Universidade Federal da Bahia
Instituto de Matemática/Escola Politécnica
Departamentos de Engenharia Mecânica & Ciência da Computação

Programa de Pós-Graduação em Mecatrônica

Sistemas Computacionais

RECONFIGURAÇÃO DINÂMICA EM SISTEMAS DE TEMPO REAL ADAPTATIVOS

Ana Carolina Sokolonski Ferreira

Dissertação de Mestrado

Salvador
24 de julho 2008

Biblioteca Central Reitor Macêdo Costa - UFBA

F413 Ferreira, Ana Carolina Sokolonski.
Reconfiguração dinâmica em sistemas de tempo real adaptativos / Ana Carolina Sokolonski Ferreira, 2008

105 f. : il.

Orientador : Prof. Dr. George Marconi de Araújo Lima.
Co-orientador : Prof. Dr. Eduardo Camponogara.
Dissertação (mestrado) - Universidade Federal da Bahia. Instituto de Matemática, Escola Politécnica, 2008.

1.Programação em tempo real. 2. Automação. 3. Programação de sistemas (Computação). 4. Mecatrônica. I. Lima, George Marconi de Araújo. II.Camponogara, Eduardo. III.Universidade Federal da Bahia, Instituto de Matemática. IV. Universidade Federal da Bahia, Escola Politécnica. V. Título.

CDU - 519.68
CDD - 519.7

Universidade Federal da Bahia
Instituto de Matemática/Escola Politécnica
Departamentos de Engenharia Mecânica & Ciência da Computação

Ana Carolina Sokolonski Ferreira

RECONFIGURAÇÃO DINÂMICA EM SISTEMAS DE TEMPO REAL ADAPTATIVOS

Trabalho apresentado ao Programa de Pós-Graduação em Mecatrônica do Departamentos de Engenharia Mecânica & Ciência da Computação da Universidade Federal da Bahia como requisito parcial para obtenção do grau de Mestre em Mecatrônica.

Orientador: *Prof. Dr. George Marconi de Araújo Lima*
Co-orientador: *Prof. Dr. Eduardo Camponogara*

Salvador
24 de julho 2008

Ao meu filho, Gabriel, que ilumina a minha vida.
Ao meu marido, Moizés, pelo seu amor incondicional.

O homem que sabe reconhecer os limites da sua própria inteligência está mais perto da perfeição.

—JOHANN GOETHE

AGRADECIMENTOS

Sinto-me estranha ao escrever estes agradecimentos. Parece que foi ontem a escrita dos agradecimentos de minha graduação. O tempo passa rápido, mais rápido do que gostaríamos. Ao mesmo tempo, o mestrado parece ter durado uma eternidade. Para realizar o meu sonho de ser mestre, adiei o meu sonho de ser mãe, fiquei longe de meu marido para comparecer a congressos, fazer trabalhos e realizar missões de estudos (os dias duravam semanas). Hoje, após tantos sacrifícios pessoais, sei que faria tudo novamente, mas nunca sem o apoio de muitas pessoas essenciais.

O meu filho, Gabriel, que está chegando, é meu grande presente. Veio na hora certa, muito desejado e amado. Agradeço a ele minha constante alegria, sempre me chutando nos momentos difíceis, lembrando-me que a vida é linda, igual a ele. Ao meu marido amado, Moizés, agradeço o apoio e a compreensão, o amor e o braço forte, que não me deixou cair. Sem ele, eu não teria chegado aqui.

Agradeço a minha mãe e minhas irmãs pelo apoio. Agradeço ao meu orientador, George, pela compreensão. Agradeço a Marcos Camada pela eterna amizade e apoio. A Eduardo, meu co-orientador, pela paciência. Ao LaSiD, por me acolher durante o desenvolvimento de meu projeto. Aos meus amigos e familiares por entenderem as minhas constantes ausências. Por fim, a CAPES, pelo financiamento de minha bolsa de mestrado.

RESUMO

Sistemas mecatrônicos têm se tornado cada vez mais complexos e têm exigido mecanismos para prover suporte à adaptação em diferentes níveis. Por exemplo, um robô pode ter que se ajustar a diferentes modos de operação, dependendo das demandas do ambiente. Quando há necessidade de se cumprir requisitos temporais em tais sistemas, a complexidade pode ser ainda mais elevada. Neste caso, tais sistemas são conhecidos como *de tempo real*. Esta dissertação descreve alguns mecanismos para prover suporte à adaptação dinâmica em sistemas mecatrônicos de tempo real, ou mais genericamente, sistemas de tempo real.

Prover adaptação dinâmica em tais sistemas não é trivial, pois os diversos componentes do sistema podem possuir diferentes características temporais ou de criticidade. Como a correção temporal de sistemas de tempo real é garantida pelos mecanismos de escalonamento, estes devem ainda estar integrados àqueles que fornecem suporte à adaptação. Nesta dissertação, usou-se *mecanismos de reconfiguração dinâmica* como meio de prover adaptação. Os mecanismos aqui propostos assumem que um sistema de tempo real pode ser estruturado como um conjunto de tarefas, cada uma das quais com múltiplas versões. Como cada uma das versões fornece ao sistema um dado benefício com um determinado custo de execução, os mecanismos de reconfiguração propostos resolvem um problema de otimização, cujo objetivo é maximizar o benefício médio do sistema sujeito ao cumprimento de suas restrições temporais, restrições estas garantidas por mecanismos de escalonamento aqui desenvolvidos.

Os mecanismos de reconfiguração e de escalonamento aqui propostos foram precisamente descritos. Suas correções foram demonstradas e seus desempenhos avaliados através de simulação. Tais resultados vêm contribuir com a área de sistemas de tempo real adaptativos ao mesmo tempo em que apontam futuros caminhos de investigação neste campo do conhecimento.

Palavras-chave: escalonamento dinâmico, sistema de tempo real, reconfiguração dinâmica.

ABSTRACT

Mechatronic systems have become increasingly complex. They have been required to provide mechanisms to support different levels of adaptation. For example, a robot may have to adjust different modes of operation depending on the demands of the environment. If there is a need to meet time requirements in such systems, when they are called *real-time systems*, their complexity is considerably increased. This work describes some mechanisms to provide support to dynamic adaptation of mechatronic real-time systems, or more generally, to real-time systems.

Providing dynamic adjustment in such systems is not an easy task. Their various components may have different temporal characteristics or criticality. As their timing correctness is guaranteed by scheduling mechanisms, support to adaptation have to work with scheduling mechanisms in an integrated way. In this dissertation *dynamic reconfiguration mechanisms* have been used as a means of providing adaptation. The proposed mechanisms assume that a real-time system can be structured as a set of tasks, each of which has multiple versions. As each version provides a given benefit for the system and requires a specified execution cost, dynamic reconfiguration is modeled as an optimization problem whose goal is to maximize the average benefit of the system subject to the its temporal restrictions.

The proposed mechanisms are precisely described, their correctness are proved and their performance are evaluated through simulation. The contribution of these results are twofold. They improve on existing techniques in the area of adaptive real-time systems and open some future research paths in this field.

Keywords: dinamic scheduling, real-time systems, dynamic reconfiguration

SUMÁRIO

Capítulo 1—Introdução 3

1.1 Sistemas Mecatrônicos e Sistemas de tempo real 3
1.2 Mecanismos de Escalonamento 4
1.3 Adaptação à Ativação de Tarefas Aperiódicas 6
1.4 Proposta e Contribuições 7
1.5 Estrutura 8

Capítulo 2—Escalonamento Flexível 9

2.1 Notação e Definições Iniciais 10
2.2 Consideração Sobre Condições de Sobrecarga 12
2.3 Isolamento Temporal Entre Tarefas 13
2.3.1 Servidores com *Carga* Fixa 16
2.3.2 Servidores com *Carga* Variável 19
2.4 Cancelamento ou Rejeição de Tarefas 20
2.5 Reconfiguração Dinâmica 23
2.5.1 Reconfiguração Considerando o Custo Computacional 25
2.5.2 Reconfiguração Considerando o Período 27
2.6 Sumário 29

Capítulo 3—Reconfiguração Dinâmica: Um Estudo Preliminar 31

3.1 Modelo 32
3.2 O Problema de Reconfiguração 34
3.2.1 Reconfiguração de Tarefas 35
3.2.1.1 Idéia Básica 35
3.2.1.2 Reconfiguração Usando o TBS 36
3.2.1.3 Reconfiguração Usando Apenas EDF 38
3.2.1.4 Simulação e Resultados 40
3.2.2 Reconfiguração Baseada em Instâncias de Tarefas 44
3.2.2.1 Simulação e Resultados 46
3.3 Sumário 48

Capítulo 4—Estudo de Testes de Escalonabilidade Em Tempo de Execução 49

4.1 Teste de Escalonabilidade Baseado em Trabalho Acumulado 50
4.2 Teste de Escalonabilidade Baseado em Seqüenciamento 54
4.3 Simulação e Resultados 55

4.3.1 Avaliando o Desempenho sem Inserção de Tarefas Aperiódicas . 56
4.3.2 Avaliando o Desempenho com Inserção de Tarefas Aperiódicas . 59
4.3.3 Avaliando o Desempenho com Alteração do Tamanho do Intervalo 60
4.4 Sumário 62

Capítulo 5—Reconfiguração Dinâmica: O Problema de Otimização 63

5.1 Reconfiguração Baseada em Trabalho Acumulado 64
5.1.1 Formulação Recursiva 65
5.2 Reconfiguração Baseada em Seqüenciamento de Instâncias 69
5.2.1 Formulação Recursiva 70
5.3 Exemplo Ilustrativo 74
5.3.1 Mecanismo de Reconfiguração Baseado em Trabalho Acumulado 76
5.3.2 Mecanismo de Reconfiguração Baseado em Seqüenciamento de Instâncias 78
5.4 Solução Aproximada 79
5.4.1 Comparação entre as soluções aproximadas das propostas de reconfiguração 80
5.5 Simulação Comparativa 83
5.6 Sumário 85

Capítulo 6—Conclusão 86

LISTA DE FIGURAS

2.1 Escalonamento das tarefas τ_1, τ_2, τ_3, τ_4 e τ_5 segundo o EDF 12
2.2 Escalonamento das tarefas, τ_1, τ_2, τ_3, τ_4 e τ_5 segundo o EDF usando servidor 15
2.3 Escalonamento das tarefas τ_1, τ_2 e τ_3 segundo o EDF, com DDS para executar as tarefas aperiódicas 17
2.4 Escalonamento das tarefas τ_1, τ_2 e τ_3 segundo o EDF, com CBS para executar as tarefas aperiódicas 18
2.5 Escalonamento das tarefas τ_1, τ_2 e τ_3 segundo o EDF, com TBS para executar as tarefas aperiódicas 20
2.6 Escalonamento das tarefas, τ_1, τ_2, τ_3, τ_4 e τ_5 segundo o EDF usando reconfiguração de custo computacional 26
2.7 Escalonamento das tarefas, τ_1, τ_2, τ_3, τ_4 e τ_5 segundo o EDF usando reconfiguração de período 29

3.1 Admissão de τ_3 assumindo degradação controlada. 37
3.2 Número de instâncias de τ_j consideradas em $[t, d_i)$ 38
3.3 Percentual de *deadlines* perdidos de tarefas periódicas para $u_a = 40\%$. 42
3.4 Percentual de rejeição de tarefas aperiódicas para $u_a = 40\%$ 43
3.5 Degradação média do sistema para $u_a = 40\%$ 44
3.6 Percentual de rejeição de tarefas aperiódicas 47

4.1 Interferência w_i^2 na execução de J_i 51
4.2 w_i^2 não interfere na execução de J_i 52
4.3 A interferência w_i^2 na execução de J_i é $w_i^2 - s_i - s_i'$ 53
4.4 Uma ilustração da condição de escalonabilidade do teste baseado em trabalho acumulado. $\Gamma(0,7) = \{J_1, J_2, J_3\}$ e $J_i = (r_i, C_i, d_i)$, onde $i = 1, 2, 3$. 53
4.5 Uma ilustração da condição de escalonabilidade do teste baseado em seqüenciamento de instâncias. $\Gamma(0,7) = \{J_1, J_2, J_3\}$ e $J_i = (r_i, C_i, d_i)$ onde $i = 1, 2, 3$. 55
4.6 Percentual de intervalos não escalonáveis por carga periódica, u_p 57
4.7 Percentual de intervalos não escalonáveis por tamanho do intervalo, $u_p = 60\%$ 58
4.8 Gráficos de taxa de descarte de tarefas aperiódicas por carga periódica 60
4.9 Percentual de descarte de tarefas aperiódicas por tamanho do intervalo de reconfiguração, $u_p = 60\%$ 61
4.10 Quantidade média de instâncias no intervalo por tamanho do intervalo de reconfiguração, $u_p = 60\%$ 62

5.1 Instâncias de $\Gamma(0, 234)$ não escalonáveis no intervalo $[0, 234)$. 75
5.2 Tempo de execução da proposta de reconfiguração baseada em trabalho acumulado usando aproximação pelo limite inferior 82
5.3 Tempo de execução da proposta de reconfiguração baseada em seqüenciamento de instâncias usando aproximação pelo limite inferior 83
5.4 Percentual de descarte de tarefas aperiódicas por carga periódica 84

Declaração

Declaro que o texto desta dissertação é de minha autoria e que todos os trechos e resultados nela utilizados advindos de outras fontes possuem referência de origem claramente expressa.

Ana Carolina Sokolonski Ferreira

Trabalhos Publicados Pela Autora

1) LIMA, George ; SOKOLONSKI, Ana Carolina ; CAMPONOGARA, Eduardo . Dynamic Reconfiguration for Adaptive Multiversion Real-Time Systems (to appear). In: 20th Euromicro Conference on Real-Time Systems (ECRTS 08), 2008, Praga, República Checa. 20th IEEE Euromicro Conference on Real-Time Systems (ECRTS 08), 2008. p. 1-10.[55]

2) LIMA, George ; SOKOLONSKI, Ana Carolina ; CAMPONOGARA, Eduardo . Online Schedulability Tests for Real-Time Systems. In: 10th Brazilian Workshop on Real-Time and Embedded Systems, 2008, Rio de Janeiro - RJ. Proc. of the 10th Brazilian Workshop on Real-Time and Embedded Systems: SBC, 2008. p. 61-68.[56]

3) SOKOLONSKI, Ana Carolina ; LIMA, George ; BARRETO, Luciano Porto . Suporte à Degradação Controlada em Sistemas de Tempo Real. In: VIII Workshop de Testes e Tolerância a Falhas (WTF 2007), 2007, Belém - Pará. Anais do VIII Workshop de Testes e Tolerância a Falhas (WTF 2007): SBC, 2007. p. 29-42.[57]

CAPÍTULO 1

INTRODUÇÃO

1.1 SISTEMAS MECATRÔNICOS E SISTEMAS DE TEMPO REAL

Um sistema mecatrônico é conhecido por integrar componentes eletromecânicos e controle computacional [60]. Segundo David M. Auslander, um sistema mecatrônico é uma aplicação que toma decisões complexas para operar sistemas físicos [5]. Tais decisões, normalmente, são solucionadas através do uso de sistemas computacionais. De fato, a possibilidade de contemplar nos sistemas computacionais mecanismos de inteligência artificial, sistemas distribuídos, agentes móveis e tolerância a falhas, por exemplo, fornece notório avanço na melhoria da funcionalidade e flexibilidade dos sistemas mecatrônicos.

Vários sistemas mecatrônicos podem ser caracterizados pela necessidade de reagir a eventos do ambiente respeitando restrições temporais previamente estabelecidas. O sistema computacional que fornece esse suporte é conhecido como sistema de tempo real. Tais sistemas são geralmente modelados como um conjunto de tarefas, ativadas como conseqüência da ocorrência dos eventos aos quais elas estão associadas. Tarefas são unidades de processamento, ou blocos de ações seqüenciais, que usam um ou mais recursos computacionais [9]. A especificação da restrição temporal de cada tarefa é geralmente expressada através de *deadlines*[1]. Assim, num sistema de tempo real, as tarefas devem apresentar correção lógica e temporal, ou seja, elas devem produzir resultados corretos dentro de intervalos de tempo pré-estabelecidos.

Sistemas mecatrônicos com restrições temporais estão presentes no nosso cotidiano, como por exemplo, sistemas de controle de vôo e sistemas de frenagem ABS de automóveis. Uma característica comum destes sistemas é sua funcionalidade de controle sujeita a rígidas restrições temporais. As tarefas deste tipo de sistema são ativadas

[1]Utilizamos aqui o termo em inglês por este ser mais usado e amplamente difundido.

contínua e periodicamente, obtendo os valores das variáveis do ambiente e atuando de forma condizente para manter os valores das variáveis controladas em níveis pré-estabelecidos. Neste modelo, o tempo máximo de execução e o período entre ativações de cada tarefa são conhecidos *a priori*.

É importante observar que este modelo de tarefas estritamente periódico é altamente previsível, o que facilita a construção de mecanismos de execução dos sistemas de tempo real. No entanto, em muitos cenários, é desejável que o sistema possa se adaptar à ocorrência de eventos aperiódicos, reagindo aos mesmos através de ativações de tarefas não-periódicas. Por exemplo, a detecção de erros pode ativar uma rotina de recuperação, que é, por definição, uma tarefa não-periódica. A incorporação de tarefas de processamento de imagem para lidar com a interface homem-máquina em sistemas de controle é outro exemplo representativo.

Esta dissertação descreve mecanismos para fornecer suporte ao tratamento de tarefas não periódicas em sistemas de tempo real. O principal foco será dado à capacidade de adaptação do sistema quando tais tarefas precisarem ser ativadas. Antes de entrar em mais detalhes sobre o tema, alguns conceitos básicos serão apresentados na próxima seção.

1.2 MECANISMOS DE ESCALONAMENTO

Os sistemas de tempo real podem ser caracterizados pela criticidade dos *deadlines* de suas tarefas. Quando a violação do *deadline* de uma tarefa tem conseqüências graves, tal tarefa é dita crítica. Caso contrário, esta é chamada de não-crítica. Um sistema com ao menos uma tarefa crítica é chamado de sistema de tempo real crítico [13].

Um importante mecanismo, responsável por garantir a correção temporal num sistema de tempo real, seja ele crítico ou não, é o escalonador. De maneira simples, o escalonador deve escolher, baseado em alguma política de escalonamento, qual tarefa deve ser executada em função do tempo. Uma das políticas de escalonamento mais

difundida é o EDF (*Earliest Deadline First*)[2] [43]. De acordo com esta política, a tarefa a ser executada é aquela que possui o *deadline* mais próximo a expirar. A razão para esta popularidade é que o EDF possui uma implementação simples e é ótimo. Ser ótimo significa que, se há um escalonamento possível, onde nenhuma tarefa perde o *deadline*, então o EDF também gera uma escala de execução onde nenhum *deadline* é violado [45]. Para verificar se existe um escalonamento possível, realiza-se um teste de escalonabilidade.

O teste de escalonabilidade é responsável por verificar se há riscos de perda de *deadline* de alguma tarefa. Este teste pode ser realizado em tempo de projeto, ou seja, antes do sistema ser executado, ou em tempo de execução, durante a execução do sistema. Por exemplo, se as tarefas do sistema forem periódicas, o teste de escalonabilidade pode ser realizado em tempo de projeto. Para um sistema $\Gamma = (\tau_1, \tau_2, ..., \tau_n)$ composto de tarefas periódicas e independentes, uniprocessado e escalonado pelo EDF, sabe-se que nenhuma destas tarefas terá seu *deadline* violado se e somente se:

$$\sum_{\forall \tau_i \in \Gamma} \frac{C_i}{T_i} \leq 1 \tag{1.1}$$

onde C_i é o custo máximo de execução da tarefa τ_i e T_i é o seu período. Obviamente, caso tarefas aperiódicas sejam consideradas, a equação (1.1) não pode ser utilizada, pois, para estas tarefas, não há como prever o instante de ativação e, muitas vezes, seu custo computacional somente é conhecido neste instante. Para sistemas que contemplam tais tarefas existe a necessidade de executar um teste de escalonabilidade em tempo de execução. Neste caso, tal teste deveria fornecer informação sobre o risco de violação de *deadlines* dinamicamente, o que poderia subsidiar o sistema para se adaptar, por exemplo, descartando tarefas menos relevantes.

Os mecanismos desenvolvidos nesta dissertação possuem o objetivo de prover capacidade de adaptação e maior flexibilidade ao sistema e, portanto, estarão baseadas em testes de escalonabilidade em tempo de execução.

[2]Utilizaremos, neste caso, a sigla em inglês, pois esta é largamente utilizada na literatura brasileira da área.

Pode-se ainda classificar os testes de escalonabilidade em necessário, suficiente e exato. Testes exatos, como é o caso da equação (1.1), fornecem informação precisa sobre o risco de violação de *deadline* no sistema. Em outras palavras, estes testes fornecem condições necessárias e suficientes para sua escalonabilidade. Testes suficientes, por outro lado, podem não reconhecer sistemas escalonáveis, mas garantem que nenhum sistema não-escalonável seria classificado como escalonável. Testes de escalonabilidade para serem executados em tempo de execução, como os desenvolvidos neste trabalho, são geralmente suficientes. Os testes necessários são aqueles que precisam ser satisfeitos para que o sistema seja escalonável, porém este não identifica sistemas não escalonáveis.

1.3 ADAPTAÇÃO À ATIVAÇÃO DE TAREFAS APERIÓDICAS

Caso o sistema não forneça mecanismos para reconhecer riscos de violações de *deadlines*, a inserção de tarefas aperiódicas pode gerar sobrecargas e, por conseguinte, perdas de *deadlines*. Existem algumas formas de se lidar com tal situação:

- Isolamento Temporal - Tal abordagem consiste em "isolar" os efeitos das tarefas aperiódicas sobre as periódicas. Uma das abordagens comumente utilizadas são os servidores de tarefas aperiódicas [45]. Estes consistem, em geral, em uma tarefa virtual que reserva parte do tempo do processador para a execução das tarefas aperiódicas por ele servidas, controlando, assim, o impacto sobre a execução das tarefas periódicas. Os servidores de tarefas aperiódicas têm um papel importante no desempenho do sistema. Além de controlar a interferência das tarefas aperiódicas sobre as tarefas periódicas, as regras de definição do servidor diminuem o tempo de resposta médio das tarefas aperiódicas, ou seja, o intervalo de tempo para que tais tarefas terminem de executar após serem ativadas. Os servidores de tarefas aperiódicas são adequados quando existem tarefas aperiódicas não-críticas, pois tais mecanismos não cumprem os *deadlines* das tarefas aperiódicas. Sistemas de tempo real modernos possuem tarefas aperiódicas

críticas, tais como rotinas de recuperação de erros, ativadas após a detecção de erros, e sua execução não deve interferir no cumprimento dos *deadlines* das demais tarefas do sistema, apesar de ser uma tarefa crítica. Desta forma, diferentes mecanismos devem ser utilizados para manter a correção destes sistemas.

- Cancelamento de tarefas - Nesta abordagem deve ser realizado um teste de escalonabilidade na ocasião da ativação da tarefa não periódica que, em caso de riscos de violações de *deadlines*, cancele a tarefa não periódica, ou um conjunto de tarefas periódicas, com o objetivo de manter o correto funcionamento do sistema. Sabe-se que tal abordagem é restritiva, pois muitas vezes as tarefas em questão são críticas, o que não proporciona a possibilidade de serem descartadas ou canceladas. Desta forma, tal abordagem nem sempre é adequada.

- Reconfiguração Dinâmica - Tal mecanismo visa adaptar o sistema à inserção da tarefa não periódica através da utilização do mecanismo de degradação controlada (*graceful degradation*), onde o sistema continua a funcionar, respeitando os *deadlines* das tarefas, mesmo com a presença de sobrecargas, aceitando uma degradação parcial de suas funcionalidades ou desempenho, durante o intervalo de reconfiguração, ou seja, durante o tempo em que as tarefas que precisaram ser degradadas estejam executando. Desta forma, consegue-se executar corretamente (cumprindo os *deadlines*) todas as tarefas ativadas no sistema, porém fornecendo uma qualidade de serviço inferior à original.

Nesta dissertação serão abordados alguns mecanismos de reconfiguração, pois estes se mostraram mais adequados para o propósito de executar corretamente todas as tarefas do sistema, respeitando seus *deadlines*, mesmo em presença de sobrecargas.

1.4 PROPOSTA E CONTRIBUIÇÕES

A proposta desta dissertação consiste em prover uma reconfiguração do sistema, em nível de escalonamento, que permita que tarefas críticas executem no sistema sem

perdas de *deadlines*. Tal reconfiguração é realizada através do uso de degradação controlada. A idéia básica é adaptar o sistema, na ocasião da ativação de uma tarefa aperiódica que cause sobrecarga no mesmo, através da degradação das tarefas periódicas, diminuindo assim a qualidade do serviço prestado, porém executando todas as tarefas do sistema corretamente. Para tanto, testes de escalonabilidade em tempo de execução foram desenvolvidos e mecanismos foram elaborados. Simulações foram utilizadas para avaliar tais contribuições.

1.5 ESTRUTURA

No capítulo 2 serão definidos os principais conceitos para o entendimento da proposta aqui apresentada, além de descrever os principais trabalhos relacionados. No capítulo 3, o problema a ser tratado na dissertação será definido e uma proposta preliminar de reconfiguração baseada em degradação controlada, com o uso de servidores de tarefas aperiódicas, será descrita. Dois testes de escalonabilidade serão desenvolvidos e comparados através de simulação no capítulo 4. Os mecanismos de reconfiguração, baseados em tais testes de escalonabilidade, serão descritos e avaliados no capítulo 5. O capítulo 6 resume as conclusões e aponta algumas perspectivas futuras.

CAPÍTULO 2

ESCALONAMENTO FLEXÍVEL

Sistemas de tempo-real modernos são caracterizados por possuírem tarefas com características diversas. Se tal sistema possuísse apenas tarefas periódicas temporalmente bem comportadas, poderia-se prever a carga do sistema, dado, por exemplo, pelo percentual do processador ocupado com a execução das mesmas. Tal fator possibilitaria a realização de testes de escalonabilidade em tempo de projeto, permitindo que a correção temporal desse sistema fosse testada em tempo de projeto. Porém, se há possibilidade de ocorrência de eventos cujos instantes de ocorrência são imprevisíveis, o sistema pode sofrer intervalos em que a execução de diversas tarefas causam sobrecargas. Nestes intervalos, pode ser impossível ao sistema executar todas as suas tarefas sem causar perda de algum *deadline*. Neste contexto, é essencial que haja o adequado suporte do escalonador para permitir adaptação da aplicação em função das demandas do ambiente. Tal suporte pode visar desde economia de energia até rapidez computacional. Por exemplo, sistemas que realizam gerenciamento de energia [52] estão presentes em satélites [54]. Adaptação dinâmica também é necessária em sistemas que realizam reconhecimento de campos militares inimigos [50], ou que operam o piloto automático de um carro [53].

Este capítulo aborda algumas técnicas para tratar tarefas aperiódicas ou esporádicas e possíveis sobrecargas. Inicialmente, apresentaremos o modelo de tarefas e a notação utilizados. Em seguida, algumas considerações sobre o comportamento temporal do sistema em sobrecarga serão feitas na seção 2.2. Posteriormente, serão discutidas algumas alternativas para proteger o sistema contra os efeitos nocivos das sobrecargas nas seções 2.3, 2.4 e 2.5. O sumário do capítulo é apresentado na seção 2.6.

2.1 NOTAÇÃO E DEFINIÇÕES INICIAIS

A notação apresentada aqui será utilizada ao longo do texto. Nos capítulos seguintes, tal notação será complementada à medida que for sendo necessário fazê-lo.

Considerou-se um sistema de tempo-real uniprocessado com preempção, composto por três tipos de tarefas, periódicas, esporádicas e aperiódicas. Este sistema é representado pelo conjunto $\Gamma = \{\tau_1, \tau_2, ...\}$, onde τ_i é uma tarefa. Cada ativação da tarefa τ_i gera uma instância da mesma, J_j. Instâncias são unicamente identificadas, ou seja, J_i e J_j são duas instâncias distintas de tarefas se, e somente se, $i \neq j$.

Se τ_i é uma tarefa periódica, o instante de ativação da sua k-ésima instância é dado por $(k-1)T_i$, onde $T_i > 0$ é o período de τ_i e $k > 0$. O *deadline* relativo de τ_i é o intervalo máximo de tempo em que qualquer instância de τ_i deve executar, representado por D_i. Em particular, o *deadline* absoluto do k-ésima instância de τ_i é dado por $d_i = (k-1)T_i + D_i$. Assumimos que $D_i \leq T_i$.

O conjunto $\Gamma(t, t') = \{J_1, \ldots, J_n\}$ representa as n instâncias de tarefas ativas no intervalo $[t, t')$. Diz-se que J_i é uma instância ativa neste intervalo se ela foi ativada num instante $r_i < t'$ e ainda não finalizou sua execução até o instante t'. Note que pode haver mais de uma instância da mesma tarefa τ_i em $\Gamma(t, t')$. Sem perda de generalidade, assumimos que o conjunto $\Gamma(t, t')$ é ordenado segundo os *deadlines* das instâncias, $d_i \leq d_{i+1}$ para $i < n$.

Um sistema é dito sobrecarregado quando este não é escalonável por alguma política de escalonamento. Sobrecargas podem ocorrer em intervalos de tempo e não necessariamente durante todo o período de execução do sistema. Sistemas que não possuem ao menos um destes intervalos são chamados de sistemas em subcarga ou carga normal. Assim, os sistemas em carga normal, por não sofrerem problemas de sobrecarga, podem executar suas tarefas sem perdas de *deadline*, bastando que sua correção temporal seja garantida por algum teste de escalonabilidade, executado em tempo de projeto [45, 14].

Cada sistema possui um modelo de escalonamento próprio, onde as características

das tarefas do sistema e sua política de escalonamento estão definidas. Dependendo da política de escalonamento e do modelo de tarefas, pode-se verificar, em tempo de projeto, para um determinado sistema, o percentual máximo de utilização do processador, ou ainda, o tempo de resposta máximo de cada tarefa. Considere um sistema com preempção composto apenas por tarefas periódicas com *deadlines* iguais aos períodos. Dado o conjunto com todas as tarefas periódicas do sistema, a demanda por processador deste conjunto é dada por $u_p = \sum_{\forall \tau_i \in \Gamma} \frac{C_i}{T_i}$, sendo C_i o tempo máximo de execução da tarefa τ_i e T_i o seu período. Se este sistema estiver operando segundo a política *Earliest Deadline First* (EDF) [43], baseada em prioridade variável, $u_p \leq 1$ garante sua escalonabilidade. Assumimos neste trabalho que a política de escalonamento é EDF.

Outra forma de avaliar a possibilidade de perdas de *deadline* de um sistema é calculando o tempo de resposta de cada uma das tarefas no pior caso. O tempo de resposta máximo é o tempo estimado que a tarefa levará para concluir sua execução, no pior caso. Caso este tempo seja menor ou igual que o *deadline* da tarefa, então esta cumprirá seu *deadline*, caso contrário a tarefa poderá perder seu *deadline* [33]. Testes de escalonabilidade baseados no tempo de resposta são mais comuns e eficientes quando o escalonamento é baseado em prioridades fixas. Desta forma, testes de escalonabilidade foram utilizados baseados na demanda do processador para o EDF.

Nem sempre é possível projetar sistemas de tempo-real modernos para operar sempre em subcarga, pois tais sistemas precisam ser reativos, lidar com falhas e outras imprevisibilidades, que podem causar sobrecargas. Desta forma, tais sistemas necessitam de técnicas dinâmicas que os adaptem às imprevisibilidades inerentes ao ambiente. Para tanto, deve haver suporte adequado em seus mecanismos de escalonamento. Nas próximas seções serão discutidos os efeitos da ocorrência de sobrecarga e alguns mecanismos de tratamento destes efeitos.

2.2 CONSIDERAÇÃO SOBRE CONDIÇÕES DE SOBRECARGA

Um sistema em regime de sobrecarga pode experimentar perdas de *deadlines* de uma ou mais instâncias de tarefas. Em sistemas escalonados segundo políticas de escalonamento de prioridade variável, como o EDF, sobrecargas podem ter conseqüências desastrosas. Por exemplo, observe a Figura 2.1, onde as tarefas periódicas τ_1 ($C_1 = 1$, $T_1 = D_1 = 10$), τ_2 ($C_2 = 3$, $T_2 = D_2 = 16$), τ_3 ($C_3 = 2$, $T_3 = D_3 = 13$) e τ_4 ($C_4 = 5$, $T_4 = D_4 = 20$) estão escalonadas de acordo com EDF e terão suas primeiras instâncias ativadas no instante $t = 0$. Este sistema é escalonável, pois $u_p = \sum \frac{C_i}{T_i} = \frac{1}{10} + \frac{3}{16} + \frac{2}{13} + \frac{5}{20} \cong 0,69 < 1$. Suponha agora que, no instante 1, a tarefa aperiódica τ_5 seja ativada, com *deadline* absoluto 16 e tempo máximo de execução $C_5 = 11$. Esta instância causa a perda de *deadline* da instância da tarefa τ_2, J_4, que precisa executar 3 unidades e atinge seu *deadline* tendo executado apenas 2 unidades, causando outras perdas de *deadline* em seqüência, das instâncias J_5 e J_6, fenômeno conhecido como "efeito dominó" [45].

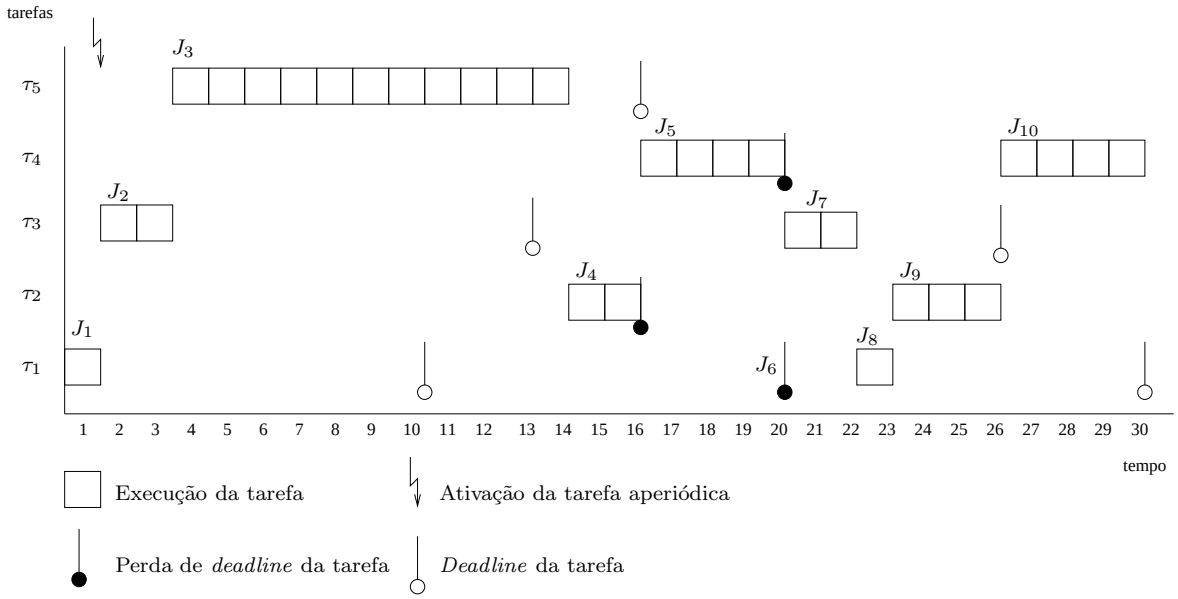


Figura 2.1. Escalonamento das tarefas τ_1, τ_2, τ_3, τ_4 e τ_5 segundo o EDF

Para proteger o sistema do efeito ilustrado na Figura 2.1, necessita-se de mecanismos de escalonamento mais apropriados. Há três tipos de técnicas para se lidar

com sobrecargas. As características da aplicação e os objetivos pretendidos em termos de escalonamento são os fatores considerados na escolha da(s) técnica(s) mais apropriada(s). Em geral, elas utilizam os seguintes princípios:

1) **Isolamento temporal entre tarefas** - A idéia básica é reservar uma porção máxima do processador para as instâncias de tarefas que podem causar sobrecarga. Caso mais do que foi reservado seja necessário, apenas as instâncias servidas por tal porção serão afetadas. No exemplo da Figura 2.1, isto significaria, por exemplo, não cumprir o *deadline* da instância de tarefa aperiódica para preservar a correção temporal das demais. Esta técnica será vista na seção 2.3.

2) **Cancelamento ou rejeição de tarefas** - Pode-se instrumentar o sistema com um teste de escalonabilidade em tempo de execução, que é executado a cada ativação de uma instância de tarefa, por exemplo. Se há risco de perda de *deadline* no sistema, pode-se decidir cancelar alguma tarefa ou rejeitar a nova tarefa, o que for mais apropriado. Técnicas deste tipo serão descritas na seção 2.4.

3) **Reconfiguração dinâmica** - Alguma aplicação pode ser composta de múltiplas versões de uma mesma tarefa, por exemplo. Neste caso, ao se detectar riscos de perda de *deadline*, pode-se escolher uma configuração alternativa para o sistema, mudando o modo de operação para alguma(s) instância(s) de tarefa(s) de forma a lidar com os efeitos da sobrecarga.

2.3 ISOLAMENTO TEMPORAL ENTRE TAREFAS

O isolamento temporal entre tarefas deve separar o comportamento das instâncias de tarefas de modo que uma não cause perdas de *deadlines* de outras. Por exemplo, suponha que não se conheça exatamente a carga relativa de determinada tarefa ou grupos de tarefas. Isto é comum quando há tarefas aperiódicas ou tarefas periódicas com custos computacionais desconhecidos previamente. Nestes casos, é interessante

isolar temporalmente as tarefas para que, o comportamento imprevisível de algumas tarefas não interfira no comportamento correto das tarefas bem comportadas.

Isolamento temporal é comumente usado para lidar com sistemas onde tarefas periódicas e aperiódicas co-existem. Nestes sistemas, como muitas vezes as instâncias aperiódicas são não-críticas e precisam de um tempo de resposta curto, técnicas para prover isolamento temporal entre tarefas devem executar as instâncias aperiódicas o mais rápido possível, sem causar perdas de *deadlines* de instâncias periódicas, que são, geralmente, críticas. De fato, instâncias de tarefas críticas não possuem urgência de execução, sendo necessário apenas que estas cumpram seus *deadlines*. Desta forma, os objetivos são: (a) minimizar o tempo de resposta das instâncias de tarefas aperiódicas; (b) não violar os *deadlines* das instâncias de tarefas periódicas.

Uma abordagem simples, que pode ser usada para isolamento temporal entre instâncias periódicas e aperiódicas é conhecida como execução em *background* [45]. De acordo com esta abordagem, instâncias aperiódicas somente são escalonadas e executadas quando não existem instâncias periódicas prontas para executar. Este método é de simples implementação e sempre apresentará um escalonamento correto, não provocando perdas de *deadlines* de instâncias de tarefas críticas. É, porém, um método pessimista e não atende ao objetivo (a), mencionado anteriormente.

Uma outra abordagem é através do uso de servidores. Um servidor de tarefas aperiódicas, ou simplesmente servidor, é um mecanismo que visa executar as tarefas aperiódicas, controlando a interferência das mesmas sobre as tarefas periódicas. A idéia é atribuir um custo computacional ao servidor. Este custo é, normalmente, conhecido como *carga* e representa o tempo máximo que o servidor disporá para executar instâncias aperiódicas em determinado intervalo de tempo. Desta forma, delimita-se a ocupação do processador para um servidor para que esta seja utilizada pelas instâncias por ele servidas. Ao ser escalonado para execução, o servidor tem sua carga decrementada na mesma quantidade e velocidade em que as instâncias por ele servidas executam. Quando a carga é nula, o servidor espera a próxima recarga para voltar ao estado pronto para executar. Desta forma, pode-se considerar o servidor como uma

tarefa virtual, que reserva um percentual de utilização do processador, porém quem realmente o utiliza são as instâncias por ele servidas.

Observe o exemplo da Figura 2.2. Para solucionar o problema de perda de *deadlines* da Figura 2.1, um servidor foi utilizado para controlar o impacto da instância aperiódica. Como pode ser observado, a execução do servidor foi adiada até o instante 12. Note que a segunda instância de τ_4 (J_{11}) e a terceira instância de τ_3 (J_{10}) tiveram suas execuções adiadas para que o servidor executasse a instância aperiódica. Desta forma, nenhuma instância periódica perdeu seu *deadline*, além da instância aperiódica ter executado até o fim. Porém, note também que o servidor não considerou o *deadline* da instância aperiódica, que foi violado.

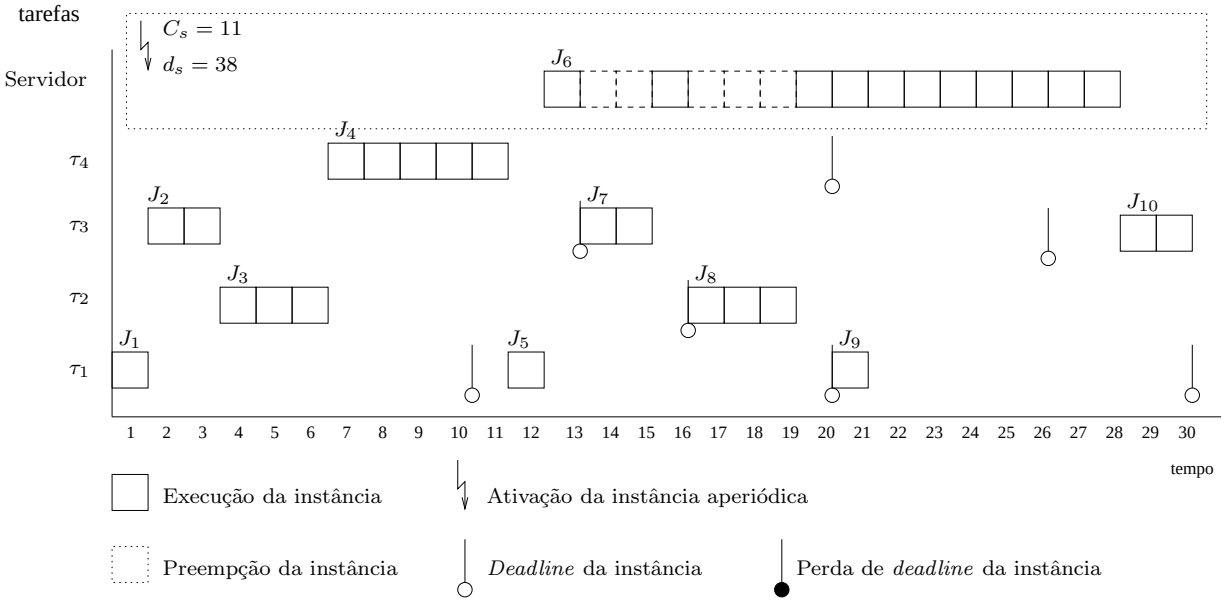


Figura 2.2. Escalonamento das tarefas, τ_1, τ_2, τ_3, τ_4 e τ_5 segundo o EDF usando servidor

Como este trabalho se concentra em escalonamento dinâmico, abordaremos apenas servidores baseados em EDF. Tais servidores podem ser divididos em duas classes, servidores com e sem preservação de tamanho de banda. O servidor com preservação de tamanho de banda, como o nome indica, é um servidor que preserva sua *carga*, enquanto o mesmo não esteja ocupando o processador, para que se possa, num tempo futuro, servir novas requisições de execução de instâncias.

Apenas os servidores com preservação de tamanho de banda serão descritos, pois os mesmos são os mais utilizados na prática. Todo servidor com preservação de tamanho de banda possui uma *carga*, C_s, um período T_s, e uma utilização de processador $u_s = \frac{C_s}{T_s}$. Classificamos os servidores com preservação de tamanho de banda em dois tipos, aqueles com *carga* fixa e os com *carga* variável.

2.3.1 Servidores com Carga Fixa

Os servidores com *carga* fixa se caracterizam pela existência de valores fixos, previamente estabelecidos, para o seu tempo máximo de execução ou *carga*, C_s, e para seu período, T_s, proporcionando assim uma utilização $u_s = \frac{C_s}{T_s}$.

O servidor *deferrable* dinâmico (DDS - *Dynamic Deferrable Server*) [31] é o tipo mais simples de servidor com preservação de tamanho de banda. Sua recarga é realizada periodicamente a cada intervalo T_s. Seu funcionamento pode ser descrito resumidamente da seguinte forma. Ao detectar que a fila de tarefas aperiódicas está vazia, o servidor não executa e mantém sua *carga* intacta. Ao chegar uma instância aperiódica na fila, o DDS é escalonado no sistema e, quando for mais prioritário e possuir carga positiva, o servidor habilita para execução a instância que encontra-se no topo de sua fila de prontos. Esta instância é executada, enquanto o servidor for a instância mais prioritária, até a *carga* do servidor esgotar ou até terminar sua execução. A recarga do servidor é realizada a cada instante kT_s, $k = 0, 1, \ldots$, e $d_{s,k}$ é o *deadline* da *k-ésima* instância do servidor.

Como ilustração do funcionamento do DDS, considere a Figura 2.3, onde τ_1, τ_2 e τ_3 são tarefas periódicas. Os atributos destas tarefas são $C_1 = 2$ e $D_1 = T_1 = 10$, $C_2 = 4$ e $D_2 = T_2 = 12$, $C_3 = 2$ e $D_3 = T_3 = 8$. O DDS tem $C_s = 2$ e $T_s = 10$. O servidor DDS servirá as tarefas aperiódicas. No momento 8, quando uma tarefa aperiódica ativa sua instância J_4, com tempo máximo de execução $C_4 = 2$, o DDS é escalonado junto com as tarefas periódicas do sistema, pois passa de estado ocioso para ativo, com *deadline* absoluto $d_{s,0} = 8 + 10 = 18$ e com carga de 2 unidades de

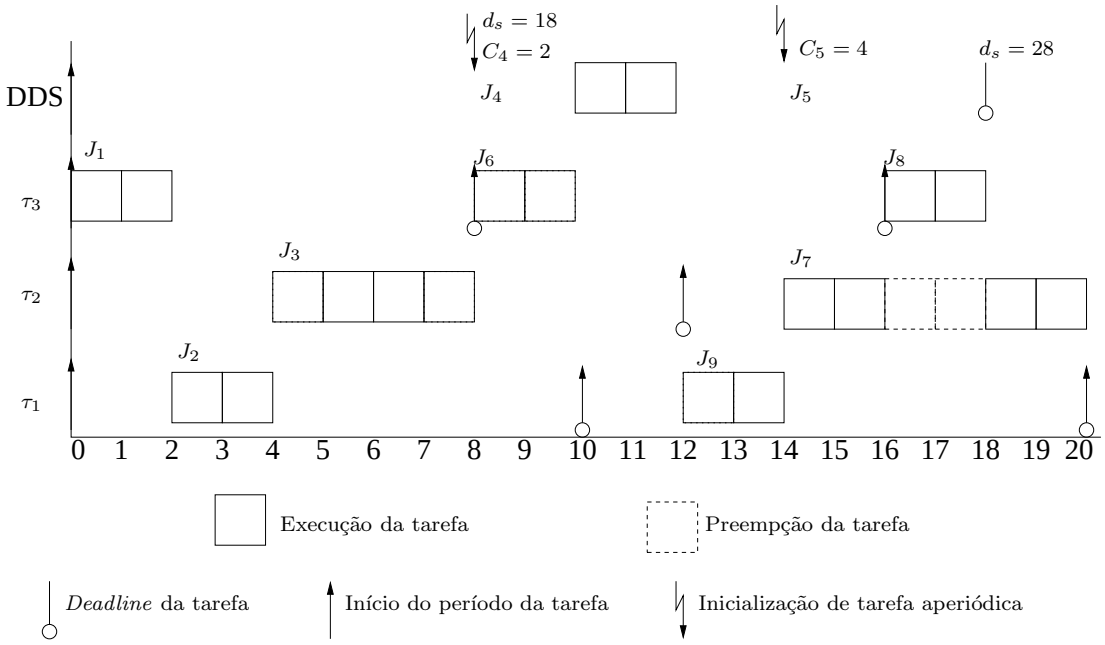


Figura 2.3. Escalonamento das tarefas τ_1, τ_2 e τ_3 segundo o EDF, com DDS para executar as tarefas aperiódicas

tempo. Quando sua prioridade é máxima, no intante 10, o mesmo executa durante 2 unidades de tempo. No instante 14, outra instância de tarefa aperiódica é ativada, J_5. Esta espera até o instante 18, pois a carga do DDS já terminou e somente será recarregado no instante 18, quando o seu novo *deadline* é calculado. Como o servidor tem carga apenas de 2 unidades, e a instância da tarefa τ_1 consome esta carga, restam ainda 2 unidades de tempo para terminar sua execução. Então o servidor é colocado em estado ocioso e a tarefa aperiódica esperará até o próximo período de recarga do servidor para que possa finalmente terminar sua execução.

Há alguns problemas relacionados ao modo como a recarga do DDS é realizada, pois duas ativações consecutivas do mesmo podem ocorrer, prejudicando a escalonabilidade do sistema [45]. Uma alternativa seria utilizar o servidor *Sporadic Server* [45], porém este tem um alto custo de implementação. Outro servidor, como o de tamanho de banda constante (CBS - *Constant Bandwidth Server*) [2], evita tais problemas e é mais flexível. O CBS é um servidor que garante que se u_s é uma fração do tempo do processador designada para o servidor, sua contribuição para a total utilização não é

maior do que u_s, mesmo na presença de sobrecarga no sistema.

O funcionamento do CBS pode ser visto na Figura 2.4, onde as mesmas tarefas periódicas do exemplo anterior foram utilizadas, com $C_s = 2$ e $T_s = 10$. No instante 7, quando uma instância de tarefa aperiódica J_4, com tempo máximo de execução $C_4 = 4$, é ativada no sistema, o CBS passa de estado ocioso para ativo e calcula um novo *deadline*. $d_{s,0} = t+T_s = 7+10 = 17$. No instante 10, o CBS passa a ser a instância mais prioritária e começa a executar J_9, porém no instante 12 sua *carga* atinge zero. Neste momento, seu *deadline* é recalculado para $d_{s,1} = d_{s,0} + T_s = 17 + 10 = 27$. Logo, o CBS deixa de ser a instância mais prioritária e sofre preempção, cedendo o processador para a execução da instância periódica mais prioritária. O servidor só voltará a executar as instâncias aperiódicas de sua fila quando ele tornar a ser a instância mais prioritária do sistema.

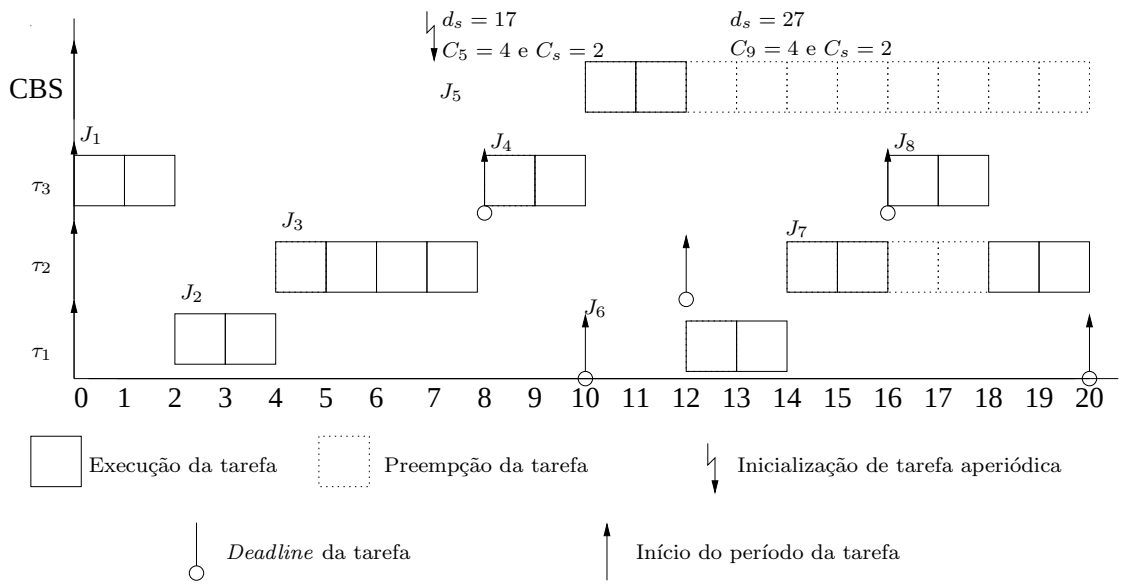


Figura 2.4. Escalonamento das tarefas τ_1, τ_2 e τ_3 segundo o EDF, com CBS para executar as tarefas aperiódicas

O CBS tem se mostrado bastante versátil e motivado outros trabalhos. Por exemplo, mecanismos de reaproveitamento de folgas entre servidores CBS foram propostos [41, 19, 40, 3] e aspectos de compartilhamento de recursos foram explorados [20, 48]. Recentemente, técnicas para calcular os parâmetros C_s e T_s foram consideradas [49]

2.3.2 Servidores com Carga Variável

Os servidores com *carga* variável se caracterizam por seu tempo máximo de execução, ou *carga*, C_s, depender do tempo máximo de execução da instância aperiódica que encontra-se no topo da fila de prontos. A idéia básica é fixar a utilização do processador u_s e calcular o *deadline* do servidor em função da carga necessária para executar a instância aperiódica a ser servida.

Dois servidores com carga variável são bastante referenciados na literatura, o TBS [58] e o CUS [27]. Como eles são muito similares, ilustraremos o funcionamento apenas do TBS, pois este, em geral, apresenta melhor desempenho.

O servidor com preservação total de tamanho de banda (TBS - *Total Bandwidth Server*) [58] é um servidor caracterizado pela sua utilização u_s, que representa a fração do processador reservada ao servidor. u_s é dado por

$$u_s = 1 - \sum_{\forall \tau_j \in \Gamma_p} \frac{C_j}{T_j}, \tag{2.1}$$

onde $\Gamma_p \subset \Gamma$ é o conjunto de tarefas periódicas.

No momento em que a *carga* é esgotada, caso o servidor esteja ativo, ele é imediatamente recarregado. Caso contrário, a *carga* é realizada assim que o servidor atingir o estado ativo. O *deadline* do servidor, para cada instância aperiódica servida, J_i, é calculado da seguinte forma:

$$d_{s,k} = \max(d_{s,k-1}, t) + \frac{C_i}{u_s}, \tag{2.2}$$

onde t é o instante corrente.

O funcionamento do TBS é ilustrado na Figura 2.5. Consideremos as tarefas periódicas τ_1, τ_2 e τ_3 usadas nos exemplos anteriores e $u_s = 0,2$. O sistema é escalonável, pois $u_p + u_s = 0,78 + 0,2 \leq 1$. Quando uma instância de tarefa aperiódica J_4 é ativada, no instante 1, com tempo máximo de execução, C_4, igual a 2, o *deadline*

do TBS é calculado, $d_{s,0} = t + (\frac{C_4}{u_s}) = 1 + (\frac{2}{0,2}) = 11$. O TBS passa de estado ocioso para ativo e é escalonado junto com as tarefas periódicas do sistema. No instante 4, o TBS tem mais alta prioridade e executa duas unidades de tempo, concluindo a execução da instância J_4. A instância de tarefa aperiódica J_9 é ativada no instante 7, com $C_9 = 4$ e é colocada na fila de prontos do servidor. Como a fila encontra-se vazia, o *deadline* do servidor é recalculado, $d_{s,1} = \max(t, d_{s,0}) + (\frac{C_9}{u_s}) = \max(7, 11) + (\frac{4}{0,2}) = 31$. O TBS passa a ter baixa prioridade e as instâncias periódicas executam.

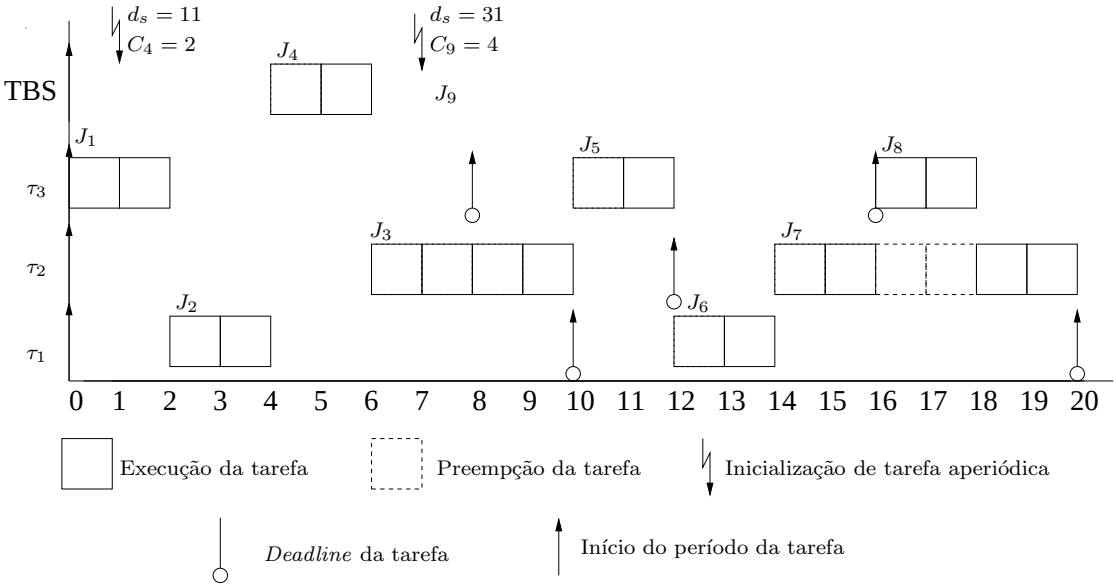


Figura 2.5. Escalonamento das tarefas τ_1, τ_2 e τ_3 segundo o EDF, com TBS para executar as tarefas aperiódicas

Da mesma forma que o CBS, extensões do TBS foram propostas para tratar compartilhamento de recursos [22, 21].

2.4 CANCELAMENTO OU REJEIÇÃO DE TAREFAS

Reconsiderando o exemplo da Figura 2.1, como foi mencionado anteriormente, proteger-se-ia o sistema dos efeitos nocivos da sobrecarga se pudéssemos cancelar a execução de alguma instância periódica ou não aceitar a instância aperiódica. Em ambos os casos, é necessário equipar o sistema com um teste de escalonabilidade em tempo

de execução. Caso haja risco de perda de *deadline*, toma-se a ação mais apropriada.

Em sistemas cuja carga periódica é conhecida, pode-se realizar testes de escalonabilidade, em tempo de projeto, para as instâncias periódicas. Neste momento, pode-se conhecer as folgas do sistema. Folga é o espaço em que o sistema se encontra ocioso entre o momento que uma tarefa termina de executar e o seu *deadline*. Se for possível utilizar o conhecimento sobre estas folgas em tempo de execução, pode-se realizar um teste de escalonabilidade em tempo de execução para avaliar a inserção das instâncias aperiódicas, rejeitando-as caso haja risco de perdas de *deadlines*.

Diversos aspectos devem ser considerados no instante em que a instância aperiódica é ativada, como a criticidade das instâncias aperiódicas ativadas e a criticidade das instâncias periódicas e aperiódicas já ativas no sistema. Tais avaliações, por muitas vezes, são muito custosas para serem realizadas em tempo de execução. Sabe-se que para ser realizado em tempo de execução, o teste precisa ser eficiente.

A técnica de execução por roubo de folga (*Slack Stealing*) [38], por exemplo, usa um algoritmo para calcular as folgas das tarefas escalonadas em tempo de projeto. Após avaliar a folga do sistema, se esta for positiva, obtemos uma quantidade de tempo total em que algumas tarefas periódicas podem ser adiadas. Este tempo é utilizado pela instância aperiódica no momento de sua ativação, causando um adiamento da execução das demais instâncias. Caso a folga seja nula, quando a instância aperiódica for ativada, ela deverá aguardar até que alguma folga seja detectada ou pode ainda ser rejeitada [24]. Esta abordagem é extremamente custosa, devido à complexidade do algoritmo de cálculo de folgas [45]. Como o custo do armazenamento da informação sobre as folgas é alto, implementar testes de escalonabilidade em tempo de execução usando tal abordagem pode não ser apropriado em sistemas complexos.

Um teste de escalonabilidade exato foi proposto em [4] para sistemas em que existem tarefas aperiódicas e periódicas. Esta proposta é baseada num escalonamento EDF. Algumas hipóteses são assumidas, tais como tarefas periódicas com períodos iguais aos *deadlines* e instante de ativação de todas as tarefas periódicas em $t = 0$. Para diminuir o custo computacional do teste realizado, os autores consideram a posição

da instância que será inserida na fila de tarefas a executar, calculando a possibilidade de conclusão com sucesso das instâncias menos prioritárias. Se ao inserir a instância aperiódica, alguma das instâncias menos prioritárias tiver possibilidades de perda de *deadline*, a instância será rejeitada. Caso contrário, ela é admitida no sistema.

Realizar testes de escalonabilidade em tempo de execução em sistemas de tempo-real não-críticos é necessário para evitar efeitos indesejados, como o efeito "dominó" citado na Seção 2.2. Porém, alguns sistemas de tempo-real não-críticos necessitam que ao menos parte de sua computação seja garantida, como por exemplo, sistemas multimídia, que podem perder *deadlines* de algumas instâncias. Porém, para se garantir a correta execução de um vídeo, por exemplo, deve-se terminar com sucesso algumas instâncias de tarefas de exibição de vídeo. Sistemas que admitem perdas de *deadlines*, mas exigem uma qualidade mínima de funcionamento podem ser denominados de sistemas de tempo-real fracamente críticos [8, 29]. Para estes, o teste de escalonabilidade não deve apenas rejeitar instâncias, deve avaliar quais as instâncias de tarefas que devem ter seus *deadlines* garantidos.

Alguns sistemas de tempo-real fracamente críticos são sensíveis a perdas consecutivas de *deadline*. Outros são sensíveis a muitas perdas de *deadline*, sendo estas consecutivas ou não. Visando ambos tipos de sistema separadamente, foram desenvolvidos algoritmos que têm por objetivo garantir certa quantidade de *deadlines* cumpridos, podendo estes serem consecutivos ou não [8]. Para tanto, os autores propõem três tipos de testes de escalonamento. O primeiro garante a conclusão com sucesso de algumas instâncias. O segundo é o algoritmo de melhor esforço, que tem por objetivo minimizar a quantidade de instâncias que perdem seus *deadlines* ao longo da execução do sistema. O terceiro é baseado em um teste de escalonabilidade que rejeita as instâncias caso, após sua inserção, haja riscos de perdas de *deadlines*. Este terceiro algoritmo pode ter dois comportamentos. Ele pode rejeitar algumas instâncias para garantir o cumprimento do *deadline* de outras, ou maximizar a quantidade de instâncias que cumprirão seus *deadlines* com sucesso.

Outros algoritmos foram desenvolvidos para evitar perdas de *deadlines* em siste-

mas de tempo-real fracamente críticos. O algoritmo *(m,k) firm* [29] garante que uma tarefa com restrição *(m,k)* perderá no máximo *m deadlines* em um intervalo de *k* invocações sucessivas. Tal algoritmo foi utilizado para escalonar mensagens em sistemas de comunicação, onde o algoritmo proposto garante que mensagens enviadas *k* vezes sucessivas seriam enviadas com sucesso *(k - m)* vezes.

Outros testes de escalonabilidade em tempo de execução, como o *Robust Earliest Deadline* (RED) [16], realizam testes de escalonabilidade baseados nos tempos residuais das instâncias e nos seus respectivos fatores de importância. O tempo residual da instância é o tempo entre o instante de término da instância e seu *deadline* absoluto. Instâncias que possuem tempos residuais negativos poderão perder seus *deadlines*. O fator de importância indica a criticidade da instância, ou seja, o quanto o término com sucesso desta representa ao sistema. Desta forma, pode-se avaliar o impacto que a inclusão da instância aperiódica causará no sistema, através do cálculo do tempo residual de cada uma das instâncias periódicas e aperiódicas ativas em seu instante de ativação. Ao constatar a possibilidade de perda de *deadlines*, o RED insere a instância aperiódica numa fila de instâncias rejeitadas, podendo esta vir a ser executada posteriormente, caso o sistema apresente resíduos de suas instâncias ativas suficientes para executar a instância do topo da fila, seguindo uma abordagem de melhor esforço.

Os testes de escalonabilidade, citados nesta seção, visam impedir ou amenizar os efeitos causados por situações de sobrecarga. O objetivo destes testes, em geral, baseia-se na possibilidade de não executar determinadas instâncias em situação de sobrecarga. Porém, sistemas de tempo-real críticos nem sempre podem cancelar/rejeitar suas instâncias. Na próxima seção, serão discutidas algumas outras alternativas que partem do princípio de que o sistema pode ser reconfigurado.

2.5 RECONFIGURAÇÃO DINÂMICA

Como visto na seção anterior, pode ser interessante cancelar/rejeitar instâncias, desde que isto possa ser feito de forma controlada. Outra forma de proteger e adaptar

o sistema é através da alteração das características de suas instâncias em tempo de execução, ou seja, reconfiguração dinâmica, cujo objetivo é modificar o sistema em execução para que este continue provendo seus serviços, possivelmente, com um nível de qualidade inferior, evitando perdas de *deadlines*, em períodos de sobrecarga. A idéia básica é degradar algumas instâncias de tarefas do sistema para garantir a execução com sucesso de todo o conjunto de instâncias em períodos de sobrecarga. Esta reconfiguração pode ser realizada de diversas formas. Como a utilização do processador de cada tarefa periódica τ_i no sistema é dada por $\frac{C_i}{T_i}$, reconfigurá-lo significaria alterar a utilização do processador através de um dos parâmetros envolvidos, o custo computacional, o período, ou ambos. A escolha do método de reconfiguração e de quais tarefas alterar é muito importante para o desempenho do sistema. Por exemplo, a decisão de quais tarefas alterar tem impacto direto na qualidade do sistema. Assim, deve-se avaliar as possíveis reconfigurações do sistema de modo a escolher aquela que menos interfira na qualidade do serviço por ele prestado.

Alguns sistemas de tempo-real possuem múltiplas versões distintas de suas tarefas, as quais representam diferentes níveis de qualidade do serviço prestado. Tais versões variam, além do tipo da qualidade do serviço prestado, o tempo computacional gasto e/ou o período das tarefas. Para prover reconfiguração dinâmica, deve-se escolher adequadamente as versões das tarefas que executarão. Normalmente, as versões das tarefas são modeladas de forma que aquelas menos custosas proporcionem um serviço de menor qualidade. As versões de uma mesma tarefa podem realizar ações totalmente diferentes, como também, podem possuir uma versão obrigatória, com menor qualidade, e ações adicionais, que melhorem a qualidade do serviço e aumentem o tempo computacional da tarefa.

Nesta seção serão apresentadas duas abordagens de reconfiguração dinâmica em nível de escalonamento. A primeira é a reconfiguração através do custo computacional para a qual as tarefas do sistema são estruturadas com múltiplas versões, cada uma das quais com diferentes custos computacionais relativos a distintos níveis de qualidade de serviço. A segunda realiza reconfiguração através do período. Neste caso, a qualidade

do serviço está associada à frequência de execução das tarefas.

2.5.1 Reconfiguração Considerando o Custo Computacional

Suponha que uma instância da tarefa aperiódica é ativada e causa sobrecarga no sistema. Suponha ainda que cada tarefa periódica possui versões alternativas de execução, cada uma das quais fornece determinado nível de qualidade de serviço. Pode-se, neste caso, avaliar se é possível selecionar versões das tarefas que executarão no intervalo de sobrecarga, degradando a qualidade do serviço, porém satisfazendo as condições de escalonabilidade. A escolha de quais versões das tarefas executarão pode levar em conta diversos aspectos de acordo com as particularidades do sistema em questão.

A idéia básica desta abordagem é ilustrada na Figura 2.6. Este exemplo é o mesmo mostrado na Figura 2.1, onde as instâncias periódicas perdiam seus *deadlines* com a inserção da instância J_3 da tarefa aperiódica τ_5. Na Figura 2.6, uma degradação controlada foi realizada, substituindo as versões das instâncias das tarefas τ_2 e τ_4 por versões diferentes, com custos computacionais inferiores, $C_4 = 2$ e $C_5 = 3$, respectivamente. Tal modificação possibilitou a execução com sucesso de todas as instâncias de tarefas do sistema, inclusive a instância J_3 da tarefa aperiódica τ_5.

A alteração do custo computacional de uma instância pode ser modelada de diversas formas. Um modelo de sistema que possibilita uma reconfiguração dinâmica através do custo computacional é baseado em computação imprecisa [26, 28, 10, 59]. Na computação imprecisa, toda tarefa tem seu tempo computacional dividido em duas partes, uma obrigatória e outra opcional. A parte obrigatória engloba tudo que é essencial e não deve deixar de ser realizada pela tarefa. A parte opcional aumenta a qualidade dos resultados produzidos pelas tarefas. Desta forma, em casos de sobrecarga, deve-se optar por executar apenas a parte obrigatória de algumas, ou ainda de todas, tarefas do sistema. Esta forma de organizar as tarefas é utilizada com frequência em sistemas de comunicação [46], sistemas de controle de congestionamento [47], tolerância a falhas em sistemas de tempo-real [44], entre outras aplicações.

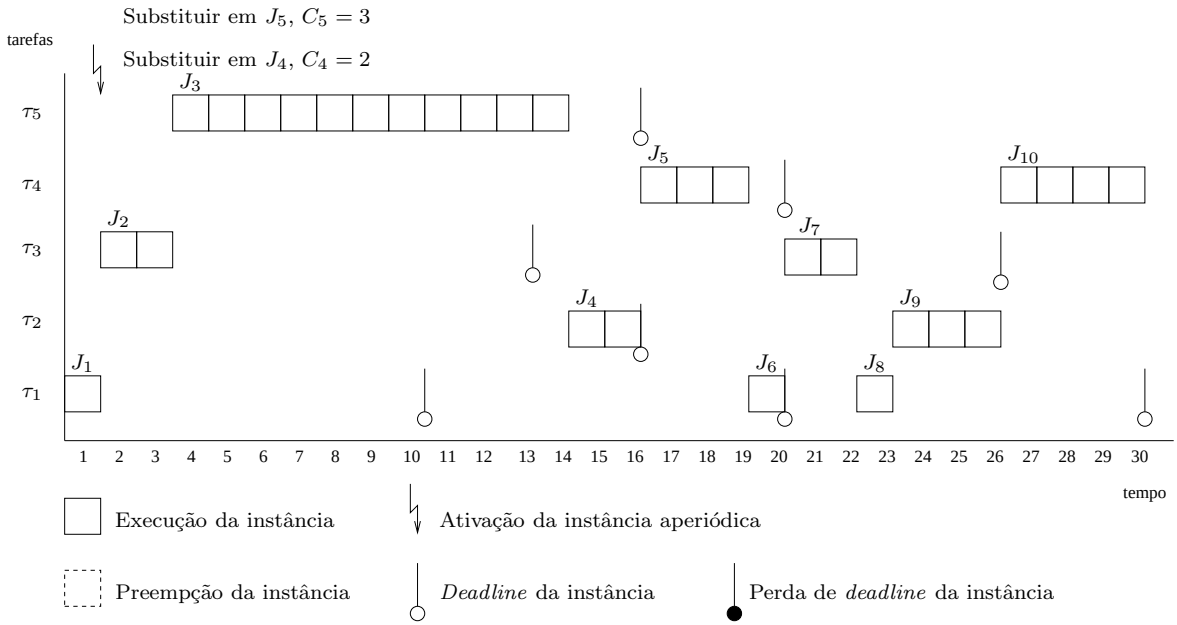


Figura 2.6. Escalonamento das tarefas, τ_1, τ_2, τ_3, τ_4 e τ_5 segundo o EDF usando reconfiguração de custo computacional

Diversas propostas para reconfiguração dinâmica usando tarefas com múltiplas versões foram feitas. Normalmente, usa-se a proposta de múltiplas versões de tarefas para prover tolerância a falhas, onde algumas versões de tarefas podem ser diferentes rotinas de recuperação [39, 11]. Este modelo de escalonamento tem sido usado em sistemas de satélite [54] e em escalonamentos baseados em energia [52]. A solução proposta por Jehuda and Israeli [32] também usa tarefas com múltiplas versões, considerando o fato que o problema de reconfiguração pode ser modelado como um clássico problema da mochila (*knapsack*) [25], onde a utilização do processador representa o tamanho da mochila.

O foco deste trabalho é prover reconfiguração em nível de escalonamento. Assim, deve-se garantir prazos temporais e obter um desempenho computacional otimizado. Outras propostas para adaptação e reconfiguração podem ser encontradas [1, 30, 51]. Entretanto, a meta destas propostas é prover um *middleware* através do qual as aplicações podem negociar recursos computacionais disponíveis. Tais propostas são inviáveis para serem utilizadas em nível de escalonamento.

Rusu *et al.* [52] propuseram um mecanismo de reconfiguração para sistemas de tempo-real que trabalham com gerenciamento de energia. A meta proposta é otimizar o benefício do sistema observando duas restrições, escalonabilidade e energia. Dois modelos de tarefas foram propostos, um baseado em *frames*, que são espaços de tempo de execuções, e outro baseado nos períodos das tarefas. No modelo de tarefas baseado em *frames*, para definir os tamanhos dos *frames*, determinou-se a restrição que todas as tarefas devem ter os mesmos períodos e *deadlines*, tal restrição pode ser inviável em sistemas de tempo-real complexos. Para este modelo, o intervalo de reconfiguração é o tamanho do *frame*. Já no modelo de tarefas baseado em períodos, o intervalo de reconfiguração é o hiperperíodo do sistema, o qual pode envolver muitas instâncias de tarefas, tornando-o demasiadamente grande e inviável para ser utilizado em nível de escalonamento.

2.5.2 Reconfiguração Considerando o Período

Diversos sistemas de tempo-real permitem que suas reconfigurações sejam baseadas, preferencialmente, na alteração dos períodos de suas tarefas. Tal reconfiguração pode ser utilizada, por exemplo, em sistemas de controle de vôo, onde a taxa de amostragem do sensor de altitude varia [17]. Quanto menor a altitude da aeronave, maior a freqüência de amostragem do sensor. Outro exemplo pode ser encontrado em sistemas de radar de sensoriamento de velocidade, no qual um histórico de velocidade é gravado numa tabela com a posição corrente, velocidade, placa do veículo, entre outras variáveis. As informações são atualizadas periodicamente, com alta taxa de atualização, para manter as informações confiáveis. Quanto mais próximo do radar, maior a freqüência de amostragem e atualização. Por outro lado, quanto mais próximo do radar, menor o esforço computacional para realizar cada atualização, pois o sensoriamento de dados proporciona menos ruído e maior precisão. Desta forma, o processamento de sinais pode diminuir a taxa de amostragem à medida que o veículo se aproxima [37].

A maioria das propostas desta área defende tal reconfiguração por julgarem ser

natural ao sistema [23, 17]. Por exemplo, um modelo flexível de tarefas, que contém uma faixa de períodos possíveis, $(T_{i_{min}}, T_{i_{max}})$, para cada tarefa, foi proposto em [7]. As instâncias, inicialmente, operam com o período mínimo, $T_{i_{min}}$, e, em caso de necessidade, este período pode ser estendido até o máximo permitido, $T_{i_{max}}$. Tais períodos foram definidos seguindo as restrições temporais e lógicas das tarefas.

Os autores Buttazzo e Abeni [15] consideraram um sistema escalonado pela política EDF associado ao uso de tarefas com coeficientes elásticos. Este modelo foi recentemente estendido para realizar a criação de um *framework* elástico, que controla as alterações dos períodos das tarefas e seus efeitos [23]. Diversos aspectos inerentes às alterações de períodos foram discutidos, como as restrições de *deadlines* das tarefas. Outro algoritmo, o D-Over [34], propôs alterar o período somando a este uma quantidade de tempo s, a qual a tarefa poderá dispor para adiar o início do próximo período. Cada tarefa pode possuir uma quantidade de tempo s diferente das demais.

Uma outra técnica se baseia no conceito de fator de adiamento (*skip factor*) da tarefa [35]. Uma tarefa que possui um fator de adiamento s poderá adiar sua ativação em s unidades de tempo, ou seja, alterar seu período. O algoritmo proposto, *D-over* [34], é um algoritmo de escalonamento baseado em prioridade dinâmica que decide qual instância irá executar através da análise dos fatores de adiamento das tarefas. Tal método consegue controlar a sobrecarga do sistema através do adiamento de algumas ativações de instâncias. Este algoritmo foi estendido para escalonar uma série híbrida de tarefas (críticas e não-críticas, periódicas e aperiódicas) [18].

No contexto do escalonamento *Rate Monotonic* (RM), Beccari *et al.* [6] propuseram uma série de heurísticas, as quais selecionam, em condições de sobrecarga, períodos diferentes de tarefas não-críticas, provendo uma maximização do benefício do sistema. Assume-se que existe um valor de benefício associado a cada versão de tarefa não-crítica. O trabalho de Kuo e Mok [36] também é baseado no RM. Para selecionar os períodos das tarefas, o mecanismo de reconfiguração requer que os períodos sejam harmonicamente relacionados uns com os outros. O modelo proposto restringe ainda os tempos de execução das tarefas, assumindo que estas são linearmente relacionadas.

Estas restrições podem tornar inviável o uso de tal solução em sistemas de tempo-real modernos.

A reconfiguração dinâmica através da alteração do período pode ser observada na Figura 2.7, que ilustra uma solução diferente da solução fornecida na seção anterior para o mesmo problema ilustrado na Figura 2.1. Neste caso, optou-se por realizar a reconfiguração alterando o período das tarefas, evitando as perdas de *deadlines* de instâncias periódicas devido à inserção da instância da tarefa aperiódica J_3. Observe que as tarefas τ_3 e τ_1 tiveram os períodos de suas instâncias alterados para 25 e 20, respectivamente. Tal alteração permitiu a execução da instância aperiódica sem causar perdas de *deadlines*.

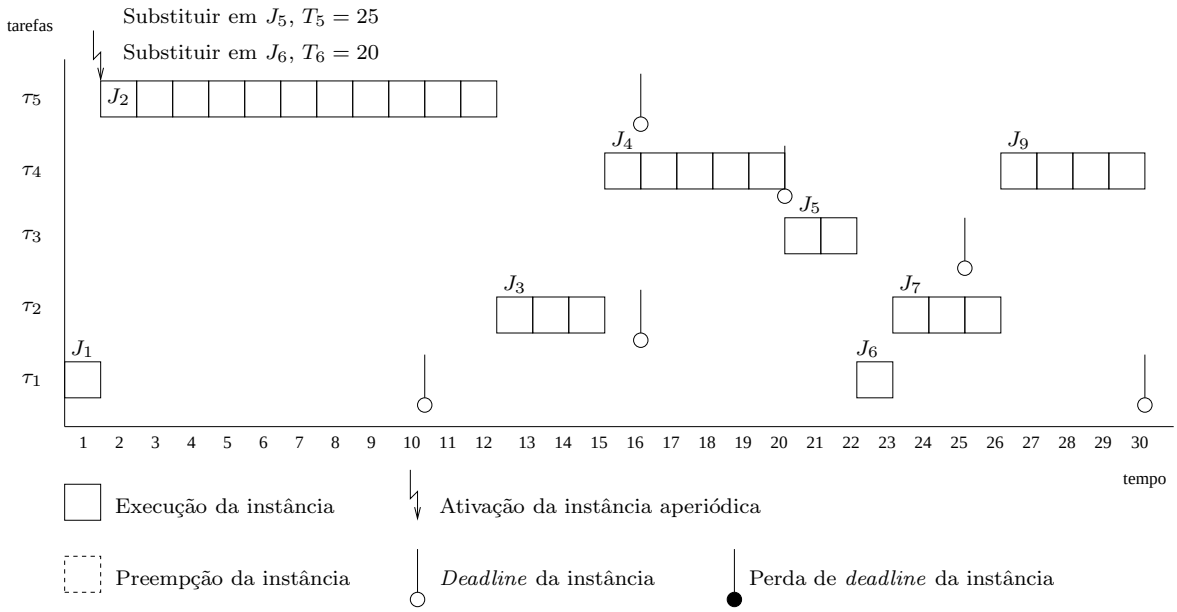


Figura 2.7. Escalonamento das tarefas, τ_1, τ_2, τ_3, τ_4 e τ_5 segundo o EDF usando reconfiguração de período

2.6 SUMÁRIO

Neste capítulo, foram descritas as principais abordagens de escalonamento para lidar com sobrecargas e adaptação em sistemas de tempo-real modernos. Estas foram divididas em três grupos principais: as baseadas em isolamento temporal das tarefas,

aquelas que fazem cancelamento ou rejeição das tarefas e, por fim, as que utilizam reconfiguração dinâmica do sistema. O primeiro grupo utiliza, em geral, servidores de tarefas aperiódicas para prover isolamento entre as tarefas, garantindo o cumprimento dos *deadlines* das instâncias periódicas, porém nenhuma consideração é feita em relação aos *deadlines* das instâncias aperiódicas. O segundo grupo utiliza testes de escalonabilidade para avaliar a possibilidade de perdas de *deadlines* quando uma instância é ativada. Nestes casos, rejeita-se, por exemplo, a instância que acaba de ser ativada. O terceiro grupo realiza a reconfiguração do sistema através da alteração do custo computacional ou do período de algumas instâncias, facilitando que todas as instâncias do sistema executem e cumpram seus *deadlines*.

Resolver o problema de reconfiguração dinâmica permite maior flexibilidade ao sistema quanto à sua adaptação. Para construir um mecanismo eficiente de reconfiguração dinâmica é necessário tratar dois problemas básicos. Deve-se equipar o sistema com um teste de escalonabilidade e um algoritmo de escolha da configuração mais adequados ao sistema naquele momento. Soluções para ambos os problemas devem ser eficientes para que possam ser utilizadas em tempo de execução. Os dois problemas supracitados serão tratados nos próximos capítulos.

CAPÍTULO 3

RECONFIGURAÇÃO DINÂMICA: UM ESTUDO PRELIMINAR

Como mencionado no capítulo anterior, mecanismos de reconfiguração dinâmica devem ser capazes de testar a escalonabilidade do sistema e selecionar a configuração mais adequada àquela situação. Apresentaremos alguns testes de escalonabilidade em tempo de execução. Todos têm o objetivo de verificar se há riscos de perdas de *deadlines*, examinando as instâncias ativas num dado intervalo de tempo. O primeiro teste, mais pessimista, assume que todas as instâncias terminam suas execuções num instante de tempo considerado. O segundo teste, utiliza o servidor de tarefas aperiódicas TBS, visto no capítulo anterior, para escalonar as instâncias aperiódicas, provendo degradação das instâncias periódicas, respeitando os *deadlines* das instâncias aperiódicas. O terceiro teste leva em consideração mais detalhes das tarefas periódicas, diminuindo o pessimismo do primeiro teste.

Como o foco do capítulo são os testes de escalonabilidade, o problema da escolha da melhor configuração, que envolve um problema combinatorial, é simplificado. Assim, reconfigurar o sistema, aqui, resume-se em encontrar uma configuração que satisfaça as condições de escalonabilidade do sistema. De qualquer forma, consideramos neste trabalho reconfiguração dinâmica assumindo que existem versões diferentes de uma mesma tarefa, com custos computacionais distintos. Os períodos não serão alterados.

Este capítulo é estruturado da seguinte forma. A seção 3.1 apresenta o modelo de tarefas e a notação utilizada. Na seção , a proposta preliminar de reconfiguração considerando instâncias contidas num dado intervalo é detalhada. Dois cenários são apresentados, simulados e avaliados. O primeiro considera a execução das tarefas simplesmente usando a política EDF e o segundo leva em consideração que o sistema possui o mecanismo TBS, subseção 3.2.1. O segundo, apresentado na subseção 3.2.2,

faz uma proposta preliminar de reconfiguração usando instâncias de tarefas, tal proposta foi simulada e comparada com a proposta da subseção 3.2.1. Finalmente, a seção 3.3 apresenta um sumário das considerações realizadas neste capítulo.

3.1 MODELO

Assumimos um sistema onde as tarefas periódicas possuem uma ou mais versões de execução. Assim, existe um conjunto não-vazio de versões distintas associadas a cada tarefa. Em outras palavras, a instância de tarefa J_i, $i = 1, \ldots, n$, pode ser ativada para executar uma de suas $k(i) > 0$ versões. Cada versão k de J_i, $0 \leq k \leq k(i)$, possui um custo máximo de execução conhecido, $C_{i,k}$. O instante de ativação da instância J_i é representado por r_i. Sem perda de generalidade, assumimos que $C_{i,1} \geq C_{i,2} \cdots \geq C_{i,k(i)} \geq 0$. Se J_n e J_j são instâncias de uma mesma tarefa, então $k(n) = k(j)$ e $C_{n,k} = C_{j,k}$, $k = 1, \cdots, \kappa(i)$. Assumimos que a execução de cada versão k de J_i fornece ao sistema um benefício, representado por $A_{i,k} \geq 0$. $C_{i,k} = 0$ indica que na versão k a tarefa J_i pode ser cancelada, produzindo um benefício nulo. O instante de interesse relativo a J_i é $s_i = \max(t, r_i)$. Em outras palavras, s_i é o instante a partir do qual o efeito da execução de J_i interferirá na escalonabilidade das instâncias ativas num dado intervalo. Este intervalo é considerado para efeito de reconfiguração. Assim, o mecanismo de reconfiguração deverá agir dentro deste intervalo, degradando tarefas, caso necessário, e montando a escalonabilidade do sistema.

Dado que cada instância J_i de tarefa periódica pode executar numa de suas $k(i)$ versões, existe um número exponencial de configurações possíveis ao sistema. Obviamente, escolher uma configuração apropriada, em tempo de execução, pode ser computacionalmente caro. Assim, para tratar o problema, assumimos inicialmente a seguinte simplificação, que será modificada em capítulos posteriores. Considere que o sistema executa as instâncias das suas tarefas periódicas com qualidade máxima. Se for necessário reduzir a qualidade de serviço destas tarefas para admitir uma tarefa aperiódica, todas as instâncias de todas as tarefas que sofrerão degradação executarão no mesmo nível de degradação no intervalo considerado. Ao término do intervalo de

reconfiguração, o sistema retorna para o nível zero de degradação, ou seja, torna a servir com qualidade máxima.

Seja t o instante de início do intervalo de reconfiguração e t' o instante de término, definimos $\Gamma(t, t')$ como o conjunto de tarefas ativas no intervalo de reconfiguração $[t, t')$. É importante notar que caso o sistema seja escalonável em $[t, t')$, todas as instâncias ativadas a partir de t' não devem ser afetadas pelas instâncias contidas em $\Gamma(t, t')$. Neste capítulo, assumimos que t é dado pela ativação de uma tarefa aperiódica e corresponde ao instante no qual o procedimento de reconfiguração é acionado. Neste caso, estamos assumindo que $|\Gamma(t, t)| \neq \emptyset$. O limite superior do intervalo de reconfiguração t', por outro lado, pode ser determinado por: Seja d o *deadline* da tarefa aperiódica ou esporádica, $D = \max_{\forall J_i \in \Gamma(t,t): d_i > d}(d_i)$ e $r = \min_{\forall J_i \in \Gamma(t,d): r_i > d}(r_i)$. Então, define-se $t' = \min(r, D)$. Esta proposta considera o número de instâncias de tarefas contidas no intervalo e isola os efeitos da execução das instâncias dentro de $[t, t')$.

Nem todas as instâncias das tarefas periódicas podem sofrer degradação. Assumimos que as instâncias que já iniciaram suas execuções em t não são passíveis de serem degradadas. Esta restrição evita tratar problemas de possíveis inconsistências causadas por interrupções da execução de tais instâncias. Assumimos que o sistema contabiliza o tempo de processador executado por cada instância ativa J_i até um instante t. Note que tal mecanismo de contabilização é usual na maioria dos sistemas operacionais. Definimos então o *tempo máximo de execução efetivo* relativo ao instante t:

Definição 1 *Seja $c_{i,k}(t)$ o tempo já executado da versão k de J_i até o instante t, com $0 \leq c_{i,k}(t) \leq C_{i,k}$. O tempo máximo a ser executado pela versão k de uma instância J_i no intervalo $[t, t')$ é dado por:*

$$C_{i,k}(t) = \begin{cases} C_{i,k} - c_{i,k}(t), & \textit{se } J_i \in \Gamma(t, t) \\ 0, & \textit{caso contrário} \end{cases} \tag{3.1}$$

Note que se a versão k de J_i finalizou sua execução antes de t, $C_{i,k} = 0$. Assumimos

aqui uma representação discreta de tempo, ou seja, qualquer valor temporal é dado em unidades inteiras não negativas. O tempo de chegada de cada instância de tarefa aperiódica J_j é desconhecido. Seus custos de execução no pior caso, C_j, são conhecidos apenas no instante de sua ativação.

O escalonador segue a política EDF *(Earliest Deadline First)* [42] para escalonar as instâncias das tarefas. Assumimos que o sistema é escalonável em condições normais de operação, assim, o conjunto $\Gamma(t, t')$ é escalonável no nível de degradação $j = 0$. Entretanto, pode haver sobrecarga e, neste caso, cabe ao teste de escalonabilidade determinar a ação a ser realizada para que o sistema volte a operar em subcarga. Assumimos que sobrecargas podem ser causadas, por exemplo, por ativações de tarefas aperiódicas que correspondem a eventos que *devem* ser tratados *para manter a consistência* do sistema. Note que esta característica impede o uso de mecanismos que assumem que tarefas aperiódicas são não-críticas, como servidores [45, 14].

3.2 O PROBLEMA DE RECONFIGURAÇÃO

Esta seção descreve o problema de reconfiguração dinâmica mais precisamente. Como já mencionado, o procedimento de reconfiguração deve ser executado sempre que há risco de perdas de *deadlines*. Portanto, é necessário estabelecer as condições de escalonabilidade do sistema. Obviamente, reconfigurar o sistema dinamicamente pode exigir um alto custo computacional. Um dos aspectos complicadores reside na necessidade de se computar os efeitos da execução de uma instância sobre as demais. Tal cálculo pode se estender por um período demasiadamente longo. Precisa-se determinar a escalonabilidade do sistema e a escolha das versões a serem executadas no intervalo de reconfiguração $[t, t')$.

3.2.1 Reconfiguração de Tarefas

3.2.1.1 Idéia Básica

Nesta seção, apresentaremos duas abordagens de reconfiguração dinâmica, ambas com seus testes de escalonabilidade e degradação de tarefas. A idéia geral destas abordagens está descrita no Algoritmo 7. Este algoritmo deve ser executado no instante t.

Busca do nível de degradação apropriado, k

1: $k = 1$
2: **while** $(k \in K_i) \wedge \ P(k)$ **do**
3: $\quad k \leftarrow k + 1$
4: **end while**
5: **if** $k \notin K_i$ **then**
6: $\quad k = -1$
7: **end if**

O algoritmo busca um nível de degradação k que seja apropriado à execução das tarefas aperiódicas. O teste de escalonabilidade P, definido na equação (3.2), é usado para verificar se o sistema permanece escalonável com o nível de degradação k. Note que o conjunto $\Gamma(t, t')$´organiza as instâncias em ordem crescente de *deadlines*. Caso $k \in K_i$ não seja encontrado para algum $J_i \in \Gamma(t, t')$, rejeita-se a tarefa aperiódica, o que é indicado fazendo-se $k < 0$. É importante enfatizar que a rejeição de tarefas aperiódicas não é desejável. Por isso, usamos a rejeição como uma das métricas de comparação de diferentes estratégias de escalonamento (ver seção 3.2.1.4).

$$P(k) : \sum_{\forall \tau_i \in \Gamma(t,t')} C_{i,k} < (t' - t) \tag{3.2}$$

3.2.1.2 Reconfiguração Usando o TBS

Assume-se que uma reconfiguração é iniciada devido à demanda da execução de uma tarefa aperiódica. O modelo de escalonamento considerado nesta seção baseia-se no conceito do servidor de tarefas aperiódicas TBS [58]. Durante o projeto do sistema, assumimos que um percentual máximo para execução de tarefas aperiódicas, u_s, é destinado ao TBS. Em geral, se as tarefas periódicas demandam u_p $(0 \leq u_p < 1)$ do processador, pode-se reservar $u_s = 1 - u_p$ para executar as tarefas aperiódicas e, desta forma, pode-se obter o uso máximo de processador sem risco de perdas de *deadlines*.

Para efeito de degradação de tarefas periódicas é necessário diferenciar aquelas passíveis de serem degradadas e aquelas que não podem ser consideradas, pois já iniciaram suas execuções antes de t. Para tanto, precisa-se, inicialmente, determinar o percentual de u_s disponível para executar a tarefa aperiódica ativada no instante t. Este valor é determinado pela adição de u_s com o valor adquirido pela degradação das instâncias periódicas. Em outras palavras,

$$u_s(t) = u_s + \sum_{\forall \tau_i \in \Gamma(t,t')} \frac{C_{i,1}(t) - C_{i,k}(t)}{T_i} \,. \tag{3.3}$$

A equação (3.4) para o cálculo de *deadline* do TBS não considera o *deadline* da instância aperiódica d_a, o que é necessário no contexto deste trabalho.

$$d_{s,j} = \max(t, d_{s,j-1}) + \left(\frac{C_a}{u_s(t)}\right) \,, \tag{3.4}$$

sendo C_a o tempo computacional da instância aperiódica.

Usando as equações 3.4 e 3.3, pode-se determinar o seguinte teste de escalonabilidade

$$P(k) \equiv d_a \geq \max(t, d_{s,j-1}) + \frac{C_a}{u_s(t)} \,, \tag{3.5}$$

onde d_i é o *deadline* da instância aperiódica no topo da fila do servidor TBS no instante t.

Como ilustração, considere a Figura 3.1, que representa a execução de um sistema com duas tarefas periódicas τ_1 e τ_2, com versões $C_{1,1} = 2$, $C_{1,2} = 1$, $C_{2,1} = 5$ e $C_{2,2} = 3$. Os períodos destas tarefas são iguais aos seus *deadlines* relativos, que valem $D_{1,1} = 7$ e $D_{2,1} = 9$, respectivamente. Considere que $r_1 = 0$ e $r_2 = 0$. No instante 1, uma instância da tarefa aperiódica τ_3 é ativada no sistema com $C_3 = 1,8$ e $d_3 = 6,8$. A tarefa τ_3 não pode ser admitida no sistema com as tarefas periódicas em nível de degradação $k = 1$, pois $u_s = 1 - 0,84 = 0,16$ e $d_{s,1} = 1 + \frac{1,8}{0,16} = 11,25$. Assim, o teste T não é satisfeito, pois $d_{s,1} > d_3$. Como no instante $t = 1$ a tarefa τ_1 está em execução, a única tarefa ativa no intervalo passível de degradação é τ_2. Ao degradar τ_2, o valor de $u_s(1) = 0,16 + \frac{(5-3)}{9} \approx 0,38$. Com esta configuração, $d_{s,1} = 1 + \frac{1,8}{0,38} \approx 5,74 < d_3$ e, portanto, τ_3 pode ser executada sem risco de perdas de *deadlines*, como ilustra a Figura 3.1.

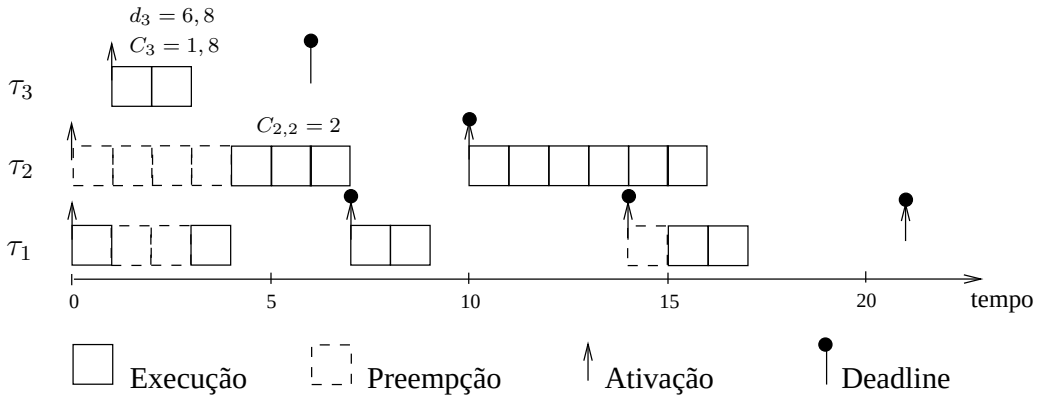


Figura 3.1. Admissão de τ_3 assumindo degradação controlada.

Como instâncias aperiódicas não são admitidas no sistema caso haja risco de violação de *deadlines*, fica claro que se o sistema é escalonável antes da ativação da instância aperiódica, ele continuará sendo após a sua admissão. É importante observar que o escalonamento de instâncias aperiódicas baseado no TBS considera tais instâncias uma por vez. Considerando o exemplo da Figura 2.5, nenhuma instância aperiódica pode ser considerada antes do instante $t = 2,8$. Desta forma, uma instância aperiódica urgente pode ter que esperar demasiadamente pela execução de uma instância aperiódica admitida anteriormente. A abordagem desenvolvida na próxima sub-seção

trata deste problema.

3.2.1.3 Reconfiguração Usando Apenas EDF

O teste de escalonabilidade em tempo de execução derivado nesta seção baseia-se no cálculo da demanda por processador no intervalo entre a ativação de uma tarefa aperiódica e o seu *deadline*. Se, de acordo com tal demanda, não há risco de violar *deadlines*, a tarefa aperiódica é admitida. O modelo de escalonamento assumido é simplesmente o EDF.

Para compreender melhor a derivação do teste de escalonabilidade, suponha uma tarefa aperiódica, τ_i, ativada no instante t, com tempo máximo de execução C_i e *deadline* absoluto d_i. O objetivo é determinar se a instância de τ_i pode ser admitida no instante t sem causar perda de algum *deadline*, seja de instâncias periódicas ativas entre $[t, d_i)$, ou de outras instâncias aperiódicas já admitidas anteriormente. Considere, por enquanto, apenas a existência de instâncias periódicas (ver Figura 3.2). Inicialmente, é preciso saber quantas instâncias periódicas executarão no intervalo $[t, d_i)$. Posteriormente, determina-se qual a demanda de processador para executá-las.

Na Figura 3.2 podem ser observadas as ativações de instâncias de uma tarefa periódica τ_j ao longo da linha do tempo. Para determinar quantas destas instâncias executam no intervalo $[t, d_i)$, é preciso considerar a última ativação de τ_j antes de t, se esta ainda não terminou sua execução até o instante t. Definimos então $r_{j,z}$ como sendo o instante desta ativação, ou seja, $r_j = (z-1)T_j \leq t < zT_j$, onde $z > 0$ e $t \geq 0$ é um instante de tempo qualquer.

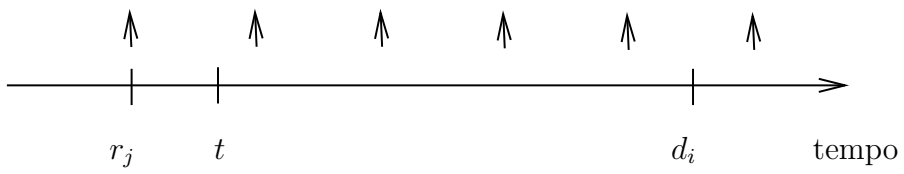


Figura 3.2. Número de instâncias de τ_j consideradas em $[t, d_i)$

Assim, o número máximo de instâncias de τ_j ativas no intervalo $[t, d_i)$ é dado por

$$\left\lceil \frac{d_i - r_j}{T_j} \right\rceil \tag{3.6}$$

Determinar a demanda por processador relativa às instâncias de τ_j requer a soma do custo computacional de cada uma delas. Todas as instâncias que já iniciaram suas execuções, sejam elas periódicas ou não, contribuirão para esta demanda com o tempo necessário para que elas terminem, $C_{j,k}(t)$, onde k é o nível de degradação da tarefa. Todas as instâncias de tarefas periódicas ativas no intervalo $[t, d_i)$ que ainda não iniciaram suas execuções demandarão $C_{j,k}$. Em resumo, a demanda por processador das tarefas periódicas no intervalo $[t, d_i)$ é dada por

$$\sum_{\forall \tau_j \in \Gamma(t,t')} C_{j,k}(t) + \sum_{\forall \tau_j \in \Gamma(t,t')} \left\lceil \frac{d_i - r_j}{T_j} \right\rceil C_{j,k} \,. \tag{3.7}$$

Considerando que o sistema é escalonável antes de admitir a tarefa aperiódica τ_i, o sistema continuará sendo escalonável se todas as tarefas consideradas na equação 3.7, incluindo τ_i, finalizarem suas execuções no intervalo $[t, d_i)$. De fato, de acordo com a política EDF, as tarefas que sofrerão influência da execução de τ_i são aquelas que têm *deadlines* maiores ou iguais a d_i. Assim, admitir o intervalo $[t, d_i)$ para executar todas as instâncias de tarefas ativas, assegura que nenhuma tarefa perderá o *deadline* após a admissão de τ_i. Caso nenhuma instância contida em $\Gamma(t, t')$ perca seu *deadline* no intervalo $[t, t')$ e nenhuma outra instância não contida em $\Gamma(t, t')$ sofra interferência das primeiras, diz-se que o sistema é escalonável no intervalo $[t, t')$.

Resumindo, para cada ativação de uma tarefa aperiódica τ_i, a relação 3.8 assegura a escalonabilidade do sistema após sua admissão:

$$P(k) \equiv \sum_{\forall \tau_j \in \Gamma(t,d_i)} C_{j,k}(t) + \sum_{\forall \tau_j \in \Gamma(t,d_i)} \left\lceil \frac{d_i - r_j}{T_j} \right\rceil C_{j,k} \leq d_i - t \,. \tag{3.8}$$

A relação 3.8 pode ser usada no Algoritmo 7. Com relação à implementação deste

algoritmo, é importante observar que, conhecendo-se $P(k)$, é desnecessário efetuar o cálculo para todos os níveis de degradação k. A demanda por processador pode ser obtida subtraindo o valor $C_{j,k-1} - C_{j,k}$ da demanda por processador encontrada para o nível $k-1$, para cada tarefa a ser degradada τ_j. Desta forma, o custo de execução é consideravelmente reduzido.

Como ilustração, considere o mesmo exemplo da Figura 2.5. No instante $t = 1$, a tarefa aperiódica τ_1 não pode ser admitida com nível de degradação $k = 1$, pois a demanda total por processador no intervalo $[t, d_1)$ é de 8 unidades de tempo, o que significa que $P(k)$ não é escalonável. Fazendo $k = 2$, reduz-se a demanda total por processador para 6 unidades de tempo tornando $P(k)$ escalonável.

3.2.1.4 Simulação e Resultados

Para avaliar as abordagens descritas, foi desenvolvido um simulador de escalonamento de tempo-real, em Java. Simulações extensivas da execução de conjuntos de tarefas foram realizadas. O modelo de simulação foi construído da seguinte forma. Foram utilizados nove conjuntos de tarefas periódicas, cada qual composto por oito tarefas. Os valores de $C_{j,1}$ foram gerados de acordo com uma distribuição exponencial com média $\frac{10}{u_p}$, onde $u_p = 10\%, 20\%, \ldots, 90\%$ é o percentual de utilização do conjunto de tarefas periódicas considerado. Para efeitos de simulação, consideramos que $C_{j,k+1} = 90\% C_{j,k}$ $(k = 1, \ldots, 9)$. Os períodos foram gerados de acordo com uma distribuição uniforme no intervalo de 80 a 500 unidades de tempo e os *deadlines* foram considerados iguais aos períodos.

O tempo de simulação considerado foi de 100.000 unidades de tempo, o que permitiu observar o comportamento das tarefas periódicas no longo prazo. Nos cenários de teste considerados, por exemplo, a tarefa com período mais longo é ativada 200 vezes durante a simulação.

A ocorrência de tarefas aperiódicas seguiu o seguinte procedimento. Durante a simulação, foram gerados eventos aleatórios de acordo com uma distribuição de *Poisson*

com parâmetro $\lambda = 1000$. Tais eventos representam uma janela de instabilidade no sistema, cujo tamanho foi definido como sendo de 100 unidades de tempo. Dentro desta janela, a ocorrência de tarefas aperiódicas foi simulada. A ocorrência de cada tarefa aperiódica seguiu uma distribuição de *Bernoulli* com parâmetro $p = 0,05$. Assim, procuramos modelar a ocorrência de tarefas aperiódicas em rajadas (*burst*). Os custos e *deadlines* de tais tarefas foram gerados de tal forma que uma instância de tarefa aperiódica J_i, gerada no instante t, requer, em média, $u_a = 40\%$ de processador no intervalo $[t, d_i)$. Desta forma, pode-se verificar sobrecargas, certamente, a partir de 60%. Os valores de C_i foram gerados de acordo com uma distribuição exponencial com parâmetro $\frac{1}{40}$. Seus *deadlines* foram gerados de acordo com uma distribuição normal com média $2,5\,C_i$ e variância $0,01\,C_i$.

A simulação da execução de todas as tarefas considerou quatro alternativas de escalonamento: (a) EDF; (b) EDF-SD; (c) EDF-CDT; (d) TBS-CD. A primeira alternativa corresponde simplesmente ao escalonador EDF, sem utilizar rejeição de instâncias aperiódicas e degradação de instâncias periódicas. A abordagem EDF-SD usa o teste de admissão proposto pela equação (3.8) para rejeitar instâncias aperiódicas que possam causar violação de *deadlines*, porém não considera a degradação de tarefas. Já a abordagem EDF-CDT usa o mesmo teste e considera degradação de instâncias periódicas. Finalmente, TBS-CD corresponde ao modelo que usa o TBS, com teste de admissão representado pela equação (3.5) realizando degradação de instâncias periódicas.

Nível de Interferência das Tarefas Aperiódicas

Observamos o nível de interferência que a execução de tarefas aperiódicas exerce na execução de tarefas periódicas. Obviamente, esta interferência é nula para as abordagens de escalonamento que realizam testes de admissão para tarefas aperiódicas (EDF-SD, EDF-CD e TBS-CD). Assim, esta seção apresenta os resultados desta interferência considerando apenas o escalonador EDF, ilustrado no gráfico da Figura 3.3, medida através da taxa de *deadlines* perdidos das tarefas periódicas. Como pode ser observado, o EDF possui um desempenho ruim, como esperado. A partir de 60% de carga das tarefas periódicas a taxa de *deadlines* perdidos cresce rapidamente, o que

ilustra o efeito dominó mencionado no capítulo 2.

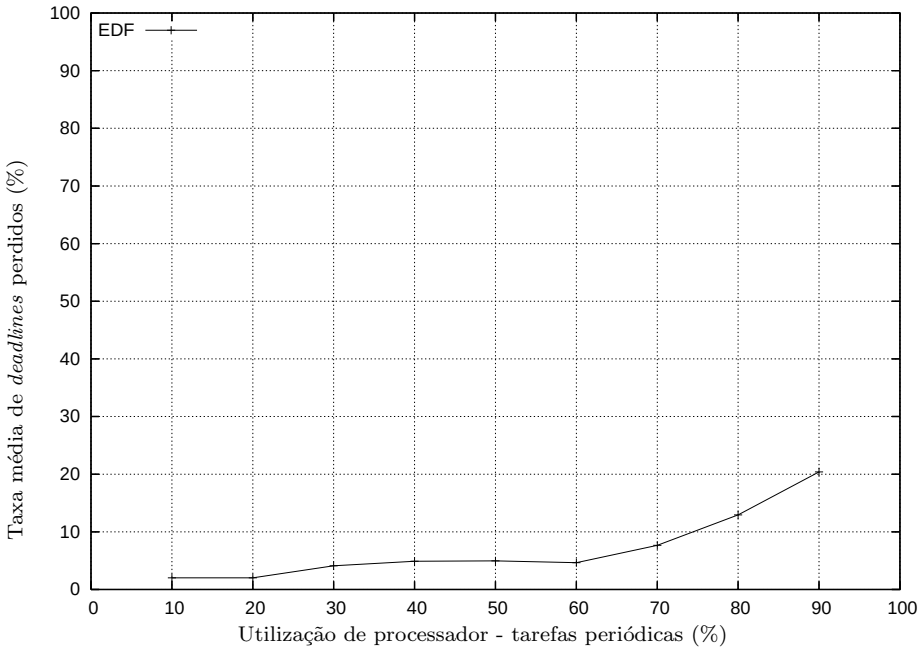


Figura 3.3. Percentual de *deadlines* perdidos de tarefas periódicas para $u_a = 40\%$

Eficiência das Técnicas de Degradação

A taxa de rejeição das tarefas aperiódicas para as três abordagens que usam testes de admissão foi medida durante a simulação. Este é um parâmetro importante de comparação, visto que o mesmo indica a eficiência do modelo de escalonamento no que se refere a admissão de instâncias aperiódicas e o respectivo efeito do algoritmo que provê degradação controlada.

O gráfico da Figura 3.4 apresenta os resultados encontrados. Para efeito de comparação, é mostrado no gráfico o desempenho do EDF com e sem o teste de admissão. Para o EDF sem teste de escalonabilidade em tempo de execução, os resultados devem ser interpretados como sendo taxa de *deadlines* perdidos de instâncias de tarefas aperiódicas, visto que todas as instâncias aperiódicas são admitidas no sistema. Apesar de o EDF apresentar resultados médios melhores que as demais abordagens com relação à taxa de rejeição de instâncias aperiódicas, deve-se considerar que instâncias periódicas estão perdendo *deadlines* (ver Figura 3.3), o que faz o uso do EDF uma abor-

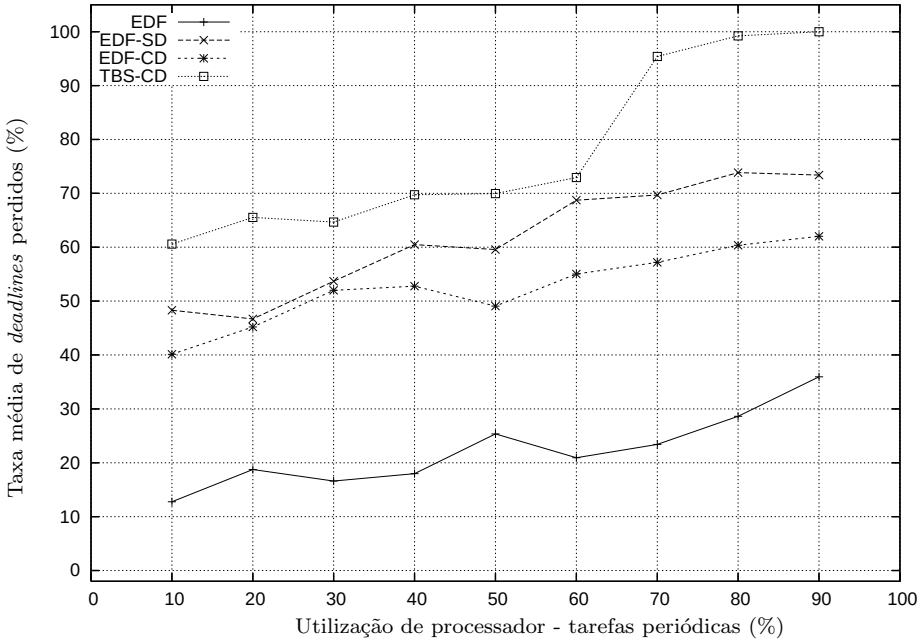


Figura 3.4. Percentual de rejeição de tarefas aperiódicas para $u_a = 40\%$

dagem inapropriada. Ao equipar o EDF com um teste de escalonabilidade em tempo de execução, pode-se notar que a taxa de rejeição de instâncias aperiódicas continua alta. Neste contexto, a implementação de métodos para prover degradação controlada é extremamente útil, como pode ser observado com a curva relativa a EDF-CD. É interessante notar que o TBS-CD mostra-se muito inferior às demais abordagens, mesmo fazendo uso de degradação controlada. Isto se deve ao fato de que tarefas aperiódicas são admitidas e executadas seqüencialmente no sistema.

Nível de Degradação Média

Aferimos o nível de degradação média do sistema, ilustrado na Figura 3.5. O nível de degradação foi medido como sendo o percentual médio de redução de $C_{j,1}$, para cada tarefa periódica τ_j, necessário para se admitir as tarefas aperiódicas. Como pode ser observado, a degradação do sistema se mantém entre 80% e 90%. Obviamente, este resultado não pode ser analisado separadamente dos resultados da rejeição de instâncias aperiódicas. De fato, as duas abordagens que contemplam degradação controlada são equivalentes quanto ao nível de degradação do sistema. No entanto, o TBS-CD rejeita

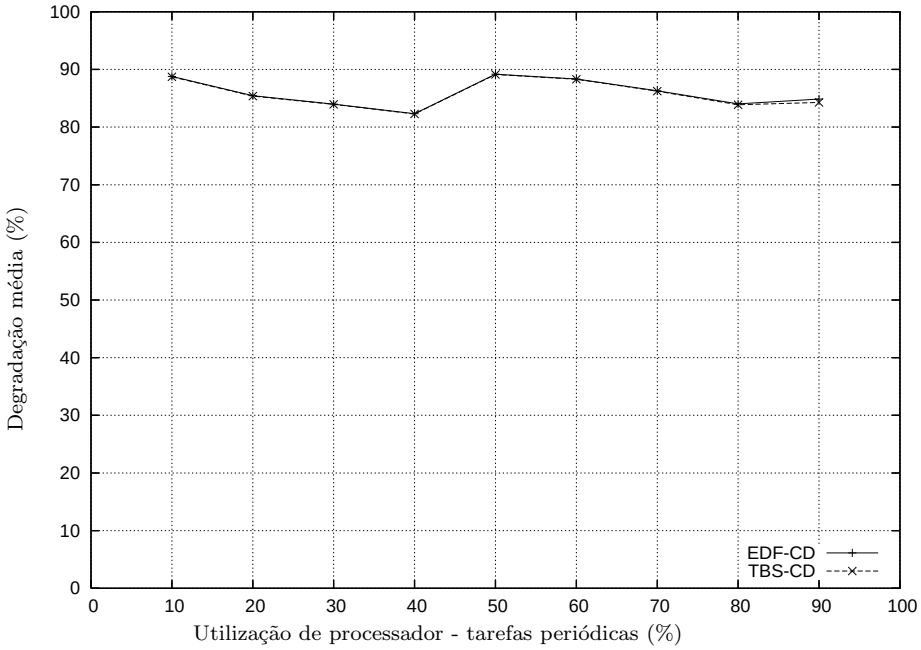


Figura 3.5. Degradação média do sistema para $u_a = 40\%$

um número maior de instâncias aperiódicas, como visto anteriormente.

Outro aspecto importante que deve ser mencionado é a simplicidade do algoritmo de degradação. Na realidade, o espaço de busca por um nível adequado de degradação é exponencial. Tal espaço foi substancialmente reduzido pelas políticas EDF-CD e TBS-CD, viabilizando-as em tempo de execução. A realização de aprimoramentos no algoritmo, certamente, reduzirá o nível de instâncias atingido, pois, em alguns casos, basta degradar uma instância para que a configuração necessária seja atingida, não sendo necessário degradar todas as instâncias de uma só vez. De qualquer forma, o benefício em prover mecanismos de degradação controlada pode compensar os seus custos, o que precisa ser melhor investigado.

3.2.2 Reconfiguração Baseada em Instâncias de Tarefas

As técnicas de reconfiguração vistas na seção anterior possuem um desempenho aquém do desejado para uma aplicação prática de reconfiguração dinâmica, pois estas degradam todas as instâncias das tarefas de uma vez só. Desta forma, para melhorar

um dos pontos fracos destes testes, foi desenvolvido um algoritmo de degradação que degrada as instâncias das tarefas, uma a uma.

Sem modificar a regra de degradar todas as tarefas ao mesmo nível ao mesmo tempo, o algoritmo aqui desenvolvido passa a degradar instância por instância, levando todas ao mesmo nível somente quando necessário. Porém deve-se salientar que a variação do nível de degradação de uma instância a outra, mesmo que de tarefas diferentes, é de apenas uma unidade, pois o nível de degradação continua sendo estabelecido de uma só vez.

Assim, o teste de admissão desta seção baseia-se no teste proposto na seção anterior, porém a demanda por processador no intervalo entre a ativação de uma tarefa aperiódica e o seu *deadline* é calculada de forma diferente. No instante de ativação da tarefa aperiódica, determinamos todas as instâncias ativas no intervalo $[t, t')$.

Para determinar a demanda total por processador no intervalo, deve-se então somar os custos computacionais das instâncias ativas no intervalo. Todas as instâncias que já iniciaram suas execuções, sejam elas periódicas ou não, contribuirão para esta demanda com o tempo necessário para que elas terminem. As demais instâncias contribuirão com o valor total a ser executado, podendo este sofrer degradação.

Considerando que o sistema é escalonável antes de admitir a instância aperiódica τ_i, o sistema continuará sendo escalonável se todas as instâncias de tarefas ativas no intervalo $[t, t')$, incluindo τ_i, finalizarem suas execuções no intervalo $[t, t')$. Para tanto, consideramos, hipoteticamente, que todas as tarefas do intervalo possuem *deadline* menor ou igual a t'.

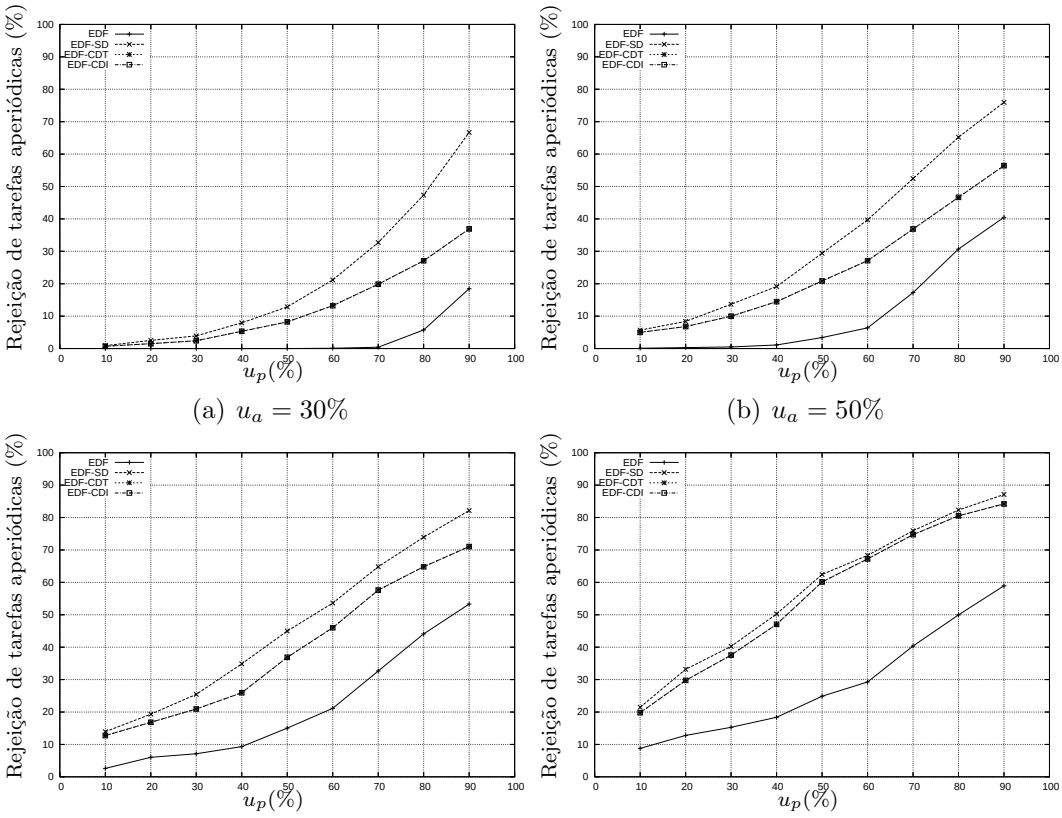
Utilizando tal algoritmo, deve-se obter o mesmo desempenho do algoritmo da subseção anterior, porém o percentual de degradação deve diminuir, pois as tarefas nem sempre precisaram degradar todas as suas instâncias ao mesmo nível de degradação.

3.2.2.1 Simulação e Resultados

Para fins de comparação, um novo simulador de escalonamento de tempo-real foi desenvolvido, adaptando o simulador utilizado na seção 3.2.1.4 para trabalhar com instâncias de tarefas. Os mesmos conjuntos de tarefas e distribuições foram utilizados, alterando apenas os algoritmos utilizados no sistema.

A ocorrência de tarefas aperiódicas foi gerada a partir de eventos aleatórios gerados de acordo com uma distribuição de *Poisson* com parâmetro $\lambda = 100$. A simulação da execução de todas as tarefas considerou quatro alternativas de escalonamento, considerando as propostas da seção anterior e as propostas desta seção: (a) EDF; (b) EDF-SD; (c) EDF-CDT; (d)EDF-CDI. A primeira alternativa corresponde simplesmente ao escalonador EDF, sem utilizar rejeição de tarefas aperiódicas nem tentar degradar tarefas periódicas. A abordagem EDF-SD usa o teste de admissão proposto pela equação (3.8) para rejeitar tarefas aperiódicas que possam causar violação de *deadlines*. Já a abordagem EDF-CDT usa o mesmo teste e considera degradação de tarefas aperiódicas. Finalmente, EDF-CDI faz o teste de escalonabilidade proposto nesta seção e realiza conjuntamente a degradação controlada de instâncias de tarefas. Os resultados são os que se seguem, considerando utilização aperiódica de $u_a = 30\%$ até $u_a = 90\%$, de 20 em 20%.

Avaliando os gráficos, pode-se perceber que a realização de degradação controlada, seja de tarefas ou de instâncias de tarefas, melhora o desempenho do sistema, pois diminui o descarte de tarefas aperiódicas. Esta melhora, que pode ser observada na Figura 3.6, fica menos acentuada com o aumento do percentual de utilização das tarefas aperiódicas. Tal comportamento era esperado, pois quanto maior o percentual de utilização da instância aperiódica a ser inserida, maior será a degradação necessária às instâncias ativas no intervalo para permitir tal inserção sem perdas de *deadlines*. Como o nível de degradação que pode ser atingido pelas instâncias periódicas é limitado, existe um limite de utilização possível à instância aperiódica para que esta seja inserida no sistema. Ultrapassando tal limite, mesmo degradando o sistema ao máximo, não é possível inserir a instância aperiódica sem causar perdas de *deadlines* das instâncias



(a) $u_a = 30\%$ (b) $u_a = 50\%$

(c) $u_a = 70\%$ (d) $u_a = 90\%$

Figura 3.6. Percentual de rejeição de tarefas aperiódicas

ativas, acarretando então no descarte da primeira.

Devemos destacar que a proposta de degradação de instâncias (EDF-CDI) apresenta mesmo percentual de rejeição de tarefas aperiódicas que a proposta de degradação de tarefas (EDF-CDT), como visto na Figura 3.6. Porém, a proposta EDF-CDI deve apresentar uma menor taxa de degradação que a proposta EDF-CDT, pois em EDF-CDI as instâncias estão sendo degradadas uma a uma, diminuindo o impacto da inserção da tarefa aperiódica no sistema em EDF-CDT. Podemos observar também que o EDF apresenta uma menor taxa de *deadlines* perdidos de tarefas aperiódicas, porém apresenta também *deadlines* perdidos de tarefas periódicas, algo eliminado pelo teste de admissão contido nas versões EDF-SD, EDF-CDT e EDF-CDI.

3.3 SUMÁRIO

Neste capítulo foram descritas e avaliadas propostas de escalonamento. Assumimos um modelo de tarefas que contempla tarefas periódicas, que devem ser executadas regularmente pelo sistema, e tarefas aperiódicas. Caso tais tarefas sejam ativadas no sistema, o escalonador adapta o sistema, degradando as tarefas periódicas, a fim de cumprir os requisitos temporais das tarefas aperiódicas. Uma das abordagens propostas baseou-se no servidor TBS, freqüentemente usado para escalonar tarefas aperiódicas não-críticas. Ilustramos, através de simulação, que esta proposta não é adequada para tratar tarefas aperiódicas críticas, como rotinas de recuperação. Ilustramos ainda que, equipando o EDF com um teste de escalonabilidade em tempo de execução, é possível melhorar bastante a eficiência do escalonador, protegendo o sistema de perdas de *deadlines* de tarefas periódicas. Por fim, mostramos uma abordagem que degrada instância por instância de uma tarefa, diminuindo assim a degradação do sistema provocada pela abordagem anterior.

Estudos são necessários para a determinação de heurísticas mais apropriadas para a determinação da ação a ser tomada no instante em que a tarefa aperiódica é ativada no sistema. Sistemas complexos podem necessitar de mecanismos mais adaptáveis, que considerem tarefas com diferentes níveis de degradação. Para atender tais sistemas, mecanismos mais adaptáveis devem ser propostos. No próximo capítulo estudaremos dois destes mecanismos.

CAPÍTULO 4

ESTUDO DE TESTES DE ESCALONABILIDADE EM TEMPO DE EXECUÇÃO

Como visto nos capítulos anteriores, os mecanismos de escalonamento estão no cerne do problema de reconfiguração dinâmica, pois os mesmos são essenciais para a garantia da correção temporal dos sistemas de tempo-real. Desta forma, o desempenho do teste de escalonabilidade interfere diretamente no desempenho do sistema. Além de garantir a correção temporal, tais testes devem executar rápido o suficiente para não afetar o sistema e a execução de suas tarefas, pois estes estão sendo executados em tempo de execução, juntamente com o escalonador. Tais fatores devem ser considerados com o mínimo de pessimismo possível.

Neste capítulo apresentaremos duas propostas de testes de escalonabilidade em tempo de execução. A primeira, o teste de escalonabilidade baseado em trabalho acumulado, calcula a interferência que as instâncias exercem umas sobre as outras conferindo a possibilidade de perdas de *deadlines*. A idéia é calcular, em tempo de execução, o trabalho acumulado para a instância J_i, considerando um intervalo de tempo $[t, t')$ e as instâncias ativas neste intervalo, $\Gamma(t, t')$. Tal trabalho acumulado é o esforço de execução de J_i e das instâncias com prioridade maior ou igual a prioridade de J_i. Se tal trabalho for inferior ao intervalo disponível para executar J_i, sabe-se que J_i será escalonável em $[t, t')$. A segunda proposta, o teste de escalonabilidade baseado no seqüenciamento de instâncias, baseia-se no seqüenciamento das instâncias ativas no intervalo $\Gamma(t, t')$, como se não houvesse preempção.

Este capítulo está estruturado da seguinte forma. Nas seções 4.1 e 4.2 serão apresentados os testes de escalonabilidade em tempo de execução descritos no parágrafo anterior. A seção 4.3 apresenta os resultados da simulação comparativa destes dois testes. Finalmente, na seção 4.4, um sumário do capítulo é apresentado.

4.1 TESTE DE ESCALONABILIDADE BASEADO EM TRABALHO ACUMULADO

Antes de apresentar o teste, algumas definições serão necessárias. Seja J_i uma instância ativa no intervalo $[t, t')$. Sejam s_i o instante de interesse relativo a J_i, definido na equação (4.1), e_i, o instante até o qual J_i deve finalizar sua execução, definido na equação (4.2), pois precisamos confinar no intervalo $[t, t')$ quaisquer execuções e efeitos obtidos pelas tarefas ativas neste intervalo, evitando a interferência destas tarefas para além deste intervalo. Por fim, s'_i, o instante no qual o sistema inicia a execução de alguma tarefa mais prioritária do que J_i, definido na equação (4.3).

$$s_i = \max(t, r_i) \tag{4.1}$$

$$e_i = \min(t', d_i) \tag{4.2}$$

$$s'_i = \max(t, \min_{d_j \leq d_i}(r_j)) \tag{4.3}$$

Como J_i, para ser escalonável no intervalo $[s_i, e_i)$, deverá finalizar sua execução até e_i, deve-se calcular a interferência máxima, w_i, sofrida por J_i no intervalo $[s_i, e_i)$. Tal interferência é denominada *trabalho acumulado* no intervalo em questão.

As instâncias de tarefas que contribuem para w_i possuem prioridades maiores ou iguais a J_i. Pode-se então dividir o trabalho acumulado do conjunto de instâncias que podem interferir na execução de J_i em duas partes, w_i^1 e w_i^2, assim definidas:

$$w_i^1 = \sum_{\forall J_j \in \Gamma(t,t'): d_j \leq d_i} C_j(s_j) \tag{4.4}$$

$$w_i^2 = \sum_{\forall J_j \in \Gamma(t,t'): d_j \leq d_i,\ s_j < s_i} C_j(s_j) \tag{4.5}$$

Em outras palavras, w_i^1 considera todas as instâncias mais prioritárias que J_i e w_i^2 contempla as interferências provocadas por aquelas que foram ativadas antes de s_i. Para calcular a interferência w_i deve-se ainda considerar quanto de w_i^2 pode ter sido executado até o instante s_i. Obviamente, o processador não pode estar ocupado no intervalo $[t, s_i)$ executando instâncias que contribuam para w_i^2 mais do que $s_i - s'_i$. Assim, a parte já executada de w_i^2 até s_i é $\min(s_i - s'_i, w_i^2)$. Portanto, w_i pode ser calculado como:

$$w_i = w_i^1 - \min(s_i - s'_i, w_i^2) \tag{4.6}$$

Podemos agora enunciar a condição de escalonabilidade do sistema da seguinte forma:

Teorema 1 *As instâncias ativas em $\Gamma(t, t') = \{J_1, J_2, ..., J_n\}$ são escalonáveis no intervalo $[t, t')$ se, para cada $J_i \in \Gamma(t, t')$, $w_i \leq e_i - s_i$.*

Prova: Para realizar a prova deste teorema, inicialmente, deve-se assumir o caso particular em que $w_i^2 = 0$. Desta forma, a equação (4.6) implica em $w_i = w_i^1$. Assim, o teorema se verifica, dado que J_i tem o intervalo $[s_i, e_i)$ para finalizar sua execução.

Considere agora que $w_i^2 > 0$, ou seja, existe alguma instância $J_j \in \Gamma(t, t')$ que inicia sua execução antes de s_i e pode interferir na execução de J_i. Neste caso, deve-se considerar o intervalo $[s'_i, s_i)$, como pode ser observado na Figura 4.1, pois durante tal intervalo o processador estará ocupado executando J_j.

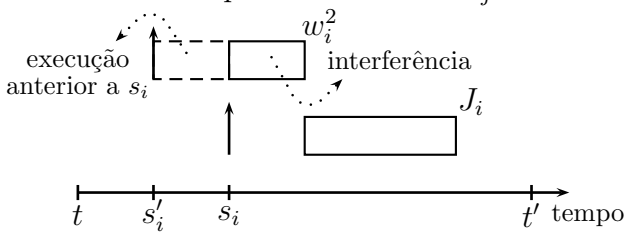


Figura 4.1. Interferência w_i^2 na execução de J_i

Dois cenários são possíveis:

Cenário 1: $w_i^2 \leq s_i - s'_i$. Em outras palavras, neste cenário não se deve considerar o valor de w_i^2 no cálculo do trabalho total, w_i, pois o intervalo de tempo $[s'_i, s_i)$ é grande o suficiente para completar a instância J_j antes da instância J_i ser ativada. Tal fato pode ser observado na Figura 4.2, onde w_i^2 é totalmente executado antes de s_i, restando, inclusive, tempo ocioso entre o final de w_i^2 e s_i. Assim, para determinar corretamente w_i, deve-se retirar de w_i^1 o valor executado antes de s_i, que é igual a w_i^2. Desta forma, $w_i = w_i^1 - w_i^2$.

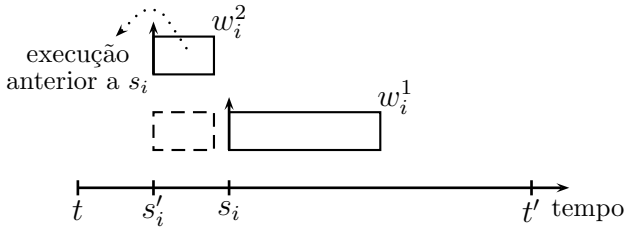


Figura 4.2. w_i^2 não interfere na execução de J_i

Cenário 2: $w_i^2 > s_i - s'_i$. Neste caso, não existe tempo suficiente para executar todas as instâncias J_j, que estão no cálculo de w_i^2, antes de J_i ser ativada. Assim o tempo em que o processador está ocupado executando as instâncias que foram ativadas antes de s_i é igual a $s_i - s'_i$. Este caso pode ser observado na Figura 4.3, onde a determinação de w_i deverá ser feita através de w_i^1 subtraído do valor correspondente às execuções antes de s_i, que corresponde a $s_i - s'_i$. Logo, $w_i = w_i^1 - (s_i - s'_i)$

Os cenários 1 e 2 acima descritos podem ser unificados pela equação (4.6), a qual contabiliza a execução de J_i e a execução das instâncias mais prioritárias do que J_i que interferem em sua execução. É claro que esta interferência não pode ser maior do que $e_i - s_i$, que é o tempo disponível para J_i iniciar e terminar sua execução. Se tal condição for repetida para todas as instâncias contidas em $\Gamma(t, t')$, prova-se o teorema. □

É interessante notar que o teste de escalonabilidade enunciado pelo teorema 1

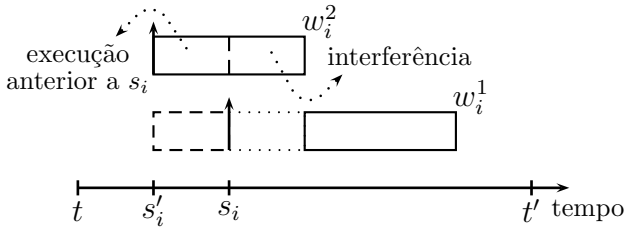


Figura 4.3. A interferência w_i^2 na execução de J_i é $w_i^2 - s_i - s'_i$

é apenas suficiente, pois, dependendo do intervalo $[t, t')$ considerado, o teste pode considerar não escalonável um intervalo que é escalonável. Por exemplo, considere a Figura 4.4, que ilustra um conjunto de instâncias escalonável no intervalo $[0, 7)$ e $[0, 4)$ pelo algoritmo EDF. As setas verticais significam os instantes de ativação das instâncias. No caso em que $[t, t') = [0, 7)$, considerando a instância J_3, o trabalho acumulado w_3 será resultado da subtração $w_3^1 - w_3^2$. O termo w_3^1 corresponde ao somatório dos custos computacionais das instâncias J_1, J_2 e J_3, totalizando 7 e w_3^2 é igual a 5. Assim, $w_3 = w_3^1 - w_3^2 = 2$, como a condição de escalonabilidade é $w_3 \leq e_3 - s_3 = 7 - 5 = 2$, a instância J_3 é escalonável no intervalo $[0, 7)$. Repetindo o mesmo processo para as demais instâncias, temos que todas as instâncias são escalonáveis neste intervalo. No entanto, se o intervalo considerado é $[0, 4)$, o teste proposto considera o sistema não escalonável. Desta forma, a definição do intervalo $[t, t')$ é importante para o desempenho do teste de escalonabilidade. Este aspecto será discutido posteriormente.

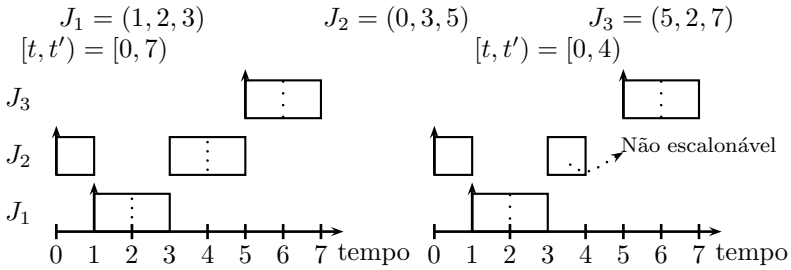


Figura 4.4. Uma ilustração da condição de escalonabilidade do teste baseado em trabalho acumulado. $\Gamma(0, 7) = \{J_1, J_2, J_3\}$ e $J_i = (r_i, C_i, d_i)$, onde $i = 1, 2, 3$.

4.2 TESTE DE ESCALONABILIDADE BASEADO EM SEQÜENCIAMENTO

Da mesma forma que na seção anterior, algumas definições serão necessárias. Considere que $\Gamma(t, t')$ está ordenado pelos *deadlines* de suas instâncias, ou seja, $d_i \leq d_{i+1}$, e se $d_i = d_{i+1}$, então $s_i \leq s_{i+1}$ para todo $i < n$. Seja t_i, $i = 1, 2 \ldots, n$, definido como segue:

$$t_i = \begin{cases} t' & \text{se } i = n \\ \min(t_{i+1}, d_{i+1}) - C_{i+1}(s_{i+1}) & \text{se } 1 \leq i < n \end{cases} \tag{4.7}$$

Seja $\min(d_i, t_i)$ o instante máximo para término de execução de J_i, podendo ser t_i quando $t_i < d_i$, enquanto $\min(d_i, t_i) - C_i(s_i)$ é o instante máximo para início de processamento de J_i.

As condições de escalonabilidade derivadas nesta seção assumem o seqüenciamento de instâncias pertencentes a $\Gamma(t, t')$ no intervalo $[t, t')$ desconsiderando a existência de preempção. Por exemplo, assume-se que J_n não pode ser escalonada antes de s_n e deve terminar até $\min(t', d_n)$. Assim, J_{n-1} não poderá iniciar sua execução antes de s_{n-1} e deverá terminar até $\min(t_{n-1}, d_{n-1})$. Note que a definição de t_i para $i < n$ leva em consideração o tempo gasto para executar J_{i+1}, ou seja, $C_{i+1}(s_{i+1})$. A escalonabilidade de todas as instâncias ativas em $\Gamma(t, t')$ é dada pelo teorema a seguir.

Teorema 2 *As instâncias ativas em $\Gamma(t, t') = \{J_1, J_2, \ldots, J_n\}$ são escalonáveis no intervalo $[t, t')$ se*

$$\forall J_i \in \Gamma(t, t') : s_i + C_i(s_i) \leq \min(d_i, t_i)\,, \tag{4.8}$$

Prova: Esta prova será realizada através da construção de um escalonamento não-preemptivo em $[t, t')$. Considere primeiro uma instância J_n. Está claro que J_n é escalonável se $s_n + C_n(s_n) \leq \min(d_n, t_n)$, pois pela equação (4.7), instâncias mais prioritárias devem terminar suas execuções até $t_{n-1} = \min(t_n, d_n) - C_n(s_n)$. Em outras palavras, existe um intervalo de tempo $I_n = [\min(d_n, t_n) - C_n(s_n), \min(d_n, t_n))$ reservado para a execução exclusiva de J_n. Agora, retire J_n da série de instâncias ativas

em $\Gamma(t, t')$ e faça $t' = t_{n-1}$. A mesma consideração se aplica para $\{J_1, J_2, \ldots, J_{n-1}\}$, as quais levam à definição de intervalos de tempo $I_i = [\min(d_i, t_i) - C_i(s_i), \min(d_i, t_i))$ que são reservados para a execução exclusiva de J_i, $i = 1, 2, \ldots, n$. Como I_i é suficiente para executar J_i, a demonstração deste teorema está concluída. □

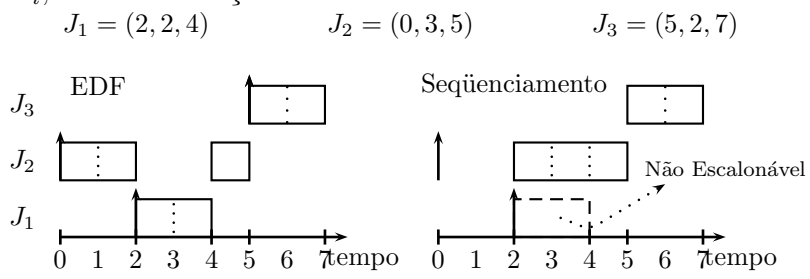


Figura 4.5. Uma ilustração da condição de escalonabilidade do teste baseado em seqüenciamento de instâncias. $\Gamma(0, 7) = \{J_1, J_2, J_3\}$ e $J_i = (r_i, C_i, d_i)$ onde $i = 1, 2, 3$.

É importante enfatizar que não consideramos sistemas não-preemptivos. A inexistência de preempção é assumida somente para derivar as condições de escalonabilidade. O sistema pode seguir normalmente a ordenação da política EDF, pois preempções são permitidas. Obviamente tal teste é restritivo, sendo somente suficiente. A Figura 4.5 ilustra um exemplo de instâncias que são escalonáveis no intervalo $[0, 7)$ pelo EDF, porém ao utilizarmos o teste de escalonabilidade baseado no seqüenciamento de instâncias, pode-se notar que o sistema é considerado não escalonável no intervalo $[0, 7)$.

4.3 SIMULAÇÃO E RESULTADOS

Nesta seção, serão apresentados resultados das simulações para fins de comparação entre as duas opções de teste de escalonabilidade apresentadas neste capítulo. Para tais simulações foi utilizado o mesmo simulador utilizado nas simuações do capítulo 3, acrescentando os algoritmos correspondentes às propostas da seção 4.1 e 4.2. O sistema, semelhantemente ao capítulo anterior, possui nove conjuntos contendo oito tarefas periódicas cada. Todos os conjuntos são escalonados pelo EDF e são escalonáveis caso não haja ocorrência de tarefas aperiódicas. A taxa de ativação de tarefas aperiódicas foi simulada através de uma distribuição de Poisson com parâmetro

$\lambda = 1000$.

O cálculo de t' é realizado da mesma forma que no capítulo anterior. Esta forma de calcular t' nos assegura um intervalo de reconfiguração maior do que o original, $[t, d)$, sendo d o *deadline* da tarefa aperiódica ativada em t, sem a inserção de novas instâncias, impedindo assim o aumento da complexidade do teste e diminui sua escalonabilidade. Note que, se o intervalo de reconfiguração for muito pequeno, nenhuma reconfiguração será possível. Por outro lado, se o intervalo for demasiadamente grande, muitas instâncias serão consideradas para reconfiguração, o que aumenta o custo computacional do procedimento de recuperação, pois o teste se torna mais complexo. Assim, políticas adequadas para a escolha de t' devem ser utilizadas.

Esta seção está dividida em três subseções. A subseção 4.3.1 avalia os testes em um ambiente escalonável, composto apenas por tarefas periódicas. Desta forma, pode-se determinar o pessimismo dos testes em situações em que todos os intervalos deveriam ser considerados escalonáveis. Na subseção 4.3.2, os testes são avaliados em um ambiente com co-existência de tarefas periódicas e aperiódicas. Obedecendo a taxa de ativação por distribuição de Poisson, as tarefas aperiódicas têm suas cargas variadas e os desempenhos dos testes são avaliados com diversas configurações de cargas periódicas e aperiódicas. Por fim, na subseção 4.3.3, avalia-se o desempenho dos testes, no mesmo ambiente da subseção 4.3.2, com diversos tamanhos de intervalos de reconfiguração, $[t, t')$. Para tanto, definiu-se um fator multiplicador, β, que cria um novo intervalo $[t, t')$ maior que o anterior e igual a $[t, t + \beta(d - t))$. Assim, pode-se avaliar um aspecto constatado na definição e simulação dos testes, que o tamanho do intervalo de reconfiguração influi no desempenho destes testes.

4.3.1 Avaliando o Desempenho sem Inserção de Tarefas Aperiódicas

Para avaliar o pessimismo dos testes de escalonabilidade, executamos os dois testes de escalonabilidade, apresentados neste capítulo. Utilizamos um sistema com carga periódica, u_p, variando de 10% a 90%. Os testes foram realizados em média 100

vezes a cada simulação. O tempo simulado foi de 100.000 unidades. Nenhuma tarefa aperiódica foi inserida no sistema, apenas foram avaliadas as tarefas periódicas, que já estavam em execução no sistema. Foram realizadas dez simulações com conjuntos diferentes de tarefas. Então, uma média foi calculada e utilizada na construção do gráfico da Figura 4.6. Os sistemas avaliados eram escalonados pelo EDF, com *deadline* igual ao período e custo computacional constante, possuindo carga total sempre menor do que 100%, isto é, $u_p < 1$. Portanto, os sistemas simulados são escalonáveis. A intenção é aferir o pessimismo dos testes de escalonabilidade propostos.

Para realizar a simulação foram utilizadas duas alternativas de escalonamento: (a) EDF-T; (b) EDF-S, correspondentes aos testes propostos nas seções 4.1 e 4.2, respectivamente. O início dos intervalos $[t, t')$ foram gerados de acordo com a distribuição de *Poisson*, com parâmetro $\lambda = 1000$.

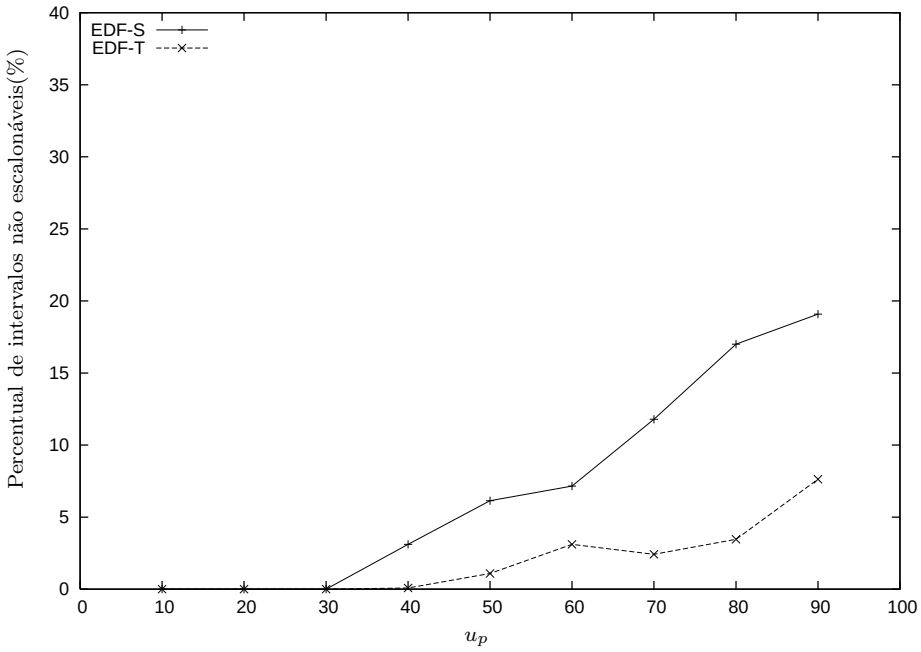


Figura 4.6. Percentual de intervalos não escalonáveis por carga periódica, u_p

Pode-se perceber, pelo comportamento do gráfico da Figura 4.6, que o pessimismo dos testes de escalonabilidade é crescente em função do aumento da carga periódica. Porém, tal pessimismo é significativamente menor para o EDF-T. Observa-se também que, inicialmente, quando a carga do sistema é menor do que 30%, ambos os testes

apresentam resposta satisfatória, dado que o sistema foi considerado escalonável em todos os intervalos gerados. Quando a utilização periódica cresce, o EDF-T apresenta um comportamento melhor. Isto se deve ao fato que a única fonte de pessimismo do teste EDF-T é a determinação do tamanho do intervalo, que pode não suportar a execução de algumas tarefas, como ilustrado na Figura 4.6. Já o teste EDF-S, além desta fonte de pessimismo, há também o fato de não se considerar a preempção das instâncias.

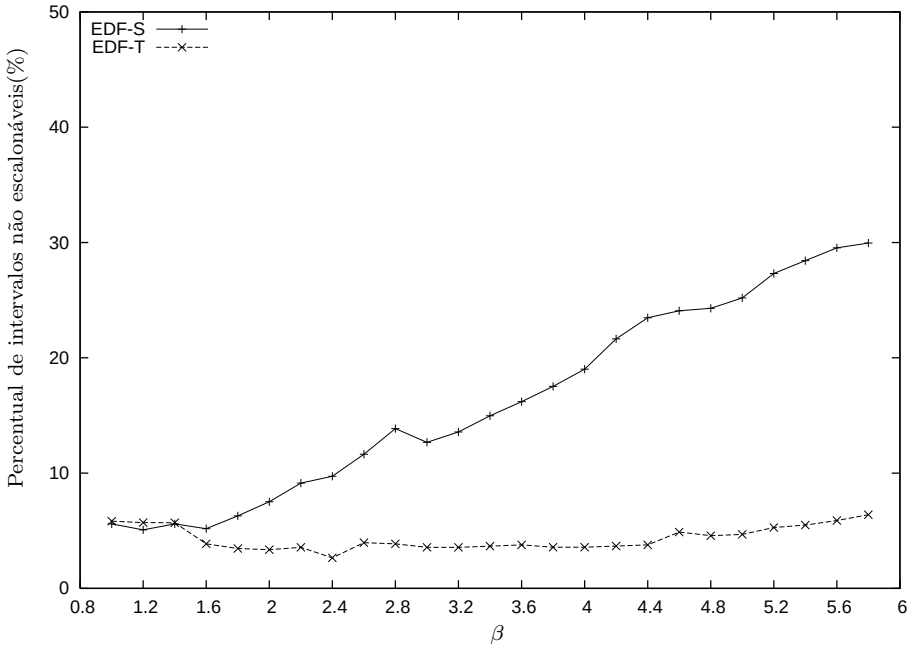


Figura 4.7. Percentual de intervalos não escalonáveis por tamanho do intervalo, $u_p = 60\%$

Para constatar o comportamento dos testes em relação ao tamanho do intervalo, variou-se o tamanho do intervalo, usando uma carga periódica fixa, $u_p = 60\%$. Para avaliar o desempenho dos testes, variou-se β de $1,0$ a $5,8$, de $0,2$ em $0,2$. Os valores plotados no gráfico da Figura 4.7 são uma média de dez simulações realizadas com conjuntos de tarefas diferentes. Os resultados indicam que o EDF-T diminui ligeiramente seu pessimismo com o aumento do intervalo, posteriormente, ficando quase estável. Porém o custo do teste cresce ao aumentarmos o intervalo, pois mais instâncias passam a ser consideradas. Já o teste EDF-S apresentou uma piora acentuada em seu desempenho ao aumentarmos o intervalo, devido ao seu maior pessimismo, como dito

anteriormente.

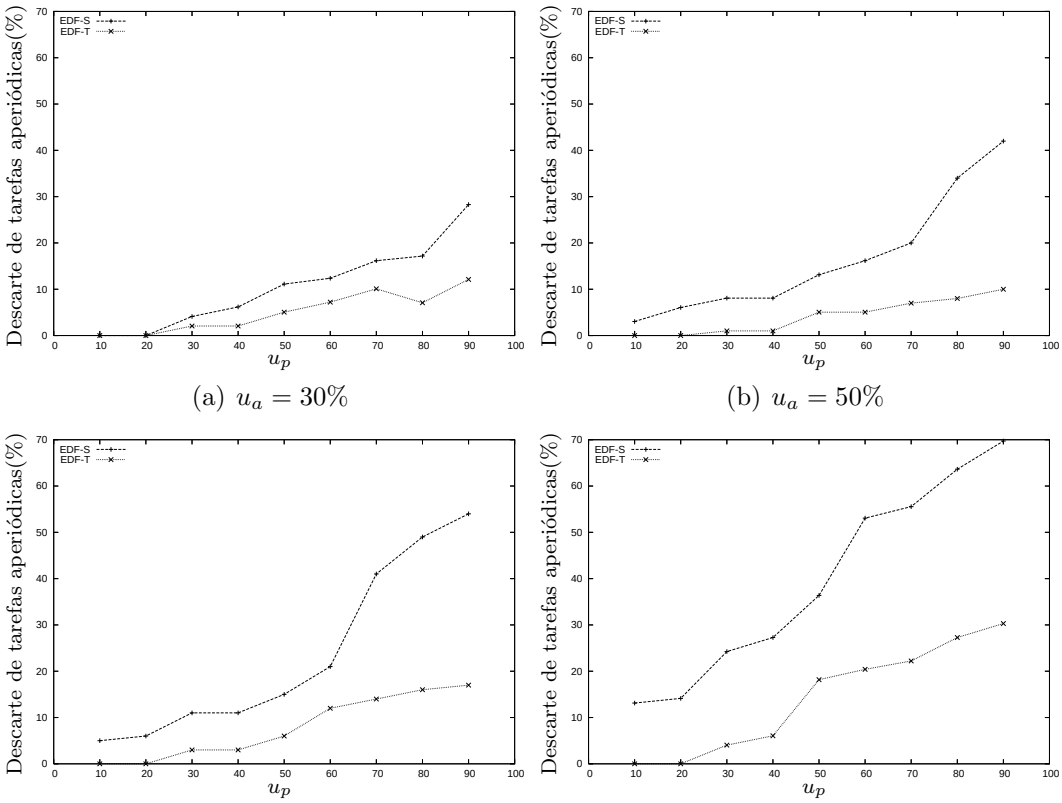
A partir dos comportamentos observados nas figuras desta seção, o EDF-T, usando $\beta = 1$, apresenta uma melhor relação entre custo de execução e quantidade de tarefas aperiódicas aceitas e executadas com sucesso. Afinal, o objetivo do teste de escalonabilidade em tempo de execução deve ser atingir uma baixa taxa de *deadlines* de tarefas perdidos ou de tarefas descartadas mantendo um custo de execução baixo, sendo assim mais eficiente.

4.3.2 Avaliando o Desempenho com Inserção de Tarefas Aperiódicas

Nesta seção foram avaliados o desempenho dos testes de escalonabilidade em ambiente com a co-existência de tarefas periódicas e aperiódicas. Usando os mesmos cenários da subseção anterior, acrescentou-se a ativação de tarefas aperiódicas de acordo com uma distribuição de *Poisson*, com parâmetro $\lambda = 1000$. Nesta simulação, instâncias de tarefas aperiódicas poderão ser descartadas, caso seja detectado, pelo teste de escalonabilidade, que o sistema corre riscos de perder *deadlines*.

Para uma completa avaliação dos testes de escalonabilidade, considerou-se diversas cargas periódicas e aperiódicas. Nos gráficos da Figura 4.8, pode-se observar a carga periódica variando de 10% até 90%, de 10% em 10%. Quatro valores para a carga aperiódica foram considerados.

Podemos observar através da Figura 4.8 que os resultados das simulações, para ambos os testes, apresenta uma piora no desempenho dos testes à medida que cresce tanto a carga periódica quanto a carga aperiódica. Pode-se perceber também que o desempenho do teste EDF-T continua significativamente superior ao desempenho do teste EDF-S, o que era esperado após as constatações da subseção anterior.



(a) $u_a = 30\%$ (b) $u_a = 50\%$

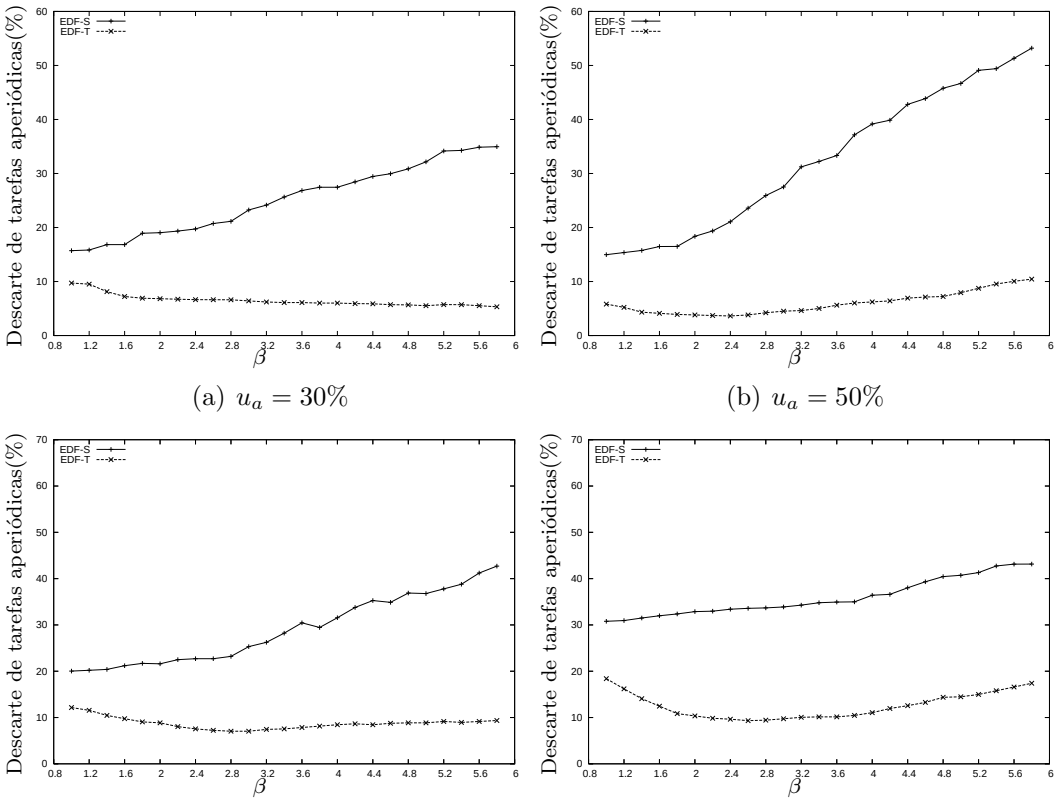
(c) $u_a = 70\%$ (d) $u_a = 90\%$

Figura 4.8. Gráficos de taxa de descarte de tarefas aperiódicas por carga periódica

4.3.3 Avaliando o Desempenho com Alteração do Tamanho do Intervalo

Para avaliar o desempenho dos testes de escalonabilidade em relação ao tamanho do intervalo de reconfiguração, novas simulações foram realizadas, alterando o tamanho de tal intervalo. Na simulação, o valor de β variou de $1,0$ a $5,8$, a carga periódica foi fixada em $u_p = 60\%$ e a carga aperiódica, u_a, variou entre 30% e 90%, garantindo assim cenários com sobrecargas variadas. Foram utilizadas as mesmas alternativas de escalonamento usadas nas subseções anteriores, EDF-S e EDF-T. Para gerar os valores dos gráficos, dez simulações foram realizadas para cada valor de β, com conjuntos diferentes de tarefas periódicas e aperiódicas. A média destes valores foi utilizada para gerar os gráficos da Figura 4.9.

Pode-se observar, a partir dos resultados mostrados na Figura 4.9, que, em geral, quanto maior o intervalo de reconfiguração, mais tarefas aperiódicas são descartadas



(a) $u_a = 30\%$ (b) $u_a = 50\%$

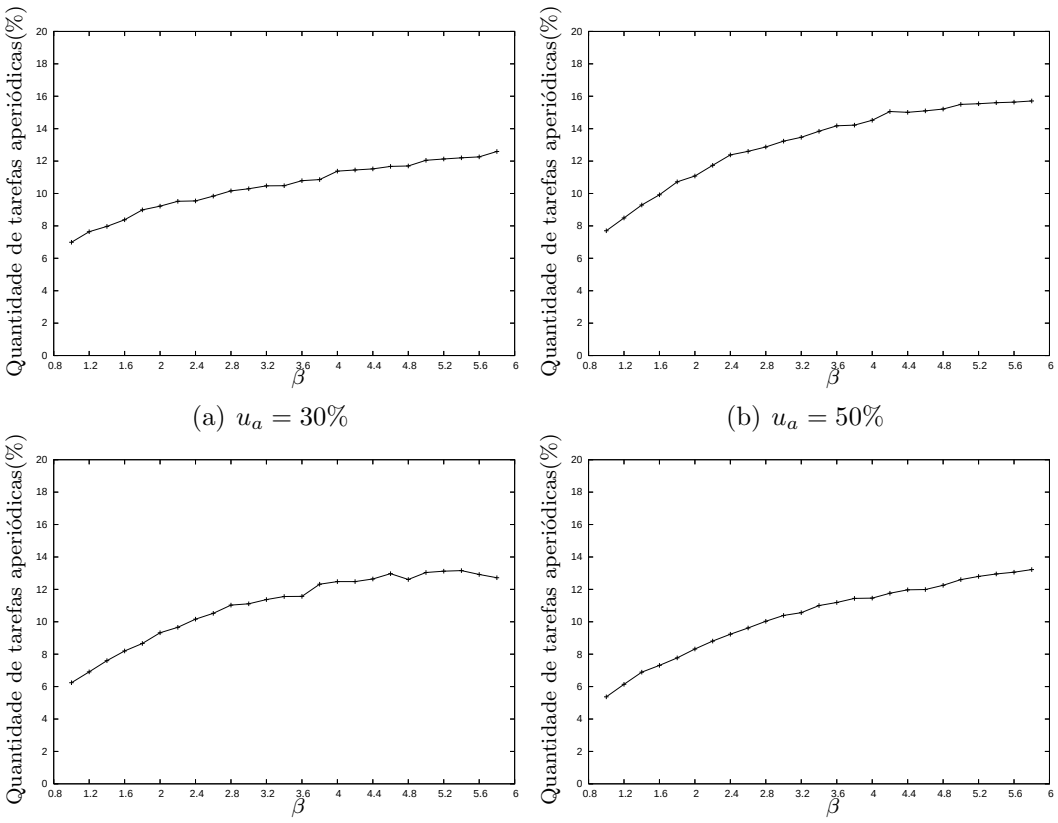
(c) $u_a = 70\%$ (d) $u_a = 90\%$

Figura 4.9. Percentual de descarte de tarefas aperiódicas por tamanho do intervalo de reconfiguração, $u_p = 60\%$

pelo EDF-S. Comparativamente, o teste EDF-T apresenta uma pequena variação em seu desempenho com o aumento do intervalo, pois novas tarefas foram inseridas, aumentando a complexidade do teste.

Sabe-se que, se o intervalo de reconfiguração $[t, t')$ for muito curto, o sistema não será escalonável, porém, quanto maior o tamanho do intervalo de reconfiguração, maior será a quantidade de instâncias envolvidas nos cálculos de escalonabilidade, como visto no gráfico da Figura 4.10, a qual mostra a quantidade de instâncias ativas no intervalo $[t, t')$ para cada ponto plotado na Figura 4.9, tornando assim o cálculo mais complexo e até inviável de ser executado em tempo de execução.

Pode-se perceber que o teste de escalonabilidade baseado em trabalho acumulado apresenta, em geral, um desempenho superior ao do teste de escalonabilidade baseado em seqüenciamento de instâncias.



(a) $u_a = 30\%$ (b) $u_a = 50\%$

(c) $u_a = 70\%$ (d) $u_a = 90\%$

Figura 4.10. Quantidade média de instâncias no intervalo por tamanho do intervalo de reconfiguração, $u_p = 60\%$

4.4 SUMÁRIO

Neste capítulo foram apresentadas duas propostas de teste de escalonabilidade em tempo de execução. Ambas propostas foram realizadas visando a escalonabilidade do sistema num dado intervalo de tempo, viabilizando a conferência da escalonabilidade do sistema em tempo de execução. A primeira proposta se baseia no cálculo do trabalho acumulado de cada instância ativa no intervalo considerado. A segunda é baseada no seqüenciamento de instãncias de tarefas, como se não houvesse preempção.

Ambos os testes foram avaliados via simulação. Os resultados obtidos mostraram que o teste de escalonabilidade baseado em trabalho acumulado apresenta um melhor desempenho. No próxino capítulo, estes testes serão estudados considerando seu uso em conjunto com os mecanismos de reconfiguração.

CAPÍTULO 5

RECONFIGURAÇÃO DINÂMICA: O PROBLEMA DE OTIMIZAÇÃO

Uma forma simplificada de reconfiguração dinâmica consiste na simples rejeição ou descarte de instâncias através de mecanismos de controle de admissão (testes de escalonabilidade em tempo de execução). Esta abordagem detalhada nos capítulos 3 e 4, pode ser útil em aplicações de tempo-real não-críticas devido à tolerância que estes sistemas apresentam em relação à perda de *deadlines* de tarefas. Para alguns sistemas, entretanto, a execução de alguns serviços de forma degradada pode ser uma solução mais adequada. Neste caso, deve-se prover um mecanismo elaborado de reconfiguração dinâmica para o sistema.

Para tanto, neste trabalho, considera-se o modelo onde cada tarefa tem uma ou mais versões que podem ser selecionadas em tempo de execução. Assim, tarefas podem ser periódicas ou aperiódicas, críticas ou não-críticas. Instâncias periódicas podem ser degradadas para permitir a execução de instâncias aperiódicas, provendo então um serviço com menor qualidade. Pode-se, por exemplo, considerar rotinas de recuperação como tarefas aperiódicas críticas, como mencionado no capítulo 1, as quais são ativadas na ocasião da detecção de um erro, quando o mecanismo de reconfiguração deverá ser acionado. Durante a recuperação, o serviço possuirá menor qualidade, porém garantirá a correta execução de todas as instâncias do sistema.

Neste capítulo, serão detalhadas duas propostas de reconfiguração dinâmica. A primeira, apresentada na seção 5.1, utiliza o teste de escalonabilidade baseado no trabalho acumulado, detalhado na seção 4.1, para verificar a escalonabilidade do sistema associado à degradação controlada. A segunda, detalhada na seção 5.2, utiliza o teste de escalonabilidade baseado em seqüenciamento de instâncias. Na seção 5.3, um exemplo simples é utilizado para ilustrar ambas abordagens. Na seção 5.4, soluções aproxima-

das são derivadas. Na seção 5.5, ambas propostas são comparadas por simulação. Por fim, na seção 5.6, será apresentado um resumo do capítulo.

5.1 RECONFIGURAÇÃO BASEADA EM TRABALHO ACUMULADO

O mecanismo de reconfiguração baseado em trabalho acumulado utiliza as condições de escalonabilidade, para cada instância de tarefa $J_i \in \Gamma(t, t')$, definidas na seção 4.1. O problema de reconfiguração dinâmica baseado nestes critérios pode ser assim formalizado:

$$P: \; f = \text{Maximize} \sum_{\forall J_i \in \Gamma(t,t')} \sum_{\forall k \in K_i} A_{i,k} x_{i,k} \tag{5.1a}$$

Sujeito a :

$$w_i \leq e_i - s_i, \; \forall J_i \in \Gamma(t, t') \tag{5.1b}$$

$$s_i = \max(t, r_i) \tag{5.1c}$$

$$e_i = \min(t', d_i) \tag{5.1d}$$

$$s'_i = \max(t, \min_{d_j \leq d_i}(r_j)) \tag{5.1e}$$

$$w_i = w_i^1 - \min(s_i - s'_i, w_i^2) \tag{5.1f}$$

$$w_i^1 = \sum_{\forall J_j \in \Gamma(t,t'): d_j \leq d_i} \sum_{\forall k \in K_j} C_{j,k}(s_j) x_{j,k} \tag{5.1g}$$

$$w_i^2 = \sum_{\forall J_j \in \Gamma(t,t'): d_j \leq d_i, \; s_j < s_i} \sum_{\forall k \in K_j} C_{j,k}(s_j) x_{j,k} \tag{5.1h}$$

$$\sum_{\forall k \in K_i} x_{i,k} = 1, \; \forall J_i \in \Gamma(t, t') \tag{5.1i}$$

$$x_{i,k} \in \{0, 1\}, k \in K_i \tag{5.1j}$$

Onde $A_{i,k}$ é o benefício atingido pela instância k da tarefa τ_i. w_i, w_i^1, w_i^2, s_i, s'_i, e_i e $x_{i,k}$ foram definidos no capítulo 4. Ao escolher a versão mais apropriada para cada instância, o mecanismo de reconfiguração atribui valores às variáveis $x_{i,k} \in$

$\{0, 1\}$, associadas a cada uma das n instâncias de $\Gamma(t, t')$. Se $x_{i,k} = 1$, então J_i foi considerada para ser escalonada no sistema com a versão k. No caso de $C_{i,k} = 0$, esta decisão acarretará no cancelamento ou rejeição de J_i. Obviamente, podem existir tarefas que não podem ser canceladas, o que pode ser modelado, facilmente, fazendo-se $\forall k, C_{i,k} > 0$. Desta forma, escolhe-se exatamente uma versão para cada instância de tarefa, $\sum_{\forall k \in K_i} x_{i,k} = 1$ para toda instância $J_i \in \Gamma(t, t')$, onde $K_i = 1, \ldots, k(i)$.

5.1.1 Formulação Recursiva

Para possibilitar a criação de um algoritmo para resolução do problema, uma formulação recursiva deverá ser desenvolvida. Esta formulação recursiva para o problema de reconfiguração P foi desenvolvida nesta seção, que leva naturalmente a um algoritmo de programação dinâmica. Assume-se a seguinte hipótese, a instância J_j não é mais prioritária do que a instância J_i sempre que $i < j$. Assume-se também que $C_{i,k}$ é um valor inteiro $\forall i, \forall k \in K_i$.

A formulação recursiva define o problema $P_i(w_i)$ como uma versão de P, que engloba apenas as instâncias do conjunto $\mathcal{J}_i = \{J_1, \ldots, J_i\}$ e para a qual o trabalho acumulado das instâncias $\mathcal{J}_i$ é no máximo w_i. $P_i(w_i)$ pode ser escrito recursivamente como uma função de $P_{i-1}(w_{i-1})$ e assim sucessivamente. Deve-se, inicialmente, formular o problema base $P_1(w_1)$, pois a partir deste as demais recursões serão criadas. Como $P_1(w_1)$ é o caso base, ele se refere a instância mais prioritária contida em $\mathcal{J}_i$. Assim, tal instância não sofrerá interferência de outras instâncias. Levando o problema a eliminar a existência de s'_1 e w^2_1, que serão zero. Levando $w_1 = w^1_1$. Desta forma, a única instância considerada neste problema é J_1, a instância mais prioritária. Tal problema é formulado como segue:

$$P_1(w_1): \; f_1(w_1) = \text{Maximize} \sum_{\forall k \in K_1} A_{1,k} x_{1,k} \tag{5.2a}$$

Sujeito a :

$$w_1 \leq e_1 - s_1, \; \forall J_1 \in \Gamma(t, t') \tag{5.2b}$$

$$s_1 = \max(t, r_1) \tag{5.2c}$$

$$e_1 = \min(t', d_1) \tag{5.2d}$$

$$w_1 \geq \sum_{\forall k \in K_1} C_{1,k}(s_1) x_{1,k} \tag{5.2e}$$

$$\sum_{\forall k \in K_1} x_{1,k} = 1, \; \forall J_i \in \Gamma(t, t') \tag{5.2f}$$

$$x_{1,k} \in \{0, 1\}, \; k \in K_1 \tag{5.2g}$$

Como a formulação base já foi realizada, podemos calcular os demais casos de forma recursiva, reduzindo o problema à soma do valor encontrado para instância atual ao valor encontrado para as instâncias anteriores. Lembre que $\mathcal{J}_i$ está ordenado segundo as prioridades das instâncias contidas nele. Assim, o problema de reconfiguração das demais instâncias é formulado da seguinte forma:

$$P_i(w_i): \; f_i(w_i) = \text{Maximize} \sum_{\forall k \in K_i} A_{i,k} x_{i,k} + f_{i-1}(w_{i-1}) \tag{5.3a}$$

Sujeito a :

$$w_i \leq e_i - s_i \tag{5.3b}$$

$$e_i = \min(d_i, t') \tag{5.3c}$$

$$s_i = \max(t, r_i) \tag{5.3d}$$

$$s'_i = \max(t, \min_{d_j \leq d_i}(r_j)) \tag{5.3e}$$

$$w_i \geq w_{i-1} + \sum_{\forall k \in K_i} C_{i,k}(s_i) x_{i,k} - \min(s_i - s'_i, w_{i-1}) \tag{5.3f}$$

$$\sum_{\forall k \in K_i} x_{i,k} = 1 \tag{5.3g}$$

$$x_{i,k} \in \{0, 1\}, \; k \in K_i \tag{5.3h}$$

$$w_{i-1} \in \{0, 1, \ldots, w_i\} \tag{5.3i}$$

Portanto, a solução da família de problema $\{P_i(w_i)\}$ pode ser explicitada conforme recursões abaixo:

$$f_1(w_1) = \begin{cases} -\infty, \text{ se } C_{1,\kappa(1)}(s_1) > w_1 \text{ ou } w_1 > \min(d_1, t') - s_1 \\ \max\{A_{1,k} : w_1 \geq C_{1,k}(s_1), \; k \in K_1\}, \text{ caso contrário} \end{cases}$$

$$f_i(w_i) = \begin{cases} -\infty, \text{ se } w_i > \min(d_i, t') - s_i \\ -\infty, \text{ se } w_i < w_{i-1} + C_{i,\kappa(i)}(s_i) - \min(s_i - s'_i, w_{i-1}), \; \forall w_{i-1} \in \{0, \ldots, w_i\} \\ \max\{A_{i,k} + f_{i-1}(w_{i-1}) : w_i \leq w_{i-1} + C_{i,k}(s_i) - \min(s_i - s'_i, w_{i-1}), \; 0 \leq w_{i-1} \leq w_i \; k \in K_i\}, \\ \quad \text{caso contrário} \end{cases}$$

A derivação de um algoritmo de programação dinâmica, **DP-Solve-P**, é realizada

a partir das recursões acima. Primeiramente, o algoritmo calcula um limite superior w_{max} para o trabalho acumulado da tarefa menos prioritária, J_n, sendo este definido por $w_{max} = \max(w_i^{max} : J_i \in \Gamma(t,t'))$, sendo $w_i^{max} = \sum_{\forall J_j \in \Gamma(t,t'):d_j \leq d_i} C_{j,1}(s_j)$. Lembre que $C_{i,1}(s_i) = \max\{C_{i,k}(s_i) : k \in K_i\}$ para todo $J_i \in \Gamma(t,t')$. Assim, w_{max} define um limite superior para todas as tarefas J_i. Os valores para $f_1(w_1)$ são então calculados para $w_1 = 0, \ldots, w_i^{max}$. Os passos seguintes computam $f_i(w_i)$ para $i = 2, \ldots, n$ conforme expressão dada pela recursão. O algoritmo também utiliza uma tabela $p_i(w_i)$ contendo o nível de degradação e uma tabela $w_i^-(w_i)$ com o valor w_{i-1} da solução do problema $P_i(w_i)$. Estas tabelas auxiliares permitem recuperar os níveis de degradação ótimos para as tarefas. O pseudo-código do algoritmo de programação dinâmica é dado a seguir.

DP-Solve-P$(t, t', n, \kappa, A, C, r, d)$

1: $s_1 \leftarrow \max(t, r_1)$
2: $w_1^{max} \leftarrow \sum_{\forall J_j \in \Gamma(t,t'):d_j \leq d_1} C_{j,1}(s_j)$
3: **for** $w_1 = 0$ to w_1^{max} **do**
4: **if** $C_{1\kappa(1)}(s_1) > w_1 \vee w_1 > \min(d_1, t') - s_1$ **then**
5: $p_1(w_1) \leftarrow 0$
6: $f_1(w_1) \leftarrow -\infty$
7: **else**
8: $p_1(w_1) \leftarrow \arg\max\{A_{1,k} : w_1 \geq C_{1,k}(s_1),\ k \in K_1\}$
9: $f_1(w_1) \leftarrow A_{1,p_1(w_1)}$
10: **end if**
11: **end for**
12: **for** $i = 2$ to n **do**
13: $s_i \leftarrow \max(t, r_i)$
14: $s_i' \leftarrow \max(t, \min_{d_j \leq d_i}(r_j))$
15: $w_i^{max} \leftarrow \sum_{\forall J_j \in \Gamma(t,t'):d_j \leq d_i} C_{j,1}(s_j)$
16: **for** $w_i = 0$ to w_i^{max} **do**
17: $p_i(w_i) \leftarrow 0$
18: $w_i^-(w_i) \leftarrow 0$

```
19:       f_i(w_i) ← −∞
20:       if w_i ≤ min(d_i, t') − s_i then
21:         for w_{i−1} = 0 to w_i do
22:           for k = 1 to κ(i) do
23:             if w_i ≥ w_{i−1} + C_{i,k}(s_i) − min(s_i − s'_i, w_{i−1}) e A_{i,k} + f_{i−1}(w_{i−1}) > f_i(w_i)
                then
24:               p_i(w_i) ← k
25:               w_i^−(w_i) ← w_{i−1}
26:               f_i(w_i) ← A_{i,k} + f_{i−1}(w_{i−1})
27:             end if
28:           end for
29:         end for
30:       end if
31:     end for
32:   end for
33:   return (p, f)
```

O tempo de execução do algoritmo é da ordem $O(nw_{max}^2\kappa_{max})$, onde n é a quantidade de instâncias contidas em $\Gamma(t,t')$, $\kappa_{max} = \max\{\kappa(i) : J_i \in \Gamma(t,t')\}$ e $w_{max} = \max(w_i^{max} : J_i \in \Gamma(t,t'))$, sendo $w_i^{max} = \sum_{\forall J_j \in \Gamma(t,t'):d_j \leq d_i} C_{j,1}(s_j)$. Tal complexidade é pseudo-polinomial, pois w_{max} cresce em função do tempo. A memória necessária para armazenar as tabelas $f_i(w_i)$, $p_i(w_i)$ e $w_i^-(w_i)$ é da ordem de $O(nw_{max})$.

5.2 RECONFIGURAÇÃO BASEADA EM SEQÜENCIAMENTO DE INSTÂNCIAS

O problema de reconfiguração desta seção foi baseado no teste de escalonabilidade proposto na seção 4.2. Desta forma, o problema de reconfiguração dinâmica Q derivado se assemelha ao teste de escalonabilidade que o originou e pode ser representado como se segue:

$$Q : f = \text{Maximize} \sum_{\forall J_i \in \Gamma(t,t')} \sum_{\forall k \in K_i} A_{i,k} x_{i,k} \tag{5.4a}$$

Sujeito a :

$$s_i + \xi_i \leq \min(d_i, t_i), \tag{5.4b}$$

$$s_i = \max(t, r_i), \tag{5.4c}$$

$$\xi_i = \sum_{\forall k \in K_i} C_{i,k}(s_i) x_{i,k}, \tag{5.4d}$$

$$t_i = \begin{cases} t' & \text{se } i = n \\ \min(t_{i+1}, d_{i+1}) - \xi_{i+1} & \text{se } 1 \leq i < n \end{cases} \tag{5.4e}$$

$$\sum_{\forall k \in K_i} x_{i,k} = 1, \forall J_i \in \Gamma(t, t') \tag{5.4f}$$

$$x_{i,k} \in \{0, 1\}, \forall k \in K_i \tag{5.4g}$$

Assim como na seção anterior, esta proposta utiliza algumas variáveis definidas no capítulo 4, como $x_{i,k}$, s_i, r_i, t_i, $C_{i,k}$ e $A_{i,k}$. A variável ξ_i é o tempo computacional necessário para executar a versão k de J_i, escolhida pelo mecanismo de reconfiguração, assim $\xi_i = \sum_{\forall k \in K_i} C_{i,k}(s_i) x_{i,k}$. Desta forma, a condição 5.4b fornece uma condição de escalonabilidade para o sistema. As condições de escalonabilidade determinadas pelo problema Q são restritivas, sendo somente suficientes.

5.2.1 Formulação Recursiva

Nesta seção será apresentada uma formulação recursiva para o problema de reconfiguração Q baseado no qual será apresentada uma solução ótima. Seja $Q_i(t^*)$ uma versão restrita do problema de reconfiguração Q, a qual leva em consideração a série de instâncias de tarefas $\mathcal{J}_i = \{J_1, \ldots, J_i\}$, tal que $\mathcal{J}_i$ deve ter sido escalonada no intervalo de tempo $[t, t^*)$. A meta é decompôr Q em uma série de subproblemas $\langle Q_1(t^*), Q_2(t^*), \ldots, Q_n(t^*) \rangle$, cada qual definido por:

$$Q_i(t^*): \; f_i(t^*) = \text{Maximize} \sum_{\forall J_j \in \mathcal{J}_i} \sum_{\forall k \in K_j} A_{j,k} x_{j,k} \tag{5.5a}$$

Sujeito a :

$$s_j + \xi_j \leq \min(d_j, t_j), \tag{5.5b}$$

$$s_j = \max(t, r_j), \tag{5.5c}$$

$$\xi_j = \sum_{\forall k \in K_j} C_{j,k}(s_j) x_{j,k}, \tag{5.5d}$$

$$\sum_{\forall k \in K_j} x_{j,k} = 1, \quad \forall J_j \in \mathcal{J}_i \tag{5.5e}$$

$$t_j = \begin{cases} t^* & \text{se } j = n \\ \min(t_{j+1}, d_{j+1}) - \xi_{j+1} & \text{se } 1 \leq j < n \end{cases} \tag{5.5f}$$

$$x_{j,k} \in \{0, 1\}, \forall k \in K_j \tag{5.5g}$$

Obviamente, $Q \equiv Q_n(t')$ e então $f = f_n(t')$. Para derivar uma formulação recursiva para $Q_n(t')$, deve-se considerar a série $\mathcal{J}_1$, a qual pode ser expressada como:

$$Q_1(t^*): \; f_1(t^*) = \text{Maximize} \sum_{\forall k \in K_1} A_{1,k} x_{1,k} \tag{5.6a}$$

Sujeito a :

$$s_1 + x_{1,k} \leq \min(d_1, t_1) \tag{5.6b}$$

$$s_1 = \max(t, r_1) \tag{5.6c}$$

$$\xi_1 = \sum_{\forall k \in K_1} C_{1,k}(s_1) x_{1,k} \tag{5.6d}$$

$$\sum_{\forall k \in K_1} x_{1,k} = 1 \tag{5.6e}$$

$$t_1 \leq t^* \tag{5.6f}$$

$$x_{1,k} \in \{0, 1\}, \forall k \in K_1 \tag{5.6g}$$

A equação (5.6d) representa o custo de execução de J_1 na versão escolhida pelo mecanismo de reconfiguração. Não é difícil notar que, quanto mais longe de s_1 for t^*, maior será o benefício para executar J_1. É claro que a escolha deste benefício deve levar em consideração as instâncias $J_i \in \Gamma(t, t')$, $i > 1$. Em outras palavras, para solucionar o problema $Q_1(t^*)$, primeiro se deve considerar o intervalo de tempo $[t, t^*)$ juntamento com ξ_2, o qual é necessário para a execução de J_2. Desta forma, o escalonamento de J_2 não pode ser feito sem observar os efeitos de J_3 com seu custo computacional ξ_3 e assim por diante. Por esta razão, pode-se derivar a formulação recursiva a seguir para os subproblemas $Q_i(t^*)$, $i > 1$.

$$Q_i(t^*): \; f_i(t^*) = \text{Maximize} \sum_{\forall k \in K_i} A_{i,k} x_{i,k} + f_{i-1}(\min(d_i, t_i) - \xi_i) \tag{5.7a}$$

Sujeito a :

$$s_i + \xi_i \leq \min(d_i, t_i) \tag{5.7b}$$

$$s_i = \max(t, r_i) \tag{5.7c}$$

$$\xi_i = \sum_{\forall k \in K_i} C_{i,k}(s_i) x_{i,k} \tag{5.7d}$$

$$\sum_{\forall k \in K_i} x_{i,k} = 1 \tag{5.7e}$$

$$t_i \leq t^* \tag{5.7f}$$

$$x_{i,k} \in \{0, 1\}, \; k \in K_i \tag{5.7g}$$

A formulação para $Q_1(t^*)$ e $Q_i(t^*)$ $(i > 1)$ pode ser expressa de forma mais compacta. A versão k de J_i pode ser escalonada no intervalo de tempo $[t, t^*)$, o que pode ser verificado pela condição $s_i + C_{i,\kappa(i)} \leq \min(d_i, t^*)$:

$$f_1(t^*) = \begin{cases} -\infty, \text{se } C_{1,\kappa(1)}(s_1) > 0 \text{ e } s_1 + C_{1,\kappa(1)}(s_1) > \min(d_1, t^*) \\ 0, \text{se } C_{1,\kappa(1)}(s_1) = 0 \text{ e } s_1 > \min(d_1, t^*) \\ \max\{A_{1,k} : s_1 + C_{1,k} \leq \min(d_1, t^*),\ k \in K_1\}, \qquad \text{caso contrário} \end{cases}$$

$$f_i(t^*) = \begin{cases} -\infty, \text{ se } C_{i,\kappa(i)}(s_i) > 0 \text{ e } s_i + C_{i,\kappa(i)}(s_i) > \min(d_i, t^*) \\ f_{i-1}(t^*), \text{ se } C_{i,\kappa(i)}(s_i) = 0 \text{ e } s_i > \min(d_i, t^*) \\ \max\{A_{i,k} + f_{i-1}(\min(d_i, t^*) - C_{i,k}(s_i)) : s_i + C_{i,k}(s_i) \leq \min(d_i, t^*),\ k \in K_i\}, \\ \quad \text{caso contrário} \end{cases}$$

A formulação descrita pelas equações acima induz um algoritmo de programação dinâmica [62], DP-Solve-Q, o qual é dado a seguir. Como pode ser observado, as linhas 1-13 do algoritmo se referem a $f_1(t^*)$ para $t^* = t, t+1, \ldots, t'$. Os valores de $f_i(t^*)$, $i > 1$, são calculados nas linhas 14-28 para o mesmo intervalo. A tabela $p_i(t^*)$ guarda os valores de $k \in K_i$ que levam ao benefício máximo para cada t^*.

DP-Solve-Q$(t, t', n, \kappa, A, C, r, d)$

1: $s_1 \leftarrow \max(t, r_1)$
2: **for** $t^* = t$ to t' **do**
3: **if** $C_{1,\kappa(1)} > 0 \wedge s_1 + C_{1,\kappa(1)}(s_1) > \min(d_1, t^*)$ **then**
4: $p_1(t^*) \leftarrow 0$
5: $f_1(t^*) \leftarrow -\infty$
6: **else if** $C_{1,\kappa(1)} = 0 \wedge s_1 > \min(d_1, t^*)$ **then**
7: $p_1(t^*) \leftarrow \kappa(1)$
8: $f_1(t^*) \leftarrow 0$
9: **else**
10: $p_1(t^*) \leftarrow \arg\max\{A_{1,k} : s_1 + C_{1,k} \leq \min(d_1, t^*),\ k \in K_1\}$
11: $f_1(t^*) \leftarrow A_{1,p_1(t^*)}$

12: **end if**
13: **end for**
14: **for** $i = 2$ to n **do**
15: $s_i \leftarrow \max(t, r_i)$
16: **for** $t^* = t$ to t' **do**
17: **if** $C_{i,\kappa(i)} > 0 \wedge s_i + C_{i,\kappa(i)}(s_i) > \min(d_i, t^*)$ **then**
18: $p_i(t^*) \leftarrow 0$
19: $f_i(t^*) \leftarrow -\infty$
20: **else if** $C_{i,\kappa(i)} = 0 \wedge s_i > \min(d_i, t^*)$ **then**
21: $p_i(t^*) \leftarrow \kappa(i)$
22: $f_i(t^*) \leftarrow f_{i-1}(t^*)$
23: **else**
24: $p_i(t^*) \leftarrow \arg\max\{A_{i,k} + f_{i-1}(\min(d_i, t^*) - C_{i,k}(s_i)) : s_i + C_{i,k}(s_i) \leq \min(d_i, t^*),\ k \in K_i\}$
25: $f_i(t^*) \leftarrow A_{i,p_i(t^*)} + f_{i-1}(\min(d_i, t^*) - C_{i,p_i(t^*)}(s_i))$
26: **end if**
27: **end for**
28: **end for**
29: retorne (p, f)

O tempo de execução deste algoritmo é da ordem de $O(n(t' - t)\kappa_{max})$, onde n é a quantidade de instâncias contidas em $\Gamma(t, t')$, $\kappa_{max} = \max\{\kappa(i) : J_i \in \Gamma(t, t')\}$. Desta forma, o algoritmo é pseudo-polinomial no tamanho da entrada. A complexidade de memória é $\Theta(n(t' - t))$ usada para manter as tabelas $f_i(t^*)$ e $p_i(t^*)$.

5.3 EXEMPLO ILUSTRATIVO

Esta seção define um simples exemplo para ilustrar os mecanismos de reconfiguração propostos. Considere um intervalo de tempo com início $t = 0$ e final $t' = 234$.

J_i	r_i	s_i	d_i
J_1	0	0	98
J_2	0	0	101
J_3	54	54	234

Tabela 5.1. Instâncias das tarefas ativas em $\Gamma(0, 234)$.

	$C_{n,k}$									
$J_i \backslash k$	1	2	3	4	5	6	7	8	9	10
J_1	31	27,9	24,8	21,7	18,6	15,5	12,4	9,3	6,2	3,1
J_2	91	–	–	–	–	–	–	–	–	–
J_3	22	19,8	17,6	15,4	13,2	11,0	8,8	6,6	4,4	2,2

Tabela 5.2. Níveis possíveis para degradação das tarefas do exemplo.

Neste intervalo, existem três instâncias ativas, ou seja, $\Gamma(0, 234) = \{J_1, J_2, J_3\}$. Os tempos de ativação destas instâncias e seus *deadlines* são descritos na Tabela 5.1.

Os custos computacionais das diferentes versões destas instâncias são descritos na Tabela 5.2. Como pode ser observado, existem dez possíveis versões para J_1 e J_3, enquanto apenas uma para J_2. Mais precisamente, $C_{i,k} = \frac{C_i(\kappa(i)+1-k)}{\kappa(i)}$, $i = 1, \ldots, 3$. Note também que o sistema não admite cancelamento das instâncias ativas, uma vez que $C_{i,\kappa(i)} > 0$ para toda tarefa $J_i \in \Gamma(t, t')$.

É importante salientar que não é possível executar J_1, J_2 e J_3 considerando as versões com mais alta qualidade sem que haja risco de perda de *deadlines*. De fato, como ilustra a Figura 5.1, o escalonador EDF selecionaria J_1 para execução, consumindo 31 unidades de tempo e o tempo restante, $101 - 31 = 70$, não é suficiente para completar a execução de J_2. O sistema deve então selecionar versões apropriadas de J_1 e J_3 de forma a maximizar o benefício do sistema. Pode-se solucionar tal problema, em geral, selecionando uma das 10^2 possibilidades de reconfiguração, porém tal seleção não é viável na prática por ser muito custosa.

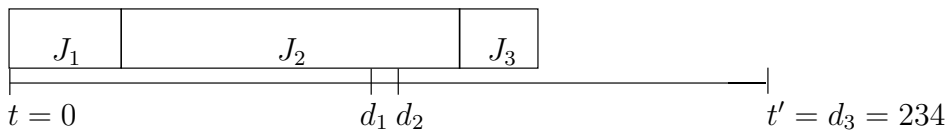


Figura 5.1. Instâncias de $\Gamma(0, 234)$ não escalonáveis no intervalo $[0, 234)$.

J_i	e_i	w_i^1	w_i^2	w_i	nível
J_1	98	9,3	0,0	9,3	$x_{1,8} = 1$
J_2	101	100,3	0,0	100,3	$x_{2,1} = 1$
J_3	234	122,3	100,3	122,3	$x_{3,1} = 1$

Tabela 5.3. Solução para a instância exemplo

De forma simplificada, consideramos que a função $A_{i,k} = \frac{C_{i,k}}{C_{i,1}}$ para os benefícios das instâncias, ou seja, a versão de J_i que produz o maior benefício é aquela com maior custo de execução, $C_{i,1}$. Outras funções benefício são possíveis e não alteram a complexidade das propostas. Utilizou-se uma versão simplificada da função benefício apenas para facilitar a demonstração do exemplo. Caso haja a necessidade de escolher uma versão com menor custo, o valor do benefício associado é reduzido proporcionalmente à redução do valor do custo de execução. Note que outras definições de função benefício são possíveis [12]. Por exemplo, pode-se atribuir valores associados à importância da execução das tarefas para o sistema, o que pode ser definido usando conhecimentos sobre a aplicação em questão.

5.3.1 Mecanismo de Reconfiguração Baseado em Trabalho Acumulado

A resolução da reformulação linear de P produz a solução dada na Tabela 5.3. De acordo com a reconfiguração ótima e segundo a política EDF, a tarefa mais prioritária, J_1, é executada no intervalo $[0,\ 9,3)$, a tarefa J_2 é executada durante o intervalo $[9,3,\ 100,3)$ e, por fim, a tarefa J_3 executa durante o intervalo $[100,3, 122,3)$. Note que as tarefas são ativadas no instante zero e cumprem seus *deadlines*.

Vamos agora ilustrar a aplicação do algoritmo de reconfiguração baseado em trabalho acumulado à instância exemplo. Apesar de, na prática, termos multiplicado os valores $C_{i,k}(s_i)$ por 10 para tornar a instância inteira, vamos apresentar a resolução com valores fracionários para facilitar a compreensão. O valor $w_{max} = 144$. Primeiramente, procuramos o valor w_3^* que induz o máximo benefício $f_3(w_3)$, sendo este dado pelos valores na faixa de $100,3$ até 144 que induzem $f_3(w_3^*) = 2,3$. Com $w_3^* = 144$, a solução ótima de $P_3(w_3^*)$ determina $x_{3,1} = 1$ pois $p_3(w_3^*) = 1$ e define $w_2^* = 100,3$ pois

	$f_i(w_i)$							
$i \backslash w_i$	0	3,1	6,2	30	90	95	98	100
1	$-\infty$	0,1	0,2	0,9	1	1,0	1,0	$-\infty$
2	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	1,1	1,2	1,2
3	∞	$-\infty$	$-\infty$	$-\infty$	$-\infty$	2,1	2,2	2,2

$i \backslash w_i$	100,2	100,3	100,4	105	111	121	131	144
1	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
2	1,2	1,3	1,3	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
3	2,2	2,3	2,3	2,3	2,3	2,3	2,3	2,3

Tabela 5.4. Tabela $f_i(w_i)$.

	$p_i(w_i)$							
$i \backslash w_i$	0	3,1	6,2	30	90	95	98	100
1	–	10	9	2	1	1	1	–
2	–	–	–	–	–	1	1	1
3	–	–	–	–	–	1	1	1

$i \backslash w_i$	100,2	100,3	100,4	105	111	121	131	144
1	–	–	–	–	–	–	–	–
2	1	1	1	–	–	–	–	–
3	1	1	1	1	1	1	1	1

Tabela 5.5. Tabela $p_i(w_i)$.

$w_3^-(w_3^*) = 100,3$. Continuando a busca recursiva, temos que $p_2(w_2^*) = 1$, logo $x_{2,1} = 1$ e também $w_2^-(w_2^*) = 9,3$ implicando $w_1^* = 9,3$. A recursão termina com $p_1(w_1^*) = 8$, fazendo $x_{1,8} = 8$. Os valores $f_i(w_i)$, $p_i(w_i)$ e $w_i^-(w_i)$ são dados nas Tabelas 5.4, 5.5 e 5.6, respectivamente. Observe que o algoritmo de programação dinâmica produziu a solução ótima dada na Tabela 5.3.

Desta forma, pode-se perceber que este algoritmo é eficiente, porém apresenta uma fonte de pessimismo na determinação do intervalo (ver seção 4.1), a mesma apresentada

	$w_i^-(w_i)$							
$i \backslash w_i$	0	3,1	6,2	30	90	95	98	100
1	–	–	–	–	–	–	–	–
2	–	–	–	–	–	3,1	6,2	6,2
3	–	–	–	–	–	94,1	97,2	97,2

$i \backslash w_i$	100,2	100,3	100,4	105	111	121	131	144
1	–	–	–	–	–	–	–	–
2	6,2	9,3	9,3	–	–	–	–	–
3	97,2	100,3	100,3	100,3	100,3	100,3	100,3	100,3

Tabela 5.6. Tabela $w_i^-(w_i)$.

	$f_i(t^*)$						
$J_i \backslash t^*$	0	10	20	60	120	180	234
J_1	$-\infty$	0,3	0,6	1,0	1,0	1,0	1,0
J_2	$-\infty$	$-\infty$	$-\infty$	$-\infty$	1,3	1,3	1,3
J_3	$-\infty$	$-\infty$	$-\infty$	$-\infty$	2,2	2,3	2,3

Tabela 5.7. $f_i(t^*)$ para o exemplo ilustrativo.

	$p_i(t^*)$							
$J_i \backslash t^*$	0	10	20	60	100	120	180	234
J_1	0	8	5	1	1	1	1	1
J_2	0	0	0	0	1	1	1	1
J_3	0	0	0	0	9	1	1	1

Tabela 5.8. $p_i(t^*)$ para o exemplo ilustrativo.

no teste de escalonabilidade que o originou.

5.3.2 Mecanismo de Reconfiguração Baseado em Seqüenciamento de Instâncias

A aplicação do algoritmo **DP-Solve-Q** no exemplo ilustrativo da seção 5.3 leva às tabelas 5.7 e 5.8, as quais contém valores para $f_i(t^*)$ e $p_i(t^*)$, respectivamente. Para ilustração, somente alguns valores serão mostrados nas tabelas. Seja $p_3(234) = 1$, o que implica $x_{3,1} = 1$, ou seja, a versão de mais alta qualidade de J_3 é selecionada para execução. Desta forma, o intervalo de tempo restante $\min(234, d_3) - C_{3,1} = 234 - 22 = 212$ é considerado para a execução das outras duas instâncias. O benefício atingido pelo sistema para J_3 é $A_{3,1} = 1$. Então, o sub-problema $Q_2(212)$ é solucionado. Como $p_2(212) = 1$, $x_{2,1} = 1$ e a versão de mais alta qualidade de J_2 inicia sua execução em $\min(212, d_2) - C_{2,1} = 101 - 91 = 10$ atingindo o benefício $A_{2,1} = 1$. Então o sub-problema $Q_1(10)$ é solucionado. Como $p_1(10) = 8$ e $x_{1,8} = 1$, J_1 inicia sua execução no instante $\lfloor \min(10, d_1) - C_{1,8} \rfloor = \lfloor 10 - 9,3 \rfloor = \lfloor 0,7 \rfloor = 0$, contribuindo com benefício $A_{1,8} = 0.3$. A solução fornecida pelo algoritmo **DP-Solve-Q** é a mesma obtida pelo problema Q, como indicado na Tabela 5.9.

J_i	ξ_i	t_i	$\min(d_i, t_i)$	$x_{i,k}$
J_1	9,3	9,3	9,3	$x_{1,8} = 1$
J_2	91,0	212,0	101,0	$x_{2,1} = 1$
J_3	22,0	234,0	234,0	$x_{3,1} = 1$

Tabela 5.9. Solução para o exemplo ilustrativo.

5.4 SOLUÇÃO APROXIMADA

A aplicação direta dos algoritmos **DP-Solve-P** e **DP-Solve-Q** fornecem execuções pseudo-polinomiais para ambos. Nesta seção serão derivadas soluções aproximadas para os problemas P e Q, dados seus limites inferiores e superiores.

A idéia principal da aproximação [61] é reduzir os tempos de execução $C_{i,k}$ dividindo-o por uma constante $\alpha > 1$, diminuindo assim o tempo computacional gasto para executar os algoritmos **DP-Solve-P** e **DP-Solve-Q**.

Considere b e c_j constantes e $x_j \in \{0, 1\}$ variável, $j = 1, \ldots, n$. Considere também a seguinte relação:

$$\sum_{j=1}^{n} c_j x_j \leq b\,, \tag{5.8}$$

Suponha que o problema é x_j $(j = 1, \ldots, n)$ tal que a relação (5.8) seja satisfeita. Primeiro, considere a seguinte relação modificada:

$$\sum_{j=1}^{n} \lceil c_j \rceil x_j = \sum_{j=1}^{n} \lceil c_j x_j \rceil \leq \lfloor b \rfloor \Rightarrow \lceil \sum_{j=1}^{n} c_j x_j \rceil \leq \lfloor b \rfloor \tag{5.9}$$

É claro que a solução que satisfaz a relação (5.9) também satisfaz a relação (5.8) (o contrário não é necessariamente verdadeiro). A relação (5.9) permite garantir um modelo de aproximação Q_α para Q e outro P_α para P, ambos como função da constante $\alpha \geq 1$, substituindo:

1) $C_{i,k}(s_i)$ por $\lceil \frac{C_{i,k}(s_i)}{\alpha} \rceil$, $\forall J_i \in \Gamma(t, t')$, $k \in K_i$;

2) r_i por $\lceil \frac{r_i}{\alpha} \rceil$ e d_i por $\lfloor \frac{d_i}{\alpha} \rfloor$, $\forall J_i \in \Gamma(t, t')$; e

3) t' por $\lfloor \frac{t'}{\alpha} \rfloor$ e t por $\lceil \frac{t}{\alpha} \rceil$.

Primeiro, observe que uma solução factível $\mathbf{x}$ para Q_α implica numa solução factível para Q, pois com as operações 1,2 e 3, o problema fica mais restritivo. Assim, o valor da função objetivo $f(\mathbf{x})$ define um limite inferior para a solução ótima de f. Por outro lado, o tempo de execução gasto pelo algoritmo de programação dinâmica para solucionar Q_α é reduzido para $O(\frac{(t'-t)n\kappa_{max}}{\alpha})$. Seguindo o mesmo raciocínio para o problema P, o tempo de execução gasto para solucionar P_α se reduz para $O(\frac{nw_{max}^2\kappa_{max}}{\alpha})$.

Da mesma forma, sabe-se que:

$$\sum_{j=1}^{n} c_j x_j \leq b \Rightarrow \sum_{j=1}^{n} \lfloor c_j \rfloor x_j \leq \lceil b \rceil \tag{5.10}$$

Assim, pode-se derivar uma relação relaxada para cada um dos problemas, P e Q, para uma constante $\alpha \geq 1$, substituindo:

1) $C_{i,k}(s_i)$ por $\lfloor \frac{C_{i,k}(s_i)}{\alpha} \rfloor$, $\forall J_i \in \Gamma(t, t')$, $k \in K_i$;

2) r_i por $\lfloor \frac{r_i}{\alpha} \rfloor$ e d_i por $\lceil \frac{d_i}{\alpha} \rceil$, $\forall J_i \in \Gamma(t, t')$; and

3) t' por $\lfloor \frac{t'}{\alpha} \rfloor$ e t por $\lceil \frac{t}{\alpha} \rceil$.

A relação relaxada apresenta um limite inferior para os problemas, já a relação aproximada apresenta um limite superior.

5.4.1 Comparação entre as soluções aproximadas das propostas de reconfiguração

Variando o valor de α, podemos construir as tabelas abaixo, que relacionam o tempo de execução gasto para solucionar o exemplo ilustrativo da seção 5.3 com o benefício atingido pelas relações de limites inferior e superior. A Tabela 5.10 se refere

ao problema de reconfiguração baseado em trabalho acumulado. A Tabela 5.11 diz respeito aos dados obtidos para o problema de reconfiguração baseado em seqüenciamento de instâncias.

	Fator de arredondamento α				
	1	8	16	32	64
Tempo de Execução	622.080	77.760	38.880	19.440	9.720
Limite Superior	2,3	2,3	2,3	2,4	2,4
Limite Inferior	2,3	2,3	2,3	2,3	2,3

Tabela 5.10. Custo de execução da reconfiguração P e benefício atingido pelos limites, superior e inferior, para o exemplo ilustrativo.

	Fator de arredondamento α				
	1	8	16	32	64
Tempo de Execução	70.200	8.775	4.387	2.193	1.096
Limite Superior	2,3	2,3	2,3	2,4	2,6
Limite Inferior	2,3	2,3	2,3	2,3	$-\infty$

Tabela 5.11. Custo de execução da reconfiguração Q e benefício atingido pelos limites, superior e inferior, para o exemplo ilustrativo.

Como pode ser concluído através das tabelas, a velocidade do algoritmo cresce consideravelmente com o crescimento de α. O tempo de execução é dado em unidades de tempo. Note que os limites, superior e inferior, têm os mesmos custos. Evidentemente, o valor de α tem que ser escolhido levando em consideração o tempo e a memória disponíveis para executar o procedimento de reconfiguração e o nível de aproximação permitido.

Note também que a diferença entre o valor ótimo ($\alpha = 1$) e o valor aproximado ($\alpha > 1$) é menor para o procedimento de reconfiguração baseado em trabalho acumulado. Porém, para este procedimento, o tempo de execução é muito alto em relação ao tempo de execução do procedimento de reconfiguração baseado em seqüenciamento de instâncias.

A fim de constatar o comportamento acima, uma simulação foi realizada para avaliação do efeito de α no tempo de execução das soluções de reconfiguração propos-

tas, usando o mesmo modelo de sistema utilizado nas simulações do capítulo 4. As simulações foram separadas entre as duas propostas de reconfiguração apresentadas.

A Figura 5.2 apresenta os resultados encontrados nas simulações da proposta de reconfiguração baseada em trabalho acumulado. Pode-se observar que o tempo de execução da função $\frac{nw_{max}^2\kappa_{max}}{\alpha}$ para cada intervalo de reconfiguração $[t, t')$ foi medido para $\alpha = 1, 4, \ldots, 32$, obtendo valores $2.970,29$, $784,57$, $378,52$, $185,55$ e $180,63$, respectivamente. O que prova a economia de tempo de execução obtida pelo uso de uma solução aproximada. Por exemplo, entre $\alpha = 1$ e $\alpha = 8$ a economia é de 87% do tempo computacional gasto para realizar o teste.

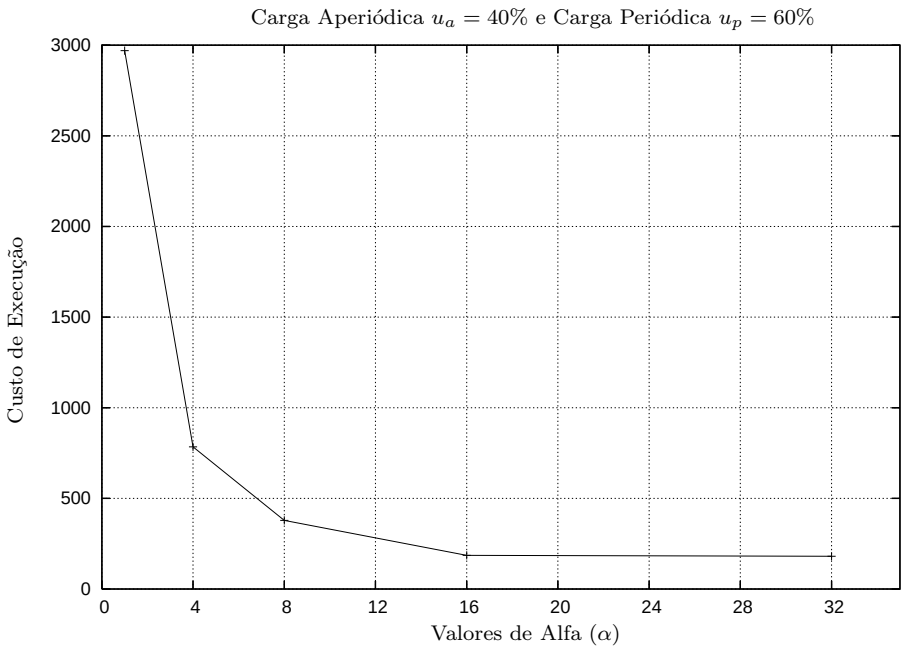


Figura 5.2. Tempo de execução da proposta de reconfiguração baseada em trabalho acumulado usando aproximação pelo limite inferior

A Figura 5.3 apresenta os resultados encontrados durante as simulações da proposta de reconfiguração baseada em seqüenciamento de instâncias. O tempo gasto pela função $\frac{(t'-t)n\kappa_{max}}{\alpha}$ para cada intervalo de reconfiguração $[t, t')$ foi medido durante a simulação. Os valores médios para $\alpha = 1, 4, \ldots, 32$ são $1.520,13$, $385,22$, $205,60$, $138,57$ e $105,04$, respectivamemte. Como pode ser notado, para o tipo de sistema que utilizamos na simulação, $\alpha = 8$ é a opção mais eficiente, considerando a relação "custo

X benefício". De fato, existe uma considerável redução do custo de execução de mais de 86%, girando em torno de 205.

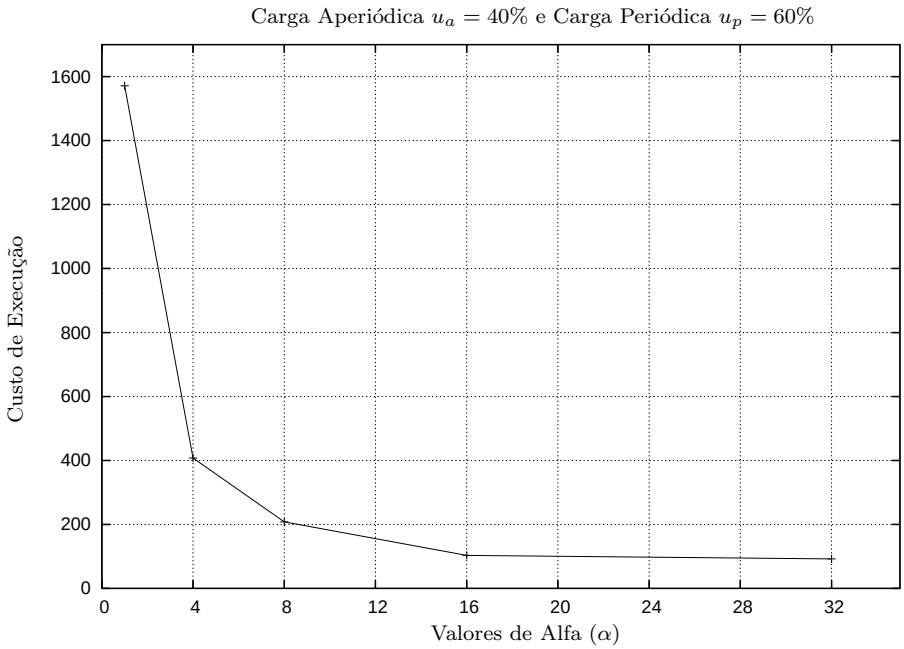
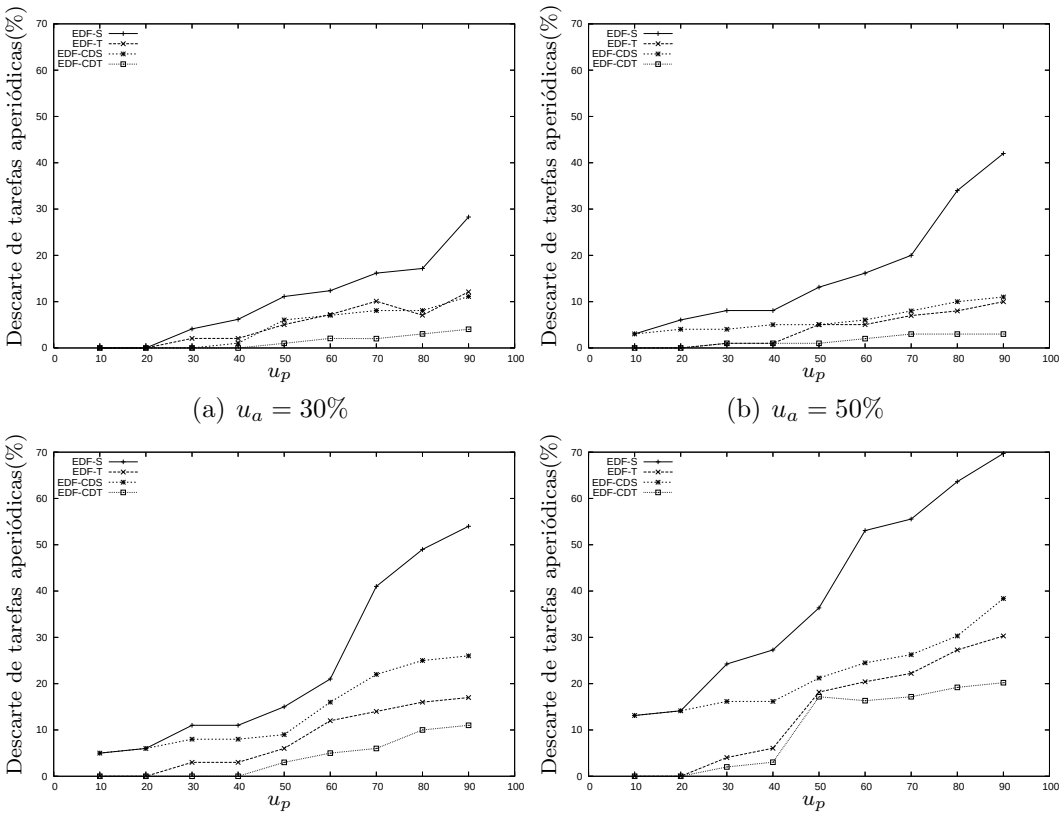


Figura 5.3. Tempo de execução da proposta de reconfiguração baseada em seqüenciamento de instâncias usando aproximação pelo limite inferior

5.5 SIMULAÇÃO COMPARATIVA

Agora que as propostas das seções 5.1 e 5.2 foram detalhadas, serão apresentados os resultados de uma simulação comparativa de ambas. Usando o mesmo modelo utilizado no capítulo 4, mas agora incorporando os mecanismos de reconfiguração propostos, permitindo assim a degradação de instâncias periódicas, foram realizadas simulações para ambas as soluções descritas. Para realizar esta comparação, foram utilizadas quatro alternativas de escalonamento: (a) EDF-S; (b) EDF-CDS; (c) EDF-T; (d) EDF-CDT. O primeiro cenário corresponde ao teste de escalonabilidade baseado em seqüenciamento de instâncias, proposto na seção 4.2, para descartar instâncias aperiódicas que possam vir a causar violação de *deadlines* de instâncias ativas no sistema. O segundo cenário, EDF-CDS, corresponde à proposta de reconfiguração

baseada em seqüenciamento de instâncias da seção 5.2, ou seja, ao algoritmo **DP-Solve-Q**. O cenário EDF-T utiliza o teste de escalonabilidade baseado no trabalho acumulado, proposto na seção 4.1. Por fim, o quarto corresponde ao algoritmo **DP-Solve-P**, ou seja, à proposta de reconfiguração baseada no trabalho acumulado da seção 5.1. Os resultados são os que seguem, considerando utilização aperiódica de $u_a = 30\%$ até $u_a = 90\%$, de 20% em 20%.



(a) $u_a = 30\%$ (b) $u_a = 50\%$

(c) $u_a = 70\%$ (d) $u_a = 90\%$

Figura 5.4. Percentual de descarte de tarefas aperiódicas por carga periódica

Observando os gráficos da Figura 5.4, pode-se notar que a proposta de reconfiguração baseada em seqüenciamento de instâncias apresenta um desempenho inferior ao desempenho da proposta de reconfiguração baseada em trabalho acumulado. Pode-se notar que o desempenho de ambos os testes melhora muito quando estes passam a degradar as instâncias periódicas. Ambos os testes degradam instância por instância, e quando a instância periódica é degradada, a carga periódica decresce. Como visto no capítulo anterior, quanto menor a carga periódica do sistema, melhor é o com-

portamento dos testes. O comportamento da reconfiguração baseada em trabalho acumulado, em geral, é melhor do que o comportamento da reconfiguração baseada em seqüenciamento de instâncias.

5.6 SUMÁRIO

Neste capítulo foram detalhadas duas propostas de reconfiguração dinâmica baseadas nos testes de escalonabilidade do capítulo 4. Para tanto, o sistema considerado possui um conjunto de tarefas, cada uma das quais com múltiplas versões. Tais versões diferem não somente no benefício fornecido ao sistema, como também no custo de execução. Os mecanismos de reconifguração descritos permitem escolher as versões mais apropriadas, maximizando o benefício para o sistema sujeito à sua escalonabilidade.

Para o desenvolvimento dos algoritmos de reconifguração de ambos os testes, utilizou-se a técnica de programação dinâmica. Desta forma, tanto algoritmos ótimos quanto de aproximação foram propostos, simulados e avaliados. Pôde-se perceber que os algoritmos de reconfiguração baseados em trabalho acumulado apresentam melhor qualidade de otimização, mas maior custo de execução quando comparados àqueles baseados em seqüenciamento de instâncias.

CAPÍTULO 6

CONCLUSÃO

Sistemas de tempo-real modernos têm sido aplicados em diversas áreas e têm se tornado cada vez mais complexos. Em diversas situações, há a necessidade de se escalonar tarefas aperiódicas críticas, preservando a correção do sistema como um todo, o que requer mecanismos de reconfiguração dinâmica. Tais mecanismos dão suporte à adaptação dinâmica do sistema, onde a qualidade de serviço é sacrificada em prol da correção do sistema.

Neste trabalho, adaptação dinâmica em sistemas de tempo-real é estudada sob a ótica dos mecanismos de reconfiguração. Assumindo que o sistema é composto de tarefas com múltiplas versões, mecanismos de reconfiguração dinâmica foram propostos. Tais mecanismos incorporam e integram soluções de análise de escalonabilidade e de problemas de otimização. Usando técnicas de programação dinâmica, foi possível demonstrar soluções ótimas para o problema de reconfiguração. Soluções aproximadas, com significativa redução de custo também foram derivadas.

Com relação aos testes de escalonabilidade, duas propostas foram descritas, uma baseada no conceito de trabalho acumulado, definido neste trabalho. A outra é baseada no seqüenciamento de instâncias de tarefas. Após simulá-las, observou-se que a primeiro é bastante superior à segunda em termos dos respectíveis níveis de pessimismo. No entanto, o custo de execução da primeira é mais alto.

Quanto aos mecanismos de reconfiguração, seguiu-se a abordagem clássica de programação. Formulações recursivas do problema foram derivadas, considerando os testes de escalonabilidade propostos. Observou-se que o custo de execução da reconfiguração usando o teste baseado no trabalho acumulado é bastante superior. Desta forma, apesar de mais pessimista, executar a reconfiguração usando o teste de escalonabilidade baseado em seqüenciamento de instâncias pode ser recomendável para

sistemas mais complexos, onde o número de reconfigurações é alto.

É importante salientar que as soluções derivadas neste trabalho seguiram uma abordagem sistemática para derivação dos algoritmos de reconfiguração. Isto permitiu o posterior desenvolvimento das soluções aproximadas do problema de reconfiguração, com custo bastante inferior às soluções ótimas, mas com garantias de qualidade mínimas.

Como continuação do presente trabalho, pode-se enumerar os seguintes tópicos como possíveis extensões:

- Derivação de Heurísticas - Apesar de o uso de programação dinâmica ser justificável, pois se obtém solução ótima para um problema de razoável complexidade, o uso de heurísticas, com custo computacional baixo, é necessário. Os algoritmos desenvolvidos neste trabalho podem servir de base para a derivação de tais heurísticas.

- Modelos Mais Complexos de Tarefas - Não foram considerados neste trabalho possíveis interdependências de tarefas. Modelos que consideram compartilhamento de recursos, por exemplo, são necessários para aumentar a aplicabilidade dos métodos desenvolvidos. Outros aspectos a serem considerados envolvem integrar as soluções com aspectos de isolamento temporal, possivelmente usando servidores de tarefas aperiódicas.

- Modelos Mais Complexos de Reconfiguração - Seria interessante passar a interrelacionar os modos de operação de cada tarefa. Por exemplo, pode haver restrição entre a versão de uma tarefa e de outras. Neste caso, tais restrições devem estar incorporadas no modelo de otimização.

Em resumo, acreditamos que os resultados do presente trabalho servirão de base para que estes e outros tópicos relacionados possam ser investigados, contribuindo, desta forma, para o aprimoramento de mecanismos de escalonamento de sistemas de tempo-real modernos.

REFERÊNCIAS

[1] Tarek F. Abdelzaher, Ella M. Atkins, and Kang G. Shin. QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control. *IEEE Transactions on Computers*, 49(11):1170–1183, 2000.

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

[3] Luca Abeni, Giuseppe Lipari, and Giorgio C. Buttazzo. Constant bandwidth vs proportional share resource allocation. In *ICMCS, Vol. 2*, pages 107–111, 1999.

[4] Björn Andersson and Cecilia Ekelin. Exact admission-control for integrated aperiodic and periodic tasks. pages 225–241. Elsevier, 2007.

[5] David M. Auslander. What is mechatronics? *IEEE Transaction On Mechatronic*, 1(1), March 1996.

[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, July 2005.

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

[8] Guillem Bernat, Alan Burns, and Albert Llamosi. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321, 2001.

[9] Philip Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery In Database Systems.* Computer Science. Addison Wesley, 1987.

[10] R. Bettati, N. Bowen, and J. Chung. Checkpointing imprecise computation. *IEEE Workshop on Imprecise and Approximate Computation*, 1992.

[11] A. Burns, R. I. Davis, and S. Punnekkat. Feasibility Analysis of Fault-Tolerant Real-time Task Sets. In *Proc. of the Euromicro Real-Time Systems Workshop*, pages 29–33, 1996.

[12] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Stringini. The Meaning and Role of Value in Scheduling Flexible Real-Time Systems. *Journal of Systems Architecture*, 46:305–325, 2000.

[13] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages.* Addison Wesley, third edition, 2001.

[14] Giorgio Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.

[15] Giorgio Buttazzo and L. Abeni. Adaptive Workload Management through Elastic Scheduling. *Real-Time Systems*, 23(1-2):7–24, 2002.

[16] Giorgio Buttazzo and J. Stankovic. RED: Robust Earliest Deadline Scheduling. In *Proc. of The Third International Workshop on Responsive Computing Systems*, pages 100–111, 1993.

[17] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Trans. Comput.*, 51(3):289–302, 2002.

[18] Marco Caccamo and Giorgio Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. pages 330–339, December 1997.

[19] Marco Caccamo, Giorgio Buttazzo, and L. Sha. Capacity sharing for overrun control. *Proceedings of the IEEE Real-Time Systems Symposium*, December 2000.

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

[21] Marco Caccamo, Giuseppe Lipari, and Giorgio Buttazzo. Sharing resources among periodic and aperiodic tasks with dynamic deadlines. In *RTSS '99: Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 284–293, Washington, DC, USA, 1999. IEEE Computer Society.

[22] Marco Caccamo, Giuseppe Lipari, and Giorgio C. Buttazzo. Sharing resources with the tb* server. In *IEEE Real-Time Systems Symposium*, 1999.

[23] T. Chantem, X. Sharon Hu, and M. D. Lemmon. Generalized elastic scheduling. In *RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 236–245, Rio de Janeiro, RJ, Brazil, 2006. IEEE Computer Society.

[24] Houssine Chetto and Maryline Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Trans. Softw. Eng.*, 15(10):1261–1269, 1989.

[25] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. mit-ee-cs. mit-pr, 1990.

[26] Rômulo Silva de Oliveira. Computação Imprecisa. *Revista de Informática Teórica e Aplicada - RITA*, 4(1), 1997.

[27] Z. Deng, J. W.-S. Liu, and J. Sun. A scheme for scheduling hard real-time applications in open system environment. In *Proceedings of Ninth Euromicro Workshop on Real-Time Systems*, Toledo, Spain, 1997. IEEE Computer Society.

[28] Wu-Chun Feng. Applications and extensions of the imprecise-computation model. Technical Report UIUCDCS-R-96-1951, 1996.

[29] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, April 1995.

[30] D. Hull, A. Shankar, K. Nahrstedt, and J. Liu. An End-to-End QoS Model and Management Architecture. In *Proc. of IEEE Workshop on Middleware for Distributed Real-time Systems and Services*, 1997.

[31] John P. Lehoczky Jay K. Strosnider and Lui Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, pages 73–91, 1995.

[32] J. Jehuda and A. Israeli. Automated Meta-Control for Adaptable Real-Time Software. *Real-Time Systems*, 14(2):107–134, 1998.

[33] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal (British Computer Society)*, 29(5):390–395, 1996.

[34] G. Koren and D. Shasha. Dover: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM J. Comput.*, 24(2):318–339, 1995.

[35] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. Technical Report TR1996-715, 1996.

[36] 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.

[37] Tei-Wei Kuo and Aloysius K. Mok. Incremental reconfiguration and load adjustment in adaptive real-time systems. *IEEE Trans. Comput.*, 46(12):1313–1324, 1997.

[38] J. Lehoczky and S. Ramos-Thuel. Scheduling periodic and aperiodic tasks using the slack stealing algorithm. In *Advances in Real-Time Systems*, pages 175–198. Prentice-Hall, 1995.

[39] G. M. A. Lima and A. Burns. An Optimal Fixed-Priority Assignment Algorithm for Supporting Fault Tolerant Hard Real-Time Systems. *IEEE Transactions on Computers*, 52(10):1332–1346, 2003.

[40] Caixue Lin and Scott Brandt. Improving soft real-time performance through better slack reclaiming. *26th IEEE International Real-Time Systems Symposium*, 2005.

[41] G. Lipari and S. Baruah. Greedy reclamation of unused bandwidth in conctant-bandwidth. *In Proceedings of the 12th ECRTS*, pages 193–200, 2000.

[42] 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.

[43] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[44] J. Liu, K. Lin, R. Bettati, D. Hull, and A. Yu. Use of imprecise computation to enhance dependability of real-time systems, 1994.

[45] Jane W. Liu. *Real-Time Systems*. Prentice Hall, first edition, 2000.

[46] N. Malcolm and W. Zhao. Version selection schemes for hard real-time communications. *IEEE Real-Time Systems Symposium*, pages 12–21, 1991.

[47] V. Millan-Lopez, W. Feng, and J. W. S. Liu. Using the imprecise-computation technique for congestion control on a real-time traffic switching element. pages 202–209, Washington, DC, USA, 1994. IEEE Computer Society.

[48] L. Nogueira and L. M. Pinho. Capacity sharing and stealing in served-based real-time systems. *In Proceedings of the 12th IEEE International parallel and Distributed Processing Symposium*, March 2007.

[49] Augusto B. Oliveira, Eduardo Camponogara, and George Lima. Dynamic reconfiguration of constant bandwidth servers. In *Proc. of the 10th Brazilian Workshop on Real-Time and Embedded Systems*, pages 37–46, Rio de Janeiro, RJ, Brasil, 2008. 10th Brazilian Workshop on Real-Time and Embedded Systems.

[50] James A. Ratches, C. P. Walters, Rudolf G. Buser, and B. D. Guenther. Aided and automatic target recognition based upon sensory inputs from image forming systems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(9):1004–1019, 1997.

[51] D. Rosu, K. Schwan, and S. Yalamanchili. FARA- A Framework for Adaptive Resource Allocation in Complex Real-Time Systems. In *Proc. of the 4th IEEE Real-Time Technology and Applications Symposium. (RTTAS 1998)*, pages 79–84, 1998.

[52] 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.

[53] M. Schwarzinger, T. Zielke, D. Noll, M. Brauckmann, and V. von Seelen. Vison-based car-following: Detection, tracking, and identification. *In Proceedings of IEEE Intelligent Vehicle Symposium*, pages 24–29, 1992.

[54] Patrick M. Shriver, Maya B. Gokhale, Scott D. Briles, Dong-In Kang, Michael Cai, Kevin McCabe, Stephen P. Crago, and Jinwoo Suh. A power-aware, satellite-based parallel signal processing scheme. pages 243–259, 2002.

[55] Ana Carolina Sokolonski, George Lima, and Luciano Barreto. Suporte à degradação controlada em sistemas de tempo real. In *Anáis do VIII Workshop de Testes e Tolerância a Falhas*, pages 1–10, Belém, PA, Brasil, 2007. VIII Workshop de Testes e Tolerância a Falhas.

[56] Ana Carolina Sokolonski, George Lima, and Eduardo Camponogara. Dynamic reconfiguration for adaptive multiversion real-time systems. In *Proc. of the 20th IEEE Euromicro Conference on Real-Time Systems (ECRTS 2008)*, pages 1–10, Praga, República Checa, 2008. 20th IEEE Euromicro Conference on Real-Time Systems.

[57] Ana Carolina Sokolonski, George Lima, and Eduardo Camponogara. Online schedulability tests for real-time systems. In *Anáis do X Workshop de Tempo-Real e Sistemas Embarcados*, pages 1–8, Rio de Janeiro, RJ, Brasil, 2008. X Workshop de Tempo-Real e Sistemas Embarcados.

[58] Marco Spuri and Giorgio Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. volume 10, pages 179–210. Real-Time Systems, 1996.

[59] John A. Stankovic. The integration of scheduling and fault tolerance in real-time systems. Technical report, Amherst, MA, USA, 1992.

[60] Jon van Amerongen. Mechatronic design. *Mechatronics*, (13):1045–1066, 2003.

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

[62] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, New York, NY, 1998.