordagem que vem sendo difundida na literatura.

Finalmente, a Parte IV traz as conclusões e sugere trabalhos futuros.

2 *Reconfiguração Dinâmica de Escalonadores Tempo Real*

Este Capítulo faz uma revisão da teoria de escalonadores tempo real, introduzindo as políticas de escalonamento de prioridade fixa Rate Monotonic, Deadline Monotonic e a política de escalonamento de prioridade dinâmica Earliest Deadline First. O problema do tratamento de tarefas não-críticas é descrito, juntamente com algumas abordagens que visam resolvê-lo. Posteriormente, a reconfiguração dinâmica de reservas de processador é discutida, passando por abordagens encontradas na literatura e finalmente terminando numa breve introdução ao que foi realizado no contexto deste trabalho.

2.1 Escalonamento Tempo Real

Um sistema de tempo real é composto por tarefas que devem reagir ao seu ambiente entregando resultados computacionais em prazos específicos [15]. De acordo com esta definição, o seu comportamento correto não depende apenas da integridade dos resultados de sua computação, mas também da entrega pontual destes resultados. Exemplos de sistema tempo real são sistemas de controle digital, de processamento de sinais e sistemas de telecomunicação. Todos estes são compostos por tarefas periódicas com prazos bem definidos que, se perdidos, incorrem numa perda de qualidade de serviço.

Mais formalmente, definimos uma *tarefa* tempo real τ como um conjunto de *instâncias* (de índice k) que em conjunto provêem a funcionalidade da tarefa. Uma instância de uma tarefa de decodificação de vídeo seria a decodificação de um quadro individual.

O *tempo de chegada* r_k de uma instância é o momento no qual ela se torna disponível para a execução. Num sistema de telecomunicação digital, por exemplo, o tempo de chegada de cada quadro depende da periodicidade definida pela taxa de quadros da codificação em questão.

O *tempo de resposta* de uma instância é igual ao período de tempo entre sua chegada e o

momento onde sua execução é completada. O *deadline* d_k de uma instância é o momento até o qual a sua execução deve terminar. Em um sistema de controle digital onde a lei de controle é computada periodicamente, cada instância de computação desta lei deve ser completada antes da chegada da sua próxima instância. Em outras palavras, neste caso as instâncias têm um deadline d_k igual ao próximo tempo de chegada r_{k+1}.

Deadlines podem ainda ser classificados como críticos ou não-críticos. Deadlines críticos são aqueles cuja perda é considerada uma falha séria, e são aplicados a instâncias cuja resposta tardia podem ter consequências catastróficas (como perda de vidas ou de grandes valores). Deadlines não-críticos são aplicados a instâncias cuja resposta tardia é indesejável, porém não causam danos sérios; a perda de deadlines causa apenas uma perda de qualidade de serviço.

Em seu artigo de 1973, Liu e Layland [26] definiram um modelo de tarefas e políticas de escalonamento que vieram a ser utilizados por uma grande quantidade de trabalhos na área de tempo real. As principais restrições impostas por Liu e Layland para resolver o problema de escalonamento usando políticas orientadas a prioridades são as seguintes:

- Chegadas de instâncias de todas tarefas para as quais existem deadlines são periódicas, com intervalos mínimos constantes entre chegadas chamados de período T_i;
- Deadlines d_k são iguais à soma do tempo de chegada r_k com o período T_i, ou seja, cada instância deve ser completada antes que a próxima instância dessa tarefa chegue;
- As tarefas são independentes entre si, no sentido que suas chegadas independem da iniciação ou término da execução de outra;
- O tempo de execução máximo das tarefas é conhecido *a priori* e não varia com o tempo;
- Tarefas aperiódicas não têm deadlines críticos, e são tratadas especialmente.

Uma tarefa (denotada por τ_i), é então definida pelo seu tempo de execução C_i e seu período T_i. A estimação do valor de C_i, chamado de tempo máximo de execução, é por si só alvo de grande esforço de pesquisa, dado que a execução de software complexo em arquiteturas complexas aumentam bastante a imprevisibilidade deste valor. A estimação da quantidade de laços iterativos em software ou do comportamento de *caches* e previsão das ramificações de execução em hardware, por exemplo, são fatores contribuintes para essa imprevisibilidade. Os valores usados são geralmente, portanto, estimativas pessimistas com uma certa margem de segurança.

Um sistema com um conjunto de n tarefas tem no máximo a seguinte utilização:

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

$U \in [0,1]$ representa a fração de tempo gasto executando o conjunto de tarefas. Um problema interessante é encontrar o menor limite superior U^{lub} para o valor de U, que tem diferente valor dependendo de qual política de escalonamento é utilizada. No mesmo artigo, Liu e Layland definem duas das mais conhecidas políticas de escalonamento de sistemas tempo real, Rate Monotonic (RM) e Earliest Deadline First (EDF) (que no texto ainda levava o nome de Deadline Driven Scheduling). A seguir são discutidas as propriedades destas políticas e exemplos clássicos de cada um deles são descritos em mais detalhes.

2.1.1 Prioridade Fixa

A classe de políticas de escalonamento de prioridade fixa engloba toda aquela política na qual todas as instâncias de uma tarefa mantêm um só valor de prioridade ao longo do tempo de vida do sistema. Isto significa que a prioridade escolhida pelo projetista do sistema ou pela política de escalonamento será mantida ao longo do tempo, o que traz alguns benefícios:

- A definição da ordem de prioridades pode ser feita associada a alguma noção de importância ou criticalidade das tarefas;
- Tarefas de baixa prioridade não interferem nas de alta prioridade.

Políticas conhecidas de prioridade fixa incluem Rate Monotonic e Deadline Monotonic discutidas a seguir.

Rate Monotonic (RM)

A política de escalonamento Rate Monotonic definida por Liu e Layland no mesmo artigo atribui prioridades para as tarefas de acordo com o seu período; quanto menor o período T_i, maior a prioridade da tarefa τ_i. É demonstrável que esta política de escalonamento é ótima, no sentido que nenhuma outra política de prioridade fixa consegue escalonar um conjunto de tarefas que Rate Monotonic não consegue escalonar, quando o deadline de todas elas é igual aos seus respectivos períodos.

O diagrama de Gantt da Figura 2.1, retirado de [15], mostra um exemplo de escalonamento pela política RM. As tarefas, τ_1, τ_2 e τ_3, são descritas na Tabela 2.1. Instâncias de todas tarefas

	C_i	T_i	U_i
τ_1	20	100	0.2
τ_2	40	150	0.267
τ_3	100	350	0.286

Tabela 2.1: Tarefas do exemplo do escalonamento por RM.

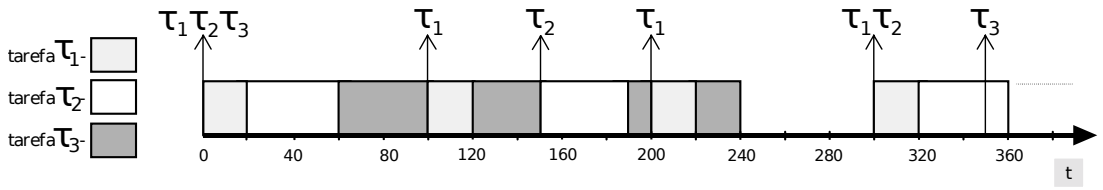


Figura 2.1: Exemplo de escalonamento através do Rate Monotonic.

chegam em $t = 0$. Por ser mais frequente (maior prioridade segundo o RM), τ_1 assume o processador. Em $t = 20$, a tarefa τ_1 conclui e τ_2 toma posse do processador por ser a mais prioritária com instâncias pendentes. A tarefa τ_3 assume em $t = 60$ e é interrompida por τ_1, que tem a chegada de uma nova instância em $t = 100$ que, por sua vez, toma o processador (preempção de τ_3 por τ_1A). τ_3 sofre mais interrupções de novas ativações das tarefas τ_2 e τ_1 em $t = 150$ e $t = 200$, respectivamente.

A prova apresentada no artigo demonstra que o menor limite superior de utilização de qualquer política de escalonamento de prioridade fixa quando deadlines são iguais aos períodos é:

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

Esta relação é suficiente, porém não necessária, para a escalonabilidade, de forma que pode existir um conjunto de tarefas com utilização acima desse limite que o RM consiga escalonar.

Um teste de escalonabilidade baseado no tempo de resposta das tarefas foi apresentado em [4] por Audsley *et al.* Este teste, necessário e suficiente, usa um algoritmo iterativo pseudo-polinomial, o que pode impedir seu uso em tempo de execução. Em [8], Buttazzo *et al.* descrevem um teste de escalonabilidade que usa um limite hiperbólico da utilização para o RM. Como o tempo de execução deste teste é $O(n)$ e ele é menos pessimista que o teste original de Liu e Layland, ele se torna um bom candidato para uso durante o tempo de execução.

Deadline Monotonic (DM)

Como foi mencionado, Rate Monotonic é ótimo para o escalonamento de tarefas de prioridade fixa no modelo de Liu e Layland. Em [22], entretanto, Leung e Whitehead argumentam que é possível que as definições sejam restritivas demais, e o caso onde elas tenham deadlines menores do que seus períodos deve ser observado. Neste artigo a política Deadline Monotonic é apresentada. Nesta política a atribuição de prioridade é feita de acordo com os deadlines das tarefas. Apesar de ser demonstrada a otimalidade do Deadline Monotonic neste caso, não é apresentado um teste de escalonabilidade ao qual um conjunto de tarefas pode ser submetido. Em [5], Audsley *et al.* apresentam um teste suficiente rápido porém pessimista e um teste suficiente e necessário porém mais complexo.

Limite Superior de Utilização

O limite superior mostrado na Equação (2.2) ocorre no pior caso, onde os períodos das tarefas são completamente não harmônicos. Em sistemas onde períodos das tarefas são múltiplos entre si, é possível atingir até 100% de utilização mesmo com políticas de escalonamento de prioridade fixa.

Ainda assim, é possível que para grandes conjuntos de tarefas mais de 25% (U^{lub} para $n = 5$ é menor que 0.744, por exemplo) do processador seja desperdiçado. Esta possibilidade levou ao desenvolvimento das políticas de prioridade dinâmica, para as quais o $U^{lub} = 1$ no modelo de Liu e Layland.

2.1.2 Prioridade Dinâmica

Ainda no seu artigo original de 1973, Liu e Layland apresentam uma solução para a subutilização causada pela atribuição de prioridades fixas; o uso de prioridades dinâmicas. No artigo é apresentada a política de escalonamento que veio a ser conhecida como Earliest Deadline First (EDF), com o intuito de aumentar a utilização alcançável em sistemas tempo real.

Earliest Deadline First (EDF)

No EDF, a qualquer dado momento a prioridade de cada instância de uma tarefa é atribuída de acordo com o seu deadline. Se o deadline de uma tarefa é o mais próximo, ela receberá a maior prioridade. Se for o mais distante, receberá a menor. A qualquer dado momento, a instância incompleta com a maior prioridade será executada. Este método de atribuição

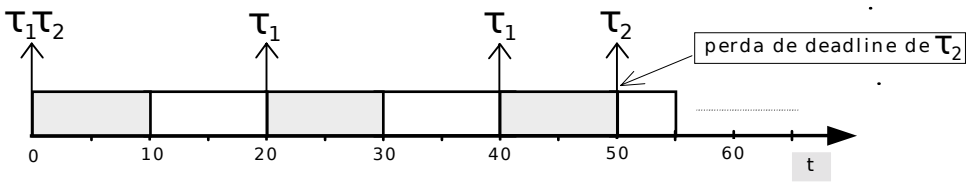
	C_i	T_i	U_i
A	10	20	0.5
B	25	50	0.5

Tabela 2.2: Tarefas do exemplo do escalonamento por EDF.

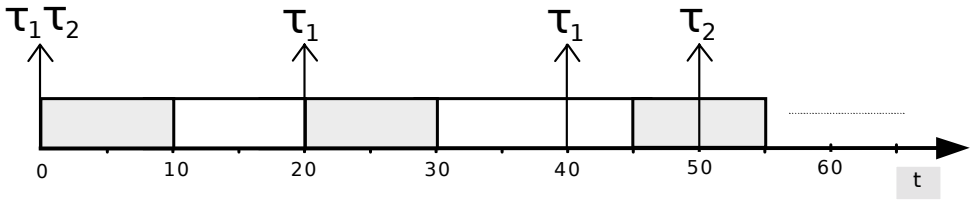
dinâmica de prioridades é bastante contrastante com os métodos de prioridade fixa vistos anteriormente, já que diferentes instâncias de uma tarefa podem ter prioridades diferentes. Uma vez que é possível alcançar 100% de utilização do processador, o único teste de escalonabilidade que precisa ser feito é um somatório das utilizações de cada tarefa, se estas forem independentes:

$$\sum_{i=1}^{n} U_i \leq 1 \tag{2.3}$$

No exemplo mostrado na Figura 2.2, o conjunto de tarefas definido na Tabela 2.2 é submetido a escalonamentos EDF e RM. A utilização individual das duas tarefas é 0.5, ocupando, portanto, 100% do processador. Usando o teste descrito em (2.2), percebe-se que o escalonamento não é garantido pelo RM (o somatório de utilizações ultrapassa o U^{lub}), mas de acordo com (2.3) é garantido com o EDF. Na Figura 2.2(a), no tempo $t = 50$, a tarefa τ_2 perde seu deadline. Ao mesmo tempo, na Figura 2.2(b) se percebe o escalonamento sem falhas pela parte do EDF.



(a) Rate Monotonic.



(b) Earliest Deadline First.

Figura 2.2: EDF escalonando um conjunto de tarefas onde RM falha.

Entre as desvantagens do EDF (bem como outras políticas de prioridade dinâmica) está o fato que todas a tarefas têm, *a priori*, a capacidade de gerar uma preempção em todas as outras. Se uma subestimação do tempo de execução no pior caso de alguma tarefa leve a Equação (2.3) a ser violada, é muito difícil prever qual tarefa perderá seu deadline.

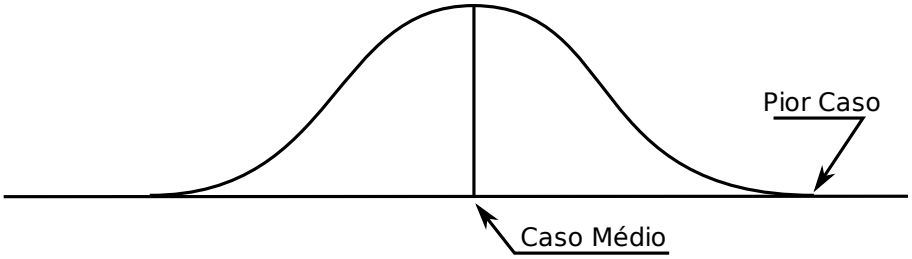


Figura 2.3: Probabilidade da ocorrência do pior caso e do caso médio.

2.1.3 Deadlines Não-Críticos vs. Deadlines Críticos

Apesar do EDF garantir 100% de utilização do processador, isto parte do princípio que todas as tarefas são de igual criticalidade, e são tratadas com a mesma importância. Isto não se observa necessariamente em todo sistema tempo real; pode ser necessário escalonar tarefas com restrições temporais críticas no mesmo processador que tarefas com restrições temporais não-críticas.

A Figura 2.3 mostra uma distribuição normal representando a probabilidade de ocorrência de diferentes valores de C_i durante várias execuções de uma tarefa. Fica claro que quando a tarefa é crítica, não há alternativa fora contemplar o pior caso nos cálculos de escalonabilidade do sistema. Isto também significa que quanto maior a distância entre o pior caso e o caso médio, mais subutilização de processador ocorrerá.

Se uma tarefa não é crítica, se torna interessante contemplar o caso médio do tempo de execução nos cálculos de escalonabilidade, visando garantir o cumprimento de deadlines apenas numa parcela das instâncias que seja compatível com a qualidade de serviço desejada.

2.2 Escalonamento Baseado em Servidores

O problema de tratar tarefas não-críticas da mesma forma que as críticas é o fato das tarefas não-críticas poderem então fazer tarefas críticas perderem seus deadlines, uma vez que seu caso médio é utilizado nos cálculos de escalonamento. Este impacto pode ser amenizado ao se separar as tarefas não-críticas das críticas, escalonando-as em um servidor de tarefas aperiódicas. Um servidor de tarefas aperiódicas é uma tarefa periódica que tem o papel de executar instâncias de tarefas não-críticas da maneira mais rápida o possível.

Da mesma forma que uma tarefa periódica, um servidor de tarefas aperiódicas S_i é definido por um período T_i e um tempo de execução fixo, chamado de orçamento Q_i. Este orçamento, descrito em unidades de tempo, da mesma forma que os tempos de execução C_i das tarefas, e é

gasto na mesma medida em que o servidor executa tarefas. Isto é, se uma tarefa tratada por um servidor executa por 3 unidades de tempo, o servidor deve subtrair 3 unidades de tempo do seu orçamento.

Dos valores de período e orçamento é possível derivar a banda u_i de um servidor, sendo $u_i = Q_i/T_i$. Assim como a utilização de tarefas periódicas, a banda de um servidor é a fração do processador ($u_i \in [0,1]$) que ele terá reservado para uso no escalonamento de instâncias de tarefas não-críticas.

Cada servidor possui ainda uma fila interna, onde as instâncias de tarefas não-críticas são enfileiradas quando chegam, e são executadas de acordo com a política estabelecida pelo servidor.

O mais simples dos servidores de tarefas aperiódicas, o Background Server [27], consiste em uma tarefa com prioridade menor que todas as outras no sistema. Uma vez que o Background Server (e por consequência as instâncias em sua fila) só recebe o processador quando nenhuma outra tarefa o estiver utilizando, é garantido que nenhuma tarefa crítica perderá seu deadline. Em contrapartida, isso também pode levar a tempos de resposta inaceitáveis para as tarefas escalonadas no servidor, tanto no caso médio quanto no pior caso.

Diversas outras abordagens foram então exploradas, e uma série de servidores com diferentes políticas foram criados com o objetivo de melhorar diferentes propriedades. Além de minimização do tempo de resposta das tarefas aperiódicas, entre estas propriedades estão a simplicidade de implementação, a garantia de isolamento temporal (não interferência de tarefas não-críticas em tarefas críticas) e minimizar o número de preempções.

A seguir serão rapidamente descritas diversas destas abordagens, todas elas concebidas com base sobre a política EDF, que serviram como base para o desenvolvimento do Constant Bandwidth Server (CBS) [2]. O CBS é o servidor utilizado no restante desta dissertação, e é portanto definido em mais detalhes na Seção 2.2.3. .

2.2.1 Deadline Sporadic Server

O Deadline Sporadic Server (DSS), de Ghazalie e Baker [17], é uma extensão para prioridade dinâmica do Sporadic Server criado por Sprunt, Sha e Lehoczky [35] para políticas de prioridade fixa.

O DSS tem como um de seus objetivos espalhar sua execução de forma uniforme ao longo do tempo. Para este fim, seu orçamento Q_i é dividido em j pedaços de tamanho $\sigma_{i,k}$. Estes

pedaços de orçamento devem sempre somar o orçamento total, de forma que a Equação (2.4) sempre se verifique:

$$\sum_{k=1}^{j} \sigma_{i,k} = Q_i,\ i = 1,\ldots,n \tag{2.4}$$

Estes pedaços são utilizados e recarregados individualmente de acordo com uma política que garante que um pedaço só será reutilizado pelo menos um período, ou seja, T_i unidades de tempo depois da última vez que ele esteve disponível. Isto leva à garantia que o efeito do servidor na escalonabilidade de tarefas críticas nunca é pior do que uma tarefa crítica de tempo de execução Q_i e período T_i, e, portanto, utilização igual a sua banda u_i.

A sua implementação relativamente complexa, uma vez que um número ilimitado de pedaços de orçamento podem ser criados, cada um com seu momento de recarga, necessitando a realização de operações de aglutinação de pedaços adjacentes.

2.2.2 Total Bandwidth Server

A idéia principal do Total Bandwidth Server (TBS), de Spuri e Buttazzo [36], é que a banda total disponível para o servidor é dedicada a uma tarefa assim que ela entra na fila do servidor. O TBS é definido pela sua banda u_i, em contraste com o DSS que recebe os parâmetros Q_i e T_i separadamente. Isto é causado pela forma pela qual o TBS aplica deadlines a instâncias de tarefas aperiódicas recém chegadas. Seja d_k o deadline a ser atribuído para a uma nova instância, d_{k-1} o deadline que foi atribuído à instância anterior, r_k o tempo de chegada da nova instância, C_k o tempo de execução da instância e u_i a banda dedicada ao servidor, a Equação (2.5) mostra esta atribuição:

$$d_k = \max\{r_k, d_{k-1}\} + \frac{C_k}{u_i} \tag{2.5}$$

O deadline do servidor é, portanto, dinâmico, e dependente do valor do tempo de execução da instância C_k. É de grande importância que a atribuição dos deadlines seja feita de forma que um deadline curto (que levaria o servidor à cabeça da lista do EDF) não leve a utilização do servidor a ultrapassar u_i, o seu valor de configuração. Como a implementação do TBS se resume à definição de deadlines corretos para as instâncias aperiódicas que chegam, ele é um dos servidores mais simples encontrados na literatura.

O Constant Utilization Server (CUS), de Sun *et al.* [14], usa essencialmente a mesma

política que o TBS, com a exceção do momento onde a recarga do orçamento é feita; enquanto o TBS recarrega o orçamento (implicitamente, pela atribuição do deadline) imediatamente se uma instância se encontra pronta na cabeça de sua fila, o CUS recarrega seu orçamento apenas sob condições especiais onde a fila está vazia ou o deadline do servidor foi alcançado.

2.2.3 Constant Bandwidth Server

O Constant Bandwidth Server [2] é um servidor de tarefas aperiódicas que busca garantir o isolamento temporal entre tarefas críticas e não-críticas, ou seja, garantir que uma sobrecarga nas tarefas não-críticas não acarretará em uma perda de deadline em uma tarefa crítica. Para atingir este objetivo, o CBS usa uma política de recarga de orçamento que garante que sua utilização máxima não ultrapassará em nenhuma hipótese o limite definido em sua configuração. Os seus parâmetros são o seu orçamento máximo Q_i e período T_i. Como visto anteriormente, a razão Q_i/T_i define a banda u_i do servidor, que deve ser utilizada no teste de escalonabilidade da política EDF.

Para garantir o isolamento temporal, o orçamento é reinicializado toda vez que acaba, porém o deadline associado ao servidor é incrementado por um valor igual ao seu período. Isto pode levar à perda do processador pelo CBS, caso outra tarefa passe a ter maior prioridade.

Formalmente, um CBS, denotado S_i, é definido por:

- Seu orçamento máximo: Q_i
- Seu período: T_i
- Sua banda: $u_i = Q_i/T_i$
- Seu orçamento atual: c_i
- Seu $k^{ésimo}$ deadline: $d_{i,k}$

Cada uma das instâncias da tarefa a ser tratada pelo CBS S_i, indexadas por j, recebe o deadline corrente de S_i na sua chegada. Em outras palavras, a tarefa herda o deadline do servidor responsável por seu escalonamento. A execução de uma instância por um tempo t decresce o valor do orçamento c_i pelo mesmo valor t. A prorrogação do deadline e a recarga do orçamento no evento da sua depleção é mostrada no Algoritmo 1.

Seja r o tempo de chegada de uma instância de tarefa servida por S_i. O Algoritmo 2 mostra a forma pela qual os valores de deadline e orçamento são atualizados se a chegada ocorre em um momento no qual a fila de instâncias do servidor está vazia.

Algoritmo 1: Condição de recarga do orçamento do CBS $S_i(Q_i, T_i)$

if $c_i = 0$ **then**
  $c_i := Q_i$
  $d_{i,k+1} := d_{i,k} + T_i$

Algoritmo 2: Tratamento de chegada de uma instância em um CBS

if *Fila de S_i está vazia* **then**
  if $c_i \geq (d_{i,k} - r)u_i$ **then**
    $c_i := Q_i$
    $d_{i,k+1} := d_{i,k} + T_i$
  else
    Mantém c_i
    Mantém $d_{i,k}$

A Figura 2.4 [2] mostra um exemplo do servidor CBS escalonando instâncias de uma tarefa não-crítica (τ_2) no mesmo sistema que uma tarefa crítica (τ_1). Ele recebe três instâncias para execução ($c_{2,1}$, $c_{2,2}$ e $c_{2,3}$), e, de acordo com as regras descritas acima, faz a manutenção de seu orçamento e deadline garantindo o isolamento temporal.

Na primeira linha da simulação pode-se observar a tarefa crítica τ_1 de $C_1 = 2$ e $T_1 = 3$ sendo executada periodicamente. Sua utilização de $U_1 = 2/3$ permite, segundo a regra de escalonabilidade do EDF, que mais $1/3$ do processador seja distribuído entre outras tarefas ou servidores. Na linha de tempo mostrada, τ_1 tem sempre o menor deadline e portanto é a tarefa mais prioritária, tendo instâncias executadas assim que chegam.

Na segunda linha é possível ver a execução da tarefa não-crítica τ_2, com sua primeira instância chegando no tempo $r_1 = 2$, com 3 unidades de tempo de execução. Todas as instâncias desta tarefa são tratadas por um CBS, de orçamento $Q_2 = 2$ e período $T_2 = 7$, o que faz sua

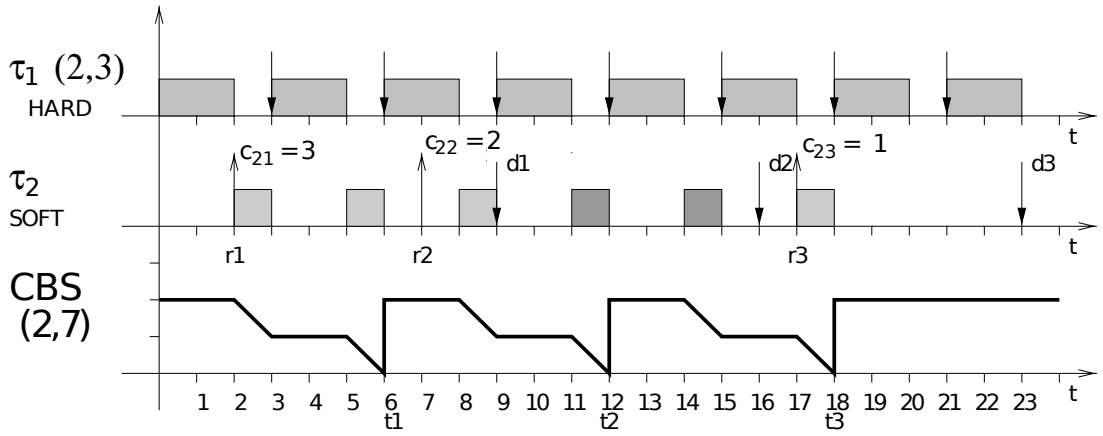


Figura 2.4: Exemplo de funcionamento de um servidor CBS.

banda ser $U_2 = 2/7$. Já que o somatório das utilizações e bandas é $2/3 + 2/7 \leq 1$, o sistema é escalonável pela política EDF.

A terceira linha da Figura 2.4 mostra o orçamento do CBS, inicialmente igual ao seu valor máximo 2. Quando a primeira instância de τ_2 chega para execução, o processador está livre e portanto o CBS pode começar a sua execução. Do instante 2 a 3 o CBS executa a primeira instância de τ_2, decrescendo seu orçamento para 1. Ele é interrompido por τ_1 (que tem deadline mais próximo, igual a 6). τ_2 volta a executar de $T = 5$ a $T = 6$, esgotando seu orçamento. Utilizando o Algoritmo 1, o orçamento é recarregado para o seu valor máximo, passando para 2, e o deadline do CBS é acrescido de um período (era 9, passa a ser 16).

Antes do CBS ter finalizado a execução da instância atual de τ_2 (que leva 3 unidades de tempo para completar) uma segunda instância de τ_2 chega, com 2 unidades de tempo de execução. Uma vez que a fila do CBS não está vazia, a instância é enfileirada e a execução continua normalmente. Nos tempos de 8 a 9 o CBS executa e termina a execução da primeira instância. Continuando com o mesmo deadline, executa a primeira unidade de tempo da segunda instância entre os instantes 11 e 12, quando seu orçamento acaba e a regra do Algoritmo 1 é novamente aplicada.

A segunda instância termina em 15, esvaziando a fila do CBS. No instante 17 chega uma nova instância de tempo de execução igual a uma unidade de tempo. Como a fila do CBS está vazia, deve-se aplicar a regra apresentada no Algoritmo 2. A condição para a recarga do orçamento não se verifica, portanto são mantidos deadline e orçamento. Quando a instância termina sua execução, o orçamento é novamente esgotado e a regra do Algoritmo 1 é aplicada recarregando o orçamento e redefinindo o deadline para $T = 23$.

Por ser de fácil implementação e ter a propriedade do isolamento temporal, o CBS foi escolhido para os ensaios práticos desta dissertação.

Existe ainda uma série de extensões do CBS que permitem a retomada eficiente de recursos não utilizados, como CASH [11], BACKSLASH [24], GRUB [25] e BASH [12]. O BASH de Buttazzo *et al.*, por exemplo, relaxa com segurança as restrições de banda que garantem o isolamento temporal no CBS simples e estende o modelo de tarefas permitindo o compartilhamento de recursos.

2.3 Reconfiguração Dinâmica de Escalonadores Tempo Real

Como pode ser visto, o escalonamento baseado em servidores de aperiódicas (em especial o CBS, que garante isolamento temporal) é capaz de resolver satisfatoriamente o problema de escalonar tarefas de tempo real críticas e não-críticas no mesmo sistema. No entanto, a reserva de banda é geralmente feita em tempo de projeto e fixada ao longo da execução do sistema. Para sistemas com tarefas multimodais tal abordagem pode não ser suficiente.

Tarefas multi-modais [32] são aquelas que consistem de vários modos de operação. Cada modo produz um diferente comportamento temporal (tempo de computação e período), fruto de um diferente conjunto de objetivos a serem alcançados pela tarefa em cada um dos seus modos. Um exemplo seria o software em execução em uma aeronave, onde os modos de cruzeiro, decolagem e pouso têm um comportamento distinto devido aos seus objetivos distintos. Na próxima seção este problema é descrito, e algumas das abordagens encontradas na bibliografia são discutidas. Neste contexto, é interessante ter mecanistmos para redefinição das bandas dos servidores em tempo de execução.

O tratamento da migração entre os diferentes modos de cada tarefa deve ser feito no nível do escalonador, uma vez que cada modo é caracterizado por diferentes atributos/demandas temporais. Esta mudança dos parâmetros de escalonamento em tempo de execução é alvo de pesquisa contínua desde pelo menos a década de 1990.

Neste ponto cabe definir a diferença conceitual entre atuar nos parâmetros do escalonador em tempo de execução – um processo tratado por protocolos de mudança de modo – e os métodos de escolha de que valores aplicar em cada parâmetro – o que é chamado neste trabalho de reconfiguração dinâmica de escalonadores tempo real. A decisão de que modo usar (o avião está decolando ou aterrissando?) cabe à reconfiguração dinâmica; aplicar o modo no escalonador (a tarefa aterrissar tem período 1s ou 100ms?) cabe ao protocolo de mudança de modo. Esta linha divisória é menos clara em algumas das abordagens discutidas a seguir mas no contexto deste trabalho serve como uma definição satisfatória do escopo.

A seguir são descritos alguns dos cuidados realizados durante as mudanças de modo, e depois algumas abordagens de reconfiguração dinâmica.

2.3.1 Protocolos de Mudança de Modo

Pode-se imaginar que a mudança do comportamento temporal de uma ou mais tarefas em um sistema de tempo real é um processo complicado; no mínimo o teste de escalonabilidade

deve ser refeito (em tempo de projeto ou execução) para garantir que os deadlines ainda serão alcançados. Em [32], Crespo e Real fazem um estudo sobre as diferentes políticas de mudança de modo existentes para sistemas com prioridade fixa, separando diferentes abordagens encontradas na bibliografia em duas classes, protocolos síncronos e assíncronos. Dependendo da forma com a qual as tarefas respondem a um "Pedido de Mudança de Modo", os protocolos caem em uma das duas categorias.

Nos protocolos síncronos, todas as tarefas do sistema que migrarão de modo o fazem simultaneamente. O exemplo mais simples dessa classe é o Idle Time Protocol, que espera algum momento no qual o processador fica livre para realizar a mudança de modo. Este protocolo tem a vantagem de ser de simples implementação, porém o tempo entre o pedido de mudança de modo e a verificação da condição que permite a mudança pode ser grande demais para ser satisfatório.

Os protocolos assíncronos fazem a mudança de modo de diferentes tarefas independentemente, causando a existência de uma mistura de tarefas em seu modo antigo (esperando a mudança de modo) e tarefas que já fizeram a mudança e se encontram no novo modo. Apesar disto permitir mudanças de modo mais rápidas, também leva a uma complexidade maior de implementação já que qualquer combinação de tarefas em modos velhos e novos pode gerar uma violação de condição de escalonabilidade.

2.3.2 Abordagens de Reconfiguração Dinâmica de Escalonadores

Com a breve descrição de protocolos de mudanças de modo acima já é possível compreender algumas das abordagens de reconfiguração dinâmica de escalonadores tempo real encontradas na bibliografia. O problema da reconfiguração dinâmica foi estudado anteriormente por diversos pesquisadores da área [6, 10, 18, 19, 23, 34], mas a maioria deles não lida com isolamento temporal ou escalonamento baseado em reserva. Algumas abordagens notáveis são descritas em mais detalhes a seguir, juntamente com uma breve descrição da abordagem que será descrita nesse trabalho.

No contexto do CBS foram feitos avanços interessantes no ajuste dinâmico dos parâmetros dos servidores utilizando teoria de controle em malha fechada [1,3,28,31,39]. A idéia principal destas abordagens é atuar nos parâmetros dos servidores ajustando sua banda de forma a fazer com que alguma métrica de qualidade de serviço seja tão próxima a um valor de referência quanto possível. Por exemplo, pode-se definir o erro de escalonamento de uma instância como a diferença entre seu instante de término e o deadline do servidor [1]. Se as instâncias de uma tarefa terminam tarde demais (ou cedo demais), mais (ou menos) banda deve ser alocada para o

servidor que a trata. A principal dificuldade em aplicar estas abordagens é determinar uma lei de controle estável de forma a permitir a modelagem da dinâmica do sistema.

Em [10], Buttazzo *et al.* apresentam uma política de escalonamento capaz de "redimensionar" as tarefas como se fossem molas. Nesta modelagem a utilização de uma tarefa representa o comprimento de uma mola real. Além disso, um comprimento mínimo e um índice de elasticidade são atribuídos a cada uma delas. O comprimento mínimo impede que a tarefa não receba nenhuma fatia do processador e o índice de elasticidade representa a tolerância da tarefa a ter seu período reduzido.

No modelo elástico, cada tarefa é capaz de modular seu período para prover diferentes taxas de qualidade de serviço. Se isto causar uma possível sobrecarga no sistema (o somatório de utilizações passará do U^{lub} do sistema), todas as outras tarefas são comprimidas de acordo com sua elasticidade e tamanho mínimo. No artigo é até apresentada uma extensão para lidar com recursos compartilhados durante este processo.

Em [34], Mossé *et al.* levantam um modelo "*energy-aware*" de escalonamento onde as tarefas têm múltiplas implementações, cada uma com um valor atribuído pelo programador. Utilizando soluções dinâmicas e estáticas os autores maximizam o valor agregado do sistema mantendo a sua escalonabilidade e também garantindo um nível mínimo de carga de bateria em um sistema móvel recarregável. Para realizar esta maximização de valor, eles modelam o sistema como um problema de otimização restrito não só pela escalonabilidade como também pelo consumo energético.

Em [18], Jehuda e Israeli apresentam uma abordagem automatizada para meta-controle de software, atribuindo uma noção de valores às tarefas e minimizando seus tempos de resposta mantendo sempre a escalonabilidade das tarefas críticas. São apresentados dois algoritmos de aproximação, e testes mostram que na prática ambos são sub-ótimos por apenas alguns pontos percentuais.

Em [21], Lehoczky *et al.* mostram um modelo multi-dimensional de sistemas computacionais, onde se visa maximizar a qualidade de serviço de várias tarefas, observando o limite de múltiplos recursos como memória, tempo de processamento e banda de rede. No mesmo artigo são apresentados algoritmos exatos e de aproximação, estes últimos com tempos de execução que possibilitariam seu uso em tempo de execução.

Diferentemente das abordagens citadas acima, este trabalho visa prover suporte a grandes ajustes aos parâmetros de servidores baseados em reserva de banda em tempo de execução. Para tanto, a modelagem do sistema deve levar em conta as propriedades de cada tarefa, como

por exemplo a existência de modos bem definidos ou de um intervalo contínuo de possíveis níveis de utilização. Também pode existir a necessidade de distribuir fatias do processador com alguma noção de justiça.

2.4 Sumário

Este capítulo apresentou a fundamentação de escalonamento de sistemas tempo real, contextualizando as políticas de prioridade fixa e dinâmica. O problema do tratamento de tarefas não-críticas também foi apresentado, seguido de uma revisão de algumas das abordagens mais conhecidas de escalonamento baseado em servidores. Em particular, o CBS foi detalhado por ser o servidor escolhido para os testes práticos deste trabalho. O problema da reconfiguração dinâmica dos parâmetros destes servidores pôde então ser definido, e uma revisão da literatura que visa resolver este problema foi feita, seguida de uma rápida descrição da proposta deste trabalho, que será detalhada a seguir.

3 *Notação, Aplicação e Modelos de Sistema*

Para ilustrar mais formalmente o tipo de aplicação que a infraestrutura deste trabalho pretende suportar, este capítulo introduz o modelo de sistema e a notação utilizada (Seção 3.1), e então descreve nestes termos uma aplicação com comportamento distintamente multi-modal: os sistemas de visão, controle e navegação de um robô móvel de competição (Seção 3.2). No Capítulo 8, esta mesma aplicação é detalhada e integrada à infra-estrutura, mostrando como ela administra o seu comportamento dinâmico. Na Seção 3.3 é apresentada uma visão introdutória dos modelos que serão apresentados na Parte II dessa dissertação.

3.1 Notação

Este trabalho considera um sistema monoprocessado composto por n servidores $\mathscr{S} = \{S_1, S_2, \ldots, S_n\}$ do tipo CBS (*Constant Bandwidth Server*) [2]. Cada servidor $S_i \in \mathscr{S}$ é definido em termos da tupla (Q_i, T_i), onde Q_i representa seu orçamento máximo e T_i seu período. Cada servidor S_i consome o máximo de $u_i = Q_i/T_i$ de processador e serve a um conjunto específico de tarefas. Em outras palavras, um sistema construído desta forma aloca, para cada conjunto de tarefas servidas pelo servidor S_i, uma banda constante determinada por u_i. Como cada servidor é escalonado por EDF (*Earliest Deadline First*) [26], pode-se usar 100% de processador para escalonar os servidores em $\mathscr{S}$. Ou seja,

$$\sum_{S_i \in \mathscr{S}} u_i \leq 1 \tag{3.1}$$

Usualmente, os parâmetros Q_i e T_i de cada servidor são definidos em tempo de projeto de acordo com as necessidades de cada aplicação. Por exemplo, geralmente uma tarefa crítica periódica pode ser servida por um servidor com Q_i igual ao custo máximo de execução da tarefa e T_i igual ao seu período. Semelhantemente, pode-se definir tais parâmetros considerando os custos médios de execução e tempo médio entre ativações de tarefas não críticas. Seja qual for

o caso, é importante ressaltar que o uso do CBS proporciona ao sistema isolamento temporal entre as tarefas servidas por diferentes servidores, o que é muito importante para sistemas de tempo real modernos. Por exemplo, pode-se, desta forma, tolerar falhas temporais de tarefas que executam além do seu custo computacional estimado, pois tais falhas não serão prejudicadas por outras tarefas do sistema.

Neste trabalho, estamos interessados em elaborar mecanismos de suporte à reconfiguração dinâmica de um sistema escalonado por um conjunto $\mathscr{S}$ de servidores do tipo CBS. Tal funcionalidade é bastante útil para atender a modificações dinâmicas da aplicação ou do ambiente. Por exemplo, dependendo do nível de iluminação ou proximidade dos objetos sendo monitorados, um sistema de vigilância pode requerer diferentes custos ou períodos de execução para as tarefas de processamento de imagem. Requisitos semelhantes de reconfiguração têm sido identificados em diversos campos de aplicação.

No contexto deste trabalho, assumimos que a aplicação pode, a qualquer momento, requisitar uma reconfiguração dos parâmetros dos servidores em $\mathscr{S}$. Para tanto, através da chamada de sistema `reconfig`$(U_1, v_1, U_2, v_2, \ldots, U_n, v_n)$, a aplicação indica quais os percentuais de processador que devem ser alocados a cada servidor S_i. Se o sistema puder alocar pelo menos U_i ao servidor S_i, há um benefício $v_i \geq 0$ para o sistema. De forma geral, assumimos que a função benefício associada a S_i assume a seguinte forma:

$$A_i(U_i, u_i, v_i) = \frac{\min(u_i, U_i)}{U_i} v_i \,, \tag{3.2}$$

onde u_i representa o percentual de processador efetivamente alocado a S_i.

Esta notação é aplicada no exemplo a seguir, apresentada rapidamente aqui para fins ilustrativos. O mesmo caso de uso é revisitado no Capítulo 8 em detalhes.

3.2 Sistemas de Visão, Controle e Navegação de um Robô Móvel

Para ilustrar o modelo de sistema utilizado neste trabalho, considere o software executado em um robô de competição acadêmica. O seu objetivo é percorrer um labirinto sem nenhum conhecimento prévio sobre a disposição de suas paredes.

Este software é composto por várias tarefas: Software de controle em malha fechada controla seus motores. Sensores infra-vermelho detectam as paredes, usando tarefas de background para periodicamente atualizar o modelo interno que o robô possui do labirinto. Um sistema de

visão captura objetos de interesse (ícones nas paredes), os catalogando e reconhecendo através do labirinto.

Estes quatro sub-sistemas (controle de motores, detecção de paredes, manutenção do mapa/definição de caminho e, finalmente, visão) compartilham o mesmo processador. Portanto, servidores CBS podem ser usados para garantir seu isolamento temporal. Na notação usada neste trabalho, um servidor S_1 poderia receber a tarefa de controle dos motores, S_2 a tarefa de visão e assim por diante.

Cada uma destas tarefas tem múltiplos modos de operação. A tarefa de controle de motores, por exemplo, não necessita de uma reserva de tempo de CPU enquanto o robô está parado. Entretanto, enquanto o robô está em movimento ela se torna crítica: se um obstáculo é detectado, ela precisa responder rapidamente para garantir que o robô pare em tempo hábil.

Pode-se perceber, portanto, que o benefício A_1 para a tarefa atribuída ao servidor S_1 (controle de motores) muda ao longo do tempo de execução do sistema. Enquanto o robô está parado, $A_1 = 0$. Enquanto está em movimento, $A_1 \gg 0$. Estas mudanças em benefício ocorrem em todas as tarefas do robô, e cada uma delas requer a reconfiguração do escalonador para que a reserva de processador seja redistribuída, maximizando o benefício agregado do sistema.

Outros exemplos de mudança de benefício podem ser encontrados no sistema de visão. Este sistema tem três níveis de análise que devem ser executados para que uma imagem na parede seja reconhecida em sua base de dados. Inicialmente, imagens de baixa resolução são capturadas em alta frequência, calculando periodicamente a chance do quadro conter uma imagem. Uma vez que esta chance ultrapassa um certo limiar, a tarefa troca de nível para confirmar a presença da imagem. Finalmente, confirmada a presença da imagem, um terceiro nível é ativado. Neste nível final, a imagem é comparada com aquelas armazenadas pelo robô, com o objetivo de identificá-la em sua base de dados. A cada mudança de nível, a utilização de processador e importância (e portanto, benefício) da tarefa crescem. No último destes níveis, a frequência da tarefa é diminuída para um quinto do seu valor original, enquanto a utilização se multiplica por dez.

Seja S_2 o servidor a tratar o sistema de visão. Pode-se imaginar que cada um destes níveis leva tanto o benefício A_2 e a utilização requerida U_2 a aumentar. Quando a soma da utilização requerida por todas as tarefas viola a condição (3.1), cabe ao sistema de reconfiguração decidir qual alocação de tempo de CPU fazer, sacrificando tarefas menos importantes (que trazem menos benefício) no processo.

Como pode-se notar, reconfigurar os parâmetros (Q_i, T_i) dos servidores em $\mathscr{S}$ para maxi-

mizar o benefício total do sistema requer a solução de um problema de otimização, no qual a equação (3.1) é uma das restrições. Diversos modelos diferentes deste problema foram criados, cada um com a capacidade de capturar diferentes aspectos de diferentes aplicações, como por exemplo, a natureza discreta ou contínua dos modos de cada uma das tarefas.

3.3 Modelos de Sistema

Este trabalho apresenta três modelos para os sistemas Tempo Real compostos por servidores CBS, os modelos Discreto, Contínuo, Híbrido, e mais a aplicação de objetivos não aditivos para estes três modelos. Segue uma descrição mais detalhada de cada um destes:

Modelo Discreto (MD) Para este modelo, assume-se que existem valores pré-determinados de (Q_i, T_i) para cada servidor S_i, o que torna as instâncias deste modelo um problema de Programação Inteira (PI). Este modelo é particularmente útil para uso em aplicações cujas tarefas possuem implementações alternativas e/ou diferentes possíveis períodos. Por exemplo, diferentes algoritmos de decodificação ou taxas de quadro podem ser observadas no processamento de um vídeo comprimido digitalmente. Custos de execução e taxas de quadros mais altos levam a uma qualidade de serviço mais alta, e vice-versa. Como será visto no Capítulo 4, o MD leva a um problema de otimização NP-Difícil. Um algoritmo exato baseado em programação dinâmica e dois esquemas eficientes de aproximação são apresentados para este problema.

Modelo Contínuo (MC) Neste modelo de sistema, as variáveis de decisão são contínuas, uma vez que os possíveis valores de (Q_i, T_i) não são pré-determinados. São definidos intervalos dentro dos quais os valores $u_i = Q_i/T_i$ devem ser fixados de forma a maximizar o benefício total do sistema. No Capítulo 5 este modelo é detalhado e uma solução ótima para instâncias do MC podem ser encontradas analiticamente com um algoritmo muito eficiente.

Modelo Híbrido (MH) Em alguns cenários, além da seleção de implementações distintas de uma tarefa durante uma reconfiguração (como no MD) também pode existir a liberdade para um ajuste fino da utilização de cada um dos modos. O MH compartilha então a pré-definição de modos com o MD, e desfruta de parte da flexibilidade presente no MC. O Capítulo 6 descreve o modelo e uma solucão baseada em programação dinâmica, com a possibilidade de um passo adicional baseado no MC para o aumento da qualidade das soluções encontradas.

Funções Objetivo Não-Aditivas (NA) Um dos aspectos não contemplados pelos modelos descritos acima é justiça, útil quando é de interesse da aplicação fazer uma distribuição equalitária de parte dos recursos de processamento. Por exemplo, é possível que a maximização do benefício total do sistema levaria à redução excessiva dos recursos alocados a um grupo de servidores. Este aspecto é contemplado pela definição de funções objetivo não-aditivas para os modelos MD, MC e MH. Estas novas funções objetivo são apresentadas no Capítulo 7, onde também se demonstra que a resolução dos novos problemas de otimização é feita de forma eficiente.

3.4 Sumário

Este capítulo apresentou a notação que será utilizada ao longo deste trabalho, seguida de um exemplo de aplicação que a infraestrutura definida aqui pretende contemplar. A natureza multi-modal das tarefas da aplicação descrita esclarece a motivação para a infraestrutura de reconfiguração proposta nesta dissertação. Vários modelos de reconfiguração foram descritos para contemplar diferentes tipos de aplicação. Suas respectivas formulações e soluções serão apresentadas a partir do próximo capítulo.

Parte II

Modelos e Algoritmos

4 Modelos Discretos

Os modelos discretos são caracterizados por configurações pré-determinadas para os servidores. Uma configuração k de um server S_i tem uma utilização $u_{i,k} = Q_{i,k}/T_{i,k}$ dada pelo orçamento $Q_{i,k}$ e período $T_{i,k}$. Desta forma, pode-se obter a seguinte formulação:

$$PD: \ fd = \text{Maximize} \sum_{S_i \in \mathscr{S}} \sum_{k \in K_i} A_{i,k} x_{i,k} \tag{4.1a}$$

Sujeito a :

$$\sum_{S_i \in \mathscr{S}} \sum_{k \in K_i} u_{i,k} x_{i,k} \leq 1 \tag{4.1b}$$

$$u_{i,k} = \frac{Q_{i,k}}{T_{i,k}} \tag{4.1c}$$

$$\sum_{k \in K_i} x_{i,k} = 1, \ S_i \in \mathscr{S} \tag{4.1d}$$

$$x_{i,k} \in \{0,1\}, \ S_i \in \mathscr{S}, k \in K_i \tag{4.1e}$$

O parâmetro $A_{i,k}$ define o benefício induzido ao se alocar $u_{i,k}$ unidades de recurso computacional ao servidor S_i. A variável $x_{i,k}$, definida pela equação (4.1e), indica a escolha de uma das $k \in K_i$ configurações relativas ao servidor S_i. Apenas uma configuração deve ser selecionada, o que é representado pela restrição (4.1d). A restrição (4.1b) garante a escalonabilidade de $\mathscr{S}$ de acordo com a política EDF. O problema clássico da mochila é trivialmente redutível ao problema de reconfiguração dinâmica discreta, logo *PD* é NP-Difícil. Sem perda de generalidade, vamos assumir que $u_{i,k} \leq u_{i,k-1}$ para todo $S_i \in \mathscr{S}$ e $k \geq 2$. Para modelar a possibilidade de cancelamento de um servidor S_i basta definir $A_{i,1} = 0$ e $u_{i,1} = 0$. Assumimos também que $\sum_{S_i \in \mathscr{S}} u_{i,1} < 1$ pois, de outra forma, os servidores não seriam escalonáveis ou não haveria potencial para otimizar as aplicações. Assume-se ainda que $A_{i,k}$ e $T_{i,k}$ são racionais.

	Servidor S_i			
	$i=1$	$i=2$	$i=3$	$i=4$
$u_{i,1}$	10/100	15/100	5/100	7/100
$u_{i,2}$	30/100	25/100	20/100	15/100
$u_{i,3}$	50/100	40/100	35/100	20/100
$u_{i,4}$	70/100	55/100	60/100	30/100
$u_{i,5}$	80/100	75/100	65/100	40/100

Tabela 4.1: Recursos computacionais associados aos servidores.

	Servidor S_i			
	$i=1$	$i=2$	$i=3$	$i=4$
$A_{i,1}$	0.125	0.2	0.076	0.175
$A_{i,2}$	0.375	0.333	0.307	0.375
$A_{i,3}$	0.625	0.533	0.538	0.5
$A_{i,4}$	0.875	0.733	0.923	0.75
$A_{i,5}$	1	1	1	1

Tabela 4.2: Benefícios associados às diferentes versões dos servidores.

4.1 Instância Exemplo

A instância exemplo apresenta $n = 4$ servidores cujas demandas computacionais são dadas na Tabela 4.1. Os benefícios aparecem na Tabela 4.2. Note que o número de versões de cada servidor $S_i \in \mathscr{S}$ é $K_i = 5$.

4.2 Algoritmos de Programação Dinâmica

Esta subseção apresenta dois algoritmos de programação dinâmica para o problema de reconfiguração discreta *PD*. O primeiro algoritmo maximiza o benefício para o sistema tomando como restrição a utilização relativa das tarefas. Por outro lado, o segundo algoritmo minimiza a utilização relativa assumindo como restrição um benefício mínimo especificado para o sistema. Chamamos a primeira versão de *primal* e a segunda de *dual*. Ambos os algoritmos resolvem versões generalizadas do problema clássico da mochila [38] onde um item pode ser selecionado com diferentes níveis discretos de peso e valor.

4.2.1 Versão Primal

O princípio por trás da programação dinâmica está na divisão do problema *PD* em subproblemas, cada um deles considerando um subconjunto dos servidores e um percentual do recurso disponível. Contudo, o projeto do algoritmo de programação dinâmica exige que

os valores $u_{i,k}$ e o lado direito da restrição (4.1b) sejam inteiros. Isto pode ser contornado multiplicando-se a desigualdade (4.1b) pelo menor múltiplo comum dos parâmetros $T_{i,k}$, digamos Λ, o que leva à desigualdade equivalente:

$$\sum_{S_i \in \mathscr{S}} \sum_{k \in K_i} \Lambda u_{i,k} x_{i,k} \leq \Lambda \tag{4.2}$$

Seja $PD(m,\lambda)$ a versão de PD que se restringe a encontrar as versões dos servidores em $\mathscr{S}_m = \{S_1, S_2, \ldots, S_m\} \subseteq \mathscr{S}$ quando a fatia de recursos disponível é $\lambda \in \{0,1,\ldots,\Lambda\}$. Formalmente, esta versão restrita é dada por:

$$PD(m,\lambda):\ fd(m,\lambda) = \text{Max} \sum_{(i,k)\in\Omega(m)} A_{i,k} x_{i,k} \tag{4.3a}$$

$$\text{S. a :}$$

$$\sum_{(i,k)\in\Omega(m)} \Lambda u_{i,k} x_{i,k} \leq \lambda \tag{4.3b}$$

$$\sum_{(i,k)\in\Omega(m)} x_{i,k} = 1,\ S_i \in \mathscr{S}_m \tag{4.3c}$$

$$x_{i,k} \in \{0,1\},\ (i,k) \in \Omega(m) \tag{4.3d}$$

onde $\Omega = \{(i,k) : S_i \in \mathscr{S},\ k \in K_i\}$ define o conjunto de todos os pares servidor e nível de configuração, enquanto $\Omega(m) = \{(i,k) \in \Omega : S_i \in \mathscr{S}_m\}$ define o conjunto de pares restrito ao subconjunto de servidores $\mathscr{S}_m$.

Obviamente $PD \equiv PD(n,\Lambda)$ e portanto $fd = fd(n,\Lambda)$. Note que $fd(m,\lambda') \geq fd(m,\lambda'')$ sempre que $\lambda'' \geq \lambda'$, mas não se pode assegurar que $fd(m',\lambda) \geq fd(m'',\lambda)$ quando $m' \geq m''$ devido à igualdade (4.3c). A versão restrita $PD(m,\lambda)$ pode facilmente ser colocada em uma forma recursiva:

$$PD(m,\lambda):\ fd(m,\lambda) = \text{Max} \sum_{k\in K_m} A_{m,k} x_{m,k} + fd(m-1, \lambda - \lambda_m) \tag{4.4a}$$

$$\text{S. a :}$$

$$\lambda_m = \sum_{k\in K_m} \Lambda u_{m,k} x_{m,k} \tag{4.4b}$$

$$\sum_{k\in K_m} x_{m,k} = 1 \tag{4.4c}$$

$$x_{m,k} \in \{0,1\},\ k \in K_m \tag{4.4d}$$

com $fd(i,\lambda) = -\infty$ se $\lambda < \Lambda u_{i,1}$ para todo $S_i \in \mathscr{S}$. O caso terminal da recursão ocorre quando $m = 0$, neste caso $fd(0,\lambda) = 0$ se $\lambda \geq 0$ e $fd(0,\lambda) = -\infty$ se $\lambda < 0$. A recursão (4.4a)–(4.4d) leva a um algoritmo de programação dinâmica com pseudo-código explicitado abaixo.

Algoritmo 3: Algoritmo de programação dinâmica para o modelo discreto

input: Servidores $\mathscr{S}$, parâmetros $A_{i,k}$, $u_{i,k}$ e Λ
initialize: $n := |\mathscr{S}|$
for $m = 1, \ldots, n$ **do**
  $fd(m,0) := -\infty$
  $p(m,0) := 0$
for $\lambda = 0, \ldots, \Lambda$ **do**
  $fd(0,\lambda) := p(0,\lambda) := 0$
for $\lambda = 1, \ldots, \Lambda$ **do**
  for $m = 1, \ldots, n$ **do**
    $fd(m,\lambda) := -\infty$
    $p(m,\lambda) := 0$
    for $k = 1, \ldots, \kappa(m)$ **do**
      if $\lambda \geq \Lambda u_{m,k}$ **and** $fd(m-1, \lambda - \Lambda u_{m,k} + A_{m,k}) > fd(m,\lambda)$ **then**
        $fd(m,\lambda) := fd(m-1, \lambda - \Lambda u_{i,k}) + A_{m,k}$
        $p(m,\lambda) := k$
return (fd, p)

No algoritmo acima, $p(m,\lambda)$ registra qual configuração, dentre as $\kappa(m)$ disponíveis, é selecionada na solução de $PD(m,\lambda)$. Por exemplo, se o servidor S_m executa segundo a configuração $k \in K_m$, então $p(m,\lambda) = k$ e $fd(m,\lambda) = fd(m-1, \lambda - \Lambda u_{m,k}) + A_{m,k}$. Note que os primeiros dois laços correspondem ao caso base da formulação recursiva (4.4a)–(4.4d). O terceiro e maior laço constitui o procedimento recursivo.

O algoritmo consome $\Theta(\Lambda n)$ unidades de memória para armazenar as tabelas $fd(m,\lambda)$ e $p(m,\lambda)$ e finaliza em $\Theta(\Lambda n \max\{\kappa_i : S_i \in \mathscr{S}\})$ passos computacionais. O tempo de execução do algoritmo é pseudo-polinomial em função da dependência de Λ.

A aplicação do algoritmo de programação dinâmica à instância exemplo dada na Subseção 4.1 produz a Tabela 4.3 com o benefício ótimo e a Tabela 4.4 com a solução ótima da família de sub-problemas $\{PD(m,\lambda)\}$. As tabelas trazem apenas as colunas que fazem parte da solução ótima. As entradas em negrito indicam as decisões ótimas associadas aos sub-problemas:

- a solução de $PD(4,100)$ define $x_{4,5} = 1$ ($u_{4,5} = 0.40$) com benefício $A_{4,5} = 1$;
- a solução de $PD(3,60)$ define $x_{3,3} = 1$ ($u_{3,3} = 0.35$) com benefício $A_{3,3} = 0.538$;
- a solução de $PD(2,25)$ define $x_{2,1} = 1$ ($u_{2,1} = 0.15$) com benefício $A_{2,1} = 0.200$;
- a solução de $PD(1,10)$ define $x_{1,1} = 1$ ($u_{1,1} = 0.10$) com benefício $A_{1,1} = 0.125$.

	$fd(m,\lambda)$				
$m\backslash\lambda$	0	10	25	60	100
0	**0**	0	0	0	0
1	$-\infty$	**0.125**	0.125	0.625	1.000
2	$-\infty$	$-\infty$	**0.325**	0.708	1.208
3	$-\infty$	$-\infty$	$-\infty$	**0.863**	1.458
4	$-\infty$	$-\infty$	$-\infty$	1.151	**1.863**

Tabela 4.3: Tabela $fd(m,\lambda)$.

	$p(m,\lambda)/u_{m,p(m,\lambda)}$				
$m\backslash\lambda$	0	10	25	60	100
0	**0**	0/	0/	0/	0/
1	0	**1/0.10**	1/0.10	3/0.50	5/0.80
2	0	0/	**1/0.15**	2/0.25	2/0.25
3	0	0/	0/	**3/0.35**	5/0.65
4	0	0/	0/	4/0.30	**5/0.40**

Tabela 4.4: Tabela $p(m,\lambda)$.

O benefício máximo gerado durante a reconfiguração é de $A_{4,5}+A_{3,3}+A_{2,1}+A_{1,1}=1.863=fd(n,\Lambda)$, como indicado na Tabela 4.3.

4.2.2 Versão Dual

Aqui apresenta-se uma formulação alternativa para resolver o problema de reconfiguração dinâmica discreta. A idéia está em transformar a restrição de escalonabilidade (4.1b) em função objetivo, enquanto o objetivo original dado por (4.1a) passa a ser modelado como restrição. Tal formulação pode levar a uma redução da complexidade computacional visto que esta passa a ser independente do valor de Λ. De fato, como será visto, a solução para a formulação dual depende apenas do número de servidores, do número de suas reconfigurações e dos valores dos benefícios associados às suas respectivas versões.

Para a síntese do algoritmo de programação dinâmica os parâmetros $A_{i,k}$ teriam de ser inteiros. Assumindo que estes parâmetros são números racionais, $A_{i,k}=N_{i,k}/D_{i,k}$, pode-se encontrar uma formulação inteira equivalente substituindo $A_{i,k}$ por $\Psi A_{i,k}$ onde Ψ é o menor múltiplo comum dos parâmetros $D_{i,k}$.

A formulação dual para o problema de reconfiguração fica:

$$\widehat{PD}(\psi):\ \widehat{fd}(\psi) = \text{Minimize} \sum_{(i,k)\in\Omega} u_{i,k}x_{i,k} \tag{4.5a}$$

Sujeito a :

$$\sum_{(i,k)\in\Omega} \Psi A_{i,k}x_{i,k} \geq \Psi\psi \tag{4.5b}$$

$$\sum_{k\in K_i} x_{i,k} = 1,\ S_i \in \mathscr{S} \tag{4.5c}$$

$$x_{i,k} \in \{0,1\},\ (i,k) \in \Omega \tag{4.5d}$$

onde $\widehat{fd}(\psi)$ é o menor consumo de recursos necessário para que os servidores produzam um benefício de pelo menos ψ.

Para se chegar a um algoritmo de programação dinâmica, teremos que desenvolver uma versão restrita e recursiva para $\widehat{PD}(\psi)$ através da noção de benefício *ajustado* $\Psi A_{i,k}$, com restrição de benefício *ajustado* mínimo assumindo valores $\delta \in \{0,1,\ldots,\Psi\psi\}$, e se restringindo ao subconjunto $\mathscr{S}_m = \{S_1,\ldots,S_m\}$ de servidores. Formalmente, esta versão restrita e recursiva é dada por:

$$\widehat{PD}(m,\delta):\ \widehat{fd}(m,\delta) = \text{Minimize} \sum_{k\in K_m} u_{m,k}x_{m,k} + \widehat{fd}(m-1,\max\{\delta-\delta_m,0\}) \tag{4.6a}$$

Sujeito a :

$$\delta_m = \sum_{k\in K_m} \Psi A_{m,k}x_{m,k} \tag{4.6b}$$

$$\sum_{k\in K_m} x_{m,k} = 1 \tag{4.6c}$$

$$x_{m,k} \in \{0,1\},\ k \in K_m \tag{4.6d}$$

Em outras palavras, $\widehat{fd}(m,\delta)$ é o esforço computacional mínimo para reconfigurar os servidores do sub-conjunto $\mathscr{S}_m$ enquanto gerando um benefício *ajustado* de pelo menos δ unidades.

Caso não seja possível induzir este benefício mínimo, então $\widehat{fd}(m,\delta)$ assume o valor $+\infty$. Obviamente, $\widehat{fd}(m,\delta) = +\infty$ sempre que $\Psi\sum_{S_i\in\mathscr{S}_m} A_{i,\kappa(i)} < \delta$ para qualquer $\mathscr{S}_m$. A condição de contorno é obtida definindo $\widehat{fd}(0,\delta) = +\infty$ para todo $\delta > 0$ e $\widehat{fd}(0,0) = 0$. Um limite superior para o benefício máximo é $\Delta = \Psi\sum_{S_i\in\mathscr{S}} A_{i,\kappa(i)}$. $\Delta^* \leq \Delta$ é o menor valor para o qual $\widehat{fd}(n,\Delta^*) \leq 1$ induz a reconfiguração ótima. Seja $\mathbf{x} = (x_{i,k} : (i,k) \in \Omega)$ um vetor com a solução ótima para $PD(n,\Lambda)$ e $\bar{\mathbf{x}} = (\bar{x}_{i,k} : (i,k) \in \Omega)$ um vetor com a solução ótima para $\widehat{PD}(n,\Delta^*)$. Então, $\sum_{S_i\in\mathscr{S}} A_{i,k}x_{i,k} = \sum_{S_i\in\mathscr{S}} A_{i,k}\bar{x}_{i,k}/\Psi$, enquanto $\sum_{S_i\in\mathscr{S}} u_{i,k}x_{i,k}/\Lambda \leq 1$ e $\sum_{S_i\in\mathscr{S}} u_{i,k}\bar{x}_{i,k} \leq 1$. O valor Δ^* pode ser encontrado algoritmicamente por meio da programação dinâmica.

Algoritmo 4: Algoritmo de programação dinâmica dual para o modelo discreto

input: Servidores $\mathscr{S}$, parâmetros $A_{i,k}$, $u_{i,k}$ e Ψ
initialize: $n := |\mathscr{S}|$
initialize: $\Delta := \Psi \sum_{S_i \in \mathscr{S}} A_{i,\kappa(i)}$
initialize: $\widehat{fd}(0,0) := \widehat{p}(0,0) := 0$
for $\delta = 1, \ldots, \Delta$ **do**
    $\widehat{fd}(0,\delta) := +\infty$
    $\widehat{p}(0,\delta) := 0$
$\delta := 0$
while $\delta \leq \Delta$ **do**
    for $m = 1, \ldots, n$ **do**
        $\widehat{fd}(m,\delta) := +\infty$
        $\widehat{p}(m,\delta) := 0$
        for $k = 1, \ldots, \kappa(m)$ **do**
            if $u_{m,k} + \widehat{fd}(m-1, \max\{\delta - A_{m,k}, 0\}) < \widehat{fd}(m,\delta)$ **then**
                $\widehat{fd}(m,\delta) := u_{m,k} + \widehat{fd}(m-1, \max\{\delta - A_{m,k}, 0\})$
                $\widehat{p}(m,\delta) := k$
    if $\widehat{fd}(m,\delta) > 1$ **then**
        $\Delta^\star := \Delta - 1$
        return $(\widehat{fd}, \widehat{p}, \Delta^\star)$
    $\delta := \delta + 1$

	$\widehat{fd}(m,\delta)$				
$m \backslash \delta / \Psi$	0	0.125	0.325	0.863	1.863
0	**0**	∞	∞	∞	∞
1	0	**0.10**	0.30	0.70	∞
2	0	0.25	**0.25**	0.70	1.45
3	0	0.30	0.30	**0.60**	1.35
4	0	0.37	0.37	0.50	**1.00**

Tabela 4.5: Tabela $\widehat{fd}(m,\lambda)$.

Inicialmente, algumas atribuições são feitas no primeiro laço *for* do algoritmo. O laço principal, o *while*, termina quando o valor de $\widehat{fd}(m,\delta) > 1$, ou seja, quando consome-se mais recursos computacionais do que disponível implicando a não escalonabilidade dos servidores. As linhas que atribuem $\widehat{fd}(m,\delta) := +\infty$ e $\widehat{p}(m,\delta) := 0$ correspondem às condições de contorno explicadas anteriormente. O laço *for* mais interno, que itera de $k = 1, \ldots, \kappa(m)$, é o responsável pelo procedimento recursivo definido pela equação (4.6a).

Como pode ser observado pelo código do algoritmo de programação dinâmica dual, são necessários $\Theta(\Delta^* n) \in O(\Delta n)$ unidades de memória. O tempo de execução é da ordem de $O(\Delta^* n \max\{\kappa(i) : S_i \in \mathscr{S}\})$.

O algoritmo de programação dinâmica produz a Tabela 4.5 com o consumo de recursos

	$\widehat{p}(m,\delta)/A_{m,\widehat{p}(m,\delta)}$				
$m\backslash\delta/\Psi$	0	0.125	0.325	0.863	1.863
0	**0**	0/	0/	0/	0/
1	0	**1/0.125**	2/0.375	4/0.875	0/
2	0	1/0.20	**1/0.20**	3/0.533	5/1.00
3	0	1/0.076	1/0.076	**3/0.538**	4/0.923
4	0	1/0.175	1/0.175	3/0.50	**5/1.00**

Tabela 4.6: Tabela $\widehat{p}(m,\delta)$.

mínimos e a Tabela 4.6 com as decisões de reconfiguração se aplicado à instância exemplo dada na Subseção 4.1. Utilizou-se $\Psi = 1000$ e $\Delta = 4000$. Observe que $\Delta^*/\Psi = 1.863 = fd(n,\Lambda)$. A reconfiguração produzida pelo algoritmo dual determina $x_{4,5} = 1$, $x_{3,3} = 1$, $x_{2,1} = 1$ e $x_{1,1} = 1$, conferindo com a solução produzida pelo algoritmo primal.

4.3 Algoritmos de Aproximação

Um algoritmo que retorna uma solução próxima da ótima é dito algoritmo de aproximação. Tais algoritmos podem ser úteis na busca de uma solução de boa qualidade quando o tempo computacional é restrito, em particular quando se tratando de problemas NP-Difíceis. No que segue apresentamos alguns conceitos fundamentais e especializamos algoritmos de aproximação do problema da mochila para o problema de reconfiguração discreta.

4.3.1 Noções Preliminares

Alguns conceitos fundamentais para o projeto de algoritmos de aproximação são apresentados tomando como base [30].

Definição 1. *Um* **problema de otimização** *P é caracterizado por:*

- **[Instâncias]** D : *um conjunto de versões de parâmetros.*
- **[Soluções]** $S(I)$: *o conjunto de todas as soluções factíveis para uma instância* $I \in D$.
- **[Valor]** f : *uma função que associa um valor a cada solução, ou seja,* $f : S(I) \to \mathbb{R}$.

Definição 2. *Dada uma instância* $I \in D$ *de um* **problema de maximização** *P, uma solução* $\omega_I^\star \in S(I)$ *é dita ótima se* $f(\omega_I^\star) \geq f(\omega)$, $\forall \omega \in S(I)$. *O valor da solução ótima será designado por* $OPT(I) = f(\omega_I^\star)$.

Definição 3. *Um* **algoritmo de aproximação** *A, para um problema de otimização P, é um algoritmo de tempo polinomial que produz uma solução* $\omega \in S(I)$ *dada uma instância I.* $A(I)$ *vai denotar o valor* $f(\omega)$ *da solução produzida por A.*

Definição 4. *Seja A um algoritmo de aproximação para um problema de maximização P. O desempenho relativo $R_A(I)$ do algoritmo A em uma instância de entrada I é definido como:*

$$R_A(I) = \frac{OPT(I)}{A(I)}$$

Para um problema de minimização, o desempenho relativo é definido como:

$$R_A(I) = \frac{A(I)}{OPT(I)}$$

O desempenho relativo é definido diferentemente para problemas de minimização e maximização de forma a se ter uma medida padrão de qualidade da solução produzida por A. Note que o desempenho relativo é pelo menos 1 e a qualidade da solução aproximada melhora à medida que o desempenho relativo se aproxima de 1.

Definição 5. *Um algoritmo A é dito* **esquema de aproximação** *se para qualquer instância I e tolerância de erro $\varepsilon > 0$, o algoritmo produz uma solução com objetivo $A(I)$ tal que:*

- $A(I) \geq (1-\varepsilon)OPT(I)$ *quando o problema é de maximização; e*
- $A(I) \leq (1+\varepsilon)OPT(I)$ *quando o problema é de minimização.*

Definição 6. *Um esquema de aproximação A é dito* **esquema de aproximação polinomial** *se para um valor fixo $\varepsilon > 0$, o tempo de execução de A é limitado por um polinômio no tamanho da instância I. A é dito* **esquema de aproximação polinomial completo** *se o tempo de execução de A é limitado polinomialmente no tamanho de I e $1/\varepsilon$.*

Para um subconjunto de pares de servidor e nível de reconfiguração $\omega \subseteq \Omega$, seja:

1. $S(\omega) = \{S_i : (i,k) \in \omega\}$ o conjunto de servidores presentes em ω;
2. $f(\omega) = \sum_{(i,k)\in\omega} A_{i,k}$ o benefício induzido por ω; e
3. $u(\omega) = \sum_{(i,k)\in\omega} u_{i,k}$ a quantidade de recursos consumidos pelos servidores para as versões especificadas em ω.

Definição 7. *$\omega \subseteq \Omega$ é uma* **reconfiguração factível**[1] *se e somente as seguintes condições são satisfeitas:*

i) $|S(\omega)| = |\mathscr{S}|$;

[1] Solução factível para *PD*

ii) $u(\omega) \leq 1$.

A condição (i) indica que o conjunto ω deve conter exatamente um par (i,k) para cada servidor S_i. A condição (ii) corresponde à condição de escalonabilidade segundo a política EDF.

4.3.2 Algoritmo Guloso Densidade

Algoritmo 5: Algoritmo AGD

input: Servidores $\mathscr{S}$, parâmetros $A_{i,k}$ e $u_{i,k}$
initialize: Ordene os pares de $\Omega - \{(i,1) : S_i \in \mathscr{S}\}$ na sequência
$\langle (i_1,k_1),\ldots,(i_{|\Omega|-|\mathscr{S}|},k_{|\Omega|-|\mathscr{S}|})\rangle$ tal que $\hat{A}_{i_p,k_p} > \hat{A}_{i_q,k_q}$ ou $\hat{A}_{i_p,k_p} = \hat{A}_{i_q,k_q}$ e $\hat{u}_{i_p,k_p} \geq \hat{u}_{i_q,k_p}$ para todo $p < q$
initialize: $\omega := \emptyset$
initialize: $b := 1 - \sum_{S_i \in \mathscr{S}} u_{i,1}$
initialize: $t := 1$
while $t \leq |\Omega|$ **and** $|\omega| < |\mathscr{S}|$ **do**
  if $S_{i_t} \notin S(\omega)$ **and** $(u_{i_t,k_t} - u_{i_t,1}) \leq b$ **then**
    $\omega := \omega \cup \{(i_t,k_t)\}$
    $b := b - (u_{i_t,k_t} - u_{i_t,1})$
  $t := t + 1$
for $S_i \in \mathscr{S} - S(\omega)$ **do**
  $\omega := \omega \cup \{(i,1)\}$
return ω

O algoritmo guloso densidade (Algoritmo 5) apresenta dois passos distintos. Primeiro, são alocados os mínimos de recursos a cada um dos servidores para assegurar a factibilidade da reconfiguração. Esta reserva de recursos se faz necessária em função da equação (4.1d) que exige que cada servidor receba uma fatia dos recursos. Segundo, são distribuidos os recursos restantes aos servidores seguindo uma ordem não crescente do benefício adicional relativo $\hat{A}_{i,k} = (A_{i,k} - A_{i,1})/(u_{i,k} - u_{i,1})$ como chave primária de ordenação e decrescente de demanda de recurso relativo $\hat{u}_{i,k} = (u_{i,k} - u_{i,1})$ com chave secundária de ordenação.

Pré-alocando $u_{i,1}$ unidades a cada servidor $S_i \in \mathscr{S}$, o problema *PD* pode ser reformulado

como:

$$PD: \ fd = \text{Maximize} \sum_{S_i \in \mathscr{S}} \sum_{k \in K_i - \{1\}} (A_{i,k} - A_{i,1}) x_{i,k}^{ad} + \sum_{S_i \in \mathscr{S}} A_{i,1} \tag{4.7a}$$

Sujeito a :

$$\sum_{S_i \in \mathscr{S}} \sum_{k \in K_i - \{1\}} (u_{i,k} - u_{i,1}) x_{i,k}^{ad} \leq 1 - \sum_{S_i \in \mathscr{S}} u_{i,1} \tag{4.7b}$$

$$\sum_{k \in K_i - \{1\}} x_{i,k}^{ad} \leq 1, \ S_i \in \mathscr{S} \tag{4.7c}$$

$$x_{i,k}^{ad} \in \{0,1\}, \ S_i \in \mathscr{S}, k \in K_i - \{1\} \tag{4.7d}$$

Observe que existe uma relação de um-para-um entre as soluções da formulação (4.1a)–(4.1e) e da formulação (4.7a)–(4.7d). Em particular, para todo servidor $S_i \in \mathscr{S}$ se $\sum_{k \in K_i - \{1\}} x_{i,k}^{ad} = 0$ então $x_{i,1} = 1$ e $x_{i,k} = 0$ para todo $k \in K_i - \{1\}$. Caso contrário, se $\sum_{k \in K_i - \{1\}} x_{i,k}^{ad} = 1$, então $x_{i,1} = 0$ e $x_{i,k} = x_{i,k}^{ad}$ para todo $k \in K_i - \{1\}$.

Algoritmo 6: Algoritmo AGD-M

input: Servidores $\mathscr{S}$, parâmetros $A_{i,k}$ e $u_{i,k}$
initialize: $(i',k') := \arg \max_{(i,k) \in \Omega} \{A_{i,k} - A_{i,1} : u_{i,k} - u_{i,1} \leq 1 - \sum_{S_t \in \mathscr{S}} u_{t,1}\}$
initialize: $\omega' := \{(i,1) : S_i \in \mathscr{S}, i \neq i'\} \cup \{(i',k')\}$
initialize: $\omega := AGD(\mathscr{S}, \{A\}, \{u\})$
if $f(\omega') > f(\omega)$ **then**
  return ω'
else
  return ω

O algoritmo guloso densidade modificado (*AGD-M*, Algoritmo 6) retorna a reconfiguração produzida por *AGD*, ω, a menos que a reconfiguração de um servidor $S_{i'}$ em um nível de execução k', (i',k'), induza um valor objetivo maior que o produzido pelo algoritmo guloso densidade. Neste caso, *AGD-M* retorna $\omega' = \{(i,1) : S_i \in \mathscr{S}, i \neq i'\} \cup \{(i',k')\}$.

Teorema 1. *O desempenho do algoritmo guloso densidade modificado e o desempenho ótimo estão relacionados pela expressão:*

$$\textit{AGD-M}(I) \geq \frac{OPT(I) + \sum_{S_i \in \mathscr{S}} A_{i,1}}{2} \tag{4.8}$$

Prova: Seja I uma instância do problema de reconfiguração *PD*. Seja $\omega^\star$ a solução produzida por *AGD-M*, onde $\omega^\star = \omega$ se $f(\omega) \geq f(\omega')$ e, caso contrário, $\omega^\star = \omega'$. Defina $(i_\omega, k_\omega) =$

$\arg \max_{(i,k)\in\Omega:k>1} \{(A_{i,k} - A_{i,1})/(u_{i,k} - u_{i,1}) : (i,1) \in \omega\}$. Claramente,

$$OPT(I) \leq f(\omega) + \frac{A_{i_\omega,k_\omega} - A_{i_\omega,1}}{u_{i_\omega,k_\omega} - u_{i_\omega,1}}(1 - \sum_{(i,k)\in\omega} u_{i,k})$$

Há dois casos possíveis. Se $\omega^\star = \omega$, então vale:

$$OPT(I) \leq 2f(\omega) - \sum_{S_i\in\mathscr{S}} A_{i,1} = 2AGD\text{-}M(I) - \sum_{S_i\in\mathscr{S}} A_{i,1}$$
$$\implies AGD\text{-}M(I) \geq \frac{OPT(I) + \sum_{S_i\in\mathscr{S}} A_{i,1}}{2}$$

Por outro lado, se $\omega^\star = \omega'$, então vale:

$$OPT(I) \leq 2f(\omega') - \sum_{S_i\in\mathscr{S}} A_{i,1} = 2AGD\text{-}M(I) - \sum_{S_i\in\mathscr{S}} A_{i,1}$$
$$\implies AGD\text{-}M(I) \geq \frac{OPT(I) + \sum_{S_i\in\mathscr{S}} A_{i,1}}{2}$$

Assim demonstrando a relação (4.8) entre $OPT(I)$ e $AGD\text{-}M(I)$. □

Corolário 1. *O algoritmo guloso densidade modificado tem desempenho relativo $R_{AGD-M} \leq 2$.*

Prova: Do teorema acima, tem-se que:

$$\frac{OPT(I)}{AGD\text{-}M(I)} \leq \frac{OPT(I)}{(OPT(I) + \sum_{S_i\in\mathscr{S}} A_{i,1})/2} \leq 2$$

Demonstrando que $R_{AGD-M} = 2$. □

O algoritmo *AGD-M* produz as reconfigurações $\omega = \{(1,1),(2,1),(3,3),(4,5)\}$ com benefício total $f(\omega) = 1,863$ e $\omega' = \{(1,1),(2,1),(3,5),(4,1)\}$ com $f(\omega') = 1,5$ quando aplicado à instância I do exemplo. Logo a solução final é ω com benefício $f(\omega) = 1,863$. Observe que a solução ótima $\omega^\star$ gerada pelo algoritmo de programação dinâmica induz um benefício de $f(\omega^\star) = 1,863$. Concluímos que o algoritmo de aproximação obteve a solução ótima global e a relação (4.8) é obviamente satisfeita: $\sum_{S_i\in\mathscr{S}} A_{i,1} = 0,576$ e $OPT(I) = 1,863$, o que implica $(OPT(I) + \sum_{S_i\in\mathscr{S}} A_{i,1})/2 = 1,2195 \leq AGD\text{-}M(I)$.

4.3.3 Esquema de Aproximação Polinomial Completo

Da mesma forma que os algoritmos de programação dinâmica, podemos estender o esquema de aproximação polinomial completo (EAPC) [37] do problema da mochila ao problema de reconfiguração dinâmica. Tal esquema toma como base a formulação dual $\widehat{PD}(\psi)$ e assume que os parâmetros de benefício são inteiros não negativos, o que é garantido com o emprego

de $\Psi A_{i,k}$ no lugar de $A_{i,k}$ e $\Psi\psi$ no lugar de ψ. Seja Δ um limite superior para o benefício máximo—um limite trivial é nA_{max} onde $A_{max} = \max\{\Psi A_{i\kappa(i)} : S_i \in \mathscr{S}\}$. Lembre que a solução de $\widehat{PD}(\psi)$ via programação dinâmica tem limite superior $O(\Delta n \kappa_{max}) = O(n^2 A_{max} \kappa_{max})$ onde $\kappa_{max} = \max\{\kappa(i) : S_i \in \mathscr{S}\}$. Se os valores dos benefícios fossem limitados por um polinômio em n, então Δ também seria limitado por um polinômio em n e, consequentemente, o algoritmo executaria em tempo polinomial. A idéia por trás do EAPC está em ignorar os bits menos significativos dos benefícios de acordo com o parâmetro ε, tornando os benefícios limitados por um polinômio em n e $1/\varepsilon$. Tal estratégia leva a um algoritmo com tempo de execução limitado por um polinômio em n e $1/\varepsilon$, enquanto produz uma solução com objetivo de pelo menos $(1-\varepsilon)OPT(I)$.

Algoritmo 7: Algoritmo EAPC

input: Servidores $\mathscr{S}$, parâmetros $A_{i,k}$, $u_{i,k}, \varepsilon$
initialize: $\alpha := \frac{\varepsilon A_{max}}{n}$
initialize: Substitua $\Psi A_{i,k}$ por $\left\lfloor \frac{\Psi A_{i,k}}{\alpha} \right\rfloor$ e $\Psi\psi$ por $\left\lfloor \frac{\Psi\psi}{\alpha} \right\rfloor$ em $\widehat{PD}(\psi)$
Denomine o problema resultante de $\widehat{PD}(\psi)_\varepsilon$
Resolva $\widehat{PD}(\psi)_\varepsilon$ utilizando o algoritmo de programação dinâmica dual, obtendo a solução ótima ω_ε
return ω_ε

Seja I uma instância do algoritmo de reconfiguração PD onde os benefícios são substituídos por $\Psi A_{i,k}$. Seja $\omega^\star$ a solução ótima deste problema. Denomine por I_ε a instância obtida a partir de I por meio do arredondamento dos benefícios e seja fd_ε a função objetivo segundo os benefícios arredondados. Considere o benefício induzido por uma configuração $(i,k) \in \omega^\star$. Ao se arredondar os benefícios, $\alpha \left\lfloor \frac{\Psi A_{i,k}}{\alpha} \right\rfloor$ pode ser menor que $\Psi A_{i,k}$ mas não mais do que α. Portanto,

$$fd(\omega^\star) - \alpha fd_\varepsilon(\omega_\varepsilon) \leq n\alpha$$

O algoritmo de programação dinâmica dual produz uma solução ω_ε de qualidade igual ou superior a $\omega^\star$ segundo os custos arredondados. Portanto:

$$\begin{aligned} fd(\omega_\varepsilon) &\geq \alpha fd_\varepsilon(\omega_\varepsilon) \\ &\geq fd(\omega^\star) - n\alpha \\ &= OPT(I) - n\left\lfloor \frac{\varepsilon A_{max}}{n} \right\rfloor \\ &\geq OPT(I) - \varepsilon A_{max} \\ &\geq (1-\varepsilon)OPT(I) \end{aligned}$$

ε	$\max\{\left\lfloor \frac{\Psi A_{max}}{\alpha} \right\rfloor : S_i \in \mathscr{S}\}$	$fd(\omega_\varepsilon)$	$n^2 \left\lfloor \frac{n}{\varepsilon} \right\rfloor \kappa_{max}$
10^{-3}	1000	1,863	320000
10^{-2}	100	1,863	32000
10^{-1}	10	1,863	3200
10^{0}	1	1,401	320

Tabela 4.7: Aplicação do esquema de aproximação polinomial à instância exemplo.

uma vez que $OPT(I) \geq A_{max}$, sob a hipótese que existe uma solução factível que contém uma configuração qualquer $(i,k) \in \Omega$ com $k \neq 1$. Note que o algoritmo de programação dinâmica dual resolve I_ε com um tempo de execução $O(n^2 \left\lfloor \frac{A_{max}}{\alpha} \right\rfloor \kappa_{max}) = O(n^2 \left\lfloor \frac{n}{\varepsilon} \right\rfloor \kappa_{max})$, que é polinomial em n e $1/\varepsilon$.

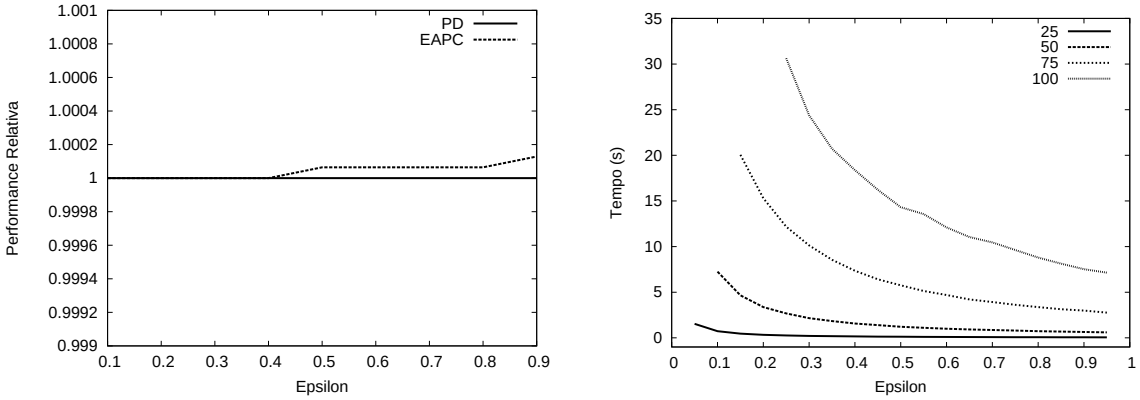
A Tabela 4.7 mostra a aplicação do esquema de aproximação polinomial à instância exemplo definida nas Tabelas 4.1 e 4.2. Pode ser visto que o valor de ε pode ser aumentado significativamente antes que exista alguma degradação na qualidade da solução. Ao mesmo tempo, o valor de $(n^2 \left\lfloor \frac{n}{\varepsilon} \right\rfloor \kappa_{max})$ chegou a um milésimo do seu valor original na primeira degradação da qualidade de solução observada.

4.4 Análise Numérica

Para avaliar o desempenho dos algoritmos de aproximação (EAPC e AGD-M), diversos testes foram conduzidos, inicialmente de forma isolada e posteriormente com o objetivo de comparar o desempenho das duas abordagens. Ambos os algoritmos foram implementados em C++ e executados sobre Linux em um processador Intel Core2Duo 2.20GHz CPU com 2GB de RAM.

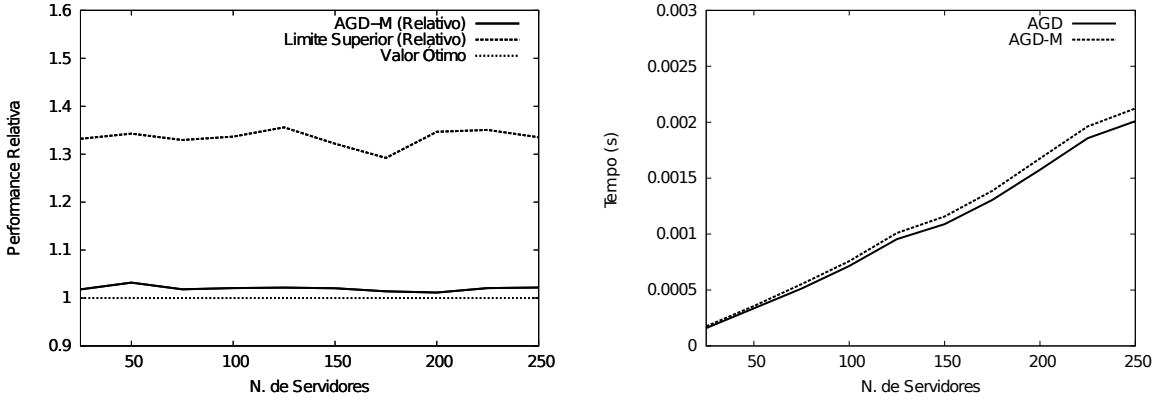
Para avaliar o desempenho do EAPC, 100 instâncias aleatórias de *PD* com $|\mathscr{S}| = 100$ e $3 \leq \kappa(i) \leq 10$ foram geradas, usando distribuições uniformes para todas as variáveis. As soluções ótimas foram alcançadas usando programação dinâmica e um valor médio foi obtido. As mesmas 100 instâncias foram então solucionadas com EAPC usando valores de ε entre 0.1 e 0.9.

Como pode se ver na Figura 4.1(a), que mostra a performance relativa do EAPC, o valor de ε pode ser significativamente incrementado antes que a degradação da qualidade da solução ocorra, e mesmo então, o decréscimo em valor é pequeno. Com $\varepsilon = 0.9$ a solução média dada pelo EAPC para as 10 instâncias de *PD* foi 99.978% do valor ótimo médio. A Figura 4.1(b) mostra o tempo de execução do EAPC para $25 \leq |\mathscr{S}| \leq 100$ e $0.05 \leq \varepsilon \leq 0.95$. Pode-se ver que o tempo de execução diminui drasticamente com o uso de valores maiores de ε, e nas instâncias



(a) Qualidade de solução para vários valores de ε. (b) Tempo de execução para vários valores de $|\mathscr{S}|$ e ε.

Figura 4.1: Avaliação do EAPC.



(a) Performance relativa do AGD-M. (b) Tempos de execução do AGD e AGD-M.

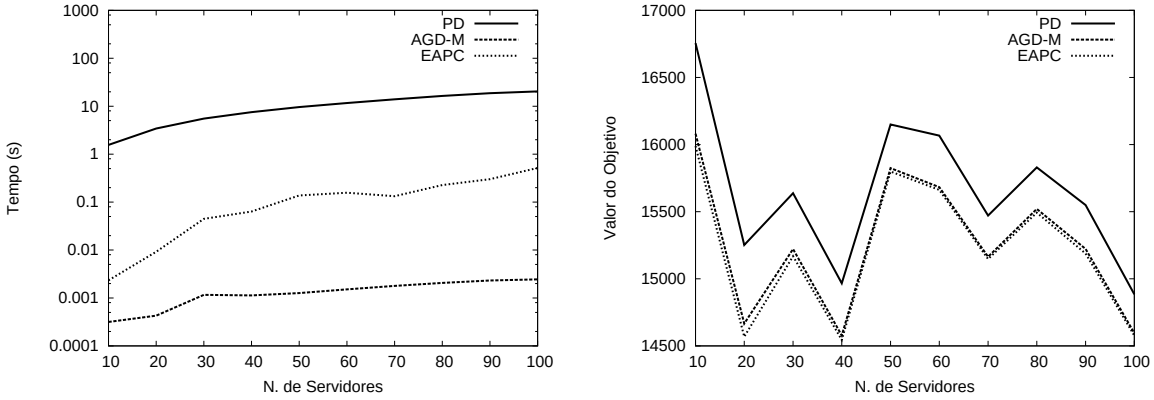
Figura 4.2: Avaliação do AGD-M.

com 100 servidores a diminuição no tempo de execução de $\varepsilon = 0.25$ para $\varepsilon = 0.95$ foi maior que 75%. É válido notar que a plotagem do tempo de execução para valores maiores de $|\mathscr{S}|$ não começa antes que ε atinja um certo valor pois, para as instâncias geradas, um valor menor de ε levaria a $\alpha < 1$ e a um problema desnecessariamente mais complexo.

Para avaliar o tempo de execução e a qualidade das soluções do algoritmo AGD, 50 instâncias aleatórias de *PD* com $25 \leq |\mathscr{S}| \leq 250$ (em incrementos de 25) e $3 \leq \kappa(i) \leq 10$ foram geradas, novamente usando distribuições uniformes para todas variáveis. As soluções ótimas foram obtidas usando programação dinâmica, e seu valor médio foi calculado.

A Figura 4.2(a) mostra o desempenho relativo do AGD-M. Pode-se ver que as soluções dadas pelo AGD-M ficam, na prática, muito mais próximas do valor ótimo do que do limite superior calculado pelo Teorema 4.8.

A Figura 4.2(b) mostra que o tempo de execução de ambos AGD e AGD-M para diferentes



(a) Tempos de execução de AGD-M e EAPC para a mesma solução.

(b) Valores de função objetivo.

Figura 4.3: Comparação entre AGD-M e EAPC.

valores de $|\mathscr{S}|$. O AGD-M é ligeiramente mais lento devido à procura inicial por uma solução trivial que ele realiza. Uma comparação com o algoritmo *PD* (não mostrado no gráfico) mostra que estes métodos têm um tempo de execução aproximadamente vinte mil vezes menor.

Finalmente, para comparar o tempo de execução do AGD-M com o do EAPC, o seguinte teste foi conduzido: 100 instâncias de *PD* com $10 \leq |\mathscr{S}| \leq 100$ (em incrementos de 10) e $3 \leq \kappa(i) \leq 10$ foram geradas. Suas soluções ótimas foram obtidas usando programação dinâmica, e seu valor valor médio para cada valor de $|\mathscr{S}|$ foi calculado. As médias das soluções dadas pelo AGD-M também foram calculadas. O EAPC foi então executado com valores cada vez maiores de ε até que a função objetivo (mostrada na Figura 4.3(b)) se tornasse menor que aquela dada pelo AGD-M para cada instância. Isto tem o efeito de mostrar quanto tempo o EAPC leva para retornar uma solução tão próxima quanto possível daquela dada pelo AGD-M. A Figura 4.3(a) mostra que, apesar de muito mais rápido que programação dinâmica, o EAPC leva pelo menos 7 e até 210 vezes mais tempo para retornar uma solução igual àquela dada pelo AGD-M, e a diferença aumenta junto com o valor de $|\mathscr{S}|$.

Os resultados indicam que o AGD-M deve ser favorecido já que ele produz soluções próximas do ótimo na maioria dos casos. Quando a diferença entre o ótimo e a solução dada pelo AGD-M é substancial, o EAPC pode ser usado uma vez que a qualidade da solução pode ser trocada por velocidade de solução através do parâmetro ε.

4.5 Sumário

Neste capítulo foi apresentado um modelo de sistema que assume a existência de valores conhecidos para a utilização de cada um dos modos $U_{i,k}$ dos servidores S_i. Foram apresentados

ainda os algoritmos de programação dinâmica primal e dual para o problema, o segundo dos quais foi utilizado na definição de um esquema de aproximação polinomial completo (EAPC). Um algoritmo de aproximação guloso (AGD-M) também foi apresentado, e uma comparação entre ele e o EAPC mostrou que o AGD-M tem uma melhor relação custo/benefício por trazer boas soluções em uma fração do tempo que o EAPC leva.

5 *Modelos Contínuos*

A versão contínua do problema de otimização é caracterizada pela ausência de configurações pré-definidas de cada servidor S_i. Em outras palavras, não há valores pré-estabelecidos para $u_i = Q_i/T_i$. Resolver a versão contínua é, portanto, encontrar os valores para cada u_i, $0 < u_i \leq U_i$ de tal forma que haja benefício máximo para o sistema. Mais formalmente,

$$PC: \ fc = \text{Maximize} \sum_{S_i \in \mathscr{S}} A_i(U_i, u_i, v_i) \tag{5.1a}$$

Sujeito a :

$$\sum_{S_i \in \mathscr{S}} u_i \leq 1 \tag{5.1b}$$

$$0 < u_i \leq U_i, \ \forall S_i \in \mathscr{S}_i \tag{5.1c}$$

Semelhantemente à versão discreta, a escalonabilidade é garantida pela equação (5.1b). O fato de não haver mais benefício em atribuir valores de $u_i > U_i$ ao servidor S_i é expressado pela restrição (5.1c).

Introduzindo limites inferiores para u_i e uma função objetivo linear, o problema (5.1a)–(5.1c) se torna um problema de programação linear que admite uma solução analítica. A versão linear é dada por:

$$PC': \ \text{Maximize} \sum_{S_i \in \mathscr{S}} A_i u_i \tag{5.2a}$$

Sujeito a :

$$\sum_{S_i \in \mathscr{S}} u_i \leq 1 \tag{5.2b}$$

$$L_i \leq u_i \leq U_i, \ \forall S_i \in \mathscr{S}_i \tag{5.2c}$$

onde L_i (U_i) é um limite inferior (superior) para alocação de processador ao servidor S_i, A_i é a taxa de benefício por unidade de processador induzida pela tarefa e u_i é o consumo de processador por S_i.

Suponha que os servidores estão em ordem decrescente de taxa de benefício, isto é, $A_i \geq A_{i+1}$ para todo $i < n$. Seja $i^* = \arg\max\{\sum_{j=1}^{i}(U_j - L_j) \leq 1 - \sum_{j=1}^{n} L_j : S_i \in \mathscr{S}\}$. Pode-se demonstrar que a solução ótima para PC' é:

$$u_i = \begin{cases} U_i, & \text{se } i \leq i^* \\ 1 - \sum_{j=1}^{i^*} U_j - \sum_{j=i^*+2}^{n} L_j, & \text{se } i = i^* + 1 \\ L_i, & \text{se } i \geq i^* + 2 \end{cases} \tag{5.3}$$

O Teorema 2 apresenta uma demonstração de otimalidade da solução (5.3).

Nota-se que $\sum_{i=1}^{n} u_i = U$, o que significa que alocar a banda de acordo com a Equação (5.3) maximiza o uso total de banda do sistema, uma condição necessária para que a solução $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ seja ótima.

Lema 1. *Seja* $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ *uma solução factível para o problema* PC'*, definido pelas Equações* (5.2a)-(5.2c)*. Se* $\sum_{i=1}^{n} u_i < U$*, existe outra solução* $\mathbf{u}' = (u'_1, u'_2, \ldots, u'_n)$ *tal que* $fc(\mathbf{u}') > fc(\mathbf{u})$ *e* $\sum_{i=1}^{n} u'_i = U$*.*

Demonstração. Seja $\sum_{i=1}^{n} u_i = U' < U$. Deve existir, portanto, u_i e $\varepsilon > 0$ tal que $U' + \varepsilon \leq U$ e $u_i + \varepsilon \leq U_i$. Seja $u'_j = u_j$ para todo $j \neq i$ e $u'_i = u_i + \varepsilon$. Uma vez que a função objetivo é monotonicamente crescente em u_i, é claro que $fc(\mathbf{u}') > fc(\mathbf{u})$. A repetição deste argumento leva a $\sum_{i=1}^{n} u'_i = U$. □

O Teorema abaixo mostra de forma concisa a otimalidade da solução derivada.

Teorema 2. *O problema* PC'*, definido pelas equações* (5.2a)-(5.2c)*, tem solução dada pela Equação* (5.3)*.*

Demonstração. A prova será feita por contradição. Considere um sistema $\mathscr{S}$ composto por n servidores cujos benefícios são ordenados de forma não-crescente em $A_1, \ldots, A_n$. Assumindo $\mathbf{u} = (u_1, \ldots, u_n)$ como a solução não ótima dada pela Equação (5.3). Deve, portanto, existir $\mathbf{u}' = (u'_1, \ldots, u'_n)$ tal que

$fc(\mathbf{u}') > fc(\mathbf{u})$. Se $\alpha_i = u'_i - u_i$, pode-se escrever

$$fc(\mathbf{u}') - fc(\mathbf{u}) = \sum_{u'_{(j)} > u_{(j)}} \alpha_{(j)} A_{(j)} + \sum_{u'_{(k)} < u_{(k)}} \alpha_{(k)} A_{(k)} > 0 \tag{5.4}$$

Se $u'_{(j)} > u_{(j)}$, pode-se concluir que $j \geq i^* + 1$ uma vez que $u_{(j)} = U_{(j)}$ para todo $j < i^* + 1$. Além disso, $u'_{(k)} < u_{(k)}$, $k \leq i^* + 1$ uma vez que $u_{(k)} = L_{(k)}$ para todo $k > i^* + 1$. Destas observações e do fato que $A_{(i+1)} \leq A_{(i)}$ para todo $i < n$ a desigualdade (5.4) implica que

i	L_i	U_i	A_i
1	0.0667	0.2000	3
2	0.0667	0.6000	2
3	0.0400	0.3333	1

Tabela 5.1: Instância exemplo do PC'.

i	u_i
1	0.2000
2	0.6000
3	0.2000

Tabela 5.2: Solução da instância exemplo do PC'.

$$A_{(i^*+1)} \sum_{u'_{(j)} > u_{(j)}} \alpha_{(j)} + A_{(i^*+1)} \sum_{u'_{(k)} < u_{(k)}} \alpha_{(k)} > 0$$

Através da Equação (5.3) sabe-se que $\sum_{i=1}^{n} u_i = U$. Pelo Lema 1, também pode-se assumir que $\sum_{i=1}^{n} u'_i = U$. Portanto,

$$\alpha = \sum_{u'_{(j)} > u_{(j)}} \alpha_{(j)} = - \sum_{u'_{(k)} < u_{(k)}} \alpha_{(k)}$$

então $\alpha(A_{(i^*+1)} - A_{(i^*+1)}) > 0$, o que é uma contradição. □

A Tabela 5.1 mostra uma instância de PC'. Para aplicar a solução (5.3), é necessário ordenar os servidores pela ordem não-crescente de benefício A_i, levando a ordem apresentada nas linhas desta Tabela.

A partir deste ponto, a definição de i^*, o índice do último servidor a receber seu U_i de utilização, é questão apenas de calcular o somatório que garante a escalonabilidade do sistema. Feito este cálculo chega-se em $i^* = 2$, o que leva à solução ótima apresentada na Tabela 5.2.

5.1 Modelo com Representação de Consumo e Período

O modelo anterior, por usar os valores de utilização u_i como variáveis de decisão, deixa em aberto os valores de tempo de computação Q_i e o período T_i das tarefas a serem reconfiguradas. A seguinte extensão do modelo contínuo visa separar estes valores de forma a permitir que cada servidor redefina o tempo de computação e o período, que passam a ser variáveis de decisão. Isto permite que o projetista do sistema mantenha o período das tarefas dentro dos limites T_i^{min} e T_i^{max} e o tempo de computação dentro dos limites Q_i^{min} e Q_i^{max}, assim limitando a degradação. Apesar desta vantagem, este novo mecanismo de reconfiguração leva a uma formulação não-

linear e não-convexa:

$$PC_2: \ fc = \text{Maximize} \sum_{S_i \in \mathscr{S}} A_i \frac{Q_i}{T_i} \tag{5.5a}$$

Sujeito a :

$$\sum_{S_i \in \mathscr{S}} \frac{Q_i}{T_i} \leq 1 \tag{5.5b}$$

$$Q_i^{min} \leq Q_i \leq Q_i^{max}, \ \forall S_i \in \mathscr{S} \tag{5.5c}$$

$$T_i^{min} \leq T_i \leq T_i^{max}, \ \forall S_i \in \mathscr{S} \tag{5.5d}$$

Podemos entretanto linearizá-lo através de uma mudança de variável:

$$u_i = \frac{Q_i}{T_i} \tag{5.6}$$

Assim, PC_2 se torna o seguinte problema de programação linear:

$$\text{Maximize} \sum_{S_i \in \mathscr{S}} A_i u_i \tag{5.7a}$$

Sujeito a : (5.7b)

$$\sum_{S_i \in \mathscr{S}} u_i \leq 1 \tag{5.7c}$$

$$Q_i^{min} \leq Q_i \leq Q_i^{max}, \ \forall S_i \in \mathscr{S} \tag{5.7d}$$

$$\frac{Q_i}{T_i^{max}} \leq u_i \leq \frac{Q_i}{T_i^{min}}, \ \forall S_i \in \mathscr{S} \tag{5.7e}$$

Vamos agora definir outra mudança de variável, fazendo:

$$Q_i = Q_i^{min} + \Delta Q_i$$

$$u_i = \frac{Q_i}{T_i^{max}} + \Delta u_i = \frac{Q_i^{min} + \Delta Q_i}{T_i^{max}} + \Delta u_i$$

Com isso o problema acima assume a forma:

$$\text{Maximize} \sum_{S_i \in \mathscr{S}} \frac{A_i}{T_i^{max}} Q_i^{min} + \sum_{S_i \in \mathscr{S}} \frac{A_i}{T_i^{max}} \Delta Q_i + \sum_{S_i \in \mathscr{S}} A_i \Delta u_i \tag{5.8a}$$

Sujeito a :

$$\sum_{S_i \in \mathscr{S}} \Delta u_i + \sum_{S_i \in \mathscr{S}} \frac{\Delta Q_i}{T_i^{max}} \leq 1 - \sum_{S_i \in \mathscr{S}} \frac{Q_i^{min}}{T_i^{max}} \tag{5.8b}$$

$$\Delta u_i - \Delta Q_i \left(\frac{1}{T_i^{min}} - \frac{1}{T_i^{max}} \right) \leq Q_i^{min} \left(\frac{1}{T_i^{min}} - \frac{1}{T_i^{max}} \right), \ \forall S_i \in \mathscr{S} \tag{5.8c}$$

$$\Delta Q_i \leq Q_i^{max} - Q_i^{min}, \ \forall S_i \in \mathscr{S} \tag{5.8d}$$

$$\Delta u_i \geq 0, \ \forall S_i \in \mathscr{S} \tag{5.8e}$$

$$\Delta Q_i \geq 0, \ \forall S_i \in \mathscr{S} \tag{5.8f}$$

Suponha que os servidores estão em ordem decrescente de taxa de benefício, isto é, $A_i \geq A_{i+1}$ para todo $i < n$. Pode-se mostrar que uma solução ótima para o problema dado por (5.8a)–(5.8f) é obtida como segue. Seja:

$$i^* = \arg\max \left\{ \sum_{j=1}^{i} \frac{Q_j^{max}}{T_j^{min}} \leq 1 - \sum_{j=i+1}^{n} \frac{Q_j^{min}}{T_j^{max}} : i \leq n \right\}$$

$$\alpha(i^*) = 1 - \sum_{i=1}^{i^*} \frac{Q_i^{max}}{T_i^{min}} - \sum_{i=i^*+1}^{n} \frac{Q_i^{min}}{T_i^{max}}$$

Há duas possibilidades:

Caso 1:

$$\frac{Q_{i^*+1}^{max} - Q_{i^*+1}^{min}}{T_{i^*+1}^{max}} \not< \alpha(i^*)$$

Então a solução é dada por:

i) $\Delta Q_i = Q_i^{max} - Q_i^{min}$ e $\Delta u_i = Q_i^{max} \left(\frac{1}{T_i^{min}} - \frac{1}{T_i^{max}} \right)$ para $i = 1, \ldots, i^*$

ii) $\Delta Q_{i^*+1} = \alpha(i^*) T_{i^*+1}^{max}$ e $\Delta u_{i^*+1} = 0$

iii) $\Delta Q_i = 0$ e $\Delta u_i = 0$ para $i = i^* + 2, \ldots, n$

Caso 2:

$$\frac{Q_{i^*+1}^{max} - Q_{i^*+1}^{min}}{T_{i^*+1}^{max}} < \alpha(i^*)$$

Então a solução é dada por:

i) $\Delta Q_i = Q_i^{max} - Q_i^{min}$ e $\Delta u_i = Q_i^{max} \left(\frac{1}{T_i^{min}} - \frac{1}{T_i^{max}} \right)$ para $i = 1, \ldots, i^*$

ii) $\Delta Q_{i^*+1} = Q^{max}_{i^*+1} - Q^{min}_{i^*+1}$ e $\Delta u_{i^*+1} = \alpha(i^*) - \frac{Q^{max}_{i^*+1} - Q^{min}_{i^*+1}}{T^{max}_{i^*+1}}$

iii) $\Delta Q_i = 0$ e $\Delta u_i = 0$ para $i = i^* + 2, \ldots, n$

Em termos da formulação (5.5a)–(5.5d), a solução ótima é dada por:

Caso 1:

$$\frac{Q^{max}_{i^*+1} - Q^{min}_{i^*+1}}{T^{max}_{i^*+1}} \not< \alpha(i^*)$$

Então a solução é dada por:

i) $Q_i = Q^{max}_i$ e $T_i = T^{min}_i$ para $i = 1, \ldots, i^*$

ii) $Q_{i^*+1} = Q^{min}_{i^*+1} + \alpha(i^*) T^{max}_{i^*+1}$ e $T_{i^*+1} = T^{max}_{i^*+1}$

iii) $Q_i = Q^{min}_i$ e $T_i = T^{max}_i$ para $i = i^* + 2, \ldots, n$

Caso 2:

$$\frac{Q^{max}_{i^*+1} - Q^{min}_{i^*+1}}{T^{max}_{i^*+1}} < \alpha(i^*)$$

Então a solução é dada por:

i) $Q_i = Q^{max}_i$ e $T_i = T^{min}_i$ para $i = 1, \ldots, i^*$

ii) $Q_{i^*+1} = Q^{max}_{i^*+1}$ e $T_{i^*+1} = \frac{Q^{max}_{i^*+1} T^{max}_{i^*+1}}{Q^{min}_{i^*+1} + \alpha(i^*) T^{max}_{i^*+1}}$

iii) $Q_i = Q^{min}_i$ e $T_i = T^{max}_i$ para $i = i^* + 2, \ldots, n$

Exemplo

Os dados da instância exemplo são:

i	Q^{min}_i	Q^{max}_i	T^{min}_i	T^{max}_i	A_i
1	1	5	15	25	3
2	2	4	20	30	3
3	2	6	10	30	2
4	1	5	15	25	1
5	2	4	20	30	1

Resolvendo o modelo (5.7b)–(5.7e) usando um solver de programação linear, obtém-se a solução abaixo com valor objetivo 2.42667:

i	Q_i	u_i	T_i
1	5	0.333333	15
2	4	0.2	20
3	6	0.36	16.6667
4	1	0.04	25
5	2	0.0666667	30

Resolvendo o modelo (5.8a)–(5.8f), obtém-se a solução abaixo com valor objetivo 1.8674 + 0.56 = 2.4274:

i	ΔQ_i	Δu_i	Q_i	u_i	T_i
1	4	0.1335	5	0.3335	14.9925
2	2	0.0667	4	0.2	20
3	4	0.16	6	0.36	16.6667
4	0	0	1	0.04	25
5	0	0	2	0.0667	29.985

A discrepância entre as soluções se deve ao emprego de poucas casas decimais na representação (5.8a)–(5.8f).

Aplicando o procedimento analítico de solução descrito acima, verifica-se que $i^* = 2$ e $\alpha(i) = 0.2933$. Uma vez que $\frac{Q^{max}_{i^*+1} - Q^{min}_{i^*+1}}{T^{max}_{i^*+1}} = 0.1333 < 0.2933 = \alpha(i^*)$, o caso 2 define a solução que segue abaixo:

i	ΔQ_i	Δu_i	Q_i	u_i	T_i
1	4	0.1333	5	0.3333	15
2	2	0.0667	4	0.2	20
3	4	0.16	6	0.36	16.6667
4	0	0	1	0.04	25
5	0	0	2	0.0667	30

5.2 Equivalência entre Modelos com Carga Relativa e com Representação de Consumo e Período

Aqui desenvolvemos a equivalência entre o modelo com carga relativa (PC'), dado por (5.2a)–(5.2c), e o modelo com representação de consumo e período (PC_2), dado por (5.5a)–(5.5d).

Lema 2. *O modelo PC' é equivalente a PC_2 se $u_i = Q_i/T_i$, $L_i = Q_i^{min}/T_i^{max}$, e $U_i = Q_i^{max}/T_i^{min}$ para todo $S_i \in \mathscr{S}$.*

Prova: Seja $(T,Q) = (T_i, Q_i : S_i \in \mathscr{S})$ uma solução para PC_2. Seja $\mathbf{u} = (u_i = Q_i/T_i : S_i \in \mathscr{S})$ a solução candidata para PC'. A restrição (5.5d) implica $1/T_i^{max} \leq 1/T_i \leq 1/T_i^{min}$ e juntamente com (5.5c) tem-se $Q_i^{min}/T_i^{max} \leq Q_i/T_i \leq Q_i^{max}/T_i^{max}$ e, portanto, $L_i \leq u_i \leq U_i$ para todo $S_i \in \mathscr{S}$. A restrição (5.2a) é satisfeita por u pois (5.5b) é sastifeita por (T,Q). Logo $\mathbf{u}$ é factível para PC'. Além disso, $\mathbf{u}$ e (T,Q) têm o mesmo valor objetivo pois $\sum_{S_i \in \mathscr{S}} A_i Q_i/T_i = \sum_{S_i \in \mathscr{S}} A_i u_i$.

Tome uma solução $\mathbf{u} = (u_i : S_i \in \mathscr{S})$ para PC'. Esta satisfaz a restrição (5.2c) que é expressa como $Q_i^{min}/T_i^{max} \leq Q_i/T_i \leq Q_i^{max}/T_i^{max}$. Há que se encontrar uma solução (T,Q) factível para PC_2 tal que $Q_i/T_i = u_i$. Fazendo $Q_i = u_i T_i$, as restrições (5.5c) e (5.5d) se tornam $Q_i^{min} \leq u_i T_i \leq Q_i^{max}$ e $T_i^{min} \leq T_i \leq T_i^{max}$. A solução $(T,Q) = (T_i, T_i u_i : S_i \in \mathscr{S})$ é factível se existir T_i tal que $\max\{T_i^{min}, Q_i^{min}/u_i\} \leq T_i \leq \min\{T_i^{max}, Q_i^{max}/u_i\}$ para todo $S_i \in \mathscr{S}$. Não existiria T_i satisfazendo estas desigualdades somente se $T_i^{min} > Q_i^{max}/u_i$ ou se $T_i^{max} < Q_i^{min}/u_i$. Mas isto ocorreria apenas se $u_i > U_i$ ou se $u_i < L_i$. Logo (T,Q) é factível para PC_2 pois a desigualdade (5.5b) também é satisfeita. Note que u e (T,Q) têm o mesmo valor objetivo.

Conclui-se que PC' e PC_2 são equivalentes. □

Para ilustrar a equivalência, tome a instância de PC_2 dada na Seção 5.1. Os dados da instância PC' equivalente são:

i	L_i	U_i	A_i
1	0.0400	0.3333	3
2	0.0667	0.2000	3
3	0.0667	0.6000	2
4	0.0400	0.3333	1
5	0.0667	0.2000	1

O algoritmo para solução de PC' determina que $i^* = 2$, produzindo a solução:

i	u_i	T_i	Q_i
1	0.3333	15	5
2	0.2000	20	4
3	0.3600	10	3.6
4	0.0400	25	1
5	0.0667	30	2

Para obter a solução para PC_2, utilizou-se $T_i = \max\{T_i^{min}, Q_i^{min}/u_i\}$ e $Q_i = T_i u_i$. O valor da função objetivo induzida por esta solução é 2.42667, que, como esperado, é o valor exato alcançado pela solução de PC_2 utilizando um solver de programação linear.

5.3 Análise Numérica

Problemas de diferentes valores de $|\mathscr{S}|$ foram gerados para verificar o tempo de execução do algoritmo. Especificamente, $25 \leq |\mathscr{S}| \leq 250$. Todos valores foram gerados usando uma distribuição uniforme tanto para valores de utilização quanto de benefício. A média do tempo de solução de 5 instâncias de cada tamanho é mostrada para minimizar erros de medição. A Figura 5.1 mostra os tempos de execução do solucionador analítico da formulação *PC*. Em comparação com os algoritmos de aproximação do modelo discreto, este algoritmo leva aproximadamente a metade do tempo para instâncias do mesmo tamanho.

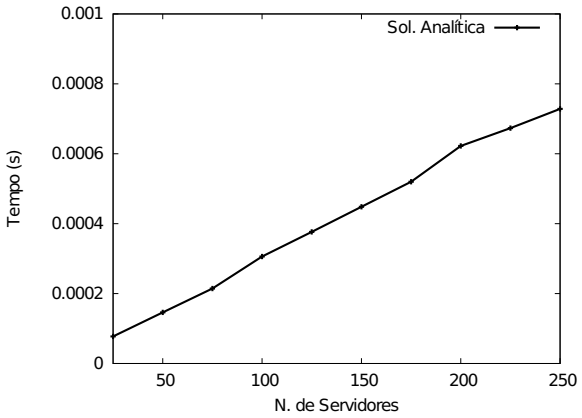


Figura 5.1: Tempo de execução da solução analítica.

5.4 Sumário

Neste capítulo foram apresentados dois modelos de sistema que não assumem a pré-determinação de modos bem definidos, mas um intervalo contínuo de possíveis utilizações que podem ser reservadas para cada tarefa. Para este modelo foi apresentado um algoritmo que alcança a solução analiticamente. Ensaios experimentais mostraram que a sua execução é ainda mais rápida que a dos algoritmos gulosos do modelo discreto do capítulo anterior.

6 *Modelos Híbridos*

Para suportar aplicações com múltiplas implementações da mesma tarefa e permitir que a utilização atribuída a cada uma seja ajustável, foi definido um modelo que permite a definição de níveis discretos de Q_i como no modelo discreto, porém admitem-se ajustes contínuos no período T_i. Formalmente,

$$PH : fh = \text{ Maximize} \quad \sum_{S_i \in \mathscr{S}} \sum_{k \in K_i} A_{i,k} \frac{Q_{i,k}}{T_{i,k}} x_{i,k} \tag{6.1a}$$

$$\text{Sujeito a : } \sum_{S_i \in \mathscr{S}} \sum_{k \in K_i} \frac{Q_{i,k}}{T_{i,k}} x_{i,k} \leq 1 \tag{6.1b}$$

$$T_{i,k}^{min} \leq T_{i,k} \leq T_{i,k}^{max},\ S_i \in \mathscr{S},\ k \in K_i \tag{6.1c}$$

$$\sum_{k \in K_i} x_{i,k} = 1,\ S_i \in \mathscr{S} \tag{6.1d}$$

$$x_{i,k} \in \{0,1\},\ S_i \in \mathscr{S},\ k \in K_i \tag{6.1e}$$

onde $A_{i,k}$ é o valor de benefício, $Q_{i,k}$ é o orçamento máximo, $T_{i,k}^{min}$ é o período mínimo e $T_{i,k}^{max}$ é o período máximo para o k^{esimo} modo do servidor S_i, sendo $x_{i,k}$ e $T_{i,k}$ variáveis de decisão. Se o servidor S_i opera no modo k, então $x_{i,k} = 1$ e o período deve-se encontrar no intervalo $[T_{i,k}^{min}, T_{i,k}^{max}]$. Sendo Ω o conjunto de modos (i,k) de todos servidores S_i do conjunto $\mathscr{S}$, as seguintes suposições são feitas:

1. $T_{i,k}^{max} \geq T_{i,k}^{min}$ para todo $(i,k) \in \Omega$;
2. $Q_{i,k-1} \leq Q_{i,k}$ para todo $S_i \in \mathscr{S}$ e $k \in K_i - \{1\}$;
3. $T_{i,k-1}^{min} \geq T_{i,k}^{max}$ para todo $S_i \in \mathscr{S}$ e $k \in K_i - \{1\}$.

Apesar do modelo ser de natureza inteiro-mista não-linear, ele pode ser redefinido como um programa inteiro-misto linear através de uma troca de variável:

$$u_{i,k} = \frac{Q_{i,k}}{T_{i,k}} x_{i,k} \implies T_{i,k} = \frac{Q_{i,k}}{u_{i,k}} x_{i,k}$$

Então, a formulação híbrida é redefinida como:

$$PH : fh = \text{ Maximize } \sum_{(i,k)\in\Omega} A_{i,k} u_{i,k} \tag{6.2a}$$

$$\text{Sujeito a : } \sum_{(i,k)\in\Omega} u_{i,k} \leq 1 \tag{6.2b}$$

$$\frac{Q_{i,k}}{T_{i,k}^{max}} x_{i,k} \leq u_{i,k} \leq \frac{Q_{i,k}}{T_{i,k}^{min}} x_{i,k},\ (i,k) \in \Omega \tag{6.2c}$$

$$\sum_{k\in K_i} x_{i,k} = 1,\ S_i \in \mathscr{S} \tag{6.2d}$$

$$x_{i,k} \in \{0,1\},\ (i,k) \in \Omega \tag{6.2e}$$

Esta reformulação é linear com $x_{i,k}$ e $u_{i,k}$ como variáveis de decisão binária e contínua, respectivamente. O modelo híbrido generaliza os modelos discreto e contínuo: ele se torna o modelo discreto se $T_{i,k}^{max} = T_{i,k}^{min}$ para todo $(i,k) \in \Omega$; e ele é redutível ao modelo contínuo se $\frac{Q_{i,k}}{T_{i,k}^{max}} = \frac{Q_{i,k-1}}{T_{i,k-1}^{min}}$ para todo $S_i \in \mathscr{S}$ e $k > 1$. Obviamente, o modelo híbrido é NP-Difícil.

6.1 Algoritmo de Programação Dinâmica

Sendo $PH_i(u)$ um sub-problema restrito aos servidores $\mathscr{S}_i = \{S_1, \ldots, S_i\}$ e com uma disponibilidade de utilização de $u\delta$, onde $u \in \{0, \ldots, \Delta\}$ e Δ é um inteiro determinando o número de intervalos nos quais a utilização total é dividida, cada intervalo com comprimento $\delta = 1/\Delta$. Formalmente, $PH_i(u)$ é definido como:

$$PH_i(u):\ fh_i(u) = \max \sum_{k \in K_i} A_{i,k}u_{i,k} + fh_{i-1}(u-u_i) \tag{6.3a}$$

$$\text{Sujeito a}: \sum_{k \in K_i} u_{i,k} \leq \delta u_i \tag{6.3b}$$

$$\frac{Q_{i,k}}{T_{i,k}^{max}} x_{i,k} \leq u_{i,k} \leq \frac{Q_{i,k}}{T_{i,k}^{min}} x_{i,k},\ k \in K_i \tag{6.3c}$$

$$\sum_{k \in K_i} x_{i,k} = 1 \tag{6.3d}$$

$$u_i \in \{0, \ldots, u\} \tag{6.3e}$$

$$x_{i,k} \in \{0,1\},\ k \in K_i \tag{6.3f}$$

onde $fh_i(u)$ recebe o valor de $-\infty$ quando $PH_i(u)$ é infactível. O caso terminal $PH_1(u)$ é idêntico ao $PH_i(u)$ dado acima, exceto na função objetivo, que se torna $\sum_{k \in K_i} A_{i,k}u_{i,k}$. Claramente, $PH_n(\Delta)$ é uma aproximação de PH que se torna mais precisa e leva $fh_n(\Delta)$ na direção de fh quando Δ tende a ∞.

O conjunto $\{PH_i(u)\}$ de sub-problemas é solucionável através de programação dinâmica, como mostrado no **Algoritmo** 8. O algoritmo retorna o valor objetivo $fh_i(u)$, juntamente com o nível ótimo $p_i(u) \in K_i$ e a decisão $u_i(u)$ para todo $S_i \in \mathscr{S}$ e $u \in \{0, \ldots, \Delta\}$. O algoritmo tem complexidade $O(n\Delta^2\kappa_{max})$ e tem uma complexidade de memória de $O(n\Delta)$, onde $\kappa_{max} = \max\{\kappa(i) : S_i \in \mathscr{S}\}$. A solução aproximada é controlada pelo parâmetro Δ, de forma que a qualidade da solução tende a aumentar junto com Δ, ao custo de maior tempo de execução.

Uma forma de reduzir o tempo de computação e melhorar a qualidade da solução consiste na combinação do algoritmo de programação dinâmica para o modelo híbrido com a solução analítica do modelo contínuo. Seja $\mathbf{x}(\Delta) = (x_{i,k} : (i,k) \in \Omega)$ um vetor que contém as decisões discretas e $\mathbf{u}(\Delta) = (u_{i,k} : (i,k) \in \Omega)$ um vetor que contém as decisões contínuas produzidas pelo algoritmo de programação dinâmica quando o nível de discretização é Δ. Resultados práticos mostram que mesmo com um valor pequeno de Δ, o algoritmo de programação dinâmica tende a chegar em um $\mathbf{x}(\Delta)$ que se aproxima do ótimo. Dado $\mathbf{x}(\Delta)$, o problema pode ser redefinido de forma equivalente ao modelo contínuo PC ao se fixar as variáveis $x_{i,k}$ de acordo com $\mathbf{x}(\Delta)$ em $PH_n(\Delta)$. Portanto, uma solução alternativa é obtida computando-se $\mathbf{x}(\Delta)$ e $\mathbf{u}^{\text{lp}}(\Delta)$ através da solução deste modelo contínuo. Seja $fh_n^{\text{lp}}(\Delta)$ o valor objetivo gerado pela solução $\mathbf{x}(\Delta)$ e $\mathbf{u}^{\text{lp}}(\Delta)$. Percebe-se que $\mathbf{u}^{\text{lp}}(\Delta)$ será ótimo se $\mathbf{x}(\Delta)$ for ótimo. Claramente, $fh_n^{\text{lp}}(\Delta) \geq fh_n(\Delta)$ e a diferença entre eles tende a aumentar enquanto o nível de discretização diminui.

Algoritmo 8: Algoritmo de programação dinâmica para o modelo híbrido de reconfiguração de servidores

input: Servidores $\mathscr{S}$, parâmetros $A_{i,k}$, $Q_{i,k}$, $T_{i,k}^{min}$, $T_{i,k}^{max}$, e Δ
initialize: $\delta := \frac{1}{\Delta}$
for $u = 0, \ldots, \Delta$ **do**
  $fh_1(u) := -\infty$
  $p_1(u) := 0$
  $u_1(u) := 0$
  for *all* $k \in K_1$ **do**
    if $Q_{1,k}/T_{1,k}^{max} \leq u\delta$ **then**
      $u_{1,k} := \min\{u\delta, Q_{1,k}/T_{1,k}^{min}\}$
      if $A_{1,k}u_{1,k} > fh_1(u)$ **then**
        $fh_1(u) := A_{1,k}u_{1,k}$
        $p_1(u) := k$
        $u_1(u) := u_{1,k}$
for $i = 2, \ldots, n$ **do**
  for $u = 0, \ldots, \Delta$ **do**
    $fh_i(u) := -\infty$
    $p_i(u) := 0$
    $u_i(u) := 0$
    for $u_i = 0, \ldots, u$ **do**
      for *all* $k \in K_i$ **do**
        if $Q_{i,k}/T_{i,k}^{max} \leq u_i\delta$ **then**
          $u_{i,k} := \min\{u_i\delta, Q_{i,k}/T_{i,k}^{min}\}$
          if $A_{i,k}u_{i,k} + fh_{i-1}(u-u_i) > fh_i(u)$ **then**
            $fh_i(u) := A_{i,k}u_{i,k} + fh_{i-1}(u-u_i)$
            $p_i(u) := k$
            $u_i(u) := u_{i,k}$
return (fh_i, p_i, u_i)

6.2 Instância Ilustrativa

Uma instância exemplo do modelo híbrido para reconfiguração de servidores é dado nas Tabelas 6.1 e 6.2. Esta instância foi obtida através da modificação da instância discreta mostrada na Seção 4.1. A solução ótima para PH é $x_{1,5} = 1$ e $u_{1,5} = 0.864286$, $x_{2,1} = 1$ e $u_{2,1} = 0.071429$, $x_{3,1} = 1$ e $u_{3,1} = 0.035714$, $x_{4,1} = 1$ e $u_{4,1} = 0.028571$ que levam ao valor da função objetivo $fh = 0.886286$. Esta solução foi obtida através da solução da instância em um solver IP.

A Tabela 6.3 mostra os valores da função objetivo para a solução $(\mathbf{x}(\Delta), \mathbf{u}(\Delta))$ produzida pelo algoritmo de programação dinâmica e a solução $(\mathbf{x}(\Delta), \mathbf{u}^{\text{lp}}(\Delta))$ produzida pela combinação do algoritmo de programação dinâmica e a solução analítica do modelo contínuo. Nota-se que

	Servidor S_i			
	$i=1$	$i=2$	$i=3$	$i=4$
$(\frac{Q_{i,1}}{T_{i,1}^{max}}, \frac{Q_{i,1}}{T_{i,1}^{min}})$	$(\frac{15}{140}, \frac{15}{130})$	$(\frac{10}{140}, \frac{10}{130})$	$(\frac{5}{140}, \frac{5}{130})$	$(\frac{4}{140}, \frac{4}{130})$
$(\frac{Q_{i,2}}{T_{i,2}^{max}}, \frac{Q_{i,2}}{T_{i,2}^{min}})$	$(\frac{30}{130}, \frac{30}{120})$	$(\frac{25}{130}, \frac{25}{120})$	$(\frac{20}{130}, \frac{20}{120})$	$(\frac{15}{130}, \frac{15}{120})$
$(\frac{Q_{i,3}}{T_{i,3}^{max}}, \frac{Q_{i,3}}{T_{i,3}^{min}})$	$(\frac{50}{120}, \frac{50}{110})$	$(\frac{32}{120}, \frac{32}{110})$	$(\frac{35}{120}, \frac{35}{110})$	$(\frac{20}{120}, \frac{20}{110})$
$(\frac{Q_{i,4}}{T_{i,4}^{max}}, \frac{Q_{i,4}}{T_{i,4}^{min}})$	$(\frac{70}{110}, \frac{70}{100})$	$(\frac{50}{110}, \frac{50}{100})$	$(\frac{60}{110}, \frac{60}{100})$	$(\frac{30}{110}, \frac{30}{100})$
$(\frac{Q_{i,5}}{T_{i,5}^{max}}, \frac{Q_{i,5}}{T_{i,5}^{min}})$	$(\frac{80}{100}, \frac{80}{90})$	$(\frac{60}{100}, \frac{60}{90})$	$(\frac{65}{100}, \frac{65}{90})$	$(\frac{40}{100}, \frac{40}{90})$

Tabela 6.1: Orçamentos e períodos.

	Server S_i			
	$i=1$	$i=2$	$i=3$	$i=4$
$A_{i,1}$	0.125	0.2	0.076	0.175
$A_{i,2}$	0.375	0.333	0.307	0.375
$A_{i,3}$	0.625	0.533	0.538	0.5
$A_{i,4}$	0.875	0.733	0.923	0.75
$A_{i,5}$	1	1	1	1

Tabela 6.2: Valores de benefício das configurações de servidores.

as decisões discretas $\mathbf{x}(\Delta)$ coincidem com o ótimo para $\Delta \geq 25$.

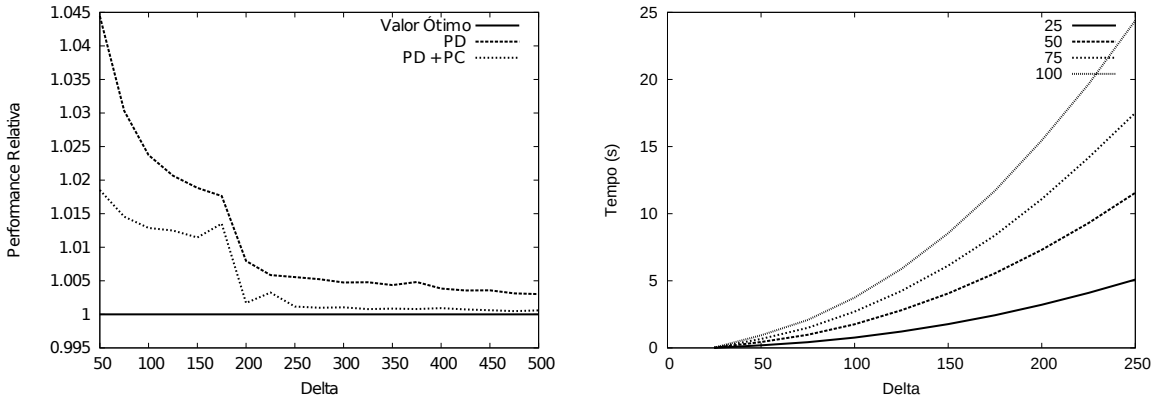
Os valores ótimos da função objetivo fh foram obtidos resolvendo o modelo linear inteiro-misto definido nas Equações (6.1a)–(6.1e) através da utilização um solver disponível no pacote NEOS [20].

6.3 Análise Numérica

Para avaliar o desempenho do algoritmo de programação dinâmica para o modelo híbrido e o efeito da adição do passo contínuo, 90 instâncias aleatórias de PH com $|\mathscr{S}| = 5$ e $3 \leq \kappa(i) \leq 10$ foram geradas usando distribuições uniformes para todas as variáveis. As soluções ótimas

Δ	δ	$fh_n(\Delta)$	$\frac{(fh-fh_n(\Delta))}{fh}$	$fh_n^{\text{lp}}(\Delta)$	$\frac{(fh-fh_n^{\text{lp}}(\Delta))}{fh}$
10	0.1000	0.636192	28.22%	0.721717	18.57%
25	0.0400	0.863692	2.55%	0.886286	0.00%
50	0.0200	0.863692	2.55%	0.886286	0.00%
100	0.0100	0.873557	1.42%	0.886286	0.00%
200	0.0050	0.878173	0.92%	0.886286	0.00%
300	0.0033	0.882703	0.40%	0.886286	0.00%

Tabela 6.3: Soluções produzidas pelo algoritmo de programação dinâmica.



(a) Efeito de Δ na qualidade de solução dos algoritmos. (b) Efeito de Δ no tempo de execução do algoritmo de programação dinâmica.

Figura 6.1: Avaliação do algoritmo de programação dinâmica para o modelo híbrido.

foram obtidas usando um solver inteiro-misto GLPK [16] , e o seu valor médio foi calculado.

A Figura 6.1(a) mostra o desempenho relativo do algoritmo de programação dinâmica por si só e o algoritmo com o passo contínuo. Como pode ser visto, o algoritmo de programação dinâmica sozinho atinge 95.74% do ótimo com $\Delta = 50$, e a adição do passo contínuo leva a solução dada a atingir 98.18% do valor ótimo com o mesmo Δ. À medida que o Δ cresce, as soluções dadas pelo algoritmo com o passo contínuo se tornam quase ótimas. O aumento no valor da função objetivo causado pelo passo contínuo custa um aumento praticamente insignificante no tempo de execução devido à natureza analítica do algoritmo.

A Figura 6.1(b) mostra o efeito do Δ no tempo de execução do algoritmo de programação dinâmica para $25 \leq |\mathscr{S}| \leq 100$. Nota-se que, para as instâncias geradas, a solução dada por programação dinâmica com $\Delta = 50$ já era mais do que 90% do valor ótimo. Um estudo de que Δ usar, analisando o intercâmbio entre tempo de execução e qualidade de solução deve ser realizado antes da aplicação do algoritmo.

6.4 Sumário

Neste capítulo foi definido um modelo híbrido, que busca incorporar não só a presença de modos bem definidos mas também um grau de liberdade para cada um desses modos. Apesar de ter de selecionar um dos modos de cada tarefa, a infraestrutura de reconfiguração dispõe de intervalos de utilização dentro dos quais ajustes finos podem ser realizados. Para a solução deste modelo foi apresentado um algoritmo baseado em programação dinâmica parametrizado em Δ, um fator de discretização que permite o aumento da precisão em troca de um aumento no tempo de obtenção da solução. Este algoritmo foi estendido por um passo que utiliza o

algoritmo analítico do modelo contínuo para melhorar a qualidade das soluções.

7 Modelos com Objetivos Não-Aditivos

Uma alocação mais homogênea de banda é obtida através do uso de funções objetivo não-aditivas. A seguir, algoritmos para a solução dos modelos discreto, contínuo e híbrido com objetivos não-aditivos são desenvolvidos.

7.1 Modelo Discreto

O modelo discreto de reconfiguração de servidores com função objetivo não-aditiva é:

$$PD^{na}: \ fd^{na} = \text{Maximize} \quad \min_{S_i \in \mathscr{S}} \sum_{k \in K_i} A_{i,k} x_{i,k} \tag{7.1a}$$

$$\text{Sujeito a}: \sum_{S_i \in \mathscr{S}} \sum_{k \in K_i} U_{i,k} x_{i,k} \leq 1 \tag{7.1b}$$

$$U_{i,k} = \frac{Q_{i,k}}{T_{i,k}} \tag{7.1c}$$

$$\sum_{k \in K_i} x_{i,k} = 1, \ S_i \in \mathscr{S} \tag{7.1d}$$

$$x_{i,k} \in \{0,1\}, \ S_i \in \mathscr{S}, k \in K_i \tag{7.1e}$$

O **Algoritmo** 9 é um procedimento guloso que encontra uma solução ótima para PD^{na}.

Teorema 3. *O algoritmo guloso soluciona* PD^{na} *se* $U_{i,k} \geq U_{i,k-1}$ *e* $A_{i,k} \geq A_{i,k-1}$ *para todo* $(i,k) \in \Omega$, $k > 1$.

Demonstração. Por contradição, assume-se que o algoritmo guloso retorna ω tal que a solução induzida $\mathbf{x} = \mathbf{x}(\omega)$ não é ótima, significando que $fd^{na}(\mathbf{x}) < fd^{na}(\mathbf{x}^\star)$ onde $\mathbf{x}^\star$ é uma solução ótima. Seja $\omega_{\geq}(\mathbf{x}^\star) = \{(i,k) \in \omega : A_{i,k} \geq fd^{na}(\mathbf{x}^\star)\}$ e $\omega_{<}(\mathbf{x}^\star) = \{(i,k) \in \omega : A_{i,k} < fd^{na}(\mathbf{x}^\star)\}$. Seja $\omega^\star = \{(i,k) : x^\star_{i,k} = 1\}$ os pares ativos em $\mathbf{x}^\star$. Seja também $(i',k') = \arg\min\{A_{i,k} : (i,k) \in \omega\}$ o par para qual o algoritmo guloso falhou ao tentar substituí-lo por $(i',k'+1)$ e seja $(i',k'') \in$

Algoritmo 9: Algoritmo guloso para a reconfiguração discreta de servidores com objetivo não-aditivo

input: Servidores $\mathscr{S}$, valores de benefício $A_{i,k}$ e utilizações $U_{i,k}$
$Q := \emptyset$ é uma fila de prioridade
for *todo* $S_i \in \mathscr{S}$ **do**
  Insira $(i,1)$ com chave $A_{i,1}$ em Q
$r := 1 - \sum_{S_i \in S} U_{i,1}$
while $r > 0$ **do**
  $(i,k) :=$ o par com a menor chave em Q
  if $(r - (U_{i,k+1} - U_{i,k})) \geq 0$ **then**
    $r := r - (U_{i,k+1} - U_{i,k})$
    Remover (i,k) de Q
    Inserir $(i,k+1)$ com chave $A_{i,k+1}$ em Q
$\omega := \{(i,k) : (i,k) \in Q\}$
retornar ω

$\omega^\star$ o par correspondente em $\mathbf{x}^\star$. Então,

$$\sum_{(i,k)\in\omega^\star} U_{i,k} = \sum_{(i,k)\in\omega^\star:(i,l)\in\omega_<(\mathbf{x}^\star)} U_{i,k} + \sum_{(i,k)\in\omega^\star:(i,l)\in\omega_\geq(\mathbf{x}^\star)} U_{i,k} \tag{7.2a}$$

$$\geq \sum_{(i,k)\in\omega_<(\mathbf{x}^\star)-\{(i',k')\}} U_{i,k} + U_{i',k''} + \sum_{(i,k)\in\omega^\star:(i,l)\in\omega_\geq(\mathbf{x}^\star)} U_{i,k} \tag{7.2b}$$

$$\geq \sum_{(i,k)\in\omega_<(\mathbf{x}^\star)-\{(i',k')\}} U_{i,k} + U_{i',k''} + \sum_{(i,k)\in\omega_\geq(\mathbf{x}^\star)} U_{i,k} \tag{7.2c}$$

$$\geq \sum_{(i,k)\in\omega-\{(i',k')\}} U_{i,k} + U_{i',k'+1} \tag{7.2d}$$

$$> 1 \tag{7.2e}$$

contradizendo a suposição que $\mathbf{x}^\star$ é factível. A desigualdade (7.2c)-(7.2d) procede do fato que $U_{i,k} \geq U_{i,l}$ para todo $(i,k) \in \omega^\star$ e $(i,l) \in \omega_\geq(\mathbf{x}^\star)$, uma vez que caso contrário o algoritmo não teria selecionado $(i,l), l > k$, já que Q continha (i',k') com $A_{i',k'} < fd^{na}(\mathbf{x}^\star) \leq \min\{A_{i,k} : (i,k) \in \omega^\star, (i,l) \in \omega_\geq(\mathbf{x}^\star), l > k\} \leq \min\{A_{i,l} : (i,k) \in \omega^\star, (i,l) \in \omega_\geq(x^\star), l > k\}$. □

Usando uma árvore de pesquisa balanceada ou um heap binário como a estrutura de dados para a fila de prioridades, o **Algoritmo** 9 executa em tempo $O(|\Omega| \lg n)$. A aplicação deste algoritmo para a instância das Tabelas 4.1 e 4.2 produz a solução $\mathbf{x}$ mostrada na Tabela 7.1. O objetivo é $fd^{na}(\mathbf{x}) = \min\{0.375, 0.333, 0.307, 0.375\} = 0.307$. Para melhorar a parte aditiva do objetivo sem comprometer a parte não-aditiva, o algoritmo pode ser complementado com uma fase de pós-processamento para usar a banda residual. Seria suficiente substituir a computação de ω com o **Algoritmo** 10.

Algoritmo 10: Pós-processamento do algoritmo guloso para o modelo discreto com objetivo não-aditivo

$\omega := \emptyset$
$r := 1 - \sum_{S_i \in S} U_{i,1}$
for *todo* $(i,k) \in Q$ **do**
  if $r \geq U_{i,k+1} - U_{i,k}$ **then**
    $\omega := \omega \cup \{(i,k+1)\}$
    $r := r - (U_{i,k+1} - U_{i,k})$
  else
    $\omega := \omega \cup \{(i,k)\}$

Pares selecionados para ω

(i,k)	$A_{i,k}$	$U_{i,k}$
(1,2)	0.375	0.30
(2,2)	0.333	0.25
(3,2)	0.307	0.20
(4,2)	0.375	0.15

Tabela 7.1: Solução de instância discreta com objetivo não-aditivo.

7.2 Modelo Contínuo

A versão do modelo contínuo com objetivo não-aditivo é:

$$PC^{na}: \ fc^{na} = \text{Maximize} \quad \min_{S_i \in \mathscr{S}} A_i u_i \tag{7.3a}$$

$$\text{Sujeito a}: \ \sum_{S_i \in \mathscr{S}} u_i \leq 1 \tag{7.3b}$$

$$L_i \leq u_i \leq U_i, \ S_i \in \mathscr{S} \tag{7.3c}$$

Um algoritmo para a solução de PC^{na} é o seguinte. Partindo dos limites inferior e superior para o objetivo não-aditivo, a e b respectivamente, o algoritmo faz uma pesquisa binária no intervalo $[a,b]$ até que a diferença entre os limites do intervalo se torne menor que uma dada tolerância ε. Nota-se que $fc^{na} \geq a = \min\{A_i L_i : S_i \in \mathscr{S}\}$ e $fc^{na} \leq b = \min\{A_i U_i : S_i \in \mathscr{S}\}$. O **Algoritmo** 11 formaliza esta pesquisa binária.

O **Algoritmo** 11 leva tempo $O(n \lg((b-a)/\varepsilon))$ para alcançar uma solução factível $\mathbf{u}$ tal que $fc^{na}(\mathbf{u}) \geq fc^{na} - \varepsilon$, onde $fc^{na}(\mathbf{u})$ é o objetivo dado por $\mathbf{u}$. Os limites para o objetivo não-aditivo para a instância mostrada na Tabela 5.1 são $a = 0.04$ e $b = 1.2$. As soluções produzidas pelo **Algoritmo** 11 para diferentes tolerâncias de erro aparecem na Tabela 7.2. O algoritmo encontrou uma solução $\mathbf{u}$ com um objetivo não-aditivo $fc^{na}(\mathbf{u}) = 0.20$. Esta solução também

Algoritmo 11: Pesquisa binária para reconfiguração contínua de servidores com objetivo não-aditivo

input: Servidores $\mathscr{S}$, benefícios A_i, e limites L_i e U_i
$a := \min\{A_i L_i : S_i \in \mathscr{S}\}$
$b := \min\{A_i U_i : S_i \in \mathscr{S}\}$
while $b - a \geq \varepsilon$ **do**
  $\xi := (a+b)/2$
  if $(\sum_{S_i \in \mathscr{S}} \frac{\xi}{A_i} \leq 1)$ *e* $(\frac{\xi}{A_i} \leq U_i$ *para todo* $S_i \in \mathscr{S})$ **then**
    $a := \xi$
  else
    $b := \xi$
$\mathbf{u} = (0 : S_i \in \mathscr{S})$
for *todo* $S_i \in \mathscr{S}$ **do**
  $u_i = a/A_i$
return **u**

induz um objetivo aditivo de $fc(\mathbf{u}) = 5 \times 0.20 = 1.0$, menor que o valor aditivo ótimo de 1.8635.

ε	iterações k	a_k	b_k
10^{-1}	1	0.0400	0.2000
10^{-2}	5	0.1900	0.2000
10^{-3}	8	0.1988	0.2000
10^{-4}	11	0.1998	0.2000
10^{-5}	14	0.2000	0.2000

Tabela 7.2: Soluções da instância contínua com objetivo não-aditivo para diferentes tolerâncias.

7.3 Modelo Híbrido

O modelo híbrido com um objetivo não-aditivo é obtido ao se substituir fh por $fh^{na} = \min_{S_i \in \mathscr{S}}\{\sum_{k \in K_i} A_{i,k} u_{i,k}\}$ na formulação (6.2a) a (6.2e). Seja PH^{na} o modelo híbrido com objetivo não-aditivo. Um algoritmo para a solução de PH^{na} é facilmente derivado do algoritmo de programação dinâmica para PH. É suficiente substituir $fh_i(u)$ por:

$$fh_i^{na}(u) = \max_{u_{i,k}: k \in K_i} \min \{ \sum_{k \in K_i} A_{i,k} u_{i,k}, fh_{i-1}^{na}(u - u_i)\}$$

na formulação $PH_i(u)$ para obter $PH_i^{na}(u)$. Ajustes similares no **Algoritmo** 8 resultam num algoritmo de programação dinâmica para PH^{na} com a mesma complexidade de tempo e memória. A aplicação deste algoritmo na instância dada nas Tabelas 6.1 e 6.2 produz os resultados mostrados na Tabela 7.3. Um solver linear inteiro-misto foi usado para obter a solução ótima, que é composta por: $x_{1,2} = 1$ e $u_{1,2} = 0.25$, $x_{2,3} = 1$ e $u_{2,3} = 0.276515$, $x_{3,3} = 1$ e $u_{3,3} = 0.291667$,

Δ	δ	$fh_n^{na}(\Delta)$	$\frac{(fh^{na}-fh_n^{na}(\Delta))}{fh^{na}}$	$fh_n^{na,\mathrm{lp}}(\Delta)$	$\frac{(fh^{na}-fh_n^{na,\mathrm{lp}}(\Delta))}{fh^{na}}$
10	0.1000	0.066600	26.74%	0.069375	23.69%
25	0.0400	0.069375	23.69%	0.069375	23.69%
50	0.0200	0.090000	0.99%	0.090909	0.00%
100	0.0100	0.090000	0.99%	0.090909	0.00%
200	0.0050	0.090909	0.00%	0.090909	0.00%
300	0.0033	0.090909	0.00%	0.090909	0.00%

Tabela 7.3: Soluções produzidas pelo algoritmo de programação dinâmica para o modelo híbrido com objetivo não aditivo.

e $x_{4,3} = 1$ e $u_{4,3} = 0.181818$ com objetivo $fh^{na} = 0.090909$. Pode-se perceber que o algoritmo de programação dinâmica encontra a solução ótima com um nível de discretização baixo se combinado com o modelo contínuo como explicado no Capítulo 6.

7.4 Sumário

Este capítulo apresenta o uso de funções objetivos não-aditivas em cada um dos modelos apresentados nos capítulos anteriores, com algoritmos rápidos para a solução de cada um deles. O uso de funções objetivos não-aditivas, isto é, que visam maximizar a mínima das relações benefício vezes a alocação, busca realizar uma distribuição mais homogênea do tempo de processador entre as tarefas, útil quando alguma noção de justiça é interessante para a aplicação.

Parte III

Aplicação e Avaliação

8 *Estudo de Caso*

Este capítulo tem o objetivo de ilustrar o uso da infraestrutura de reconfiguração em uma aplicação, desde a definição de estados e modos e a escolha de um modelo de otimização até a simulação de escalonamento para avaliar os efeitos das escolhas feitas nos primeiros passos do processo. Para utilizar esta infraestrutura na aplicação de robótica móvel introduzida no Capítulo 3, é necessário primeiro isolar as tarefas que a compõe e os seus modos. Depois disso, um modelo de otimização pode ser escolhido, aplicado e avaliado.

Como visto no Capítulo 3, a aplicação de visão, controle e navegação de um robô móvel é composta por cinco tarefas: Controle de Motores, Detecção de Obstáculos, Definição de Caminho, Manutenção de Mapa e Visão. Uma vez que do ponto de vista do escalonador as tarefas de Controle de Motores e Detecção de Obstáculos apresentam comportamento idêntico, elas serão agrupadas em uma única tarefa.

8.1 Modelo de Escalonamento

Para formalizar esta aplicação no contexto da infraestrutura de reconfiguração, é necessário definir os valores de benefício e utilização de cada tarefa para todos os seus possíveis modos de operação. Além disso, as transições que causam a mudança desses valores precisam ser identificadas, já que serão nesses pontos de transição que uma reconfiguração será requisitada. Os valores apresentados aqui são uma aproximação de valores aferidos de uma aplicação real para o controle de robôs móveis desenvolvida na Universidade Federal de Santa Catarina [33], que atualmente usa um sistema simples de executivo cíclico no seu escalonamento. Os valores de benefício são definidos com base na importância de cada tarefa para a aplicação como um todo.

Sensor/Atuador

A tarefa Sensor/Atuador (união das tarefas de Controle de Motores e Detecção de Obstáculos) tem apenas um modo de operação que, dependendo do estado do sistema, pode ter dois valores diferentes de benefício e utilização. Quando o robô está parado, o valor de benefício desta tarefa para o sistema é zero, uma vez que qualquer reserva de tempo feita para ela seria desperdiçado. Quando o robô se move, o seu valor se torna bastante elevado uma vez que a tarefa é crítica para a aplicação; ela é a única capaz de evitar colisões, e se o robô colide, a competição é perdida. A Tabela 8.1 define os valores para esta tarefa durante ambos estes estados.

	Servidor S_1 - Tarefa Sensor/Atuador	
	Parado	Movendo
$k=1$	$U_{1,1}=0 \quad A_{1,1}=0$	$U_{1,1}=0.1 \quad A_{1,1}=100$
	$T=\infty$	$T=0.1s$

Tabela 8.1: Utilizações e valores de benefício para os estados tarefa de Sensor/Atuador.

Enquanto o robô se move, o valor de $A_1 = 100$ é atribuído ao modo. Este valor será usado como uma marca inicial de criticalidade para as outras tarefas. O fato de ser o único modo para esta tarefa, entretanto, garante que ele será selecionado pelo algoritmo de otimização sempre que estiver disponível.

Definição de Caminho

A tarefa de Definição de Caminho geralmente se mantém ociosa, já que ela só é ativada quando o robô detecta um obstáculo no seu caminho atual. Enquanto um caminho está sendo percorrido e nenhum obstáculo é detectado, todos os seus modos têm valor de benefício nulo. Uma vez que um obstáculo é detectado, a definição de um novo caminho a percorrer se torna de grande importância, porém não-crítica. Isto permite a definição de dois modos para esta tarefa, onde o mais valioso tem alta utilização e benefício por ser capaz de calcular um caminho rapidamente. Um modo de menor utilização e benefício também é definido, e quando utilizado leva a tarefa a demorar mais para definir um caminho. A Tabela 8.2 define os valores de utilização e benefício para esta tarefa.

Note que o modo $k = 3$ da tarefa de Definição de Caminho efetivamente ocupará o processador todo ($U_{2,3} = 1$), mas tem um benefício menor que a tarefa Sensor/Atuador ($A_{2,3} = 90$, enquanto para o Sensor/Atuador, $A_{1,1} = 100$). Isto é uma maneira de atribuir valores relativos entre as tarefas. Uma vez que durante movimentações pelo menos 10% do processador será reservado, o modo $k = 3$ será ativado apenas se o robô se encontrar parado.

Servidor S_2 - Tarefa de Definição de Caminho		
	Ocioso	Definindo Caminho
$k=1$	$U_{2,1}=0 \quad A_{2,1}=0$	$U_{2,1}=0 \quad A_{2,1}=0$
$k=2$		$U_{2,2}=0.5 \quad A_{2,2}=50$
$k=3$		$U_{2,3}=1 \quad A_{2,3}=90$
	$T=\infty$	$T=0.5s$

Tabela 8.2: Utilizações e valores de benefício para os estados tarefa de Definição de Caminho.

Também é válido notar que $k=1$ é um modo de benefício e utilização nulos para esta tarefa. Uma vez que ela não recebe dados externos como as outras, ela pode ser adiada sem perda de dados (como as perdas causadas por sobrecargas no *buffer* de entrada dos sensores ou da câmera, por exemplo). Se esta opção é selecionada pelo solucionador, esta tarefa não receberá reserva de banda até pelo menos um novo pedido de reconfiguração.

Manutenção de Mapa

A tarefa de Manutenção de Mapa se comporta de maneira bastante similar. Existe uma utilização nominal causada pela necessidade de adicionar qualquer novo segmento à representação interna do mapa toda vez que os sensores infravermelho detectam um obstáculo. Porém, periodicamente, a representação interna do mapa se tornará complexa o suficiente para justificar uma simplificação. Isto é feito para liberar memória e acelerar os cálculos de definição de caminhos.

A Tabela 8.3 define os valores de utilização e benefício para esta tarefa. Enquanto o mapa se mantém simples, os modos de utilização mais alta não são de interesse. Uma vez que o mapa é complexo o suficiente, simplificá-lo requer mais tempo de processador e traz um benefício maior ao sistema.

Servidor S_3 - Tarefa de Manutenção de Mapa		
	Nominal	Simplificação Necessária
$k=1$	$U_{3,1}=0.1 \quad A_{3,1}=40$	$U_{3,1}=0.1 \quad A_{3,1}=40$
$k=2$		$U_{3,2}=0.25 \quad A_{3,2}=50$
$k=3$		$U_{3,3}=0.5 \quad A_{3,3}=70$
	$T=1s$	$T=1s$

Tabela 8.3: Utilizações e valores de benefício para os estados tarefa de Manutenção de Mapa.

Visão

Finalmente, o sistema de Visão tem três estados, migrando entre eles de acordo com a probabilidade da imagem captada conter um dos símbolos que ele deve reconhecer. A utilização

nominal é baixa, fruto da análise de imagens de baixa resolução capturadas em alta frequência. Os estados seguintes, Grosso e Fino, necessitam de resoluções cada vez maiores em frequências cada vez mais baixas.

A Tabela 8.4 define estes três estados. Cada um deles tem um modo específico, garantindo a utilização que a tarefa necessita no pior caso e, portanto, levando ao desempenho máximo. No caso nominal, apenas o processamento em alta frequência de imagens de baixa resolução importa. Uma vez que a análise destas imagens encontra algo de interesse, alcança-se o estado Grosso. Se a presença de uma imagem é detectada neste modo, o estado Fino é alcançado para que seja realizada a sua identificação.

Uma vez que o processo de identificação de imagens é feito através de um algoritmo de inteligência artificial de alta complexidade, o modo mais consumidor do estado Fino é capaz de utilizar 100% do tempo do processador. O mesmo não pode ser dito do algoritmo utilizado no modo Grosso, onde a reserva de uma utilização tão alta levaria ao desperdício de processador.

As disponibilidades dos modos $k = 1$ e $k = 2$ no estado Fino permitem que a tarefa seja realizada com uma reserva de tempo menor, causando um atraso no seu tempo de resposta.

Servidor S_4 - Tarefa de Visão

	Nominal	Grosso	Fino
$k = 1$	$U_{4,1} = 0.25 \quad A_{4,1} = 55$	$U_{4,1} = 0.25 \quad A_{4,1} = 55$	$U_{4,1} = 0.25 \quad A_{4,1} = 55$
$k = 2$		$U_{4,2} = 0.5 \quad A_{4,2} = 75$	$U_{4,2} = 0.5 \quad A_{4,2} = 75$
$k = 3$			$U_{4,3} = 1 \quad A_{4,3} = 95$
	$T = 0.25s$	$T = 0.5s$	$T = 1s$

Tabela 8.4: Utilizações e valores de benefício para os estados tarefa de Visão.

Discussão

Cada uma destas tarefas pode alcançar qualquer um dos seus estados de forma independente entre si. A utilização da infraestrutura de reconfiguração para definir quanto tempo de processador alocar para cada tarefa em tempo de execução de acordo com seu valor de benefício libera o projetista do sistema do trabalho de considerar todas as combinações de estados estaticamente.

Enquanto o exemplo desta seção é simples, um sistema complexo com diversas tarefas, estados e modos faria a tarefa de atribuir fatias de processador manualmente extremamente demorada e suscetível a erros. A atribuição de valores para as tarefas, entretanto, é significativamente mais simples e não suscetível à explosão combinatória.

A questão de que modelo matemático usar na reconfiguração deste sistema persiste. Devido à natureza discreta dos modos de cada tarefa, o Modelo Discreto apresentado no Capítulo 4 será

utilizado. O uso de um esquema de aproximação permitirá soluções satisfatórias com uma baixa sobrecarga, o que é importante num sistema móvel com capacidade de processamento limitada.

8.2 Exemplo de Execução

Para ilustrar o uso da infraestrutura de reconfiguração nesta aplicação, uma linha de tempo com várias migrações entre estados (e, portanto, pedidos de reconfiguração) é apresentada na Figura 8.1 e detalhada a seguir. Uma simulação do escalonamento desta seção do tempo de execução do robô será apresentada, ilustrando como um número de reconfigurações modifica os parâmetros de escalonemento maximizando o valor agregado do sistema.

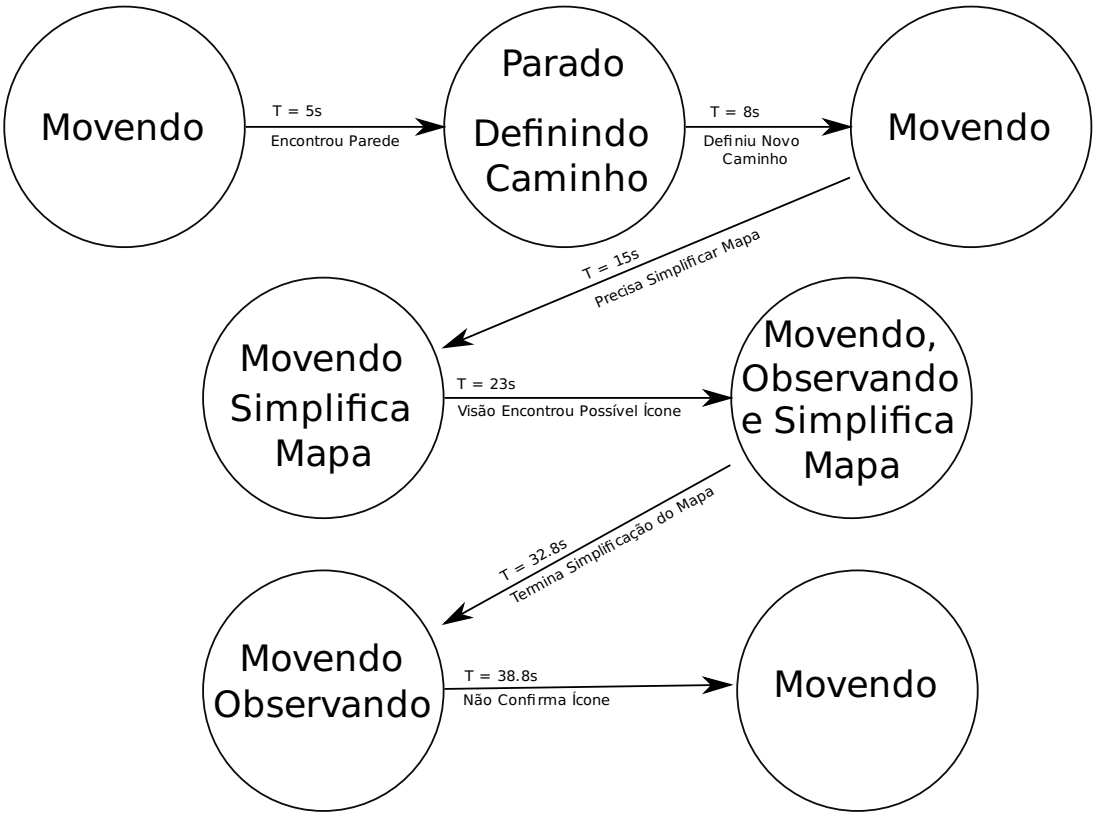


Figura 8.1: Evolução de estados do robô.

No início do tempo de vida do robô, ele se encontra parado, com um mapa vazio e sem nenhuma imagem de interesse no seu campo de visão. Sua primeira instrução é para que ele comece a se movimentar; a primeira reconfiguração faz com que os estados e modos para cada tarefa se tornem aqueles mostrados na primeira coluna da Tabela 8.5. Neste estado, a utilização total do sistema é $U = 0.45$.

Quando $T = 5s$, depois de algum movimento, o robô detecta uma parede em sua frente. Isto leva a tarefa Sensor/Atuador a parar, e a Definição de Caminho a alcançar seu estado De-

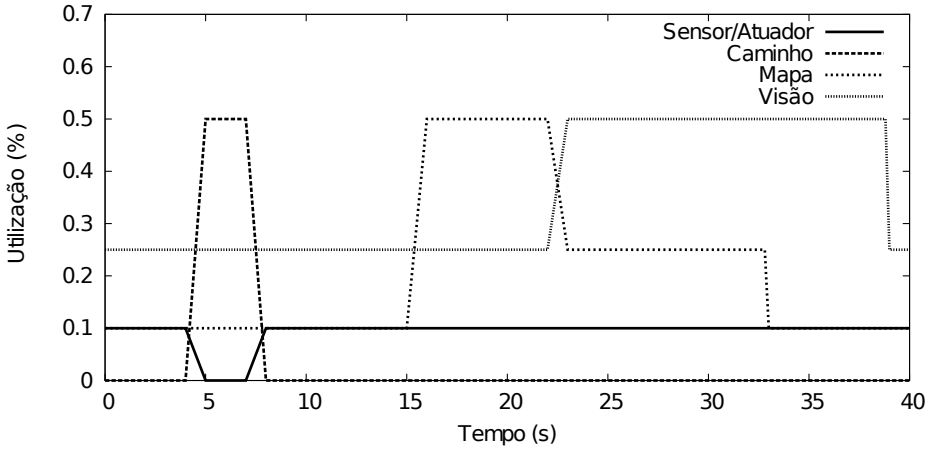


Figura 8.2: Linha de tempo das alocações de orçamento.

	Estado 1 ($T = 0s$)		Estado 2 ($T = 5s$)		Estado 3 ($T = 15s$)		Estado 4 ($T = 23s$)	
Tarefa	Estado	Modo	Estado	Modo	Estado	Modo	Estado	Modo
Sens./Atu.	Movendo	$U_1 = 0.1$	Parado	$U_1 = 0$	Movendo	$U_1 = 0.1$	Movendo	$U_1 = 0.1$
Def. Cam.	Ocioso	$U_2 = 0$	Definindo	$U_2 = 0.5$	Ocioso	$U_2 = 0$	Ocioso	$U_2 = 0$
Mapa	Nominal	$U_3 = 0.1$	Nominal	$U_3 = 0.1$	Simplif.	$U_3 = 0.5$	Simplif.	$U_3 = 0.25$
Visão	Nominal	$U_4 = 0.25$	Nominal	$U_4 = 0.25$	Nominal	$U_4 = 0.25$	Grosso	$U_4 = 0.5$
Total		$U_t = 0.45$		$U_t = 0.85$		$U_t = 0.85$		$U_t = 0.85$

Tabela 8.5: Mudanças de estado para cada tarefa durante o tempo de execução da aplicação.

finindo Caminho. Esta mudança de estado leva a uma reconfiguração, que resolve o problema de otimização e redistribui o tempo de processador da forma mostrada na segunda coluna da Tabela 8.5. Nesta mudança de modo, o terceiro modo da tarefa de Definição de Caminho (com $U_3 = 1$) estava disponível para seleção porém não foi escolhido porque o estado Nominal das tarefas de Definição de Caminho e de Visão tornam essa escolha impossível. Apesar da tarefa de Definição de Caminho ter a capacidade de ocupar todo o processador, isto nunca ocorrerá devido à alocação mínima destas outras tarefas.

Depois de definir que caminho percorrer, o robô começa a se mover novamente em $T = 8s$. Os parâmetros de escalonamento retornam àqueles definidos na primeira coluna da Tabela 8.5. A Figura 8.2 mostra como a infraestrutura de reconfiguração desalocou a utilização previamente reservada para a tarefa Sensor/Atuador em $T = 5s$ e depois a realocou em $T = 8s$, quando o algoritmo de Definição de Caminho completou seus cálculos.

Em $T = 15s$ o modelo interno do labirinto se torna grande o suficiente para justificar uma simplificação. A tarefa de Manutenção de Mapa alcança o estado Simplificação Necessária e pede uma reconfiguração. A alocação de processador se torna aquela mostrada na terceira coluna da Tabela 8.5. Quando $T = 23s$ a tarefa de Visão muda seu estado para Grosso.

Como pode-se perceber ao contrastar as duas últimas colunas da Tabela 8.5, a infraestrutura de reconfiguração teve de desalocar tempo de processador da tarefa de Manutenção de Mapa

para que o benefício total do sistema fosse maximizado. Isto pode ser atribuído ao benefício mais alto dado a modos com valores de utilização iguais; quando todo o resto é igual e as restrições permitem, o solucionador da otimização sempre escolherá a opção de maior valor de benefício.

Em $T = 32.8s$, a simplificação da representação interna do mapa termina, então U_3 se torna 10%. Em $T = 38.8s$, o sistema de Visão para de seguir o que tinha levantado seu interesse, e U_4 volta aos 25% nominais. A Figura 8.2 mostra estas desalocações de orçamento, e o estado do sistema retorna ao original, da primeira coluna da Tabela 8.5.

A Figura 8.2 mostra como a infraestrutura de reconfiguração faz a administração das reservas de tempo de processador ao longo do tempo de vida do robô, mas um segundo gráfico, mostrando o efeito destas reservas no atraso das tarefas também deve ser analisado. A Figura 8.3 mostra as perdas de deadline causadas pela retirada de tempo de processador da tarefa de Manutenção de Mapa. Até a chegada da tarefa de Visão em $T = 23$, a soma de utilizações no sistema era sempre menor que 100%, portanto não havia perda de deadlines. A reconfiguração que realocou tempo de processador da tarefa de Manutenção de Mapa para a tarefa de Visão também teve o efeito de distribuir sobrecarga entre as duas, e, sem ela, a tarefa de Visão teria sofrido atrasos mais severos, um maior número de perdas de deadline e um tempo de resposta mais longo.

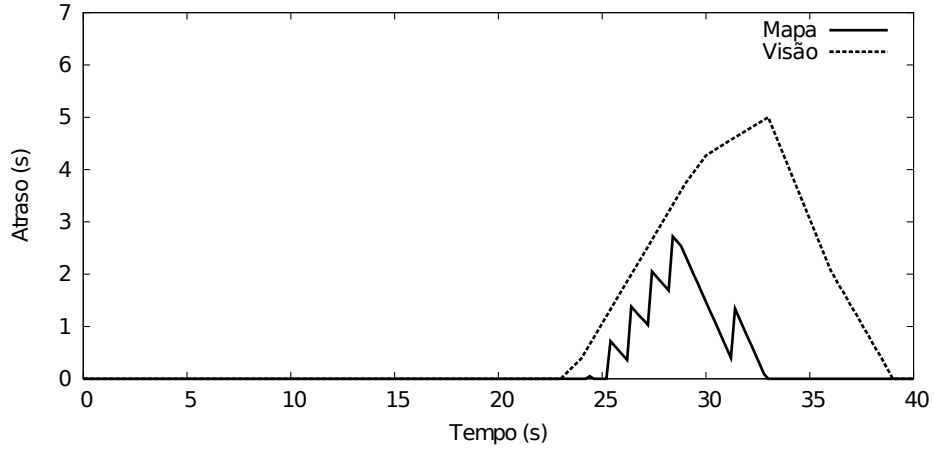


Figura 8.3: Linha de tempo do atraso das tarefas.

Este é o propósito da infraestrutura de reconfiguração; quando uma situação de sobrecarga se apresenta, cabe a ela pesar a importância de cada tarefa para o sistema como um todo e redistribuir tempo de processador entre elas. O efeito disso é mostrado entre os segundos 23 e 38.8 da simulação, onde perdas de deadline são compartilhadas entre as tarefas de Manutenção de Mapa e Visão em vez de ocorrer em alguma delas exclusivamente.

8.3 Sumário

Este capítulo expandiu e detalhou a descrição do sistema de visão, controle e navegação de um robô móvel que foi inicialmente apresentado como motivação no Capítulo 4. Foi analisado um segmento do tempo de vida da aplicação, demonstrando a mudança de estados simultânea de várias das tarefas da aplicação, culminando na necessidade de se realocar uma parcela do tempo de processador para maximizar o benefício agregado do sistema, mesmo na ocorrência de perdas de deadline.

9 Avaliação com Dados Reais

Para avaliar o desempenho da infraestrutura de reconfiguração em um ambiente mais realístico, o decodificador de vídeo FFmpeg [7] foi modificado para gerar um histórico dos tempos de chegada e tempos de computação dos quadros de um fluxo de vídeo de alta definição. Alta variabilidade na utilização por período foi observada, como se pode ver nas figuras 9.1 e 9.2. Os valores de utilização foram calculados através da divisão do tempo de computação de cada quadro pelo intervalo entre seu tempo de chegada e o tempo de chegada do próximo quadro. Enquanto a utilização média foi de 19.7%, o período de maior carga teve utilização de 88%. O tempo de computação médio entre todos os quadros foi de 8.4ms, e o período médio foi 42.5ms (uma taxa de aproximadamente 23.5 quadros por segundo).

Nesta simulação, três CBS recebem instâncias na maneira especificada no histórico de execução. Como consequência, toda vez que a utilização passa de 33.3%, deadlines serão perdidos, já que a soma das utilizações ultrapassará 100%. Isto permitirá que os benefícios da reconfiguração sejam diretamente percebidos pelo seu efeito na perda de deadlines. O período de cada CBS foi configurado para 42ms, o período nominal do vídeo, e seus orçamentos foram calculados utilizando a utilização retornada pelo solver do modelo escolhido.

O modelo escolhido para esta demonstração foi o *PC*, de natureza contínua. Resultados similares podem ser atingidos com as outras formulações, porém com maiores tempos de execução do mecanismo de reconfiguração devido à maior complexidade dos outros modelos. Inicialmente, todos os servidores têm benefício igual $A_i = 1$ e recebem fatias iguais do processador de aproximadamente 33%. O último 1% de CPU foi atribuído a um quarto servidor, dedicado à solução do problema de otimização que é criado a todo pedido de reconfiguração.

Aos 50 segundos, uma reconfiguração é requisitada, aumentando o valor do CBS número 3 de forma que $A_3 = 2$. Este tipo de pedido de reconfiguração representa um cenário de sobrecarga onde a pontualidade de uma aplicação específica é considerada mais crítica do que a das outras. O solucionador analítico neste caso particular (com $|\mathscr{S}| = 3$) levou 19.5us, que corresponde a 0.2% do tempo médio necessário para a decodificação de um quadro. Depois da solução

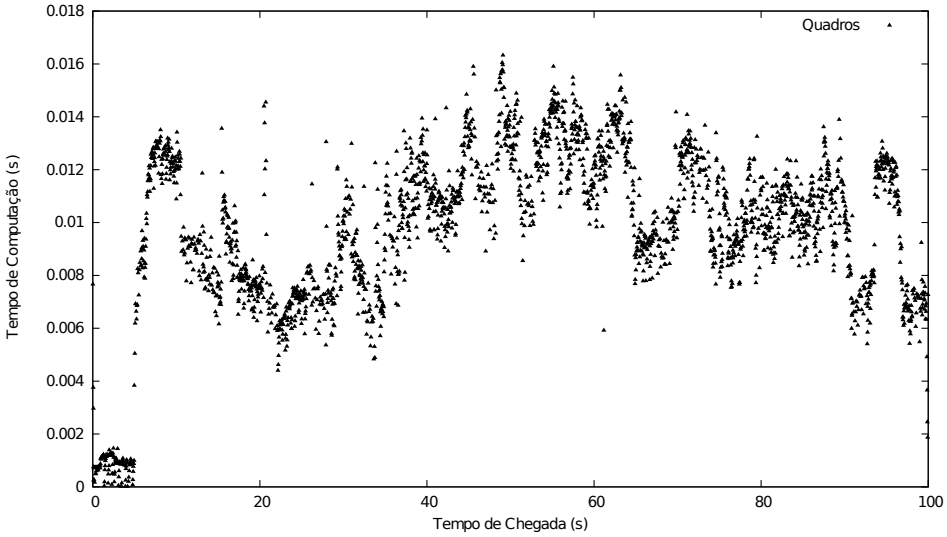


Figura 9.1: Tempo de computação de decodificação de vídeo.

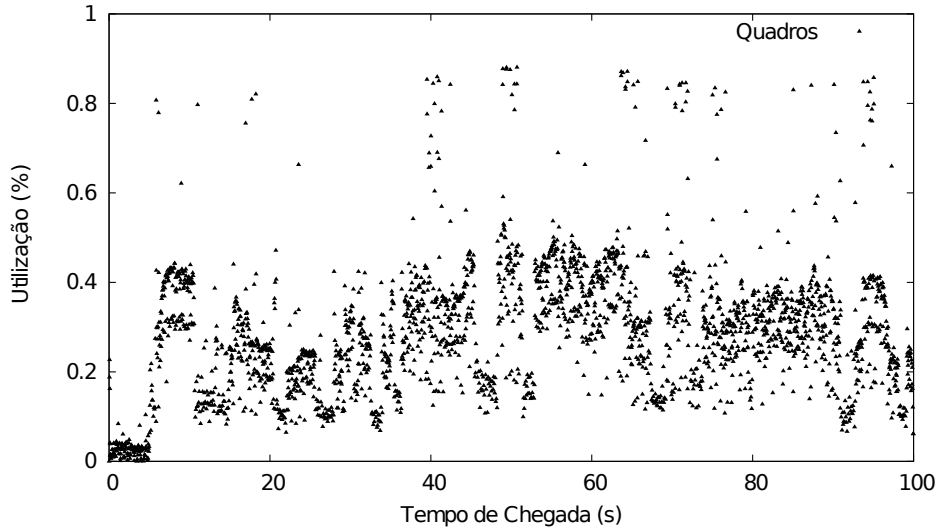


Figura 9.2: Utilização necessária para a decodificação de vídeo.

do problema de otimização, os novos parâmetros dos servidores são aplicados. A simulação foi executada até que 100 segundos do histórico fossem processados. O mesmo cenário foi executado sem a reconfiguração para fins de comparação.

A Tabela 9.1 mostra os parâmetros do CBS e os resultados de simulação. Duas métricas de qualidade de serviço foram usadas: taxa de perda de deadlines (T.P.D.) e atraso médio (A.M.). As medições foram realizadas antes e após o pedido de reconfiguração. L_i e U_i foram definidos como aproximadamente a metade e o dobro da utilização média da tarefa de decodificação, respectivamente. O atraso médio, mostrado em segundos, foi calculado como medida de severidade do atraso das instâncias. Pode-se perceber que a taxa de perda de deadlines do CBS 3 caiu de 24.2% para 3.9% após a reconfiguração, e o atraso médio foi 2% do que o observado nos

outros servidores. A melhoria obtida no CBS 3 se torna evidente ao se comparar os resultados de antes e depois do pedido de reconfiguração.

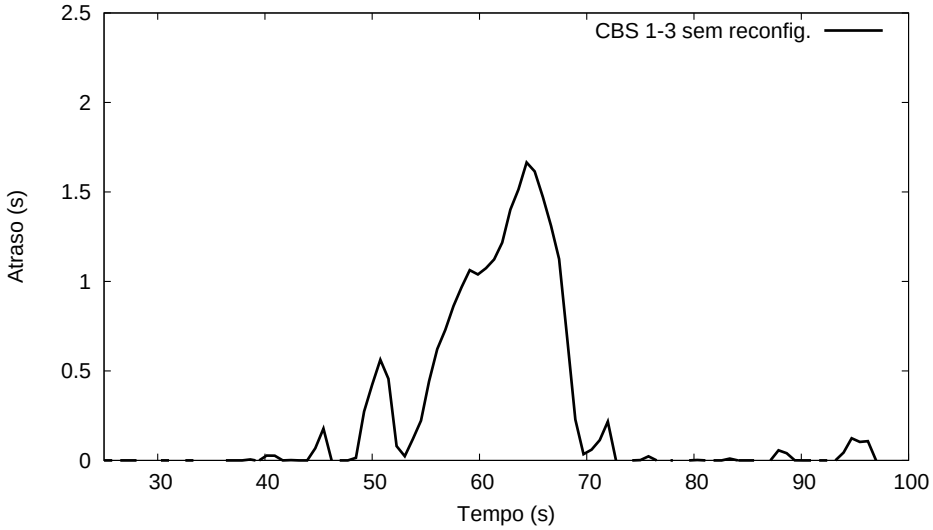
Tabela 9.1: Parâmetros e resultados de simulação.

			Antes da Reconf.				Depois da Reconf.				Sem Reconf.	
	L_i	U_i	A_i	u_i	T.P.D.	A.M.	A_i	u_i	T.P.D.	A.M.	T.P.D.	A.M.
CBS 1	0.065	0.395	1	0.328	0.246	0.293	1	0.295	0.562	0.516	0.422	0.202
CBS 2	0.065	0.395	1	0.328	0.239	0.296	1	0.295	0.711	0.542	0.416	0.198
CBS 3	0.065	0.395	1	0.328	0.242	0.296	2	0.395	0.039	0.011	0.418	0.201

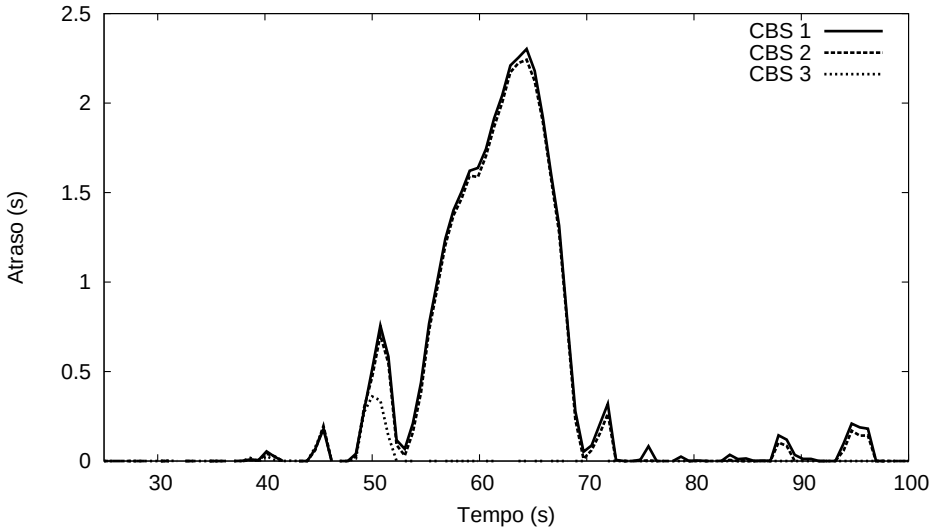
A Figura 9.3 mostra o padrão de atrasos para os 3 servidores. A Figura 9.3(a) mostra o sistema sem reconfiguração, e a Figura 9.3(b) o padrão de atraso do mesmo sistema com a reconfiguração. Todos servidores são representados pela mesma linha no caso sem reconfiguração uma vez que eles demonstram desempenho praticamente idêntico. É válido notar que enquanto o CBS 1 e 2 tiveram desempenho pior que na simulação sem reconfiguração, o servidor que teve sua criticalidade aumentada no pedido de reconfiguração deixou completamente de perder deadlines 1.6 segundo depois dos seus novos parâmetros serem aplicados. Este atraso de 1.6 segundo se deve às instâncias acumuladas devido à sobrecarga presente antes da reconfiguração.

9.1 Sumário

Este capítulo apresentou a aplicação da infraestrutura em um histórico real de um decodificador multimídia, mostrando sua capacidade de lidar com sobrecargas transientes limitando tanto o atraso médio quanto a taxa de perdas de deadlines das tarefas julgadas mais importantes pelo projetista do sistema ou pelo usuário.



(a) Simulação sem reconfiguração.



(b) Simulação com reconfiguração.

Figura 9.3: Simulação do histórico do FFmpeg

10 Integração com Feedback Scheduling

Como mencionado no Capítulo 2, progresso tem sido feito na aplicação da teoria de controle realimentado para realizar o ajuste dinâmico de parâmetros de servidores. Ao atuar nos parâmetros dos servidores, ajustando sua banda, é possível fazer com que diversas métricas de qualidade de serviço se aproximem do seu valor desejado. Esta abordagem é melhor aplicada quando não existe conhecimento prévio suficiente sobre o comportamento temporal das tarefas para fazer alocações de processador ótimas em tempo de projeto, e é esperado que variações graduais na utilização das tarefas ocorram dinamicamente.

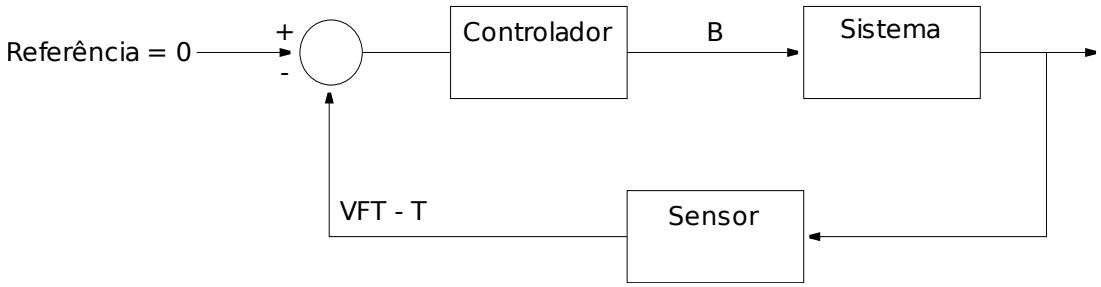


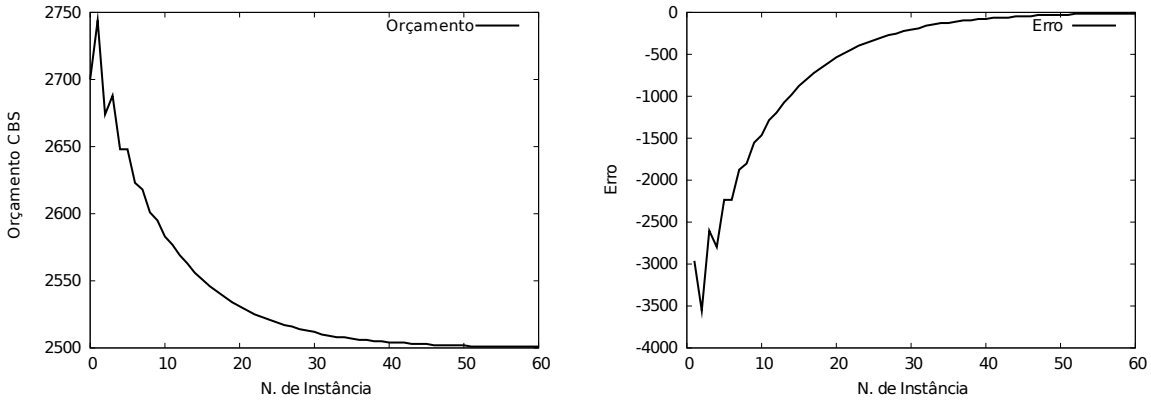
Figura 10.1: Controle realimentado aplicado a escalonamento baseado em servidores

10.1 Análise Experimental de Feedback Scheduling

A Figura 10.1 mostra a estrutura do controlador descrito em [3] por Abeni *et al.*. O erro de escalonamento é definido como a diferença entre o *Virtual Finishing Time* (VFT) e o período efetivo da tarefa. O *Virtual Finishing Time* VFT_i de uma instância é o instante no qual ela terminaria se executada em um processador dedicado de velocidade igual à parcela reservada por uma banda u_i. Se a tarefa termina cedo demais ($VFT_i < T_i$), o controlador dará menos banda ao servidor que a recebe. Se a tarefa termina tarde demais ($VFT_i > T_i$), o controlador o dará uma fatia maior do processador.

A principal dificuldade de aplicar esta abordagem é determinar um modelo para a dinâmica do sistema de forma que uma lei de controle estável possa ser definida. A escolha de polos

do controlador pode levar a um controlador instável ou um controlador restritivo demais, no sentido que o tempo até a convergência no valor de referência seja inaceitavelmente longo.



(a) Controle de orçamento através de Feedback Scheduling.

(b) Erro medido no uso de Feedback Scheduling.

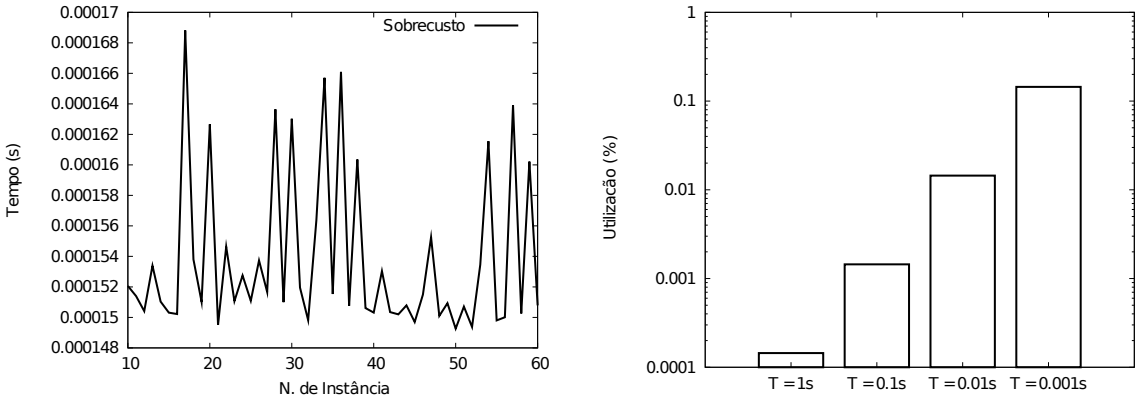
Figura 10.2: Feedback Scheduling.

A Figura 10.2(a) mostra o controlador atuando no orçamento de um servidor CBS de forma a diminuí-lo de um valor inicial de $u_i = 2700$ e estabilizá-lo no valor alvo de 2500. A Figura 10.2(b) mostra o erro de escalonamento gradualmente subindo até atingir o seu valor alvo, zero. Um valor de erro negativo significa que a tarefa está adiantada e, portanto, deve receber menos tempo de processador. Devido à sua configuração conservadora, este controlador levou 60 períodos da tarefa para neutralizar o erro de escalonamento.

10.2 Sobrecusto

A Figura 10.3(a) mostra o tempo gasto realizando cálculos para o controlador em cada período da tarefa. Uma vez que apenas algumas operações simples são necessárias para cada período, o sobrecusto do Feedback Scheduler é muito pequeno, com uma média de um décimo de milissegundo por período.

Uma vez que os cálculos são feitos uma vez por período, a fatia de processador que deve ser dedicada ao controlador é relativa à frequência da tarefa. A Figura 10.3(b) mostra que seria necessária uma tarefa com $T = 1ms$ para fazer com que o controlador tomasse mais de 10% do processador.



(a) Tempo gasto por período. (b) Sobrecusto relativo.

Figura 10.3: Sobrecusto de Feedback Scheduling.

10.3 Composição das Abordagens

Enquanto a infraestrutura de reconfiguração baseada em otimização definida neste trabalho tem como objetivo realizar ajustes grandes nos parâmetros dos servidores, um controlador em malha fechada faz um melhor trabalho de correção automática de pequenos erros de escalonamento. A integração destas duas abordagens leva a uma infraestrutura de escalonamento robusta que provê suporte a aplicações com tempos de computação e períodos altamente dinâmicos.

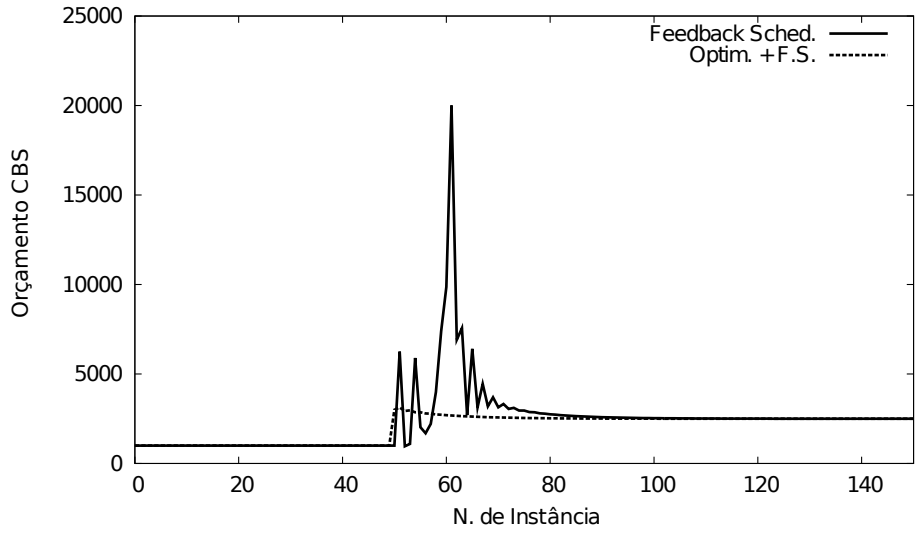


Figura 10.4: Composição das duas abordagens.

A Figura 10.4 mostra a composição das duas abordagens [13]. As primeiras 50 instâncias são escalonadas sem erro com um orçamento de 1000 unidades de tempo. As instâncias a partir da 51ª necessitam de um orçamento de 2500, representando um aumento instantâneo no tempo de computação da tarefa. O Feedback Scheduler, quando usado de forma isolada, oscila até convergir no orçamento correto na instância de número 142.

Para ilustrar a composição da infraestrutura de reconfiguração deste trabalho e o controle realimentado, foi simulada uma superestimação do orçamento necessário da parte do projetista do sistema. Em vez de se aplicar o valor correto de 2500 para o orçamento depois da mudança de modo, usa-se um orçamento de 3000 unidades de tempo. O controlador em malha fechada atua então com muito menos oscilação e converge no valor correto de 2500 na instância 120.

Como pode-se ver na figura, uma resposta adequada para uma variação drástica na utilização é alcançada pela infraestrutura descrita neste trabalho, enquanto o controle realimentado é melhor utilizado para fazer ajustes finos no escalonador de forma a contemplar pequenas variações na utilização das tarefas. O melhor desempenho é obtido ao se integrar ambas as abordagens. Enquanto a abordagem baseada em otimização carrega um alto custo de processamento para grandes números de servidores, ela é executada apenas uma vez a cada grande mudança na utilização das tarefas. O Feedback Scheduling, pelo outro lado, é executado uma vez a cada instância porém requer muito pouco tempo de processador.

10.4 Sumário

Este capítulo mostrou que a união do Feedback Scheduling – a abordagem que usa controle realimentado para atuar nos parâmetros dos servidores – com a infraestrutura definida neste trabalho tem resultados favoráveis, gerando uma infraestrutura combinada capaz de lidar com mais tipos de comportamento dinâmico das tarefas que qualquer uma das abordagens em isolamento. Enquanto a infraestrutura baseada em otimização trata grandes ajustes com mais facilidade, ela não pode ser usada com frequência e necessita de um conhecimento prévio do comportamento temporal da tarefa. O Feedback Scheduling, por outro lado, faz pequenos ajustes contínuos automaticamente e com muito pouco sobrecusto.

Parte IV

Conclusões

11 Conclusão

À medida que a complexidade das aplicações tempo real cresce, mais imprevisível se torna o comportamento temporal de suas tarefas. Em virtude disto, se torna difícil e até impossível a estimativa correta do tempo máximo de execução das suas instâncias, tornando ainda mais importante o suporte em nível de escalonador para lidar com a variabilidade do tempo de execução das tarefas.

Os servidores de tarefas aperiódicas mostram bom desempenho ao lidar com tarefas com alta variabilidade na utilização do processador, porém a definição estática de seus parâmetros traz problemas quando se deve suportar tarefas com diferentes padrões de comportamento temporal. Por exemplo, diferentes modos de execução podem ter utilizações médias entre instâncias drasticamente diferentes, o que não é tratado satisfatoriamente por servidores de tarefas aperiódicas exclusivamente, necessitando de mecanismos de reconfiguração dinâmica no escalonador.

Nesta dissertação foi apresentada uma infraestrutura para a reconfiguração dinâmica de escalonadores tempo real que utiliza otimização para realizar a divisão de tempo de processador entre tarefas. Para este fim foram definidos três modelos de sistema, visando contemplar diferentes tipos de aplicações: o Modelo Discreto, o Modelo Contínuo e o Modelo Híbrido. Em adição a estes modelos foi ainda estudado o uso de funções objetivo não-aditivas para que uma noção de justiça fosse incorporada à solução dos problemas de otimização.

O Modelo Discreto representa o sistema como um conjunto de servidores onde o intervalo de possíveis utilizações é discretizado; existe um número finito de configurações dentre as quais uma deve ser selecionada. Para este modelo, que leva a um problema de otimização NP-Difícil, foram definidos dois esquemas de aproximação, um deles guloso e outro polinomial. Foi demonstrado que o desempenho do algoritmo guloso foi próximo do ótimo numa fração muito pequena do tempo de execução de um algoritmo de programação dinâmica, permitindo o seu uso em sistemas reais onde o tempo de processador é escasso e o tempo de resposta das reconfigurações deve ser curto.

O Modelo Contínuo, o mais simples dentre os apresentados, representa o sistema como um conjunto de servidores que podem ter sua banda definida como qualquer valor dentro de um intervalo contínuo. Uma solução analítica foi apresentada para este modelo, e sua equivalência com um modelo onde orçamento e período são apresentados separadamente foi comprovada. A avaliação numérica do algoritmo também mostrou sua aplicabilidade em sistemas reais. Apesar do Modelo Contínuo ser mais simples, e portanto ser aplicável a um número menor de sistemas, a velocidade do algoritmo ótimo torna sua utilização atraente.

O Modelo Híbrido permite a definição de modos discretos como o Modelo Discreto, porém com um certo grau de liberdade contínua como no Modelo Contínuo. Para este modelo foi apresentado um algoritmo baseado em programação dinâmica, que pode ter seu resultado melhorado por um passo adicional baseado na solução do Modelo Contínuo. Por ser o mais geral dos modelos, ele pode ser aplicado em mais sistemas do que os modelos listados anteriormente. Esta maior aplicabilidade vem em consequência de uma maior complexidade, o que é refletido nos tempos de execução dos seus algoritmos. Ainda assim, num sistema suficientemente complexo este modelo pode ser o único dos apresentados capaz de capturar o comportamento temporal dos modos das aplicações.

Foi apresentada ainda a possibilidade do uso de funções objetivo não-aditivas em cada um dos modelos apresentados, visando a distribuição mais uniforme dos recursos em sistemas onde isso é de interesse. Algoritmos rápidos para os três modelos foram apresentados.

Para exemplificar o uso da infraestrutura e fazer uma avaliação de como ela se adapta a uma aplicação real, um estudo de caso foi realizado usando dados aproximados de uma aplicação real, a saber, o sistema de visão, controle e navegação de um robô móvel. Para avaliar realisticamente os benefícios do uso da infraestrutura, históricos de aplicações multimídia reais foram usados em simulação, demonstrando a aplicabilidade deste trabalho no tratamento de sobrecarga em sistemas reais. Foi realizada ainda a integração de uma política de escalonamento baseada em controle realimentado na infraestrutura, o que levou a uma infraestrutura ainda mais robusta para o tratamento de comportamento temporal dinâmico.

Como trabalho futuro fica o tratamento de consumo estocástico de tempo de processamento, onde cada valor de utilização de cada modo seria uma variável aleatória. Isto permitiria um controle direto da qualidade de serviço do sistema através da probabilidade do cumprimento dos deadlines. A extensão dos modelos para contemplar políticas de escalonamento para sistemas multiprocessados também teria resultados importantes.

Referências Bibliográficas

[1] L. Abeni and G. Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proc. of the 6th IEEE International Conference on Real-Time Computing Systems and Applications (RTCSA 99)*, pages 70–77. IEEE Computer Society, 1999.

[2] L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, 2004.

[3] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a reservation-based feedback scheduler. In *Proc. of the 23rd IEEE Real-Time Systems Symposium (RTSS 02)*, pages 71–80. IEEE Computer Society, 2002.

[4] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, 1993.

[5] N.C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: The deadline-monotonic approach. In *in Proc. IEEE Workshop on Real-Time Operating Systems and Software*, pages 133–137, 1991.

[6] G. Beccari, S. Caselli, and F. Zanichelli. A technique for adaptive scheduling of soft real-time tasks. *Real-Time Systems*, 30(3):187–215, 2005.

[7] Fabrice Bellard. Ffmpeg, 2009. [Online; accessado 07-Junho-2009].

[8] Enrico Bini, Giorgio Buttazzo, and Giuseppe Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In *ECRTS '01: Proceedings of the 13th Euromicro Conference on Real-Time Systems*, page 59, Washington, DC, USA, 2001. IEEE Computer Society.

[9] G. Buttazzo. Research trends in real-time computing for embedded systems. *ACM SIGBED Review*, 3(3), 2006.

[10] G. Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. *Real-Time Systems*, 23(1-2):7–24, 2002.

[11] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proc. of the 21st IEEE Real-Time Systems Symposium (RTSS 00)*, pages 295–304, 2000.

[12] M. Caccamo, G. C. Buttazzo, and D. C. Thomas. Efficient reclaiming in reservation-based real-time systems with variable execution times. *IEEE Trans. Comput.*, 54(2):198–213, 2005.

[13] A. B. de Oliveira, E. Camponogara, and G. Lima. Dynamic reconfiguration in reservation-based scheduling: An optimization approach. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'09)*, 2009.

[14] Z. Deng, J. W. Liu, and S. Sun. Dynamic scheduling of hard real-time applications in open system environment. Technical report, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1996.

[15] J. M. Farines, J. Fraga, and R. S. de Oliveira. *Sistemas de Tempo Real*. Escola de Computação, first edition, 2000.

[16] Free Software Foundation. Glpk (gnu linear programming kit), 2009. [Online; accessado 07-Junho-2009].

[17] T. M. Ghazalie and T. P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9(1):31–67, 1995.

[18] J. Jehuda and A. Israeli. Automated meta-control for adaptable real-time software. *Real-Time Systems*, 14(2):107–134, 1998.

[19] T.-W. Kuo and A. K. Mok. Incremental reconfiguration and load adjustment in adaptive real-time systems. *IEEE Transactions on Computers*, 46(12):1313–1324, 1997.

[20] Argonne National Laboratory. Neos server for optimization, 2009. [Online; accessado 07-Junho-2009].

[21] Chen Lee, John Lehoczky, Ragunathan Rajkumar, and Dan Siewiorek. On quality of service optimization with discrete QoS options. In *Proceedings of the IEEE Real-time Technology and Applications Symposium*, pages 276–286, 1999.

[22] Joseph Y. T. Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237 – 250, 1982.

[23] G. Lima, E. Camponogara, and A. C. Sokolonski. "Dynamic Reconfiguration for Adaptive Multiversion Real-Time Systems". In *Proc. of The 20th IEEE Euromicro Conference on Real-Time Systems (ECRTS 08)*, pages 115–124, Prague, Czech Republic, 2008.

[24] Caixue Lin and Scott A. Brandt. Improving soft real-time performance through better slack reclaiming. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS '05)*, pages 410–421, Washington, DC, USA, 2005. IEEE Computer Society.

[25] G. Lipari and S. Baruah. Greedy reclamation of unused bandwidth in constant-bandwidth servers. In *Proc. of the 12th Euromicro Conference on Real-Time Systems*, pages 193–200, 2000.

[26] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogram in a hard real-time environment. *Journal of ACM*, 20(1):40–61, 1973.

[27] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[28] Chenyang Lu, John A. Stankovic, Tarek F. Abdelzaher, Gang Tao, Sang H. Son, and Michael Marley. Performance specifications and metrics for adaptive real-time systems. In *IEEE Real-Time Systems Symposium*, pages 13–13, 2000.

[29] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves for multimedia operating systems. Technical Report CMU-CS-93-157, Carnegie Mellon University, 1993.

[30] R. Motwani. Approximation algorithms. Book in preparation, 1992.

[31] L. Palopoli, T. Cucinotta, and A. Bicchi. Quality of service control in soft real-time applications. In *Proc. of the 42nd IEEE Conf. on Decision and Control*, pages 664–669, 2003.

[32] Jorge Real and Alfons Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, 2004.

[33] Robota. Main page - robota, 2009. [Online; accessado 07-Junho-2009].

[34] C. Rusu, R. Melhem, and D. Mossé. Multi-version scheduling in rechargeable energy-aware real-time systems. *Journal of Embedded Computing*, 1(2):271–283, 2005.

[35] Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1(1):27–60, June 1989.

[36] M. Spuri and G.C. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Proc. IEEE Real-Time Systems Symposium*, pages 2–11, Dec 1994.

[37] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[38] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, 1998.

[39] P. Zhou, J. Xie, and X. Deng. Optimal feedback scheduling of model predictive controllers. *Journal of Control Theory and Applications*, 4(2):175–180, 2006.