Escalonamento Dinâmico de Tarefas Periódicas e Esporádicas[1]

Sibelius Lellis Vieira[2]

Departamento de Engenharia de Computação e Automação Industrial
FEE – UNICAMP

Orientador - Prof.Dr. Maurício Ferreira Magalhães[3]

Campinas,SP - 1994

[1]Dissertação apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas como requisito parcial para a obtenção do título de Doutor em Engenharia Elétrica.

[2]O autor é Bacharel em Física pela Universidade Federal de Goiás e Mestre em Física pela Universida Estadual de Campinas.

[3]Professor do DCA - FEE - UNICAMP.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Vieira, Sibelius Lelis
V673e Escalonamento dinâmico de tarefas periódicas e esporádicas / Sibelius Lelis Vieira.--Campinas, SP: [s.n.], 1994.

Orientador: Maurício Ferreira Magalhães.
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica.

1. Sistema de tempo real.. I. Magalhães, Maurício Ferreira. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica. III. Título.

Aos meu pais e à Amanda!

What we anticipate seldom occurs;
what we least expect generally happens.

Benjamin Disraeli

Agradecimentos

Ao Maurício, pela orientação, apoio e incentivo durante a elaboração deste trabalho.

Aos amigos do grupo de tempo-real, Adilson, Alencar e Juan, pelas discussões técnicas e pelo apoio.

À Eloísa e à Márcia, sempre dispostas a colaborar.

À Capes, pelo auxílio financeiro concedido através da bolsa.

Ao CNPq/RHAE, pelo financiamento da estadia na Universidade do Texas.

Ao Prof. Mok, pela acolhida na Universidade do Texas.

À FUNAPE/UFG, pelo financiamento parcial de viagem para apresentação de trabalho em congresso.

À todos os amigos da Faculdade de Engenharia Elétrica/Unicamp.

Aos meus colegas e amigos do IMF/UFG, particularmente ao Prof. Thierson, pelas críticas e sugestões.

Resumo

Atualmente, uma abordagem metodológica formal é fundamental para se construir sistemas computacionais de caráter crítico. Estes sistemas, denominados Sistemas de Tempo-Real Crítico(STRC), baseiam-se no fato de que as restrições temporais quantitativas devem ser obedecidas. Tal característica se deve ao fato de que STRCs são, em geral, sistemas dedicados integrados em sistemas maiores que monitoram e controlam um ambiente físico.

Este trabalho está inserido dentro do paradigma de flexibilidade e previsibilidade para STRCs que consistem, respectivamente, na capacidade de adaptação do sistema a um ambiente variável e, ao mesmo tempo, na capacidade de prever que determinados comportamentos, que comprometam a operação do sistema, nunca irão ocorrer. Vamos nos concentar na busca de soluções para testes de escalonabilidade que sejam apropriados para STRCs em ambientes dinâmicos. Portanto, estaremos lidando com sistemas onde as tarefas computacionais têm parâmetros que variam no tempo.

Serão abordados vários modelos de escalonamento de tarefas, tanto periódicas quanto esporádicas, partindo-se de situações mais simples, onde as tarefas são independentes e, portanto, completamente preemptíveis, até situações onde as tarefas podem apresentar relações de precedência.

Inicialmente, investigamos a escalonabilidade de um conjunto de tarefas periódicas com prazo arbitrário, através do fator de utilização. Propomos um teste de escalonabilidade adequado aos propósitos de um escalonamento *on-line* eficiente. A partir daí, verificamos que um servidor esporádico dinâmico pode ser empregado para o teste de escalonabilidade de tarefas em tempo de execução, e apresentamos uma condição suficiente para o escalonamento de tarefas esporádicas em tempo de execução, bem como, reformulamos uma condição necessária e suficiente, de tal forma a atender os requisitos de teste rápido. Através de simulações, analisamos o nível de eficiência das condições suficientes e da condição exata proposta.

Para garantir que as tarefas esporádicas essenciais possam cumprir seus prazos, mesmo em situações de sobrecarga, propomos um algoritmo de escalonamento que garanta que tarefas mais importantes possam ser escalonadas com sucesso. Verificamos que o algoritmo proposto, embora seja computacionalmente rápido, tem uma eficiência muito próxima à eficiência de um algoritmo ótimo para o problema referido.

Por fim, tratamos o caso em que algumas tarefas periódicas mantêm relações de interdependência, de tal forma a produzir um teste global de escalonabilidade para tarefas periódicas que possuem relações de precedência e tarefas esporádicas com grau de funcionalidade diferente, procurando maximizar a utilização da UCP e garantir a escalonabilidade de tarefas essenciais.

Abstract

Nowadays, the design and implementation of Hard Real-Time(HRT) Systems can only be accomplished through a formal methodological approach. These HRT systems are based upon the fact that the quantitative temporal behaviour must be met. Otherwise, severe negative consequences may occur and be spread in the environment. Such behaviour is due to the fact that HRT systems are, generally, embebbed in larger systems that control physical processes.

This work is closely tied to the flexibility and predictability paradigm. This paradigm requires that the HRT system must be flexible enough to adapt to its environment, and, at the same time, that the desired behaviour will be guaranteed to occur, insuring a safe system. Flexibility relates itself to dynamic reconfiguration(on-line), while predictability will guarantee that the behaviour is according to what is expected. In this way, these aspects may conflict with each other.

In this work, we search for schedulability tests that are suitable when dealing with dynamic environment HRT systems. Thus, we allow systems whose computational tasks might have varying parameters, for example, variable worst case execution time. The research core contributes to the scheduling theory of periodic and sporadic tasks, particularly when on-line tests are necessary.

It will be seen several scheduling models of periodic and sporadic tasks, from the very simple assumptions, where tasks are independent and completely preemptible, up to cases where precedence relations occurs.

Initially, we investigate the schedulability of a periodic task set with arbitrary deadlines, through the utilization factor. We propose a schedulability test that is suitable to online scheduling. Then, we drift to a dynamic sporadic server, that can be used as a schedulability test at execution time, and we provide a sufficient condition for sporadic test scheduling as well as redesign an exact condition in order to fullfill the timetable. Through the simulations, we analyse how accurate our condition is working.

In order to guarantee that the most important sporadic tasks will met their temporal constraints, even operating at overload, we propose a scheduling algorithm that will try to guarantee important tasks and not to jeopardize urgent tasks. We show that our algorithm is computationally fast and has a great efficient average.

Finally, we deal with periodic tasks that present precedence relations, such that they can be embebbed in a system with sporadic and independent periodic tasks and the overall schedulability may be tested on-line. So, our thesis is very concerned about flexible environments, as we employ on-line tests, as well as predictable issues, dictated by the applications.

Lista de Figuras

1.1 Esquema Geral de um STRC 3
1.2 Esquema de um Controle de Reator através de Robôs 7
1.3 Esquema de um Controle de Objeto Aéreo 7

2.1 Representação de uma Tarefa Periódica 16
2.2 Representação de uma Tarefa Esporádica 17
2.3 Esquema de Classes de Complexidade 19
2.4 Escalonamento de um Conjunto de Tarefas Periódicas Independentes . 21
2.5 Escalonamento de um Conjunto de Tarefas Periódicas Cooperantes . . 23
2.6 Escalonamento de um Conjunto de Tarefas Periódicas com Taxa Monotônica 24
2.7 Funções de Importância 30

3.1 Escalonabilidade Válida por T_1, T_2 41
3.2 Escalonabilidade Perdida por T_2 41
3.3 Escalonabilidade Perdida com Prazo Arbitrário 43
3.4 Condição $p_1 \leq d_2$ 45
3.5 Condição $p_2 > p_1 > d_2$ 47
3.6 Condição $p_1 \geq p_2$ 48
3.7 Algoritmo Suficiente 55

4.1 Exemplo de um escalonamento utilizando PPA 62
4.2 Espaço Livre Disponível 64
4.3 Espaço Livre da Condição Suficiente 67
4.4 Valores de Eficiência e Complexidade para U=0.5979 72
4.5 Valores de Eficiência e Complexidade para U=0.6167 73
4.6 Valores de Eficiência e Complexidade para U=0.6249 74
4.7 Valores de Eficiência e Complexidade para U=0.7080 75
4.8 Valores de Eficiência e Complexidade para U=0.8750 77
4.9 Algoritmo Global 80
4.10 Algoritmo Controle-Periódico 81

4.11 Algoritmo PP 82
4.12 Algoritmo Controle-Esporádico 83
4.13 Algoritmo ESPExato1 85
4.14 Algoritmo ESPExato2 87
4.15 Especificação Formal do Escalonamento 89

5.1 Função Degrau para Prazo Rígido 99
5.2 Algoritmo de Garantia de Importância 101
5.3 Algoritmo de Busca Exaustiva 104
5.4 Valores de Complexidade de ALG1 até ALG5 108
5.5 Valores de Benefício de ALG1 até ALG4 109
5.6 Valores de Eficiência e Complexidade para Folga 1 110
5.7 Valores de Eficiência e Complexidade para Folga 2 111
5.8 Valores de Eficiência e Complexidade para Folga 3 112
5.9 Valores de Eficiência e Complexidade para Folga 4 113
5.10 Valores de Eficiência e Complexidade para Folga 5 114

6.1 Comunicação entre Duas Tarefas 121
6.2 Comunicação entre Duas Tarefas através do Algoritmo Ótimo 122
6.3 Escalonamento com Reserva de Tempo 124
6.4 Comunicação Direta com Servidora 125
6.5 Comunicação com Tarefas Intermediárias 125
6.6 Algoritmo Controle-Periódico-Cooperante 128

Conteúdo

1 Introdução 1

1.1 Motivação 1

1.2 O que é um Sistema de Tempo-Real Crítico? 1

1.2.1 Práticas Correntes no Desenvolvimento de STRC 9

1.3 Aspectos Relevantes no Desenvolvimento de STRC 11

1.3.1 Especificação e Análise de Requisitos 11

1.3.2 Projeto Físico e Manutenção 12

1.4 Objetivos deste Trabalho 12

1.5 Organização do Trabalho e Contribuições 13

2 Revisão das Principais Técnicas de Escalonamento 15

2.1 Tarefas Periódicas 15

2.2 Definição de Tarefa Esporádica 16

2.3 Escalonamento de Tarefas em Tempo-Real 18

2.3.1 Escalonamento e Complexidade Computacional 18

2.4 Escalonamento de Tarefas Periódicas 20

2.4.1 Tarefas Periódicas Cooperantes 21

2.4.2 Tarefas Periódicas Independentes com Associação Fixa de Prioridade 23

2.5 Escalonamento de Tarefas Esporádicas 24

2.5.1 Garantia Pré-tempo de Execução para Tarefas Esporádicas . . . 25

2.5.2 Teste de Escalonabilidade em Tempo de Execução 26

2.6 Escalonamento de Tarefas Esporádicas com Importância 29

2.6.1 Escalonamento Baseado no Fator Densidade 30

2.6.2 Integração de Importância e Urgência 31

2.6.3 Função Valor de Importância 32

2.7 Comentários Finais 33

3 Escalonabilidade para Tarefas com Prazo Arbitrário **35**
3.1 Introdução . . . 35
3.2 Condições Suficientes para Escalonabilidade com Prazo Arbitrário . . . 37
3.2.1 Condição Suficiente com Bloqueio . . . 38
3.2.2 Condição Suficiente com Variação Uniforme do Prazo . . . 38
3.2.3 Condição Suficiente com Períodos Alterados . . . 39
3.2.4 Condição Suficiente com Análise de Interferência . . . 39
3.3 Condição de Escalonabilidade para um Sistema de 2 Tarefas . . . 40
3.3.1 Casos Particulares . . . 42
3.3.2 Cálculo do Valor Mínimo para o Fator de Utilização . . . 43
3.3.3 Casos Comuns . . . 44
3.3.4 Casos Extremos . . . 47
3.4 Condição de Escalonabilidade para um Sistema de 3 Tarefas . . . 48
3.4.1 Casos Comuns para T_1, T_2 . . . 48
3.4.2 Casos Incomuns para T_1, T_2 . . . 49
3.5 Condição de Escalonabilidade para um Sistema de n Tarefas . . . 50
3.5.1 Valores dos Períodos entre Prazos Adjacentes . . . 51
3.5.2 Períodos Ordenados . . . 53
3.6 Comparação entre as Metodologias . . . 54
3.6.1 Metodologia do Bloqueio . . . 56
3.6.2 Variação Uniforme do Prazo . . . 56
3.6.3 Análise da Interferência . . . 57
3.7 Comentários Finais . . . 58

4 Escalonamento *On-line* de Tarefas Esporádicas **59**
4.1 Introdução . . . 59
4.2 Metodologias *Off-line* para Escalonamento de Tarefas Esporádicas . . . 61
4.2.1 Substituição de Tarefas Periódicas . . . 61
4.2.2 A Abordagem de Disponibilidade de Carga . . . 62
4.2.3 Abordagem de Máximo Tempo Livre . . . 63
4.2.4 Escalonamento por Lista de Quadros Livres . . . 64
4.3 Metodologias para Escalonamento de Tarefas Esporádicas *On-line* . . . 65
4.3.1 Teste Suficiente . . . 65
4.3.2 Condição Exata . . . 67
4.3.3 Avaliação dos Testes Suficiente e Exato . . . 70
4.4 Generalização para um Sistema Esporádico de n Tarefas . . . 76
4.5 Implementação do Sistema . . . 78
4.5.1 Implementação Integrada . . . 82
4.5.2 Escolhas Flexíveis . . . 84
4.6 Formalização da Consistencia da Política de Escalonamento . . . 86

4.6.1 Especificação Formal do Sistema 88
4.7 Comentários Finais 91

5 Tarefas com Nível de Importância **93**
5.1 Introdução 93
5.2 Algoritmos Críticos: Uma Revisão 94
5.2.1 Função Densidade 95
5.2.2 Criticalidade Absoluta e Urgência 96
5.2.3 Nível de Importância Geral 97
5.3 Algoritmos de Importância 100
5.3.1 Algoritmo de Garantia por Ordem de Importância 100
5.3.2 Busca Exaustiva 102
5.4 Modelagem dos Experimentos 105
5.4.1 Satisfação do Conjunto Periódico 105
5.4.2 Metodologia de Geração de Tarefas 106
5.4.3 Aplicando a Abordagem do Menor Prazo 106
5.4.4 Aplicando a Ordem Arbitrária 107
5.4.5 Geração de Tarefas Esporádicas 107
5.5 Resultados e Comentários Finais 110

6 Tarefas com Relação de Precedência **117**
6.1 Introdução 117
6.2 Análise de Escalonabilidade para 2 Tarefas Cooperantes 119
6.3 Análise de Escalonabilidade para k Tarefas Cooperantes 123
6.3.1 Comunicação com Servidor Simples 124
6.3.2 Comunicação Geral entre Tarefas 126
6.3.3 Análise de Escalonabilidade para n Tarefas Gerais 126
6.4 Complexidade do Modelo de Bloqueio Máximo 127
6.5 Análise dos Tempos de Computação e Pontos de Comunicação 129
6.5.1 Múltiplos Pontos de Comunicação 130
6.5.2 Múltiplas Versões de Segmentos 130
6.6 Comentários Finais 132

7 Conclusão **133**
7.1 Introdução 133
7.2 Revisão dos Objetivos 133
7.3 Resultados Obtidos 133
7.4 Sugestões para Continuidade do Trabalho 135

Capítulo 1

Introdução

1.1 Motivação

Sistemas de Tempo-Real Crítico(STRC) são definidos como ambientes computacionais cuja correção depende não apenas dos resultados lógicos da computação, mas também do momento em que estes resultados são produzidos[Sta88b]. Tais sistemas são característicos de aplicações onde se exige o controle de um processo físico. Como exemplo, temos os sistemas de controle e comando, controle de vôo, rastreamento, navegação e sistemas de alta complexidade, como controle de estação espacial e sistemas de defesa aérea. A maior parte dos sistemas computacionais de Tempo-Real Crítico são de propósito específico e de alta complexidade, requerendo um alto grau de tolerância à falhas e são, tipicamente, parte integrante de um sistema maior. Sistemas de Tempo-Real Críticos devem armazenar informações contendo as características das aplicações e do ambiente do qual são parte integrante. Uma grande parte dos STRC atuais baseia-se em um conhecimento adquirido a priori e, portanto, dependente de projetos estáticos. A natureza estática de muitos destes sistemas contribui para seu alto custo e falta de flexibilidade. A próxima geração de STRCs deve ser projetada para ser dinâmica e flexível[SR91].

1.2 O que é um Sistema de Tempo-Real Crítico?

Sistemas de Tempo-Real Crítico são caracterizados pelo fato de que graves consequências podem ocorrer se as propriedades lógicas e temporais do sistema não forem satisfeitas[SR88]. Tipicamente, um Sistema de Tempo-Real consiste de um sistema controlado e um sistema de controle. Por exemplo, em um ambiente fabril, o sistema

controlado é o chão de fábrica com seus robôs, estações de montagem e partes a serem montadas, enquanto o sistema de controle engloba o computador, os processos computacionais e as interfaces homem/máquina que gerenciam e coordenam as atividades do chão de fábrica. Portanto, o sistema controlado pode ser visto como um ambiente com o qual o sistema computacional interage.

Geralmente, o uso de computadores no processo de controle do ambiente é opcional, sendo que sua ação pode ser desabilitada, resultando apenas em uma perda de desempenho. Como exemplo, podemos citar o uso de STRC em navegação de objetos aéreos. Existem, contudo, aplicações onde o uso do computador torna-se mandatório, como, por exemplo, em sistemas de controle de vôo de última geração. Devido às características de determinadas aeronaves a ação humana, somente, não é suficiente para controlar um pouso sem riscos de falha de segurança.

Um sistema de controle interage com seu ambiente baseado em informações disponíveis sobre este, através dos vários sensores ligados ao ambiente. É fundamental que o estado do ambiente visto pelo sistema de controle seja consistente com o estado real do ambiente[SKL85]. Caso contrário, os efeitos sobre o ambiente das atividades do sistema de controle podem ser desastrosos, ou não corresponderem ao desejado. Portanto, o monitoramento periódico do ambiente, bem como o processamento dentro do tempo previsto da informação sensoreada são necessários. A Figura 1.1 ilustra o esquema geral de um STRC. Observamos que o processo controlado interage diretamente com os sensores e os atuadores, gerenciados pelo controlador. Este último executa uma das tarefas do conjunto de tarefas e informa seu estado através do *display*, indicando ao operador a situação do processo.

Os requisitos de correção do sistema em um Sistema de Tempo-Real Crítico também acontecem em função do impacto físico das atividades do sistema de controle sobre seu ambiente[Gar81]. Por exemplo, se um computador controlando um robô não consegue comandar o robô no sentido de pará-lo ou iniciá-lo no tempo certo, o robô pode colidir com outro objeto no chão de fábrica. É desnecessário dizer que tal ação pode resultar em sérias consequências[Lev84].

Na maior parte destes sistemas, as atividades que devem ocorrer de uma forma temporizada coexistem com aquelas que não são temporalmente críticas[RS91]. Uma aplicação de contabilidade ou uma aplicação de longo prazo executando em um computador de uma fábrica computadorizada são exemplos de aplicações temporalmente não críticas. Vamos denotar ambos os tipos de atividades como tarefas e uma tarefa com requisições temporais como tarefa de tempo-real. Na literatura, tarefas de tempo-real são geralmente conhecidas como tarefas críticas. Em condições ideais, o computador deveria executar tarefas críticas de tal forma que cada tarefa satisfaça seu requisito temporal, ao mesmo tempo que tarefas não-críticas seriam executadas

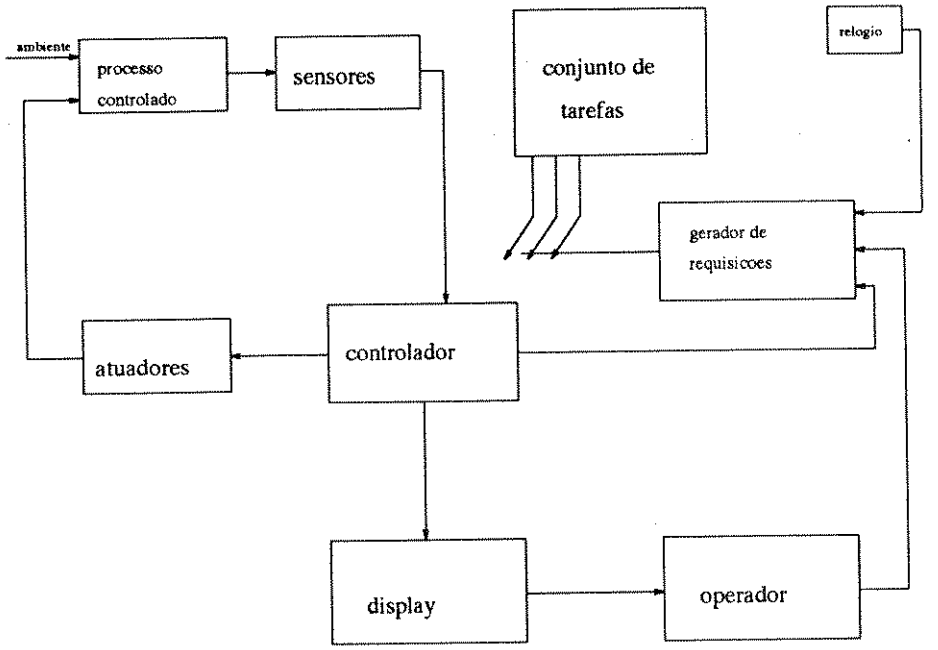

Figura 1.1: Esquema Geral de um STRC

de modo a minimizar o seu tempo médio de resposta. A necessidade de satisfazer os requisitos individualmente de tarefas críticas é uma questão que torna o problema de projeto de STRC um problema complexo[SR88]. Outras questões incluem tolerância a falhas e a necessidade de operar em condições de incerteza. Contudo, vamos nos concentrar nas questões temporais, uma vez que elas permeiam todos as questões do sistema e ambiente.

As características temporais para as tarefas podem ser arbitrariamente complicadas, mas as mais gerais caracterizam as tarefas como periódicas ou aperiódicas[Mok83]. Uma tarefa aperiódica tem um prazo que deve regular a terminação ou início da tarefa, ou pode ter uma restrição no ínicio e tempo máximo para terminação. No caso de tarefas periódicas, o período pode significar uma execução completa.

Tarefas de aplicação de baixo nível, tais como as que obtêm informação dos processos através de sensores, ou aquelas que ativam elementos no ambiente, tipicamente têm restrições temporais rígidas ditadas pelas características do ambiente. Uma grande parte do processamento de monitoração é periódico por natureza. Por exemplo, um radar que monitora vôos produz dados à uma taxa fixa. Um monitor de temperatura de um reator nuclear deve ler periodicamente a temperatura do líquido resfriante do reator para detetar qualquer mudança rapidamente. Algumas destas tarefas periódicas podem existir desde a inicialização do sistema, enquanto outras devem ser criadas dinamicamente. O monitor de temperatura é uma instância de uma tarefa permanente. Um exemplo de uma tarefa que pode ser criada dinamicamente é uma tarefa que monitora um vôo particular. Esta tarefa passa a existir a partir do momento que a aeronave entra na região de controle aéreo e cessa sua atividade quando a aeronave deixa a região. Portanto, em ambientes estáticos, o STRC tem conhecimento prévio de grande parte dos parâmetros da aplicação, o que não acontece quando o ambiente é dinâmico.

Tipos mais complexos de restrição temporal podem ocorrer. Por exemplo, um jato de pintura de carro deve ser disparado entre os tempos t_1 e t_2. Requisitos aperiódicos podem advir de eventos dinâmicos tais como um objeto caindo na frente de um robô em movimento ou um operador humano apertando um botão em um console. Requisitos relacionados ao tempo podem também ser especificados em termos indiretos. Por exemplo, um valor pode ser ligado à terminação de uma tarefa, o qual pode aumentar ou diminuir com o tempo. O valor pode ser indicativo da qualidade de uma resposta, por exemplo, uma resposta inacurada mas rápida, pode ser considerada mais importante que uma resposta lenta que seja mais precisa. Em outras situações, a perda de X prazos pode ser tolerada, mas não a perda de $X + 1$ prazos.

Esta discussão nos leva à questão do que acontece quando as restrições temporais não são satisfeitas. A resposta depende, na maior parte das vezes, do tipo de aplicação.

É desnecessário dizer que um sistema de tempo-real que controla uma usina nuclear ou um míssel, não pode deixar de satisfazer suas retrições temporais. Os recursos necessários para tarefas muito críticas, em tais sistemas, devem ser prealocados de tal forma que as tarefas possam executar sem atraso. Em casos extremos, contudo, alguma ação de exceção pode ser tomada[LC89]. Por exemplo, em uma fábrica automatizada, se for estimado que o comando correto para um robô não poderá ser gerado em tempo, pode ser apropriado fazer com que o robô pare(desde que isto não cause um tipo diferente e indesejável de desastre). Isto é, uma instância de uma tarefa em tempo-real produzindo um resultado de qualidade baixa, mas em tempo hábil. No caso de uma tarefa periódica monitorando uma aeronave, dependendo da trajetória da mesma, a perda de um ou dois processamentos de leitura pode não causar problemas graves.

Em resumo, Sistemas de Tempo Real Crítico diferem dos sistemas tradicionais nos padrões de temporização ligados às tarefas, na capacidade do sistema de assumir compromissos de execução dentro de intervalos bem definidos e na possibilidade de consequências desastrosas advindas do não cumprimento das exigências temporais. Isto implica em uma forte correlação entre desempenho e correção, diferentemente de sistemas tradicionais. Portanto, Sistemas de Tempo-Real Crítico resolvem os problemas de perda de prazo de forma dependente da aplicação alvo. Contudo, deve ser dito que quanto mais cedo o sistema determine que um prazo será perdido, maior flexibilidade ele tem para lidar com a exceção pertinente.

Além de restrições temporais, uma tarefa pode também possuir os seguintes tipos de restrições e requisitos:

- Restrições de Recursos: Uma tarefa pode requerer acesso a certos recursos além da própria UCP, como dispositivos de E/S, estruturas de dados, arquivos e base de dados[ZRS87].
- Relações de Precedência: Uma tarefa complexa, por exemplo, requerendo acesso à vários recursos, é mais facilmente manipulada se for dividida em subtarefas múltiplas relacionadas por condições de precedência e cada uma requisitando um subconjunto dos recursos[XP90].
- Restrições de Concorrência: tarefas deveriam ter permissão para acessar recursos desde que a consistência de acesso aos recursos não seja violada[Kuo91].
- Importância: dependendo da funcionalidade de uma tarefa, o cumprimento do prazo pode ter uma importância relativa para o sistema. Por exemplo, uma tarefa que reage à uma situação de emergência, tal como fogo no chão de fábrica é, sem sombra de dúvida, mais importante que uma tarefa que controla os movimentos de um robô sob condições normais de operação[SR91].

Características das várias tarefas de aplicação são usualmente conhecidas a priori e podem ser escalonadas estaticamente ou dinamicamente. Embora a especificação estática do escalonamento seja o caso típico para tarefas periódicas, o contrário é verdade para tarefas esporádicas. Quando o monitor de temperatura do nosso exemplo prévio do reator nuclear sente algum problema no núcleo do reator, ele invoca outra tarefa(aperiódica) para ativar os elementos apropriados do reator para corrigir o problema, por exemplo, forçando a entrada de mais líquido resfriante no núcleo. Por outro lado, o prazo de uma tarefa que controla o robô em um chão de fábrica pode ser determinado dinamicamente dependendo da velocidade, direção e posição do robô. O comando que deve ser dado ao robô no sentido de forçá-lo a virar ou a parar deve ser gerado antes do tempo crítico, caso contrário, situações indesejáveis poderão ocorrer.

Como exemplo prático de STRC, podemos citar o controle de um processo de leitura de formas de duto em um núcleo de um reator, feito através de robôs, e o controle do sistema de navegação de objetos aéreos. Ilustramos o processo de STRC aplicado em robótica, onde um robô mede a forma dos dutos dentro de um reator nuclear circundando o núcleo através de sensores. Um conjunto de tarefas periódicas simplificado ilustra os aspectos mais importantes deste processo, como leitura e comando do servomecanismo, processamento, e sistema de medida. A Figura 1.2 ilustra o esquema básico de um controlador de reator através de robôs. Em outro exemplo, temos um sistema de navegação de objetos aéreos, mais especificamente o subsistema de comunicação, que fornece informação de posição ao usuário. A Figura 1.3 apresenta aspectos de um controle de objetos aéreos.

Em um sistema estático, as características do sistema controlado são conhecidas a priori e, portanto, a natureza das atividades e a sequência nas quais tais atividades ocorrem podem ser determinadas *off-line*, ou previamente à operação do sistema. É desnecessário colocar que tais sistemas são inflexíveis, mesmo se eles estão em regime de baixo *overhead* em tempo de execução. Na prática, a maioria das aplicações envolve um número de componentes que podem ser estaticamente especificados, juntamente com alguns componentes dinâmicos. Se manipulados apropriadamente, um sistema com um alto grau de utilização de recursos e baixo tempo de gerenciamento pode ser produzido para tais aplicações.

Uma larga porcentagem dos Sistemas de Tempo-Real Crítico correntemente implementados são estáticos por natureza, o que provavelmente não será uma solução razoável a ser adotada em ambientes da nova geração, quando flexibilidade e adaptabilidade serão características fundamentais. Isto se dá em função da dimensão e complexidade de tais sistemas, e do fato de que tais sistemas deverão funcionar em ambientes dinâmicos e fisicamente distribuídos. Mais importante, eles deverão ser facilmente sujeitos à manutenção e expansão, devido à sua natureza evolucionária e

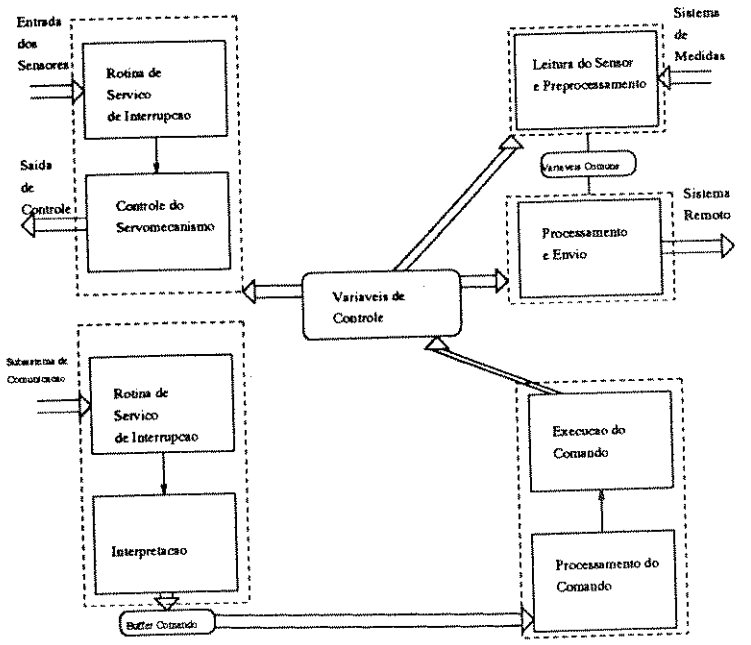

Figura 1.2: Esquema de um Controle de Reator através de Robôs

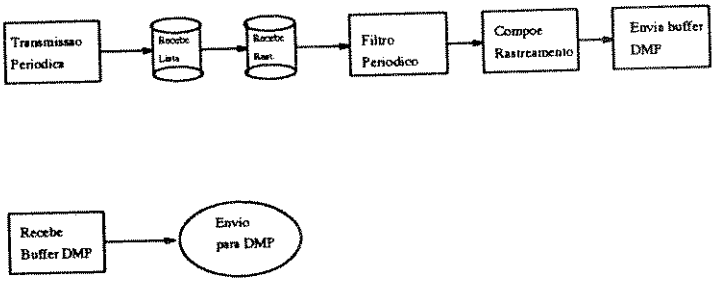

Figura 1.3: Esquema de um Controle de Objeto Aéreo

1.3 Aspectos Relevantes no Desenvolvimento de STRC

O projeto de um STRC envolve um número de questões no que concerne ao ciclo de desenvolvimento de sistemas, que vai desde a especificação informal, baseada nos requisitos, até a manutenção do sistema. Portanto, a análise de projeto de STRC deve ser específica, objetivando as particularidades das restrições temporais que devem ser atendidas. Para tal tipo de análise, vários princípios têm sido delineados, abordando cada fase do ciclo de desenvolvimento do sistema. Particularmente, duas questões merecem atenção: o desenvolvimento de cada fase levando em conta o paradigma de correção temporal quantitativa e a integração das fases do projeto, de modo a fornecer uma base metodológica para a automação completa do processo de desenvolvimento de STRC.

A seguir, descreveremos as fases do desenvolvimento de STRC e seus princípios básicos, bem como sua relação com o paradigma de STRC.

1.3.1 Especificação e Análise de Requisitos

Nos STRC, a especificação formal, o projeto e a análise dos requisitos é dificultada em função da concorrência exibida pelas aplicações de tempo-real, bem como pela presença de restrições quantitativas de tempo[Hen80]. Em termos de programação, os STRC são programas dependentes de tempo de execução(ou tempo-real)[Wir77]. Além das propriedades de correção lógicas que devem ser obedecidas, precisa-se também garantir que as propriedades temporais relacionadas à ocorrência física de um evento sejam satisfeitas. O projeto derivado da especificação deve fornecer garantias de satisfação das propriedades mesmo em condições de falhas do ambiente, de equipamento ou de condições humanas. Como Sistemas de Tempo-Real são críticos também em termos de falhas, a especificação deve levar em conta a existência de tais eventos e deve fornecer mecanismos para a sua recuperação. Além disto, a engenharia de programação pode promover o uso de técnicas de programação que produzam sistemas mais seguros e confiáveis. Em termos de linguagens, os requisitos mínimos para a especificação envolvem a independência de implementação, uma vez que isto facilita a manutenção, e o suporte à execução de programas de tempo-real. A independência de implementação pode ser conseguida com linguagens de alto nível, que além de fornecerem uma capacidade de análise formal, são mais aplicáveis a sistemas heterogêneos. O suporte à execução de programas em tempo-real exige uma limitação na quantidade de tempo de execução de um objeto de tempo-real, seja ele um programa, módulo ou procedimento. Um último fator importante como característica de uma linguagem, envolve

a capacidade de expressar as restrições temporais, inclusive o suporte as decisões do núcleo de tempo-real, dentro da própria linguagem[AC93].

1.3.2 Projeto Físico e Manutenção

Como um dos objetivos mais importantes no projeto de um STRC é a previsibilidade, a questão da escalonabilidade é a chave mestra do projeto de tempo-real. Neste sentido, o sistema operacional e a arquitetura básica devem cooperar de modo a proporcionar uma integração entre o escalonamento de UCP e a disponibilidade de recursos de tal forma que, mesmo em condições de pior caso, as requisições temporais sejam garantidas. O paradigma de núcleos de tempo-real deve ser responsável por garantir que o gerenciamento de memória, arquivos e acesso dos recursos, de forma geral, não sejam arbitrariamente longos. A presença de falhas nos acessos deve ser considerada e a arquitetura para suportar um sistema operacional com este objetivo deve ser dedicada. Com a tendência de distribuir as aplicações, estimulada pela existência de um ambiente de processos fisicamente distribuídos[ACa92], cria-se também a necessidade de utilizar protocolos de comunicação cujo tempo de transmissão segura de mensagens seja previsível[ZR87]. A integração destes protocolos com o núcleo do sistema operacional, módulos de recursos e de aplicações, em condições de falha, é uma questão ainda aberta.

1.4 Objetivos deste Trabalho

Este trabalho tem como objetivo fornecer um conjunto de técnicas de escalonabilidade para vários tipos de tarefas que compõe um STRC. Os diferentes tipos de tarefas compartilham uma característica: são tarefas dinâmicas, no sentido de que sua existência pode ser transitória, e seu momento de criação indeterminado de antemão. Esta característica reforça o requisito de adaptabilidade dos STRCs, mas impõe uma séria dificuldade na garantia de previsibilidade, uma vez que o sistema se torna extremamente sujeito a sobrecargas inesperadas. O nosso objetivo é garantir um mínimo de previsibilidade para STRC através de testes de escalonabilidade que possam ser efetuados em tempo de execução, sem afetar o desempenho do sistema e que apresentam resultados satisfatórios.

Os tipos de tarefas abordados neste trabalho serão compostos de tarefas periódicas e esporádicas dinâmicas, como foi observado. As tarefas periódicas poderão conter relação de precedência ou não, dependendo da aplicação. Apresentaremos uma metodologia para a escalonabilidade de tarefas periódicas com relação de precedência que

se encaixa no comportamento desejado para STRCs adaptáveis.

Um número muito grande de aplicações de tempo real consiste em um conjunto de tarefas periódicas e esporádicas independentes e dinâmicas. O cumprimento do prazo para todas as tarefas em tais circuntâncias é difícil de ser garantida, mas é possível de se prever, em tempo de execução, quais as tarefas que deverão cumprir seus prazos, baseando-se em uma política de priorização de tarefas. Apresentaremos uma proposta de escalonamento e teste de escalonabilidade para esta situação.

Tendo como referências os avanços da teoria do prazo monotônico, inicialmente propomos uma metodologia de escalonamento baseada na associação de prioridades a tarefas que não varia no tempo, e analisamos a escalonabilidade de um conjunto de tarefas periódicas e esporádicas de acordo com esta metodologia. Posteriormente, generalizamos a metodologia para uma associação variável de prioridades, em função do grau de utilização da UCP.

A política de priorização de tarefas pode estar relacionada à importância que a tarefa tem para o sistema. Desta forma, a previsibilidade garante que, em regime de sobrecarga, as tarefas mais importantes serão escalonáveis. É também objetivo deste trabalho a proposta de um algoritmo que permita a garantia desta previsibilidade.

Por fim, estendemos os resultados para aplicações que permitem que tarefas sejam cooperantes, através de acesso comum a variáveis de dados e de controle.

1.5 Organização do Trabalho e Contribuições

Sob o ponto de vista de organização, este trabalho foi dividido em quatro partes distintas, que podem ser integradas em um sistema único. As quatro partes são referentes ao tipo de tarefa constante no modelo.

A primeira parte modela as tarefas periódicas independentes e esporádicas como um conjunto de tarefas periódicas equivalentes, o que reduz o problema da escalonabilidade a tarefas periódicas com prazo arbitrário. Utilizando um algoritmo de prazo monotônico, propomos o teste de escalonabilidade para o conjunto que seja eficaz e rápido.

Na segunda parte, tratamos de tarefas periódicas dinâmicas e independentes em um sistema com tarefas esporádicas. Apresentamos a condição de escalonabilidade para o conjunto total de tarefas e verificamos a sua aplicabilidade através de simulações. Por fim, propomos uma formalização do modelo de escalonabilidade através de uma linguagem de comportamento formal.

Na terceira parte, propomos um algoritmo que visa garantir as tarefas mais im-

portantes quando da possibilidade de ocorrência de sobrecarga, uma vez que, sendo dinâmico, o sistema apresenta variações de demanda de UCP no tempo. A garantia de escalonabilidade das tarefas mais importantes é um requisito fundamental para a previsibilidade, e deve ser uma decisão rápida. Uma comparação com outros algoritmos similares é apresentada.

A quarta parte lida com a questão da escalonabilidade de tarefas periódicas com relações de precedência. Propomos um algoritmo de escalonamento juntamente com um teste de escalonabilidade, compatíveis com os critérios de previsibilidade e flexibilidade. Apresentamos as condições normais através das quais o modelo pode ser empregado.

Capítulo 2

Revisão das Principais Técnicas de Escalonamento

Em Sistemas Operacionais, entendemos como tarefa a representação de um programa computacional sequencial que pode ser caracterizado por seus parâmetros como, por exemplo, o valor corrente dos registradores de máquina, informações de estado, contabilidade e apontadores para área de dados[Tan92]. A sequencialidade da tarefa garante a unicidade de seu fluxo de controle. A comunicação entre tarefas pode ocorrer baseada na semântica da aplicação. Tarefas são processos concorrentes e podem ser cooperantes ou independentes.

Do ponto de vista semântico, uma tarefa está associada a uma funcionalidade única para o sistema. Definimos instância de uma tarefa a cada vez que uma tarefa é ativada para execução[Mok83]. Portanto, todas as instâncias de uma mesma tarefa têm a mesma funcionalidade. Em Sistemas de Tempo-Real Crítico, as instâncias envolvidas devem tratar do controle e monitoramento das operações vitais do sistema, bem como de operações especiais comandadas pelo operador ou por módulos do sistema.

2.1 Tarefas Periódicas

As operações de monitoramento e controle garantem que uma variável representando parte do estado do sistema esteja consistentemente correta, uma vez que tais operações devem ser executadas com uma frequência mínima. Portanto, devem existir tarefas, ou seja, representações de programas que são responsáveis pela atualização e controle do estado do sistema, e que executam em um determinado período. Consideramos tais tipos de tarefas como tarefas periódicas e associamos a elas uma semântica que

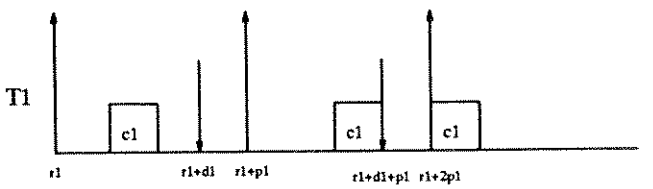

Figura 2.1: Representação de uma Tarefa Periódica

garante a sua execução completa em intervalos de tempo denominados períodos.

Em termos de prazo(*delay*), devemos garantir que uma tarefa periódica possa executar cada uma de suas infinitas instâncias dentro do período correspondente, o que impõe uma ordem de serialidade das instâncias, bem como um tempo de início e prazo bem definidos para cada uma das instâncias. Qualquer que seja a aplicação para o sistema, a tarefa só pode ser escalonada após seu tempo de início(tempo de pronto), e deve terminar sua execução até ,no máximo, o tempo de início da próxima instância. Em termos práticos, os tempos de execução das instâncias são modelados como pior caso, de forma que a garantia de execução vale para qualquer instância em particular.

Formalmente, a tarefa periódica T é caracterizada por uma quádrupla (c, r, d, p), onde c representa o pior caso do tempo de execução, r é o tempo de início da primeira instância da tarefa, d é o prazo relativo ao início de qualquer instância, e p é o período, de tal forma que a instância k será requisitada para executar entre $r + (k-1)p$ e $r + (k-1)p + d$, onde $d \leq p$. O escalonamento de tarefas periódicas será válido se todas as instâncias de todas as tarefas periódicas puderem satisfazer suas requisições temporais. A Figura 2.1 ilustra a representação da Tarefa Periódica $T_1 = (c_1, r_1, d_1, p_1)$.

Até o ponto corrente, descrevemos as tarefas como se elas fossem unidades independentes umas das outras. É também possível especificar uma relação de cooperação entre as tarefas, dependendo da aplicação particular. Questões como exclusão mútua e relações de precedência são possíveis entre tarefas cooperantes. Veremos adiante o tratamento de escalonabilidade para tarefas concorrentes independentes, e posteriormente, descreveremos os métodos para concorrência com cooperação.

2.2 Definição de Tarefa Esporádica

Verificamos que a exigência da satisfação do prazo implica em uma tarefa com prazo rígido, enquanto que a conveniência da satisfação do prazo é característica de tarefas com prazo flexível. Em termos de ativação, além das tarefas periódicas, o sistema

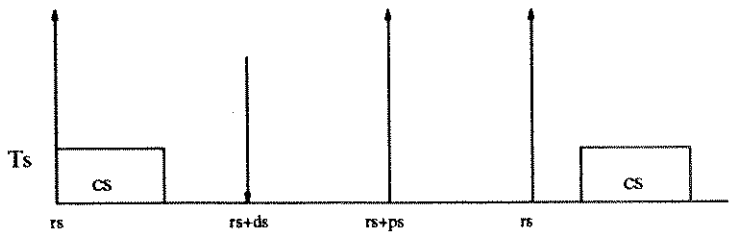

Figura 2.2: Representação de uma Tarefa Esporádica

comporta tarefas cujo tempo de chegada é indeterminado e aleatório. Consideramos uma tarefa como sendo aperiódica se ela tem tal característica, associada a um prazo que preferencialmente deve ser cumprido, embora não seja uma obrigatoriedade. Uma tarefa aperiódica T_a pode ser caracterizada como (c_a, r_a, d_a), onde c_a é o seu tempo de execução, r_a o tempo de chegada e d_a é o prazo relativo. Para tarefas aperiódicas, é importante que o tempo médio de resposta seja minimizado, embora isto nem sempre implique em um cumprimento rígido do prazo[SSL89].

Do ponto de vista semântico, as tarefas aperiódicas representam computações que ocasionalmente serão executadas, em resposta a eventos externos ou internos, cuja repercussão para o sistema não seja fundamental[SSL89]. Por exemplo, a geração de estatística como apoio à manutenção, sob comando do operador, ou a falha em componentes do sistema que não afetem diretamente o seu funcionamento, podem ser vistas como tarefas aperiódicas.

Tarefas aperiódicas com prazo rígido são conhecidas como Tarefas Esporádicas. Formalmente, uma tarefa esporádica é vista como $T_s = (c_s, r_s, d_s, p_s)$, onde c_s, d_s, r_s são parâmetros análogos aos das tarefas aperiódicas. No sentido de se garantir a escalonabilidade das tarefas esporádicas, é necessário haver um intervalo mínimo entre chegadas de duas instâncias da mesma tarefa esporádica. Isto é fornecido pelo fator p_s. A existência de p_s garante que a tarefa esporádica não vai monopolizar os recursos do sistema. A Figura 2.2 ilustra a execução da tarefa esporádica T_s dada acima.

A tarefa esporádica também ocorre aleatoriamente como resultado de requisições externas ou internas. Interrupções de dispositivos, falhas de componentes de hardware e software e intervenção do operador são exemplos comuns de tais eventos. Para a tarefa esporádica, é fundamental que o prazo seja obedecido.

2.3 Escalonamento de Tarefas em Tempo-Real

Um algoritmo de escalonamento especifica uma ordenação de execução para a UCP, baseado nas requisições das tarefas correntemente ativas[Jr77]. Esta ordenação atua sobre uma estrutura de dados conhecida como fila de pronto, que aponta para as tarefas que podem ser executadas em um determinado instante. Em qualquer momento, o processador executa a tarefa de maior prioridade contida na fila de pronto, contanto que a preempção seja permitida. Logo, é função do algoritmo de escalonamento mapear requisições de tarefas em prioridades.

Em Sistemas de Tempo-Real Crítico, as instâncias das tarefas que requisitam serviço tem um prazo determinado para serem servidas. Dizemos que um escalonamento é válido se, para um conjunto de tarefas e um dado algoritmo, este algoritmo consegue escalonar todas as tarefas de tal modo a cumprir todas as restrições temporais do conjunto. Um conjunto de tarefas é factível se existe um algoritmo de escalonamento que produz um escalonamento válido. Quando um determinado algoritmo de escalonamento produz um escalonamento válido, dizemos que o conjunto de tarefas é escalonável sob tal algoritmo[LW82].

No sentido de verificarmos se um conjunto de tarefas é factível, devemos procurar por um escalonamento válido para este conjunto de tarefas. Para isto, empregamos o conceito de algoritmo ótimo. Um algoritmo de escalonamento é ótimo se o escalonamento produzido por este algoritmo for válido, sempre que qualquer outro algoritmo também produz um escalonamento válido para este conjunto de tarefas. Portanto, o algoritmo ótimo deve ser sempre utilizado para o teste de escalonabilidade[LW82].

2.3.1 Escalonamento e Complexidade Computacional

Dadas as características das tarefas de tempo-real, periódicas e esporádicas, faz-se necessário determinar se tais tarefas poderão satisfazer seus requisitos temporais. Uma tarefa satisfaz seus requisitos temporais se todas as suas instâncias são executadas dentro do intervalo assinalado para a execução. Em um sistema com um conjunto de tarefas de tempo-real, é preciso que se associe uma ordem de execução de tarefas, de tal forma a alocar a UCP para tarefas que necessitem de serviço. Em termos de Sistemas Operacionais, o nosso objetivo é encontrar um algoritmo de escalonamento que ordene as tarefas da fila de pronto de acordo com uma política de prioridade que permita às tarefas o cumprimento de seus requisitos temporais.

A teoria de escalonamento de tarefas tem como objetivo encontrar algoritmos eficientes para escalonar com sucesso um conjunto de tarefas, quando isto é possível, bem como a formalização da complexidade de determinados problemas de

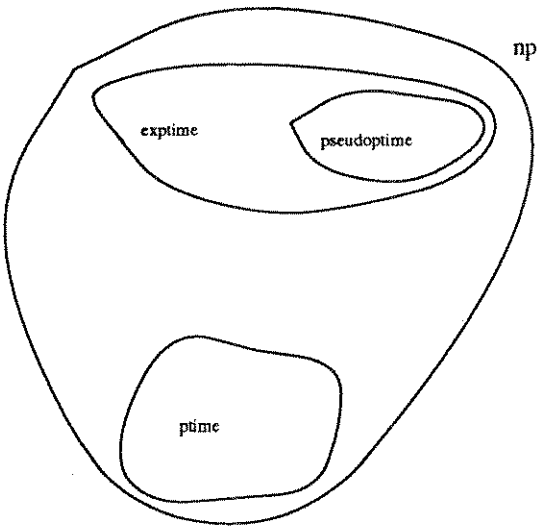

Figura 2.3: Esquema de Classes de Complexidade

escalonamento[Cof76]. Os problemas de escalonamento serão caracterizados neste trabalho, em termos de complexidade como PTIME, PSEUDOPTIME e EXPTIME. Esta terminologia é comumente empregada em sistemas de tempo real[Wan91];[BRH91].

Os problemas PTIME são problemas pertencentes à classe P, de tal forma que suas soluções executam em um tempo polinomial em relação ao tamanho da entrada. Problemas de ordenação são PTIME, por exemplo. Os problemas EXPTIME são problemas pertencentes às classes dos problemas NP-completos e NP-árduos. Não faremos distinção entre estas duas classes aqui, uma vez que suas complexidades são equivalentes. Tais problemas não tem soluções que executam em um tempo polinomial, necessariamente, no tamanho da entrada. Por exemplo, problemas como o achar um mínimo circuito hamiltoniano em um grafo são EXPTIME. Os problemas PSEUDOPTIME são problemas EXPTIME que possuem a propriedade de terem sua explosão controlada pelo conteúdo da entrada[GJ79]. Doravante, vamos considerar que um problemas pertencente a classe PTIME possui um algoritmo PTIME e assim por diante. Ver na Figura 2.3 a representação das classes.

Podemos relacionar a complexidade computacional com algoritmos de escalonamento de tarefas observando que determinados conjuntos de tarefas têm características que permitem que o escalonamento seja encontrado rapidamente. Estas características são relacionadas com o fato das tarefas serem independentes ou não, a relação do prazo com o período, a existência de preempção e não-preempção, entre outras coisas.

Por exemplo, para um conjunto de tarefas periódicas com o prazo igual ao valor do período, permitindo a preempção em qualquer ponto, é possível verificar se existe um escalonamento válido em um tempo linear no número de tarefas deste conjunto[LL73].

2.4 Escalonamento de Tarefas Periódicas

O escalonamento de um conjunto de tarefas periódicas depende diretamente do tipo de associação de prioridade escolhido[LW82]. Em escalonamento dirigido por prioridade, a tarefa com maior prioridade é a próxima a executar na UCP. Portanto, tanto a ordem do escalonamento quanto a escalonabilidade vão estar relacionados com o tipo de associação. Por associação de prioridade fixa(APF), entende-se que cada tarefa recebe um valor de prioridade que será mantido fixo. Quando, por outro lado, usamos associação de prioridade variável(APV), a prioridade de uma tarefa depende do momento corrente, e pode mudar. Também se denomina escalonamento estático e dinâmico, respectivamente, à associação fixa e variável. Nesta seção, vamos ater-nos à associação de prioridade variável.

Pode-se mostrar que se uma APV baseia seu mapeamento de prioridades na política de próximo prazo, ou seja, a tarefa com maior prioridade é a que tem a instância corrente com prazo mais próximo, o algoritmo de escalonamento é ótimo. Outra alternativa de APV que gera uma associação ótima é conhecida como algoritmo de menor folga. Definimos a folga de uma instância no tempo t como sendo a quantidade máxima que esta instância pode ser atrasada de tal forma que a mesma ainda consiga cumprir seu prazo. É fundamental que se empregue uma política ótima no escalonamento de tarefas, quando tal política existe, pois toda a escalonabilidade deve ser testada em relação à esta política, uma vez que se a escalonabilidade é factível através da política ótima, então ela será factível através de qualquer outra política.

Para tarefas independentes, com prazo coincidente com o período($d = p$) para todas as tarefas, e operando na forma de um sistema síncrono($r = 0$), ou seja, todas as tarefas são ativadas simultaneamente, é possível garantir a escalonabilidade do sistema através do teste do fator de utilização, definido como sendo a soma da porcentagem de utilização de todas as tarefas, dentro do seus períodos respectivos. Sendo $T_i = (c_i, r_i, d_i, p_i)$ a i-ésima tarefa do sistema T, que contem n tarefas, definimos o fator de utilização como $U = \sum_{i=1}^{n} c_i/p_i$[LL73]. Se e somente se $U \leq 1$ o sistema acima será escalonável. Este teste é claramente linear no tamanho da entrada e portanto pode ser utilizado em tempo de execução, garantido um baixo valor de *overhead*. A Figura 2.4 ilustra o escalonamento de um sistema com duas tarefas, $\{T_1, T_2\}$, de acordo com o algoritmo de próximo prazo. Neste exemplo, consideramos $T_1 = (c_1, 0, p_1, p_1)$ e $T_2 = (c_2, 0, p_2, p_2)$.

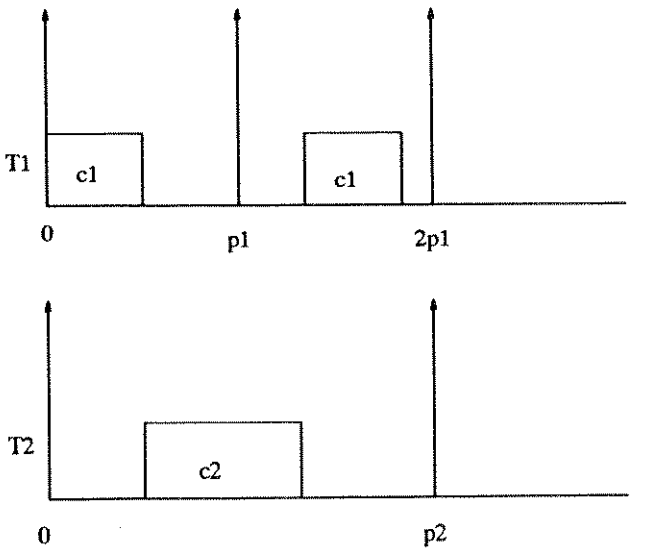

Figura 2.4: Escalonamento de um Conjunto de Tarefas Periódicas Independentes

Para sistemas gerais, onde o prazo pode ser arbitrariamente menor que o período e as tarefas não estão necessariamente sincronizadas na inicialização, o teste de escalonabilidade ainda se baseia na aplicação de um algoritmo ótimo de APV, como o algoritmo de próximo prazo. Contudo, há a necessidade de se testar o escalonamento de todas as instâncias de todas as tarefas em um intervalo que é, no mínimo, duas vezes maior que a janela do sistema periódico[LM80]. Isto impõe uma condição de intratabilidade para o problema que impede a realização do teste em tempo de execução. Algumas alternativas para este teste que impõe uma restrição no tamanho do escalonamento analisado são PSEUDOPTIME e, portanto, inconvenientes para uso em tempo de execução. Apenas no caso em que o número de tipos distintos de tarefas é fixo, podemos resolver o problema da escalonabilidade em tempo polinomial, no caso do sistema ser completo, ou seja, os tempos de início de todas as tarefas são conhecidos.

2.4.1 Tarefas Periódicas Cooperantes

Na ausência de restrições na seleção dos pontos de preempção dos processos, fato comum quando tratamos de tarefas independentes, observamos que existem algoritmos eficientes que podem ser utilizados em tempo de execução. Contudo, quando as tarefas são cooperantes, ou seja, a aplicação de tempo real requer que um conjunto de tarefas

compartilhe dados globais de uma forma concorrente, pode haver a necessidade de se garantir a exclusão mútua ao acesso das seções críticas comuns das tarefas. Esta característica implica na existência de blocos onde a preempção não é permitida e reduz consideravelmente o grau de paralelismo do sistema. Inicialmente, podemos pensar em primitivas de comunicação do tipo semáforo para tratar o problema da seção crítica das tarefas. Sabe-se que o problema de decidir se é possível escalonar um conjunto de tarefas periódicas com um algoritmo de próximo prazo, utilizando semáforos apenas para assegurar a exclusão mútua é intratável[Mok83]. Em geral, a coordenação das tarefas paralelas através de semáforos é pouco estruturada, uma vez que o semáforo pode ser usado tanto para exclusão mútua como para estabelecer relações de precedência.

As primitivas de comunicação baseadas na troca de mensagens estabelece uma relação de precedência entre blocos de uma tarefa e é conveniente para a sincronização entre as tarefas. Esta sincronização é bloqueante, simétrica, e exige que toda a computação anterior à primitiva em todas as tarefas deve preceder as computações posteriores à primitiva, também em todas as tarefas. Este tipo de sincronização bloqueante só tem sentido ocorrer entre tarefas periódicas conhecidas por harmônicas, ou seja, cujos períodos sejam múltiplos, de modo a garantir que os prazos poderão ser obedecidos. Para um conjunto arbitrário de tarefas periódicas, contendo prazos arbitrários, o algoritmo de próximo prazo não é ótimo para o escalonamento, quando a sincronização por troca de mensagens é utilizada. Na verdade, o algoritmo ótimo é uma variação do algoritmo de próximo prazo denominado algoritmo de próximo prazo revisado[Mok83]. A revisão de prazos objetiva a eliminação de vínculos de precedência de tal forma que os prazos revisados automaticamente levem em conta tais restrições de precedência. A análise inclui uma revisão de prazos de todas as instâncias de todas as tarefas, e a construção do escalonamento dentro da janela periódica das tarefas[YTA87]. Na Figura 2.5, ilustramos o escalonamento de duas tarefas cooperantes, $T_1 = (c_1, 0, p_1, p_1)$ e $T_2 = (c_2, 0, p_2, p_2)$, onde $c_1 = c_{11}, c_{12}$ e $c_2 = c_{21}, c_{22}, c_{23}, c_{24}$. A cooperação se dá na forma de relações de precedência entre blocos das tarefas. Por exemplo, podemos dizer que c_{22} deve executar apenas depois de c_{11} e c_{21}.

Outra metodologia para garantir a escalonabilidade de processos cooperantes se baseia na exclusão mútua e relações de precedência dos blocos de tarefas, denominados segmentos, dentro da janela periódica das tarefas[XP90]. As relações entre segmentos são dadas pelos conjuntos de exclusão(EXCLUDE) e precedência(PRECEDE), e o algoritmo encontra um escalonamento válido, se tal existir, que minimiza o atraso de todos os segmentos. Se o atraso mínimo de todos os segmentos é maior que zero, pela definição de atraso, não existe um escalonamento válido para o problema. Caso contrário, o algoritmo vai encontrar o escalonamento válido, em uma quantidade de tempo intratável. Observações empíricas mostraram que o algoritmo tem uma e-

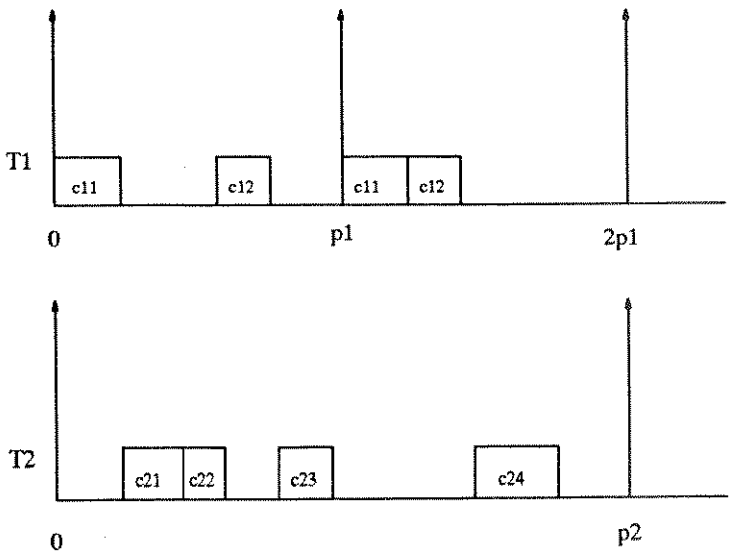

Figura 2.5: Escalonamento de um Conjunto de Tarefas Periódicas Cooperantes

ficiência razoável para uso *off-line*, obtendo uma melhoria em relação a métodos de busca extensiva[MF75].

2.4.2 Tarefas Periódicas Independentes com Associação Fixa de Prioridade

A Associação de Prioridade Fixa(APF), também denominada de escalonamento estático, se traduz pela associação de prioridades às tarefas baseada em uma característica interna da tarefa, que é independente do tempo corrente. Sabe-se que para conjuntos de tarefas periódicas onde o prazo é igual ao período, o algoritmo que atribui maior prioridade às tarefas com menor período é ótimo[LL73]. O teste de escalonabilidade do algoritmo de Taxa Monotônica pode ser suficiente, consumindo um tempo PTIME, ou exato, levando à um procedimento PSEUDOPTIME.

Estamos supondo tacitamente que o sistema é síncrono, ou seja, as tarefas periódicas têm início simultâneo. Para prazos arbitrários, a APF ótima ainda se baseia no prazo, e é conhecida como Prazo Monotônico[LW82]. Esta APF associa a prioridade maior para tarefas com menor prazo. Quando o prazo e o período são iguais, o algoritmo de Prazo Monotônico se transforma em Taxa Monotônica. Em termos de escalonabilidade, sabe-se que existe um algoritmo que tem complexidade PSEUDOPTIME para decidir se um sistema síncrono é escalonável em um processador. Em

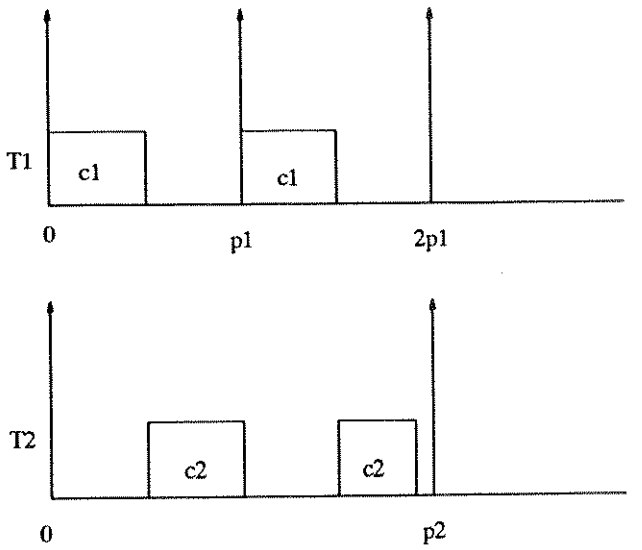

Figura 2.6: Escalonamento de um Conjunto de Tarefas Periódicas com Taxa Monotônica

termos de sistemas assíncronos, não foi encontrado ainda nenhum algoritmo ótimo. Neste caso, o problema de decisão de escalonabilidade é intratável, salvo em casos especiais. Na Figura 2.6, o conjunto de tarefas $T = \{T_1, T_2\}$ é escalonado através do algoritmo de Taxa Monotônica. Como T_1 tem sempre prioridade maior que T_2, verificamos que T_1 executa sempre que necessitar, inclusive preterindo T_2.

Para o teste de escalonabilidade *on-line*, é importante que a decisão seja PSEUDOPTIME, no pior caso, desde que existam garantias de limitação de que a complexidade não vai evoluir para a intratabilidade. Algumas condições suficientes para o teste de escalonabilidade foram desenvolvidas, baseando-se em interferências entre tarefas. Uma tarefa qualquer sofre interferência de todas as tarefas mais prioritárias e, portanto, seu prazo pode ser perdido apenas se tarefas mais prioritárias prejudicarem sua computação. O processo de cálculo de interferências pode ser PTIME, PSEUDOPTIME ou intratável, dependendo do nível de precisão que se quer estabelecer.

2.5 Escalonamento de Tarefas Esporádicas

O escalonamento de tarefas esporádicas, para o caso de monoprocessamento preemptivo, possui vários algoritmos ótimos[DM89]. Entre estes, podemos citar o algoritmo

de próximo prazo, que executa sempre a tarefa cuja folga é a menor. O algoritmo de próximo prazo é ótimo e isto se baseia no fato de que, para monoprocessamento, é sempre possível transformar-se um escalonamento possível em outro que siga a política do prxímo prazo. Mesmo não sendo um algoritmo ótimo para o caso de multiprocessamento, o algoritmo de próximo prazo é eficiente em termos de minimização do número de preempções, o que também vale para monoprocessamento[DM89].

2.5.1 Garantia Pré-tempo de Execução para Tarefas Esporádicas

Existem duas formas de tratar o escalonamento de tarefas em tempo real: a primeira procura garantir em tempo anterior à execução que todas as tarefas irão cumprir seus prazos e a segunda procura estabelecer em tempo de execução quais as tarefas que deverão cumprir seus prazos, de forma a tomar as ações apropriadas para as tarefas que não forem garantidas. A primeira solução é apropriada para sistemas estáticos, onde todos os parâmetros das tarefas são conhecidos, e se baseia em uma reserva de recursos para a garantia do cumprimento dos prazos. Vamos analisar aqui este tipo de solução, enfocando sistemas com tarefas periódicas fixas e esporádicas, que são dinâmicas por natureza.

É possível garantir *offline* o escalonamento de um conjunto de tarefas esporádicas através de uma reserva de tempo de UCP(reserva de recurso), utilizando uma transformação que habilite a representação das tarefas esporádicas por tarefas periódicas equivalentes. Desta forma, se as tarefas periódicas equivalentes são escalonáveis, podemos garantir que as tarefas esporádicas equivalentes também o serão. Uma forma inicial de substituição sugere que cada tarefa esporádica $T_s = (c_s, r_s, d_s, p_s)$ possa ser representada por uma tarefa periódica $T_p = (c_p, r_p, d_p, p_p)$, onde $c_p = c_s$, $r_p = 0$, $p_p = min(l_s + 1, p_s)$ e $d_p = c_s$, onde $l_s = d_s - c_s$. Em geral, a tarefa esporádica pode ser substituída por uma tarefa periódica que obedeça os seguintes parâmetros: $(1)d_s \geq d_p \geq c_s$, $(2)c_p = c_s$ e $(3)p_p \leq d_p - d_s + 1$. Esta técnica é um exemplo de uma técnica geral conhecida como escalonamento de latência[Mok84].

Devemos observar dois pontos em relação à transformação acima: o primeiro se refere ao fato de ser uma transformação parcial, no sentido de que as tarefas periódicas equivalentes oferecem mais recursos do que as tarefas esporádicas poderiam requerer. Isto implica em uma condição suficiente para o escalonamento das tarefas esporádicas, embora não seja uma condição necessária. O segundo ponto diz respeito à garantia absoluta de que as tarefas esporádicas, cuja transformação habilita a validade de escalonamento para as tarefas periódicas, são escalonáveis, independentemente destas tarefas virem a ocorrer. Isto se dá em função da reserva do armazenamento *off-line*

de recursos(tempo da UCP), garantidos pela existência das tarefas periódicas equivalentes e de sua escalonabilidade. Embora seja um fator importante para uma garantia absoluta, este armazenamento pode acarretar perda de tempo de utilização da UCP, uma vez que não existe uma previsão de quando, ou mesmo se a tarefa esporádica vai ocorrer. Endereçamos primeiramente a questão da escalonabilidade exata, através de uma técnica de substituição total.

Dentro do contexto de garantia *off-line*, o objetivo fundamental seria encontrar um algoritmo *off-line* que pudesse produzir automaticamente algoritmo *on-line* para o escalonamento da UCP. Os algoritmos *off-line* determinariam a validade do escalonamento das tarefas esporádicas, de forma tal que o escalonamento válido pudesse ser construído pelo algoritmo *on-line*. O algoritmo *on-line* resultante deveria escalonar o conjunto dinâmico de tarefas de forma a satisfazer todas as requisições temporais. Como o algoritmo de menor prazo é ótimo, o teste de validade *off-line* deve ser satisfeito de forma a garantir a validade do escalonamento.

A substituição total implica em uma condição necessária e suficiente para o escalonamento do conjunto de tarefas esporádicas. Ela se baseia na construção de um conjunto finito de requisições tais que a escalonabilidade deste conjunto garanta a escalonabilidade do conjunto esporádico e vice-versa. Foi mostrado que esta metodologia leva à um procedimento PSEUDOPTIME para a escalonabilidade do conjunto de requisições equivalentes[BMR90]. A questão básica da subutilização da UCP, quando o conjunto esporádico não se apresenta, continua. Esta questão só pode ser modificada se a garantia não for absoluta, ou seja, se o teste de garantia é feito em tempo de execução.

2.5.2 Teste de Escalonabilidade em Tempo de Execução

Em vários sistemas dinâmicos, o grau de carga da UCP é variável, impondo a necessidade de um teste de escalonabilidade em tempo de execução. Uma vez que as tarefas esporádicas são essencialmente dinâmicas, e sua influência sobre a carga real do sistema só pode ser conhecida em tempo de execução, é necessário que se estabeleça um teste de escalonabilidade de tal forma que a tarefa esporádica não seja previamente garantida, mas que permita uma avaliação da escalonabilidade em tempo de execução, de tal forma a garantir sua escalonabilidade, também em tempo de execução. Algumas tarefas esporádicas poderão não ser garantidas segundo este esquema, sendo que a vantagem é que podemos saber quais são e adotarmos medidas complementares. Desta forma, não haverá uma reserva de recursos disponível para as tarefas esporádicas, que poderiam não ser utilizados em circuntâncias particulares. A base dos testes de escalonabilidade em tempo de execução é fornecer uma garantia

da validade do novo escalonamento causado pela chegada de uma tarefa esporádica. Veremos as abordagens propostas abaixo.

Nesta abordagem de estimativa implícita de tempo disponível[SRC85], um sistema distribuído consistindo de um conjunto de nodos(processadores), cada um com seu escalonador local, deve fornecer uma condição para a aceitação de tarefas esporádicas que chegam em tempos indeterminados em cada nodo. O conjunto de tarefas periódicas nos nodos é fixo e sua escalonabilidade é garantida na inicialização. O escalonador local deve checar a possibilidade da tarefa esporádica executar no nodo cumprindo seu prazo e não afetando os prazos das instâncias das tarefas periódicas previamente garantidas. Caso a tarefa possa ser garantida naquele nodo, a Tabela de Tarefas do Sistema(TTS), que contém informações sobre as instâncias de todas as tarefas dentro da janela periódica, deve ser atualizada e reordenada, para refletir a nova condição de escalonamento.

As entradas da Tabela de Tarefas do Sistema são ordenadas de acordo com o valor de prazo crescente, contanto que tais instâncias já tenham chegado, ou seja, a TTS reflete, em um determinado momento, apenas as instâncias ativas do sistema. Um parâmetro importante que é parte da TTS é o valor do máximo tempo de atraso que uma instância pode sofrer, mantendo ainda o cumprimento de seu prazo. Este parâmetro é importante para a determinação da escalonabilidade da tarefa esporádica que chega. A escalonabilidade da tarefa recém-chegada é possível desde que exista tempo disponível suficiente para sua execução previamente ao seu prazo. Este tempo disponível é implicitamente avaliado baseado nos tempos máximos de atraso das outras instâncias. Contudo, este tempo pode ser explicitamente computado quando da resposta de nodos remotos a pedidos de transferência de tarefas. Este cálculo envolve o tempo estimado de chegada da tarefa ao nódo(TECT), e o prazo da tarefa(PT). Para o cálculo do tempo disponível, leva-se em conta as instâncias futuras das tarefas garantidas, periódicas ou não, o tempo de computação de outras tarefas que potencialmente podem chegar como resultado de ofertas anteriores, a porcentagem do tempo de UCP requerido pelas tarefas periódicas entre PT e TECT, a estimativa de tempo de UCP requerido por tarefas locais entre TECT e PT, de tal modo que o teste final reflete uma série de condições locais ou não, para a garantia da tarefa esporádica localmente.

Na abordagem do cálculo exato de tempo máximo disponível, procura-se obter a folga máxima dada em determinado intervalo onde a tarefa esporádica deve ser escalonada[CC89]. O objetivo do escalonamento *on-line* de tarefas esporádicas na presença de um sistema de tarefas periódicas, base da metodologia de teste em tempo de execução, é fornecer uma garantia de que tarefas com características dinâmicas possam ser escalonadas com sucesso, sem alterar a escalonabilidade das tarefas estáticas, especificamente as tarefas periódicas conhecidas *off-line*. Desta forma, é necessário

estabelecer parâmetros que possam julgar, em tempo de execução, se uma dada tarefa esporádica pode comprometer o prazo de uma ou mais instâncias de tarefas periódicas e, em caso positivo, julgar se a tarefa esporádica pode ser aceita em tais condições. A abordagem do cálculo exato do tempo máximo disponível pretende investigar e estabelecer uma condição necessária e suficiente para que uma tarefa esporádica seja escalonável em um sistema estático de tarefas periódicas de tal forma que as tarefas periódicas não sejam comprometidas pela inclusão da nova tarefa. Em termos de escalonamento ótimo, o algoritmo de menor prazo garante que se tal escalonamento válido existe, então ele será produzido. Desta forma, a tarefa esporádica não vai comprometer as instâncias periódicas se os atrasos que estas sofrerem, em função da tarefa esporádica, não forem suficientes para implicar em perda de prazo. Os atrasos permissíveis em cada intervalo possível para a tarefa esporádica são calculados, supondo-se que o conjunto periódico é síncrono, de tal forma que podemos montar uma base de dados com todos os intervalos e recuperar a informação de quanto tempo está disponível em cada intervalo, dependendo do momento que a tarefa esporádica chegar. Se a computação da tarefa esporádica não exceder este limite, sua escalonabilidade está garantida[SRC85].

Através do escalonamento dinâmico baseado em listas de regiões ocupadas, procura-se permitir o teste de escalonabilidade *on-line* através da tentativa de reescalonamento das instâncias de tarefas previamente garantidas, juntamente com a tarefa recém-chegada[SZ92]. O algoritmo escalona conjuntos de instâncias das tarefas ordenadas por prazo. Dado tal conjunto, procura-se escalonar com maior prioridade a tarefa cuja requisição está presente com menor prazo. Desta forma, pode-se verificar se todas as instâncias de tal conjunto são escalonáveis, conhecendo-se seu tempo de requisição. Procura-se então formar uma estrutura conhecida como Lista de Regiões Ocupadas(LRO), que consiste exatamente nos intervalos onde a UCP estará ocupada em função do atendimento das instâncias presentes, obedecendo ao algoritmo de prioridades estabelecido acima. Tal algoritmo está especificado na lista de menor prazo(LMP), que contém a ordem na qual as instâncias devem ser executadas.

A chegada de uma tarefa esporádica em um tempo r qualquer dispara um processo que deve avaliar se a tarefa esporádica é escalonável sem afetar a escalonabilidade do sistema corrente. Baseada na lista de regiões ocupadas e na lista de menor prazo, que é atualizada de acordo com o prazo da tarefa esporádica, faz-se um reescalonamento das instâncias, a partir de r, objetivando a verificação do cumprimento dos prazos. Se alguma instância não cumpre o prazo, a tarefa esporádica não será escalonável, e o sistema retorna ao estado prévio à sua chegada. Se todas as instâncias reescalonadas cumprem seus prazos, a tarefa esporádica será aceita, e as listas atualizadas para refletir a nova situação. O reescalonamento das instâncias depende da quantidade de computação que a tarefa esporádica requer[SZ92]. Tal computação força um atraso

em todas as instâncias, desde que não exista um tempo de UCP livre, ou seja, a especificação de quais instâncias serão afetadas deverá ser capturada da Lista de Regiões Ocupadas. Esta metodologia permite uma abordagem *on-line* desde que as tarefas periódicas sejam previamente conhecidas para formar a Lista de Regiões Ocupadas.

2.6 Escalonamento de Tarefas Esporádicas com Importância

Sistemas de Tempo-Real consistindo de tarefas periódicas e esporádicas suportando aplicações críticas devem procurar garantir, em tempo anterior à execução das tarefas, as requisições temporais das mesmas. Tal metodologia de projeto é difícil de ser obedecida, uma vez que os recursos para a satisfação de todas as garantias serão subutilizados quando o sistema operar em carga bem abaixo do pior caso previsto. A subutilização dos recursos torna-se um dos principais obstáculos para à garantia *off-line* do escalonamento das tarefas, quando o sistema é dinâmico. Torna-se então necessário um procedimento que permita que a garantia das tarefas possa ser testada em tempo de execução, o que assegura ao sistema um alto nível de previsibilidade, uma vez que as tarefas ativas correntemente seriam garantidas, mas não proporcionaria uma garantia geral para todas as tarefas.

Na seção anterior, discutimos o escalonamento de tarefas periódicas e esporádicas sob a ótica da garantia incondicional de tarefas periódicas *off-line*, sendo que as tarefas esporádicas seriam garantidas *on-line* caso o teste assegurasse que as tarefas esporádicas poderiam ser escalonadas desde que não houvesse prejuízo para as tarefas já garantidas. Isto implica em uma priorização das tarefas periódicas, bem como das tarefas esporádicas com menor tempo de chegada. Em situações de sobrecarga, as tarefas que provocaram a sobrecarga, ou seja, as tarefas esporádicas ainda não garantidas, podem vir a ser não garantidas. Portanto, a prioridade de garantia é variável e depende da condição corrente de sobrecarga, e não leva em conta a semântica particular da tarefa.

Este nível de imprevisibilidade sobre a escalonabilidade de tarefas é indesejável em Sistemas de Tempo Real Crítico, especialmente quando existem diferenças semânticas entre as tarefas. Estas diferenças advêm da natureza da tarefa e de sua funcionalidade em relação ao sistema. Em geral, as tarefas esporádicas estão associadas a condições de exceção e/ou tolerância a falhas, execução de rotinas de manutenção e supervisão, podendo ser ativadas pelo operador ou pelo controle automático do sistema. Dependendo da funcionalidade, a necessidade de cumprimento do prazo tem uma importância relativa entre as tarefas, embora todas sejam tarefas críticas. Em geral, a importância

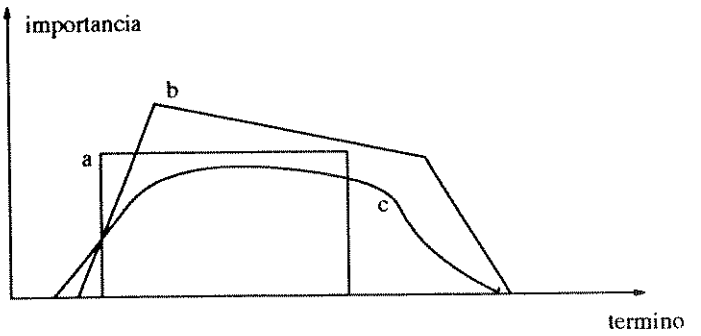

Figura 2.7: Funções de Importância

está associada aos requisitos do ambiente interno na forma que ele afeta a tarefa e à relação entre a tarefa e o sistema como um todo, incluindo seu ambiente[JLT85]. Procurando um escalonamento que priorize as tarefas mais importantes, adiciona-se um nível de maior previsibilidade ao sistema, desde que as especificações de carga periódica e esporádica sejam mantidas dentro de limites aceitáveis. Ver na Figura 2.7 alguns exemplos de funções de importância, onde *a*, *b* e *c* ilustram diferentes funções de importância.

2.6.1 Escalonamento Baseado no Fator Densidade

A primeira técnica desenvolvida com o intuito de fornecer uma maximização do benefício gerado para o sistema através do término das tarefas baseou-se na simulação de um conjunto de tarefas periódicas e esporádicas em condição de sobrecarga, usando vários algoritmos e testando sua utilidade. Definiu-se o valor densidade de uma tarefa como sendo o quociente do valor de importância pelo tempo de computação. Foi observado que, dado um conjunto de tarefas cujo prazo não é levado em conta, pode ser mostrado que o escalonamento no qual os processos com o maior valor de densidade são priorizados vai produzir um valor de benefício melhor do que qualquer outro algoritmo[JLT85]. Embora esta observação seja interessante, o prazo não é explicitamente levado em conta. Consideramos a associação de prioridade acima como o algoritmo de densidade.

As implicações das observações acima indicam que se não ocorrer uma sobrecarga, o algoritmo de próximo prazo tem sucesso em escalonar todas as tarefas de modo que o benefício será maximizado. Caso a sobrecarga aconteça, o algoritmo de densidade produzirá um alto valor para a função benefício, embora algumas tarefas poderão

perder seus prazos sem necessidade. Destas implicações e das observações seguem dois algoritmos[JLT85]:

- BEValue1 - Algoritmo de densidade, tendo como vantagens a atuação em sobrecarga. Contudo, este algoritmo peca em relação ao algoritmo de prxímo prazo em situações de subcarga.
- BEValue2 - Algoritmo misto de próximo prazo e densidade, atuando como um ou outro dependendo da probabilidade da ocorrência de sobrecarga.

O resultado de simulações envolvendo um conjunto de tarefas geradas de forma aleatórias permite concluir que a forma da função de importância e o nível de carga afetam muito os algoritmos. Em geral, o algoritmo BEValue2 tem um rendimento melhor, em função de sua heurística e adaptabilidade.

2.6.2 Integração de Importância e Urgência

No modelo que integra importância e urgência, a semântica da garantia se baseia na política de garantia incondicional. Neste caso, a tarefa uma vez garantida, terá a certeza do cumprimento de seu prazo. Isto implica que as tarefas com chegada corrente não podem afetar a escalonabilidade de tarefas previamente garantidas[BSR88]. Isto impõe uma dificuldade quando tarefas com nível de importância diferentes estão presentes, supondo que uma tarefa garantida esteja impedindo a escalonabilidade de uma tarefa recém-chegada, com importância maior. Neste caso, a garantia inicial não deve ser absoluta, ou seja, a tarefa com menor importância poderia ser retirada para o sistema poder atender a tarefa de maior importância. Esta semântica implica em uma garantia condicional. Esta política procura, ao contrário da anterior, levar em conta os prazos das tarefas, ou seja, sua expressão de urgência, bem como sua importância. Contudo, a idéia não se baseia em um mapeamento explícito de urgência e importância na prioridade, mas na tentativa de escalonamento das tarefas segundo a regra ótima do menor prazo e alterar as tarefas garantidas de acordo com a importância.

A determinação da escalonabilidade da tarefa recém-chegada inicia-se tentando se garantir que a tarefa pode ser escalonada de acordo com seu prazo, independente de sua importância. Caso isto seja verdadeiro, o escalonamento é válido. Senão, é feita uma tentativa de garantir a tarefa ás custas de tarefas previamente garantidas, com importância menor. Se um conjunto suficiente de tarefas menos importantes puder ser encontrado e removido do sistema, a tarefa recém-chegada será garantida. Caso contrário, a tarefa recém-chegada será descartada. Os algoritmos que tratam a

remoção diferem apenas na forma de remover as tarefas menos importantes. Foram desenvolvidas duas variações de remoção[BSR88], apresentadas abaixo.

- Remoção 1 - remove-se temporariamente a tarefa com o menor nível de importância que pode afetar a escalonabilidade da tarefa recém-chegada, cumulativamente, até que o prazo da tarefa recém-chegada seja satisfeito.
- Remoção 2 - remove-se temporariamente a primeira tarefa com nível de importância menor que a da tarefa recém-chegada, cumulativamente, até que o prazo da tarefa recém-chegada seja satisfeito.

Em resumo, ambas as políticas procuram garantir todas as tarefas baseando-se no menor prazo. Caso isto não seja possível, pode-se optar por remover as tarefas menos importantes entre todas, ou apenas as tarefas com importância menor que a da recém-chegada por ordem de prazo. Esta diferença pode ser importante para sistemas distribuídos, mas não afeta sistemas centralizados[BSR88].

2.6.3 Função Valor de Importância

Em Sistemas de Tempo-Real Crítico, as tarefas devem terminar sua execução até o seu prazo. Neste sentido há a necessidade de se associar uma função que indique a importância de se completar a tarefa em um tempo especificado [CM91]. Esta função não é necessariamente bem comportada, podendo mapear um comportamento bem geral para o nível de importância da tarefa. Entretanto, estamos interessados no caso onde a função é modelada como um degrau, caracterizando uma tarefa com prazo crítico. Assim, o término da tarefa previamente ao seu prazo fornece um valor de importância ao sistema que, cumulativamente com todos os outros valores fornecidos através da execução das outras tarefas com cumprimento do prazo, estabelece um benefício global para o sistema. Observe que se a tarefa completa sua execução posteriormente ao vencimento do prazo, seu benefício para o sistema será nulo.

Várias abordagens têm sido propostas para escalonar tarefas de tal forma que o grau de importância seja considerado. Em qualquer caso, o objetivo é escalonar com sucesso um grande número de tarefas que sejam mais importantes. Matematicamente, isto significa maximizar algumas variáveis que levam em conta o nível de importância das tarefas satisfazendo seus requisitos temporais. Neste trabalho, adotaremos o conceito de que a função benefício que incorpora a contribuição de tarefas individuais deve ser maximizada, ou seja, existe um escalonamento que produz o máximo valor para a função benefício. Em geral, este problema é intratável, e procura-se empregar alguma heurística que forneça uma solução subótima.

Em termos de otimização geral da função benefício, esta abordagem permite uma ordenação de tarefas do ponto de vista de uma política que procura maximizar o benefício através da análise de ordenações de precedência, de tal forma que a ordem das tarefas reflita as relações de precedência que forneçam um alto valor para o benefício. Neste sentido, para cada conjunto de duas tarefas possível, calculamos a função δ_{ij}, que estabelece qual tarefa entre i e j deveria vir primeiro na ordenação. Feito isto para todas as tarefas não ordenadas ainda, priorizamos as tarefas com maior valor de precedência, ou seja, a tarefa que precede um maior número de tarefas segundo o valor de δ_{ij}. Esta tarefa será a próxima da lista, e um novo cálculo é feito para as tarefas restantes, até esgotarmos todas as tarefas.

2.7 Comentários Finais

Existem, basicamente, duas formas distintas de tratar o problema do escalonamento de sistemas de tempo real no que concerne à previsibilidade e adaptabilidade, a saber, escalonamento *on-line* e escalonamento *off-line* de tarefas. O escalonamento *off-line* se baseia no conhecimento prévio de todos os parâmetros das tarefas para garantir se o escalonamento é válido. A aplicabilidade do escalonamento off-line é grande, uma vez que qualquer conjunto de tarefas cujos parâmetros são completamente conhecidos pode ser avaliado. Um conjunto de tarefas periódicas com parâmetros constantes pode sempre ser testado com respeito à escalonabilidade. A propriedade de frequência de requisição periódica garante que as avaliações possíveis são limitadas para a garantia *off-line* do escalonamento.

Tarefas esporádicas, por outro lado, são dinâmicas por natureza, ou seja, não têm um tempo de pronto conhecido e, além disto, o seu escalonamento só pode ser garantido levando-se em conta o pior caso, ou seja, superestimando recursos para a tarefa. A semântica das tarefas esporádicas permite que, por um lado, cada instância seja garantida separadamente e, por outro lado, que o pior caso transforme uma tarefa esporádica em uma periódica. Os procedimentos para a garantia de tarefas esporádicas, na hipótese delas serem transformadas em periódicas, requerem uma pré-alocação de recursos, indesejável quando o uso de tais recursos deva ser otimizado.

O escalonamento *on-line* surge como uma alternativa de teste de escalonabilidade para sistemas estritamente dinâmicos, onde a alocação prévia de recursos é inaceitável, dada a grande possibilidade de não utilização de tais recursos. Neste caso, conjuntos de tarefas periódicas variáveis em tempo de execução, bem como tarefas esporádicas, podem ter sua escalonabilidade avaliada *on-line*, embora a garantia da escalonabilidade seja perdida.

Os argumentos contra e a favor da escalonabilidade *on-line* e *off-line* são vários. O aspecto mais importante para sistemas de tempo real é, consensualmente, a previsibilidade do sistema, independente de como a escalonabilidade é garantida. O escalonamento *off-line* garante a previsibilidade, muitas das vezes através da subutilização de recursos. O escalonamento *on-line* procura otimizar a utilização de recursos, mas não garante previsibilidade absoluta, embora certa garantia possa ser obtida através do uso de técnicas de nível de importância de tarefas e múltiplas versões. Desta forma, pode-se garantir um conjunto mínimo de tarefas *off-line*, baseando-se em aspectos de criticalidade, e procurar a otimização de recursos *on-line*, o que acreditamos ser mais coerente com a proposta de previsibilidade. Neste trabalho pretendemos explorar metodologias de desenvolvimento de condições que garantam a previsibilidade dentro de padrões aceitáveis para os STRC, aplicáveis a situações em que o ambiente é dinâmico e, portanto, flexível.

Capítulo 3

Escalonabilidade para Tarefas com Prazo Arbitrário

3.1 Introdução

Neste capítulo, iniciaremos uma discussão que visa estabelecer formalmente as noções de garantia de escalonamento para tarefas periódicas independentes e esporádicas que atuam em um ambiente dinâmico e flexível. Entende-se que tal ambiente possui requisitos que variam com o tempo, de tal forma que seu modelamento se baseia na existência de tarefas esporádicas e de tarefas periódicas cujos parâmetros possam ser alterados.

Como exemplo de tais sistemas, podemos pensar em um sistema de controle de tráfego aéreo. O ambiente controlado se compõe das aeronaves, torres de controle e equipamentos de monitoramento das aeronaves. Como o número de aeronaves em um espaço sob jurisdição de uma torre de controle é variável, e se altera frequentemente, temos um ambiente dinâmico. A representação do monitoramento e atuação como tarefas periódicas para cada aeronave implica em um conjunto de tarefas periódicas que é dinâmico[KM91]. Juntamente com funções que podem ser ativadas em um tempo não-determinístico, representadas por tarefas esporádicas, faz-se necessário garantir a escalonabilidade das tarefas, levando-se em conta seus parâmetros correntes.

Um conjunto de tarefas periódicas e esporádicas com tal natureza requer um tratamento que permita uma decisão de escalonabilidade em tempo de execução. Como já foi enfatizado anteriormente, este tipo de escalonabilidade só pode ser aplicado se duas questões forem satisfeitas:

- O teste deve ser rápido, pois é feito em tempo de execução

- O teste deve fornecer um limite de escalonabilidade adequado

A garantia da escalonabilidade para os ambientes dinâmicos, sendo função dos parâmetros correntes, deve ser estabelecida em tempo de execução, e reavaliada sempre que se fizer necessário, ou seja, sempre que requisições de aumento de carga no sistema acontecerem, através de chegada de tarefas esporádicas e periódicas[XP91]. Sistemas de Tempo Real requerem, por seu turno, decisões rápidas em tempo de execução. Logo, ambientes dinâmicos vão exigir testes de escalonabilidade que forneçam garantias para tarefas escalonáveis em tempo de execução de tal forma que o cálculo da garantia seja rápido o suficiente para não comprometer o funcionamento normal do sistema.

Inicialmente, devemos decidir qual tipo de associação de prioridade deve ser empregado, no sentido de se estabelecer uma ordem de execução das tarefas que estão na fila de pronto. Entre as possibilidades, temos a associação fixa(APF) e a associação variável(APV). Considerando as vantagens das associação fixa sobre a variável, como a maior facilidade de implementação, dadas as características das arquiteturas e sistemas operacionais atuais e a determinação de qual tarefa ou tarefas não irão satisfazer suas restrições temporais em caso de sobrecarga, vamos nos ater neste capítulo fundamentalmente a este tipo de escalonamento, também chamado de escalonamento estático[CSR87]. Contudo, é sabido que as associações fixas tem um desempenho inferior às de associação variável, considerando a utilização da UCP como o fator de escalonabilidade[LL73], embora no caso médio este desempenho seja razoável[LSD89].

Vamos considerar que o conjunto de tarefas periódicas e esporádicas original seja modelado como um conjunto de tarefas periódicas equivalentes, onde as tarefas periódicas são conservadas e as tarefas esporádicas são transformadas em tarefas periódicas equivalentes. Se o conjunto equivalente for escalonável, então o conjunto original também o é[Mok84]. Esta técnica, conhecida como escalonamento de latência, possibilita que as tarefas periódicas equivalentes tenham um prazo inferior ao seu período[Mok83]. Desta forma, vamos considerar neste capítulo a questão da escalonabilidade aplicada a um conjunto de tarefas periódicas, síncronas, com prazo arbitrário, não superior ao período[LW82].

Este modelamento de tarefas periódicas encontra aplicação em outras situações. Como exemplo, podemos citar o acesso da tarefa a E/S[SLR86], onde o prazo da saída é o fim do período, e o prazo da execução 'e, portanto, menor que o período, uma vez que a execução deve anteceder a saída.

Dentro do contexto de teste em tempo de execução, o nosso objetivo é fornecer um tratamento para a escalonabilidade que seja rápido, estabelecendo o intervalo de validade do teste. Para ser rápido, este teste não pode ser baseado no princípio de analisar a escalonabilidade no intervalo crítico, ou seja, o intervalo definido para cada

tarefa, onde a perda de prazo deve ser verificada[LSD89]. Iremos verificar que para prazos arbitrariamente pequenos, qualquer teste baseado no fator de utilização será extremamente inconveniente, uma vez que a condição necessária para a escalonabilidade não é fornecida no teste e pode estar longe da condição suficiente.

Queremos fornecer um limite mínimo para o fator de utilização que represente uma condição suficiente para o escalonamento do conjunto de tarefas periódicas com prazo arbitrário[LL73]. Inicialmente, notamos que quando o prazo é arbitrário, a política ótima de atribuição de prioridades é conhecida como Prazo Monotônico. Esta política associa prioridades maiores às tarefas com menor prazo[LW82]. Trata-se de uma atribuição fixa de prioridades, como visto anteriormente.

Vamos tratar principalmente os casos mais comuns de prazo arbitrário, uma vez que são estes casos que refletem mais comumente as aplicações. Formalmente, não estabeleceremos uma regra para qualquer tipo de prazo, ou seja, os prazos poderiam ser arbitrariamente menores que os períodos. Entretanto, para situações reais, e particularmente na análise de sistemas de muitas tarefas, o prazo será limitado de tal forma a corresponder a uma pequena alteração no valor do período.

3.2 Condições Suficientes para Escalonabilidade com Prazo Arbitrário

Inicialmente, apresentamos algumas metodologias que visam estabelecer uma série de condições suficientes para o escalonamento de um sistema de tarefas periódicas com prazo arbitrário, menor que o período. Para prazos maiores que o período, as condições propostas em [Leh90] podem ser empregadas. As condições suficientes são, em geral, bem piores que as condições exatas, ou seja, condições suficientes e necessárias de escalonabilidade. Uma das formas mais comuns de se estabelecer o nível de validade de uma condição suficiente emprega simulações com sistemas de tarefas gerados aleatoriamente[BSR88]. A vantagem das condições suficientes em relação às condições exatas é que as primeiras são, em geral, procedimentos que podem ser executados em tempo polinomial ou, em pior caso, em tempo PSEUDOPTIME[LW82]. Para sistemas de tempo real, esta propriedade é fundamental. Resta-nos avaliar, dentre as condições suficientes apresentadas e propostas neste trabalho, qual oferece a melhor relação complexidade/desempenho e, se possível, descrever uma metodologia que seja flexível o suficiente para abranger métodos diferentes, seguindo uma hierarquia de testes, partindo do mais simples para o mais complexo, em termos de custos computacionais, possíveis de serem empregados em um sistema de tempo real. A seguir, apresentamos os resultados de métodos suficientes desenvolvidos para tratar o proble-

ma acima. Vamos nos referir às tarefas periódicas como $T = (c, d, p)$, em função de tratarmos neste capítulo de tarefas síncronas, ou seja, $r = 0$.

3.2.1 Condição Suficiente com Bloqueio

Dado um conjunto de tarefas $T = \{T_1, ..., T_n\}$, onde $T_j = (c_j, d_j, p_j)$, a formulação original de condição suficiente para taxa monotônica(quanto menor o período, maior a prioridade da tarefa) garantia que quando o fator de utilização U fosse não superior à $n(2^{1/n} - 1)$, onde n é o número de tarefas, o conjunto de tarefas seria escalonável[LL73]. Baseado nesta condição e no princípio de que uma tarefa com prazo inferior ao período pode ser considerada como uma tarefa que tem um bloqueio associado[SG90a], uma nova condição suficiente foi desenvolvida para as tarefas com $d_i \leq p_i$[SG90a]. Seja $k_i = p_i/d_i$ e B_i o bloqueio associado à tarefa T_i. O conjunto é escalonável se:

$\sum_{i=1}^{n}(c_i/p_i) + max(B_i/p_i) \leq n(2^{1/n} - 1)$

onde $U = \sum_{i=1}^{n}(c_i/p_i)$ e $B_i = p_i - d_i$. Obtemos finalmente que, se $U \leq n(2^{1/n} - 1) - max(1 - 1/k_i)$, o conjunto é escalonável.

Este resultado de escalonabilidade é válido para o conjunto T globalmente. Eventualmente, este resultado pode ser melhorado observando que o conjunto T é escalonável se todas as tarefas forem escalonáveis. A escalonabilidade individual das tarefas fornece condições mais gerais, como verificamos no caso a seguir. Seja $l < n$, e se a condição acima é satisfeita, então $\sum_{i=1}^{l}(c_i/p_i) + max(B_i/p_i) \leq l(2^{1/l} - 1)$. Como $l(2^{1/l} - 1) \geq n(2^{1/n} - 1)$, uma análise de escalonabilidade por subconjuntos de T é mais precisa.

3.2.2 Condição Suficiente com Variação Uniforme do Prazo

A variação uniforme do prazo é uma restrição imposta à condição de prazo arbitrário, de tal forma que os prazos das tarefas sejam uniformemente menores que os períodos respectivos[Leh90]. Desta forma, sendo d_i o prazo da tarefa T_i e p_i seu período, a razão p_i/d_i é igual para todas as tarefas. Seja $k = p_i/d_i$. Obviamente, $k \geq 1$. A condição suficiente para o escalonamento de um conjunto de tarefas periódicas as quais possuem o mesmo valor de k é dada abaixo, se uma das duas condições ocorrerem:

- Se $k \leq 2$ e $U \leq n((2/k)^{1/n} - 1) + (1 - 1/k)$
- Se $k \geq 2$ e $U \leq k$

então o conjunto das n tarefas é escalonável[LS86];[SRL90].
Observe que existe uma diferenciação entre prazos menores que a metade do valor do período e prazos maiores que este valor.

Novamente aqui, esta condição suficiente, quando aplicada sobre um conjunto T garante que se T é escalonável, então qualquer subconjunto de T também o será, pois para $l < n$, se a condição acima é satisfeita, então vale $\sum_{i=1}^{l}(c_i/p_i) \leq l((2/k)^{1/l} - 1) + (1 - 1/k)$.

3.2.3 Condição Suficiente com Períodos Alterados

Dado um conjunto de tarefas como acima, podemos transformar o conjunto original T em um conjunto T', de tal forma que $T_i' = (c_i, d_i, d_i)$, ou seja, $p_i' = d_i$. Como temos o prazo igual ao período, o conjunto T' é escalonável se $U' \leq n(2^{1/n} - 1)$. Contudo, se T' for escalonável, então T também é escalonável. Portanto, uma condição suficiente para a escalonabilidade de T é que $\sum_{i=1}^{n}(c_i/d_i) \leq n(2^{1/n} - 1)$. Aqui também esta condição pode ser analisada em termos de subconjuntos de T'.

Uma análise de qual seria o maior conjunto escalonável dentre os subconjuntos de T é possível através do exame da escalonabilidade de cada tarefa, de tal forma a maximizar o subconjunto de T escalonável.

3.2.4 Condição Suficiente com Análise de Interferência

Nesta abordagem, um conjunto de condições suficientes podem ser verificadas em uma série de testes[ABRW92]. Vamos nos ater ao teste que fornece a complexidade mais baixa, comparável aos procedimentos que vamos desenvolver. Este método baseia-se na propriedade de escalonamento fixo que dita que a perda do prazo, se ocorrer, ocorrerá na primeira instância. Ainda mais, tarefas mais prioritárias interferem nas instâncias de tarefas menos prioritárias de modo que estas podem ser prejudicadas com relação ao uso do tempo de processador e, portanto, ao cumprimento do prazo. O problema é que estas interferências não podem ser calculadas explicitamente de forma exata em um tempo hábil. Portanto, aqui se estima um valor de interferência que cada tarefa poderá sofrer, no máximo, dentro de seu prazo de execução. Se a computação da tarefa for menor que a quantidade de tempo livre que é garantida no intervalo, a tarefa é escalonável. Matematicamente, para um sistema de n tarefas, a escalonabilidade é válida se:

$c_i + I_i \leq d_i$ para $i = 1..n$

onde I_i é a interferência das tarefas mais prioritárias que T_i sofre no prazo d_i. A complexidade deste método é claramente dependente de como se calcula I_i. Como exemplo mais simples, se $I_i = \sum_{j=1}^{i-1} \lceil d_i/p_j \rceil c_j$, então a complexidade deste método é $O(n^2)$.

3.3 Condição de Escalonabilidade para um Sistema de 2 Tarefas

Dado um sistema $T = \{T_1, T_2\}$ de duas tarefas caracterizadas por $T_i = (c_i, d_i, p_i)$, onde $c_i \leq d_i \leq p_i$, sendo c_i o tempo de computação, d_i o prazo relativo e p_i o período da tarefa T_i, estaremos interessados em resolver a questão da escalonabilidade para este conjunto de tarefas, quando utilizamos uma associação de prioridade fixa. Para o caso em que o prazo é geral, a associação ótima é conhecida como Prazo Monotônico, e é caracterizada por atribuir uma prioridade maior à tarefa com menor prazo. Obviamente, quando os prazos relativos são iguais aos respectivos períodos, o Prazo Monotônico se torna Taxa Monotônica.

Encontrar uma solução para a escalonabilidade que seja dada na forma de um teste suficiente envolve o cálculo de um fator de utilização mínimo, abaixo do qual tem-se a certeza de que o sistema é escalonável e acima do qual podemos mostrar que não se garante que o sistema será escalonável. Isto é feito através do conceito de utilização completa do processador. Vamos considerar $d_1 < d_2$ e inicialmente, seja $d_i = p_i$, para as duas tarefas em T. Sabemos que o fator de utilização U é dado por:

$$U = c_1/p_1 + c_2/p_2$$

O limite de 0.83 é conhecido como o mínimo fator de utilização através do qual podemos garantir a escalonabilidade de T. Isto significa que qualquer conjunto T tal que $U_T \leq 0.83$ é escalonável através do algoritmo de Taxa Monotônica. É possível mostrar que existe um sistema T com $U_T = 0.84$ que não é escalonável, e, portanto, este é o limite mínimo de máxima utilização garantida. Os valores dos parâmetros das tarefas neste limite são:

$$d_2 = p_2 = 1.414 p_1 = 1.414 d_1$$
$$c_1 = p_2 - p_1 \text{ e } c_2 = p_1 - c_1$$
$$U = 2(2^{1/2} - 1) = (2^{1/2} - 1) + (2^{1/2} - 1)$$

Nota-se que o mínimo do fator de utilização ocorre quando uma partição uniforme

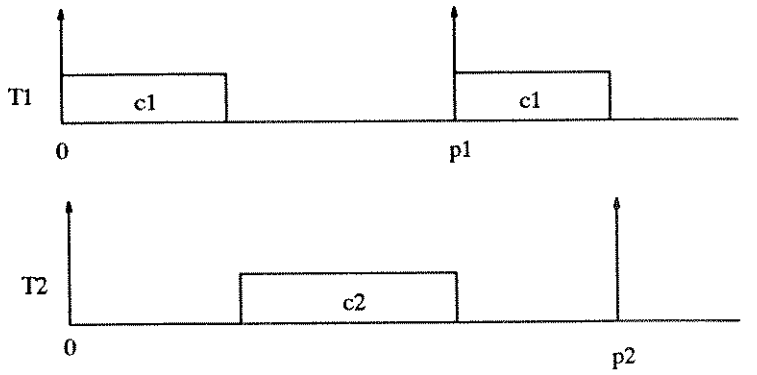

Figura 3.1: Escalonabilidade Válida por T_1, T_2

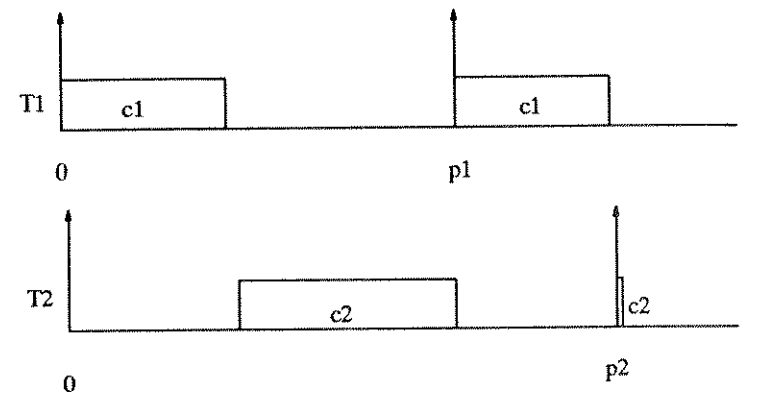

Figura 3.2: Escalonabilidade Perdida por T_2

da UCP é feita entre as tarefas, de modo que cada tarefa terá o mesmo fator de utilização, e a utilização de uma unidade a mais torna o sistema não escalonável.

Ilustramos na Figura 3.1 o processo de utilização completa para o sistema T dado por $T_1 = (414, 1000, 1000)$ e $T_2 = (586, 1414, 1414)$, onde a escalonabilidade é garantida por $U = 0.828$. Quando aumentamos em uma unidade o tempo de computação de T_2, fazendo $T_2 = (587, 1414, 1414)$, verificamos que o prazo de T_2 não será cumprido, de modo que o limite imposto no fator de utilização é suficiente, mas não necessário para garantir a escalonabilidade. Esta perda de prazo é ilustrada na Figura 3.2.

Para o caso geral, vamos denominar $k_i = p_i/d_i$ o fator de aperto da tarefa T_i. Quanto maior o fator de aperto, mais difícil se tornará o cumprimento do prazo por

parte da tarefa. O mínimo fator de aperto é 1, indicando que a tarefa deve cumprir seu prazo dentro do seu intervalo de requisição periódica. No caso geral, $k_i \geq 1$. Temos como objetivo inicial analisar de que forma o aumento do fator de aperto para um sistema de duas tarefas vai influir na escalonabilidade garantida pela condição suficiente dada acima. Consideraremos alguns casos particulares:

- $k_1 \geq 1$ e $k_2 = 1$
- $k_1 = 1$ e $k_2 \geq 1$
- $k_i \geq 1$

A análise a seguir pretende esclarecer de que forma a alteração dos prazos afeta a condição de escalonabilidade, fornecendo uma metodologia de busca da solução dentro de um novo contexto, que se baseia em um prazo cujo valor impossibilita a tarefa de executar em qualquer intervalo de seu período.

3.3.1 Casos Particulares

Como primeiro caso particular, vamos analisar o comportamento das tarefas em T quando $k_2 = 1$ e k_1 é arbitrário. Como $d_2 = p_2$, a tarefa T_2 não deve perder o prazo se $U \leq 0.828$, e, portanto, a sua escalonabilidade não é afetada. Para T_1, é possível a perda de prazo se $d_1 \leq 0.828p_1$, e $c_1 = 0.828p_1$. Logo, temos uma nova condição de escalonabilidade que depende de k_1. Contudo, se $d_1 > 0.828p_1$, a condição anterior se mantém. Portanto, T_1 é escalonável se $U_1 \leq U_{min}(1)$, onde $U_i = c_i/p_i$ e $U_{min}(1) = 1/k_1$. Por outro lado, T_2 será escalonável se $U_1 + U_2 \leq 0.828$, e T é escalonável se T_1 e T_2 o forem. Logo, consideramos $U_{min}(2) = 0.828 = 2(2^{1/2} - 1)$.

No segundo caso particular, T_1 não perde o prazo em circuntância alguma, uma vez que é a mais prioritária. Seja $T_1 = (414, 1000, 1000)$ e $T_2 = (c_2, d_2, 1414)$. Verificamos que $c_2 = 586$ e $d_2 = 1414$, a escalonabilidade de T é garantida. Suponha agora que d_2 possa mudar, tornando-se por exemplo $d_2 = 1400$. Observe que agora não podemos executar a primeira instância de T_2 depois de 1400, por exemplo. Suponhamos que T_2 perde o prazo executando após 1400 e terminando em 1401. Isto é possível tendo-se $c_1 = 400$ e $c_2 = 601$. Portanto, $d_2 = 1400$ torna possível a existência de um fator de utilização $U = 0.825$, menor do que o valor mínimo, que não garante a escalonabilidade. Na Figura 3.3, observamos o escalonamento de T_1 e T_2 acima. Na verdade, como vemos na Figura 3.3, a diminuição de d_2 força um decréscimo no fator de utilização de T_1 maior do que o aumento no fator de utilização de T_2, enquanto

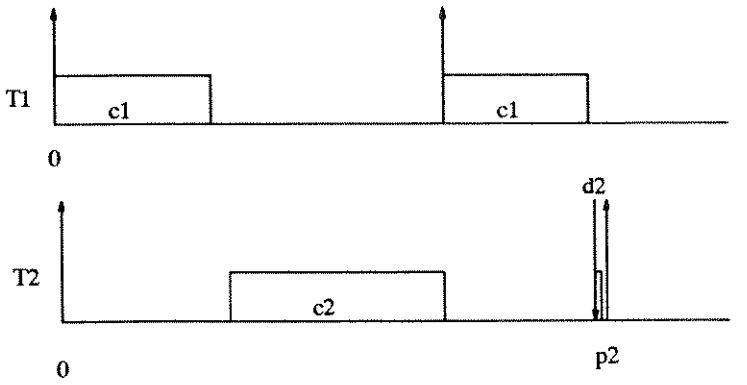

Figura 3.3: Escalonabilidade Perdida com Prazo Arbitrário

mantendo completamente a utilização do processador. Isto diminui o valor do mínimo fator de utilização que habilita a condição suficiente.

O caso geral, onde T_1 e T_2 são arbitrários, deve levar em conta tanto o valor de d_1 que pode invalidar T_1, como o valor de d_2, responsável pela diminuição do valor mínimo do fator de utilização. Vamos avaliar mais detalhadamente o caso geral e suas possibilidades no decorrer do capítulo.

3.3.2 Cálculo do Valor Mínimo para o Fator de Utilização

O valor mínimo para o fator de utilização deve ser obtido seguindo a metodologia empregada em trabalhos anteriores [LL73], procurando minimizar o fator de utilização, mantendo-se a utilização completa do processador. Neste sentido, vamos considerar o sistema T anterior, com $d_1 = p_1$ inicialmente e $d_2 = lp_1 + q$, onde l é um inteiro positivo e $q < p_1$. Este é o caso mais geral para o sistema T. Vamos mostrar que, dado um sistema T desta forma, podemos obter um sistema T^* tal que seu fator de utilização seja menor, e a utilização completa é mantida.

Teorema 3.3.1 *Seja um sistema $T = \{T_1, T_2\}$, onde $T_i = (c_i, d_i, p_i)$, com $d_2 = lp_1 + q$ como acima, então se o sistema T^* tal que $T_1^* = T_1$ e $T_2^* = (c_2^*, p_1+q, p_2^*)$, é escalonável, T também é escalonável.*

Prova. Na verdade, queremos mostrar que se para um sistema qualquer que tenha $d_2 < 2p_1$, o sistema é escalonável, então se $d_2 = lp_1 + q$ o sistema também será escalonável. Portanto, suponhamos que $d_2 = p_1 + q$, onde $q < p_1$, e $\{T_1, T_2\}$ são

escalonáveis, ou seja, $U_{T^*} \leq U_{min}$. Vamos manter T_1 e alterar T_2 de tal forma que $d_2 = lp_1 + q$ e $p_2 = k_2 d_2$. Se mantivermos a utilização completa até o prazo, $c_2 = c_2^* + (l-1)(p_1 - c_1)$. Logo, a utilização da tarefa T_1 continua inalterada, e a da tarefa T_2 dependerá do novo valor de c_2. Originalmente, ou $c_2 = p_1 - c_1$ ou $c_2 = d_2 - 2c_1$. Se $c_2 = p_1 - c_1$, então $c_2 = lc_2^*$, e como $p_2 = lp_2^* - (l-1)k_2 q$, então o mínimo do fator de utilização com utilização completa ocorre quando $d_2 < 2p_1$. Por outro lado, se $c_2 = d_2 - 2c_1$, novamente teremos um mínimo de utilização quando $d_2 < 2p_1$. Podemos também substituir T_1 por $T_1^* = (c_1, ld_1, lp_1)$, ou seja, o novo período(e prazo) de T_1^* é l vezes o anterior. Logo, $c_2^* = c_2 + (l-1)c_1$, no sentido de se manter a utilização completa. Portanto, $U_{T^*} = c_1^*/lp_1 + c_2^*/p_2$, ou seja, $U_{T^*} = c_1^*/lp_1 + (c_2 + (l-1)c_1/p_2$. Portanto, $U_{T^*} = c_1/lp_1 + (l-1)c_1/p_2 + c_2/p_2 \leq c_1/lp_1 + (l-1)c_1/lp_1 + c_2/p_2 = c_1/p_1 + c_2/p_2 = U_T$. Logo, o menor valor para o fator de utilização mantendo a utilização completa ocorre quando $d_2 < 2p_1$.

Todas as análises subsequentes usarão este resultado. Vamos relaxar um pouco as condições acima, observando que, de fato, podemos ter um conjunto escalonável mesmo se $d_1 \leq p_1$. Seja $U_{min}(2)$ o valor mínimo de U que garante a escalonabilidade de um conjunto de duas tarefas.

Teorema 3.3.2 *Seja um sistema T como acima, tal que $d_2 < 2p_1$, e $U_{min}(2)$ o valor mínimo de U para garantir a escalonabilidade de T_2. O conjunto T é escalonável se $d_1 \geq U_{min}(2)p_1$.*

Prova. Como T_2 é escalonável, independentemente do valor de c_1, apenas T_1 poderia perder seu prazo. Isto acontece apenas se $c_1 > d_1$, o que acarretaria um valor de $U > U_{min}(2)$. Portanto, o conjunto é escalonável, se $U \leq U_{min}(2)$.

Vamos agora proceder ao cálculo do valor mínimo para U. Inicialmente, faremos a análise baseados em tres possibilidades:

- $p_1 \leq d_2$.
- $d_2 \leq p_1 \leq p_2$.
- $p_2 \leq p_1$.

3.3.3 Casos Comuns

Vamos calcular o fator mínimo de utilização para o caso em que $p_1 \leq d_2$. Para isto, estabelecemos os seguintes parâmetros:
$c_1 = m_1 d_1$

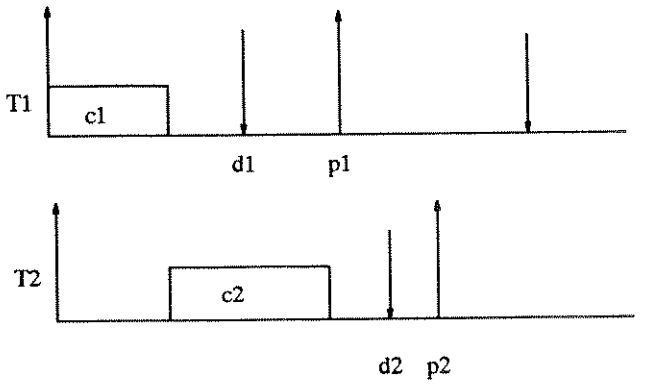

Figura 3.4: Condição $p_1 \leq d_2$

$p_1 = k_1 d_1 = d_1$
$d_2 = K d_1$, com a restrição de que $d_2 < 2p_1$, como colocado no teorema 3.3.1. Ver na Figura 3.4 uma ilustração desta condição.

$p_2 = k_2 d_2 = K k_2 d_1$

Temos três possibilidades para o valor de c_1:

- $c_1 < d_2 - p_1$
- $c_1 > d_2 - p_1$
- $c_1 = d_2 - p_1$

Para $c_1 < d_2 - p_1$, o valor de c_2 deve ser $d_2 - 2c_1$. Portanto, teremos:

$c_1 = m_1 d_1$ e $c_2 = (K - 2m_1)d_1$

Logo, $U = m_1/k_1 + (K - 2m_1)/K k_2$, ou rearranjando os termos:

$U = 1/k_2 + m_1(K k_2 - 2k_1)/K k_1 k_2$

Vemos que o valor de U vai aumentar com um aumento de $c_1(m_1)$ se a quantidade $K k_2 - 2k_1$ for positiva.

Para $c_1 > d_2 - p_1$, temos $c_2 = p_1 - c_1$. Logo, $c_1 = m_1 d_1$ e $c_2 = (k_1 - m_1)d_1$, gerando:

$U = m_1/k_1 + (k_1 - m_1)K k_2$

$U = K/k_1 + (2k_1/k_2)(1/K) - (k_2 + 1)/k_2$

Como $Kk_2 - k_1 > 0$, esta escolha faz com que o valor de U cresça com o aumento de c_1. Portanto, devemos diminuir c_1. Logo, o valor ótimo para c_1 é dado por $c_1 = d_2 - p_1$. Neste caso, $c_2 = (2k_1 - K)d_1$.

$U = m_1/k_1 + (2k_1 - K)/Kk_2$

$U = K/k_1 + (2k_1/k_2)(1/K) - (k_2 + 1)/k_2$

A análise a seguir é válida para a seguinte situação:

$p_1 \leq p_2 \leq 2p_1$, portanto, temos como hipótese que $p_2 \leq 2p_1$. Posteriormente, vamos verificar o que acontece se $p_2 > 2p_1$.

O valor mínimo de U é função de K, dados os valores de k_1 e k_2. Observe que como supusemos que $k_1 = 1$, K define o valor ótimo para a relação d_2/p_1 na presença de um fator de aperto para T_2. O valor mínimo de U deve ser obtido diferenciando-se U com relação à K:

$dU/dK = 1/k_1 - 2(k_1/k_2)(1/K)^2 = 0$

Obtemos o valor ótimo de $K = k_1(2/k_2)^{1/2}$. Para $k_1 = 1$, teremos $K = (2/k_2)^{1/2}$. O valor para U_{min} é:

$U_{min} = (1/k_2)(2(2k_2)^{1/2} - (k_2 + 1))$

Observamos que para $k_2 = 1$, teremos $U_{min} = 2(2^{1/2} - 1)$

Retornamos agora a análise para o caso em que $Kk_2 - 2k_1 > 0$, ou seja, $p_2 > 2p_1$. O valor mínimo continua valendo, desde que $k_2 \leq 2$. Portanto, estaremos supondo que $k_2 \leq 2$. Posteriormente, analisaremos o resultado para k_2 qualquer.

Vamos analisar o caso em que k_1 é geral. A única modificação a ser feita está relacionada com qual fator de utilização devemos trabalhar. Um valor muito alto de k_1 pode minimizar o fator de utilização dado por $1/k_1$. Como estamos supondo que $c_1 \leq d_1$, não precisamos nos preocupar com este aspecto. Contudo, no caso geral, este problema existe. Para este caso, dados os valores de k_1 e k_2, procuramos garantir que $d_1 \geq U_{min}(1)p_1$. Com relação à T_2, também é necessário a garantia de $d_2 \geq U_{min}(2)p_2$. Veremos que isto é suficiente para reconhecer $U_{min}(i)$ como o valor mínimo do fator de utilização que garante as tarefas cujo fator de utilização é menor.

Podemos resumir nossa avaliação para o caso analisado, com as seguintes conclusões:

- T_1 é escalonável se $U_1 \leq U_{min}(1) = 1/k_1$, como dado na formula geral para U_{min}.
- T_2 é escalonável se $U_1 + U_2 \leq U_{min}(2)$.
- As condições acima valem para $k_2 \leq 2$. Caso contrário, $U_{min}(2) = 1/k_2$.

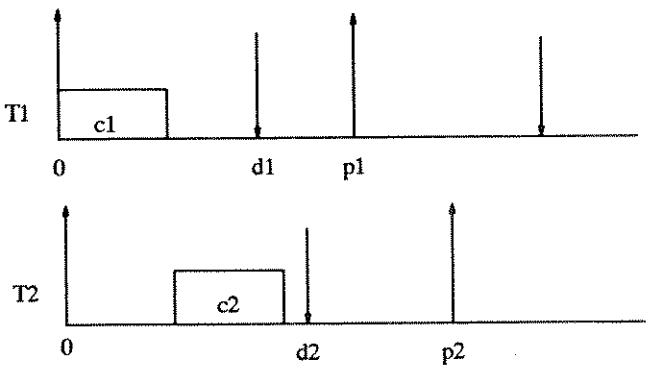

Figura 3.5: Condição $p_2 > p_1 > d_2$

3.3.4 Casos Extremos

Vamos considerar dois casos extremos:

- $d_2 \leq p_1 \leq p_2$
- $p_2 \leq p_1$

No primeiro caso, salientamos o significado dos algoritmos preemptivos dirigidos por prioridade, onde podemos ter preempção quando uma tarefa de maior prioridade requisita serviço da UCP no momento em que esta serve uma tarefa de menor prioridade. Como $p_1 \geq d_2$ e como T_1 tem maior prioridade, não haverá preempção. Logo, até d_2 deveremos executar apenas uma instância de cada uma das tarefas. Portanto, $c_1 + c_2 \leq d_2$, ou seja, $c_2 = d_2 - c_1$. Logo, $U = c_1/p_1 + (d_2 - c1)/p_2$, ou $U \geq c_1/p_1 + (d_2 - c_1)/p_2$, pois $p_1 \leq p_2$. Logo, $U \geq c_1/p_1 - c_1/p_2 + 1/k_2$. Portanto, o mínimo fator de utilização ocorre quando $c_1 = 0$, ou seja, $U = 1/k_2$. Neste caso, podemos entender o valor mínimo do fator de utilização como sendo $U_{min}(1,2) = 1((2/k_2)^1 - 1) + (1 - 1/k_2)$, ou seja, $U_{min}(1,2) = 1/k_2$. Ver Figura 3.5.

No segundo caso, teremos novamente que $c_2 = d_2 - c_1$, conduzindo à $U = c_1/p_1 + (d_2 - c_1)/p_2$, levando à $U \leq c_1/p_1 + (d_2 - c_1)p_1$, ou seja, c_1 deve ser máximo para U ser mínimo. Isto ocorre quando $c_1 = d_1$, ou seja, teremos $U_{min}(2) = 1/k_1 + (K-1)k_2K$. Ver Figura 3.6.

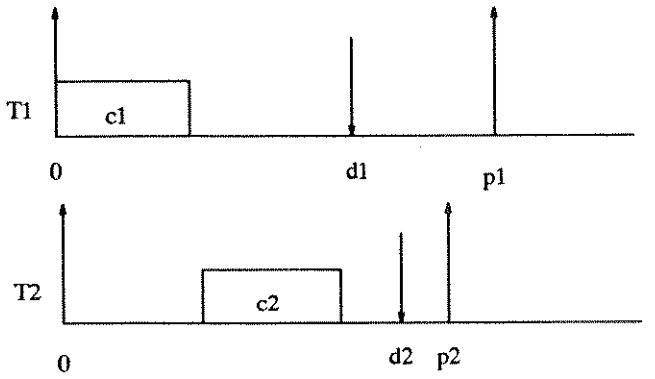

Figura 3.6: Condição $p_1 \geq p_2$

3.4 Condição de Escalonabilidade para um Sistema de 3 Tarefas

Dado um sistema $T = T_1, T_2, T_3$, onde $d_1 \leq d_2 \leq d_3$, vamos procurar verificar as questões de escalonabilidade, de forma a encontrar uma relação que possa levar a uma indução para um sistema de n tarefas. Inicialmente, notamos que a escalonabilidade de T_1, T_2 pode ser testada independentemente da escalonabilidade de T_3, dadas as características da política de prazo monotônico. Dependendo das relações entre d_2, p_1 e p_2, podemos inferir a escalonabilidade do conjunto T. Antes de continuarmos, vamos denominar $U(k)$ como sendo a soma do fator de utilização das k tarefas com menor prazo, ou seja, $U(k) = \sum_{i=1}^{k} c_i/p_i$. Vamos definir $U_{min}(i,j)$ como sendo o fator de utilização mínimo para garantir a escalonabilidade do conjunto $T = \{T_1..T_j\}$, levando se em conta que apenas i tarefas terão um valor não nulo de tempo de execução para minimizar o valor de U. Por exemplo, $U_{min}(1,1) = U_{min}(1) = 1/k_1$. Também, $U_{min}(2,2) = U_{min}(2)$. Contudo, quando $p_1 > d_2$, teremos que garantir a escalonabilidade de T_2 através de $U_{min}(1,2)$.

3.4.1 Casos Comuns para T_1, T_2

Vamos, inicialmente, supor que para T_1, T_2 vale a condição $d_1 \leq p_1 \leq d_2 \leq p_2$. Como $d_3 \leq p_3$, podemos ter as alternativas:

- $d_1 \leq p_1 \leq d_2 \leq p_2 \leq d_3 \leq p_3$

Devemos calcular o fator de utilização para a escalonabilidade de T_3, uma vez que a escalonabilidade de T_1 e T_2 já foram avaliadas e consideradas válidas. A escalonabilidade de T_3 depende dos valores de c_1 e c_2. Supondo que $d_3 \leq 2p_1$, nossa hipótese mais geral, tentaremos calcular $U_{min}(3)$, o fator de utilização que garante a escalonabilidade de T_3. Como observado anteriormente, o valor ótimo de c_1 para minimizar $U_{min}(3)$ é $c_1 = p_2 - p_1$, pois minimiza $U_{min}(3)$ e ao mesmo tempo permite a utilização total. Por outro lado, $c_2 = d_3 - p_2$ é o valor de c_2 que tem o mesmo efeito. Isto acontece porque T_3 é independente de T_1, mas não de T_2. Por exemplo, se $c_2 = p_3 - p_2$, não estaríamos minimizando $U_{min}(3)$, pois poderíamos ter um valor de $U_{min}(3)$ tal que o fator de utilização fosse menor e o sistema(T_3 neste caso), perderia seu prazo. Logo, $c_3 = d_3 - 2(c_1 + c_2)$.

Procedendo as avaliações para o cálculo de $U_{min}(3)$, obtemos finalmente:

$U_{min}(3) = 3((2/k_3)^{1/3} - (2k_3 + 1)/3k_3)$

Novamente, consideramos $c_1 = p_2 - p_1$ porque este valor minimiza o fator de utilização ao mesmo tempo que permite utilizar completamente a UCP. Se $c_1 > p_2 - p_1$ ou $c_1 < p_2 - p_1$, o fator de utilização aumentaria[LL73]. Contudo, para T_2 o conceito de utilizar completamente a UCP depende do valor de d_3, que impõe o limite de utilização.

- $d_1 \leq p_1 \leq d_2 \leq d_3 \leq p_2 \leq p_3$

  Novamente, supomos que T_1, T_2 são escalonáveis, ou seja, $U(1) \leq U_{min}(1,1)$ e $U(2) \leq U_{min}(2,2)$. A pergunta é: qual o valor mínimo que garante que T_3 não perde o prazo? Como $p_2 > d_3$, o fator de utilização mínimo ocorre quando $c_2 = 0$, ou seja, o sistema é escalonável com o maior fator de utilização garantido, como se fosse um sistema de duas tarefas, T_1, T_3. Logo, teremos a condição $U(3) \leq U_{min}(2,3)$ como suficiente para garantir a escalonabilidade de T_3.

- $d_1 \leq p_1 \leq d_2 \leq d_3 \leq p_3 \leq p_2$

  Neste caso, teremos o contrário do caso anterior, ou seja, c_2 deve ser máximo, de tal forma a minimizar $U_{min}(3)$. Situações deste tipo são incomuns e não serão analisadas para um sistema de muitas tarefas.

3.4.2 Casos Incomuns para T_1, T_2

Seja agora a seguinte relação para T_1 e T_2: $d_1 \leq d_2 \leq p_1 \leq p_2$. Novamente, T_1 é escalonável se $U(1) \leq U_{min}(1,1)$ e T_2 é escalonável se $U(2) \leq U_{min}(1,2)$. Dos seis possíveis casos que temos para considerar, vamos excluir os casos em que a ordem dos prazos não coincide com a ordem dos períodos. Portanto, temos as alternativas:

- $d_1 \leq d_2 \leq p_1 \leq p_2 \leq d_3 \leq p_3$

  Como $d_3 \geq p_2$, o valor de $U_{min}(3)$ deve ser obtido com $c_1 = p_2 - p_1$ e $c_2 = d_3 - p_2$, ou seja, $U(3) \leq U_{min}(3,3)$ é a condição suficiente.

- $d_1 \leq d_2 \leq p_1 \leq d_3 \leq p_2 \leq p_3$

  Neste caso, o valor mínimo de $U_{min}(3)$ deve ser encontrado quando $c_2 = 0$, ou seja, $U(3) \leq U_{min}(2,3)$ é a condição suficiente.

- $d_1 \leq d_2 \leq d_3 \leq p_1 \leq p_2 \leq p_3$

  Novamente, o valor mínimo para o fator de utilização ocorre quando $c_2 = 0$ e também $c_1 = 0$. Logo, $U(3) \leq U_{min}(1,3)$.

Concluímos nossa discussão observando que a escalonabilidade do conjunto T dependerá das relações entre os prazos e os períodos e as condições de escalonabilidade para cada tarefa independentemente.

3.5 Condição de Escalonabilidade para um Sistema de n Tarefas

No caso geral, podemos ter um número arbitrário de tarefas periódicas no sistema, cada qual com um prazo ligeiramente menor que o período. Adiante, vamos definir formalmente o que se entende por ligeiramente menor. Por hora, é suficiente dizer que isto imporá um intervalo de validade para os prazos, de tal forma a que certas condições sejam obedecidas. Suporemos a existência de um conjunto de tarefas $T = \{T_1, \ldots, T_n\}$, tal que $d_1 < d_2 < d_3 < \ldots < d_n$ e $p_i = k_i d_i$ para todo $i = 1 \ldots n$. Em todo o processo, vamos supor que os períodos obedecem a mesma relação de ordem dos prazos, ou seja, $p_1 < p_2 < \ldots, p_n$. Como o algoritmo Prazo Monotônico é ótimo, vamos aplicá-lo em nossas avaliações. Dois casos devem ser considerados:

- $d_i \leq p_i < d_{i+1}$ para $i = 1 \ldots n-1$ e $d_n \leq p_n$.

- $p_i \leq p_j$ para $i < j$.

Ambos os casos são representativos de tarefas relativamente bem comportadas. Vamos analisá-los a seguir.

3.5.1 Valores dos Períodos entre Prazos Adjacentes

Aqui consideramos o caso em que $d_i \leq p_i \leq d_{i+1}$, supondo que $d_n < 2p_1$. Veremos que esta última relação nos dará a melhor(mínimo) relação para o Fator de Utilização.

$U = \sum_{i=1}^{n} c_i/p_i$

No sentido de obtermos o valor mínimo de U utilizando completamente a UCP, devemos escolher valores dos tempos de execução de tal forma que uma mudança nestes valores não diminui o valor de U enquanto utilizando completamente a UCP. Tais valores são:

$c_i = p_{i+1} - p_i$ para $i = 1 \ldots n-2$

$c_{n-1} = d_n - p_{n-1}$

$c_n = d_n - 2(c_1 + \ldots + c_{n-1})$

Portanto, obtemos o valor de U dado como:

$U = \sum_{i=1}^{n-2}(p_{i+1} - p_i)/k_i d_i + (d_n - p_{n-1})/k_{n-1}d_{n-1} + (d_n - 2(c_1 + \ldots + c_{n-1}))/k_n d_n$

Como $\sum_{i=1}^{n-1} c_i = (p_2 - p_1) + \ldots + (d_n - p_{n-1}) = d_n - p_1$ e denominando $k_{i,j} = d_i/d_j$, temos:

$U = \sum_{i=1}^{n-2}(p_{i+1} - p_i)/k_i d_i + (d_n - p_{n-1})/k_{n-1}d_{n-1} + (2p_1 - d_n)/k_n d_n$

$U = \sum_{i=1}^{n-2}(k_{i+1}d_{i+1} - k_i d_i)/k_i d_i + (d_n - k_{n-1}d_{n-1})/k_{n-1}d_{n-1} + (2k_1 d_1 - d_n)/k_n d_n$

$U = \sum_{i=1}^{n-2}(k_{i+1}d_{i+1})/k_i d_i + d_n/k_{n-1}d_{n-1} + (1-n) + (2k_1 d_1/k_n d_n - 1/k_n$

$U = \sum_{i=1}^{n-2} k_{i+1}k_{i+1,1}/k_i k_{i,1} + k_{n,1}/k_{n-1}k_{n-1,1} + (1-n) + (2k_1 - k_{n,1})/k_n k_{n,1}$

Para $n = 2$, temos que $U = k_{2,1}/k_i k_{i,1} - 1 + (2k_1 - k_{2,1})/k_2 k_{2,1}$

Chamando $k_{2,1} = K$, como $k_{1,1} = 1$ teremos $U = K/k_1 + 2k_1/k_2 K - (k_2 + 1)/k_2$

Estas equações retornam ao sistema de duas tarefas já avaliado. No sentido de obtermos um valor mínimo de U, devemos minimizar U através das variáveis $k_{i,1}$, ou seja, da relação entre os prazos das tarefas e o prazo da primeira tarefa. Isto é suficiente, uma vez que estas variáveis são as únicas das quais U depende. A dependência de U com outras variáveis é implícita, portanto. Assim, procedemos a derivação parcial de U com respeito à cada uma das variáveis.

O fator de utilização pode ser visto como: $U = k_2 k_{2,1}/k_1 + k_3 k_{3,1}/k_2 k_{2,1} + \ldots + k_{n,1}/k_{n-1}k_{n-1,1} + (1-n) + (2k_1 - k_{n,1})/k_n k_{n,1}$

$\partial U/\partial k_{i,1} = k_i/k_{i-1}k_{i-1,1} - k_{i+1}k_{i+1,1}/k_i k_{i+1,1}^2$ para $i = 2 \ldots n-2$

Como $\partial U/\partial k_{i,1} = 0$, vêm que:

$k_{i,1}^2 = (k_{i-1}k_{i+1}/k_i^2)/k_{i-1,1}k_{i+1,1}$

Temos ainda:

$\partial U/\partial k_{n-1,1} = k_{n-1}/k_{n-2}k_{n-2,1} - k_{n,1}/k_{n-1}k_{n-1,1}^2$

Isto nos leva à:

$k_{n-1,1}^2 = (k_{n-2}/k_{n-1}^2)k_{n-2,1}k_{n,1}$

Por fim, para $i = n$ vem:

$\partial U/\partial k_{n,1} = 1/k_{n-1}k_{n-1,1} - 2k_1/k_n k_{n,1}^2$

$k_{n,1}^2 = (2k_1 k_{n-1}/k_n)k_{n-1,1}$

Obtemos, para cada i as seguintes relações:

$k_{2,1}^2 = (k_1 k_3/k_2^2)k_{3,1}$

$k_{3,1}^2 = (k_2 k_4/k_3^2)k_{2,1}k_{4,1}$

$\vdots$

$k_{n-2,1}^2 = (k_{n-3}k_{n-1}/k_{n-2}^2)k_{n-3,1}k_{n-1,1}$

$k_{n-1,1}^2 = (k_{n-2}k_{n-1}^2)k_{n-2,1}k_{n,1}$

$k_{n,1}^2 = (2k_1 k_{n-1}/k_n)k_{n-1,1}$

Elevamos a equação para $k_{n-1,1}$ à potencia de 2:

$k_{n-1,1}^4 = (k_{n-2}/k_{n-1}^2)^2 k_{n-2,1}^2 k_{n,1}^2$

Através da equação para $k_{n,1}$, obtemos:

$k_{n-1,1}^3 = (2k_1/k_n)(k_{n-2}^2/k_{n-1}^3)k_{n-2,1}^2$

Elevamos a equação para $k_{n-2,1}$ à potência de 3:

$k_{n-2,1}^6 = (k_{n-3}^3 k_{n-1}^3/k_{n-2}^6)k_{n-3,1}^3 k_{n-1,1}^3$

Através da equação para $k_{n-1,1}$ anterior, obtemos:

$k_{n-2,1}^4 = (2k_1/k_n)(k_{n-3}^3/k_{n-2}^4)k_{n-3,1}^3$

Elevamos a equação para $k_{n-3,1}$ à potencia de 4:

$k_{n-3,1}^8 = (k_{n-4}^4 k_{n-2}^4/k_{n-3}^8)k_{n-4,1}^4 k_{n-2,1}^4$

Através da equação para $k_{n-2,1}$, vem que:

$k_{n-3,1}^5 = (2k_1/k_n)(k_{n-4}^4/k_{n-3}^5)k_{n-4,1}^4$

Procedendo desta forma, obtemos finalmente:

$k_{3,1}^{n-1} = (2k_1/k_n)(k_2^{n-2}/k_3^{n-1})k_{2,1}^{n-2}$

Logo, da equação original para $k_{2,1}$, vem:

$k_{2,1}^{2(n-1)} = (k_1^{n-1}k_3^{n-1}/k_2^{2(n-1)})(2k_1/k_n)k_2^{n-2}k_{2,1}^{n-2}/k_3^{n-1}$

$k_{2,1} = 2^{1/n}(k_1/k_2)(1/k_n)^{1/n}$

Retornando às outras equações, teremos:

$k_{3,1} = 2^{2/n}(k_1/k_3)(1/k_n)^{1/n}$

$\vdots$

$k_{i,1} = 2^{(i-1)/n}(k_1/k_i)(1/k_n)^{(i-1)/n}$

para $i = 1 \ldots n-1$

$k_{n,1} = 2^{(n-1)/n}k_1(1/k_n)^{(n-1)/n}$

Voltando à U, vem:

$U = (k_2/k_1)k_{2,1} + (k_3/k_2)(k_{3,1}/k_{2,1}) + \ldots + (k_{n-1}/k_{n-2})(k_{n-1,1}/k_{n-2,1}) + (1/k_{n-1})(k_{n,1}/k_{n-1,1}) + (1-n) + (2k_1 - k_{n,1})/k_n k_{n,1}$

$U = (2/k_n)^{1/n} + \ldots + (2/k_n)^{1/n} + (1-n) + (2/k_n)^{1/n} - 1/k_n$

$U = n((2/k_n)^{1/n} - ((n-1)k_n + 1)/nk_n)$

3.5.2 Períodos Ordenados

Vamos considerar um sistema no qual a ordenação dos períodos segue a ordenação dos prazos, ou seja, se $d_i \leq d_j$, então $p_i \leq p_j$. Seja K o primeiro índice tal que $p_k \geq d_n$. Vamos verificar que as contribuições das tarefas de T_k a T_{n-1} devem aumentar o fator de utilização.

Teorema 3.5.1 *Seja um sistema T dado como no parágrafo anterior. Se U_1 é dado através de uma configuração de T tal que $c_k > 0$, para $p_k \geq d_n$, então para U_2 dado por $c_k = 0$ e $c_n = c_n + c_k$, teremos $U_2 \leq U_1$.*

Prova. Como $p_k \geq d_n$, teremos apenas uma execução de T_k em d_n. Como $p_n \geq p_k$, por hipótese, a contribuição para o fator de utilização de T_k é maior que a de T_n se a computação de T_k puder ser alterada para T_n e vice-versa. Portanto, toda a computação de T_k deve ser nula, para o mínimo do fator de utilização. Como isto vale para todo K, todas as tarefas com $p_k \geq d_n$ não contribuem para o cálculo do fator de utilização mínimo.

Para o sistema T dado acima, o fator de utilização mínimo deve levar em conta apenas as tarefas com períodos menores que o prazo máximo. Para estas tarefas, o comportamento das condições se mantém como na seção passada. O fator mínimo

será disponível se $c_i = p_{i+1} - p_i$, para $i < k$ e $c_{k-1} = d_n - p_{k-1}$. O valor de c_n é dado como anteriormente. Portanto, o comportamento do sistema é análogo ao anterior, retirando-se as $n - k + 1$ tarefas. Logo, o fator mínimo será:

$U = l((2/k_n)^{1/l} - ((l-1)k_n + 1)/lk_n)$

Só serão analisados escalonabilidades de tal forma que a ordem seja mantida, ou seja, se $d_i \leq d_j$ então $p_i \leq p_j$. Seja um conjunto de n tarefas tal que a ordem seja mantida. A questão é estabelecer a escalonabilidade para este conjunto de tarefas, dados os valores de U_{min}. Inicialmente, definimos $U_{min}(a,b) = a((2/k_b)^{1/a} - ((a-1)k_b + 1)/ak_b)$. Analisamos a escalonabilidade de cada tarefa separadamente:

- T_1 é escalonável se $U(1) \leq U_{min}(1,1)$
- T_2 é escalonável se $U(2) \leq U_{min}(2,2)$ para o caso em que $d_2 \leq p_1$ e $U(2) \leq U_{min}(1,2)$ para o caso em que $d_2 \geq p_1$
- T_j é escalonável se $U(j) \leq U_{min}(i+1,j)$, onde i é o índice tal que $p_i \leq d_j \leq p_{i+1}$.

Este procedimento contempla todos os casos possíveis onde a ordem seja mantida, e tem complexidade polinomial na solução da escalonabilidade, pois emprega apenas uma ordenação e uma busca para cada tarefa. O algoritmo suficiente é apresentado detalhadamente na Figura 3.7. Na figura, o conjunto de tarefas periódicas escalonáveis é dado por τ_{pa}. A chegada de uma nova tarefa, periódica ou esporádica, é representada por T_{ch} e a terminação de uma tarefa por T_{pt}. Na chegada da tarefa, verificamos a escalonabilidade do conjunto testando os valores dos fatores de utilização parciais em relação ao valor de U_{min} adequado. Caso o conjunto não seja escalonável, retornamos ao conjunto original.

3.6 Comparação entre as Metodologias

Vamos comparar as diversas metodologias apresentadas, tanto a desenvolvida neste trabalho, como as já descritas inicialmente, com relação aos parâmetros de custo, representado aqui pela complexidade computacional e desempenho, indicado pela capacidade de escalonar um grande número de tarefas através da condição suficiente. Faremos uma análise em separado para cada uma das metodologias descritas, comparando-as com a proposta deste trabalho. Enfatizamos que, em se tratando de um sistema de tempo real crítico dinâmico, onde as decisões muitas das vezes devem ser tomadas em tempo de execução, é fundamental que a complexidade computacional seja limitada convenientemente, de modo a não invalidar o comportamento do sistema.

Algoritmo PrazoMonotônicoSuficiente;
começo
τ_{pa} = conjunto de tarefas periodicas
se (chegada)
$T_{ch} = (C_{ch}, P_{ch}, k_{ch})$
$\tau_{pa} = \tau_{pa} \cup T_{ch}$
ordene τ_{pa} por prazo crescente
escalonavel=1
para cada $T_i \subset \tau_{pa}$
$U(i) = \sum_{j=1}^{i} U_j$
seja j tal que $p_{j-1} \leq d_i < p_j$
se $(U(i) > U_{min}(j,i))$
escalonavel=0
fimse
fimpara
se (escalonavel=0)
$\tau_{pa} = \tau_{pa} - T_{ch}$
fimse
se (partida)
$\tau_{pa} = \tau_{pa} - T_{pt}$
fimse
fim

Figura 3.7: Algoritmo Suficiente

3.6.1 Metodologia do Bloqueio

A abordagem do bloqueio considera que o prazo de uma tarefa pode ser menor que o seu período através de um bloqueio, considerado como um intervalo de tempo dentro do período que a tarefa terá que esperar por outras tarefas. O custo computacional deste teste é linear, mas seu desempenho não é interessante quando comparado com o método de variação arbitrária do prazo. Abaixo, mostramos que o método VAP é mais geral do que o método de bloqueio, uma vez que qualquer conjunto escalonável pela estratégia de bloqueio será também escalonável pela estratégia VAP.

Teorema 3.6.1 *A abordagem VAP é mais geral que a abordagem de bloqueio(BLQ).*

Prova. Seja τ um conjunto de tarefas periódicas tal que $k_i = p_i/d_i$. Seja $K = max(k_i)$. Portanto, $max(1 - 1/k_i) = 1 - 1/K$. Como $B_i/p_i = 1 - 1/k_i$, e supondo que BLQ garante que τ é escalonável, teremos que $U \leq n(2^{1/n} - 1) - (1 - 1/K)$[SG90a]. Sabemos que $1/k_n \geq 1/K$, então $n((2/k_n)^{1/n} - 1) + (1 - 1/k_n) \geq n((2/K)^{1/n} - 1) + (1 - 1/K)$. Por outro lado, $n((2/K)^{1/n} - 1) + (1 - 1/K) \geq n(2^{1/n} - 1) - (1 - 1/K)$. Portanto, $U \leq n((2/k_n)^{1/n} - 1) + (1 - 1/k_n)$, ou seja, τ é escalonável através de VAP. Para verificar que VAP é mais geral, seja τ constituído de $\{T_1, T_2\}$, onde $T_i = (c_i, d_i, p_i)$. Se $1/k_1 = 0.8$ e $1/k_2 = 0.7$. Logo, τ é escalonável por VAP se $U \leq 0.66$. Consideremos $T_1 = (20, 40, 50)$ e $T_2 = (17, 49, 70)$. Logo, $U = 0.642$. Contudo, τ não é escalonável por BLQ.

3.6.2 Variação Uniforme do Prazo

Em termos de complexidade, a variação uniforme do prazo(VUP) apresenta o mesmo custo computacional que a metodologia proposta aqui, que se aplica à variação arbitrária do prazo(VAP). Ambos se baseiam em um teste do fator de utilização como indicativo de escalonabilidade e, portanto, são algoritmos lineares na entrada.

O desempenho da metodologia de variação uniforme do prazo nunca é melhor que o desempenho da metodologia proposta neste trabalho, em função desta última ser uma generalização da primeira e englobá-la. De fato, podemos mostrar que, dado um conjunto de tarefas periódicas no qual a variação uniforme do prazo garante a escalonabilidade, então a proposta apresentada aqui também garantirá a escalonabilidade. O contrário não será verdadeiro, conforme observaremos adiante.

Teorema 3.6.2 *A VAP garante a escalonabilidade de um conjunto de tarefas cuja escalonabilidade é garantida pela metodologia de variação uniforme do prazo(VUP).*

Prova. Seja T um conjunto de tarefas periódicas tal que $K = p_i/d_i$ para todas as tarefas em T. Como T é escalonável através da variação uniforme do prazo, então teremos que $U \leq n((2/K)^{1/n} - 1) + (1 - 1/K)$[Leh90]. Portanto, como $k_n = K$, $U \leq n((2/k_n)^{1/n} - 1) + (1 - 1/k_n)$. Logo, a escalonabilidade também será garantida pela metodologia apresentada. Considere agora $k_i = p_i/d_i$ para cada tarefa T_i. Seja T' o conjunto derivado de T, fazendo-se $k_i = K$, onde $K = max(k_j)$ para todas as tarefas. Portanto, os prazos de todas as tarefas de T' são mais restringidos que os prazos das tarefas similares de T. Logo, se T' for escalonável, T também será. Como T' é escalonável se $U \leq n((2/K)^{1/n} - 1) + (1 - 1/K)$. Como $k_n \leq K$, T será escalonável através de VAP.

Podemos verificar que o conjunto pode ser escalonável através da condição deste trabalho e não o ser através da variação uniforme observando o seguinte contra exemplo:

Seja um conjunto de duas tarefas, tal que $p_1 = k_1 d_1$ e $p_2 = k_2 d_2$. Suponhamos $k_1 = 0.9$ e $k_2 = 0.95$. Logo, $k_1 > k_2$. Este conjunto é escalonável através da abordagem VAP se $U \leq 0.807$. Entretanto, só se pode garantir a escalonabilidade por VUP se $U \leq 0.783$. Escolhendo apropriadamente o valor de U, podemos ter um conjunto que não é escalonável por VUP, mesmo que a abordagem de VAP o garanta. Por exemplo, se $c_1 = 20$, $p_1 = 50$, $c_2 = 28$ e $p_2 = 70$, $U = 0.8$ e, portanto, não é escalonável por VAP.

3.6.3 Análise da Interferência

A metodologia que garante a escalonabilidade através da análise da interferência de tarefas mais prioritárias sobre uma tarefa qualquer se baseia em uma série de cálculos de interferências, cada qual com um diferente grau de complexidade[Aud90]. O cálculo exato da interferência é um processo PSEUDOPTIME e, portanto, completamente inviável para os nossos propósitos. A forma mais simples de se calcular a interferência tem uma complexidade quadrática no tamanho da entrada, sendo portanto $O(n^2)$. Dentre as diferentes formas de se especificar a interferência, as variações de complexidade são mínimas, pois envolvem vários cálculos que podem ser otimizados. Entretanto, do ponto de vista prático, podem não ser convenientemente aplicadas em sistemas de tempo real. Portanto, vamos nos ater a formas mais simples de especificação da interferência, a saber, a interferência provocada por tarefas mais prioritárias sobre a tarefa T_j será, supondo que as tarefas estejam ordenadas por prazo crescente:

$$I_j = \sum_{k=1}^{j} \lceil d_j/p_k \rceil$$

A escalonabilidade de um conjunto de n tarefas pode ser testada de acordo com as equações:

$c_i/d_i + I_i/d_i \leq 1$ para todas as tarefas.

3.7 Comentários Finais

A metodologia de variação arbitrária de prazo é especialmente útil quando o número de tarefas periódicas a ser tratado é grande, de tal forma que a diferença entre $O(n)$ e $O(n^2)$ faça sentido. Objetivando-se o princípio de previsibilidade, o emprego de abordagem VAP conjugado com a abordagem de múltiplas versões conduz à uma versão escalonável, de tal forma que é sempre possível escolher-se a versão do conjunto periódico de tal forma que a escalonabilidade seja garantida.

Pudemos verificar que o teste proposto neste capítulo é o mais geral dentro de sua classe de complexidade, uma vez que permite prazos arbitrariamente menores que o período, desde que a ordem dos prazos seja equivalente à ordem dos períodos. Entretanto, pudemos também verificar que o valor do fator de utilização que garante a escalonabilidade se torna extremamente baixo quando o prazo difere substancialmente do período, e esta influência é mais sentida para tarefas com prazo maior, ou seja, tarefas com um alto fator de aperto deveriam apresentar um baixo valor do período, ou seja, quanto menor é o prazo, maior poderia ser a diferença entre o prazo e o período, relativamente. De qualquer maneira, a partir de um determinado ponto, a abordagem de prioridades fixas torna-se impraticável.

Capítulo 4

Escalonamento On-line de Tarefas Esporádicas

4.1 Introdução

Verificamos que o uso de uma associação de prioridades fixa, embora apresente algumas vantagens, torna-se incoveniente para o tratamento de escalonabilidade em tempo de execução para tarefas periódicas equivalentes com prazo razoavelmente inferior ao período. Nestes casos, o fator de utilização da UCP pode ser drasticamente baixo para garantir uma condição suficiente de escalonabilidade.

Neste capítulo, procuraremos analisar a questão da escalonabilidade de sistemas dinâmicos em função de uma associação de prioridade variável. Para tarefas periódicas, associação de prioridades do tipo algoritmo de próximo prazo apresenta a garantia de ser ótima em relação a outras associações. Vamos aqui novamente apresentar uma metodologia que procurará estabelecer formalmente as noções de garantia de escalonamento para tarefas periódicas e esporádicas. Devemos observar que, devido às suas características, o algoritmo de próximo prazo não prevê, de antemão, qual ou quais tarefas estarão comprometidas no caso do conjunto inteiro não puder ser escalonável. Discutiremos algumas noções de garantia, em caso de sobrecarga.

As abordagens clássicas para o tratamento de tarefas esporádicas no sentido de se produzir um escalonamento válido consistem em várias metodologias, entre as quais: a) a substituição de uma tarefa periódica previamente garantida pela tarefa esporádica, b) a avaliacão da disponibilidade de carga do sistema baseada em um teste de escalonamento em um intervalo e c) a determinação do máximo tempo livre possível no intervalo de validade da tarefa. Em alguns casos, é possível garantir em tempo prévio à

execução se o escalonamento é válido, uma vez que todas as tarefas esporádicas devem usar o tempo de processador reservado para a tarefa periódica equivalente[BMR90]. Este aspecto é importante para a previsibilidade. Em outros casos, a avaliação é feita em tempo de execução e pode ser demorada, e além disto, pode não ser exata, ou seja, o teste de garantia pode falhar mesmo se algum algoritmo pudesse escalonar a tarefa[Mok83]. Por último, pode-se ter um teste exato, requerendo uma avaliação *off-line* e *on-line* do tempo de espaço livre em um intervalo e comparando se a tarefa esporádica atende aos requisitos deste espaço livre[CC89].

Para se aplicar a abordagem *off-line*, deve-se ter um conhecimento completo do sistema, ou seja, o sistema deve ser conhecido previamente ao tempo de execução e os parâmetros das tarefas devem manter-se constantes. Quando as aplicações requerem que as tarefas periódicas mudem seus parâmetros, este conhecimento prévio perde seu significado. Portanto, uma abordagem dinâmica que permita o teste de escalonamento em tempo de execução é mais conveniente nestes casos, refletindo as condições correntes do sistema.

As noções de garantia para sistemas que tomam decisões em tempo de execução são diferentes da forma de garantia pré-tempo de execução, uma vez que neste último caso a garantia é sempre incondicional, ou seja, uma vez garantida, a tarefa continua em tal estado enquanto o sistema estiver em trabalho. Quando as decisões sobre escalonabilidade são dinâmicas, deve-se decidir quais tarefas têm prioridades sobre as outras, no sentido de garantir as tarefas mais prioritárias. As regras de garantia devem levar em conta fatores como, por exemplo, a incondicionalidade da garantia, ou seja, a garantia estabelecida uma vez deve ser mantida até que a tarefa deixe o sistema ou seu prazo seja vencido. Outros fatores se relacionariam com a função da tarefa(periódica ou esporádica). As tarefas periódicas, representando computações que devem ser realizadas com frequência definida têm, em geral, prioridade sobre as tarefas esporádicas. Entre as tarefas esporádicas, pode-se assumir uma garantia baseada na ordem de chegada, ou seja, uma vez que uma tarefa esporádica foi garantida, ela cumprirá seu prazo, independente da chegada de outras tarefas esporádicas. Outras formas de garantia, contudo, são possíveis [Sta88a].

Estabelecida a forma de garantia que deve ser obedecida por tarefas periódicas e esporádicas, é importante poder verificar se o projeto do sistema permite que a garantia será cumprida conforme especificado. Os algoritmos de escalonamento deverão refletir a forma de garantia, e isto é indicado através da especificação formal do comportamento destes algoritmos em face dos eventos possíveis. Na finalização do capítulo, apresentaremos uma maneira de especificar formalmente um sistema como o discutido [VTB83].

4.2 Metodologias *Off-line* para Escalonamento de Tarefas Esporádicas

Esta seção é uma revisão das condições apresentadas em trabalhos prévios, relativos ao escalonamento de tarefas esporádicas. O problema consiste em determinar se uma tarefa esporádica pode ser escalonada em um sistema de tarefas periódicas que tem a garantia de cumprimento de suas condições temporais. Este conjunto de tarefas periódicas é conhecido previamente ao tempo de execução e suporemos que ele é estático. Vamos também supor que o escalonamento é preemptivo e a atribuição de prioridade é baseada na política de próximo prazo, uma vez que esta política é ótima para as condições consideradas aqui[LL73].

Consideramos o escalonamento de uma tarefa esporádica $T_s = (c_s, r_s, d_s, p_s)$, onde c_s é o tempo de computação, r_s é o tempo de chegada(pronto) e d_s é o prazo relativo à r_s, ou seja, a tarefa deve executar no intervalo $[r_s, r_s + d_s]$. No sentido de satisfazer tal condição de execução, precisamos estar certos de que no dado intervalo, pode-se incluir a quantidade c_s sem prejudicar a garantia de escalonamento válido das tarefas periódicas. Estas últimas são representadas pelo conjunto $\tau = \{T_{pi}\}$, onde $i = 1..m$ e $T_{pi} = (c_{pi}, r_{pi}, d_{pi}, p_{pi})$. Nós definimos a janela periódica P como sendo o mínimo múltiplo comum dos períodos do sistema τ. Precisamos garantir a escalonabilidade de T_s mantendo o conjunto τ escalonável. Ilustra-se o escalonamento de um conjunto τ_{ex} de tarefas periódicas no intervalo $[0, 30]$, através da política de próximo prazo na variante o mais cedo possível(PPA-PP Adiantado) e o tempo livre gerado por este escalonamento[CC89]. Na Figura 4.1, ilustramos o escalonamento das tarefas $T_1 = (6, 0, 10, 10)$ e $T_2 = (4, 0, 16, 16)$.

A seguir, nós descreveremos brevemente quatro abordagens principais para o escalonamento do problema colocado.

4.2.1 Substituição de Tarefas Periódicas

Diz-se que o objetivo do estudo sobre garantia de cumprimento de restrições temporais de tarefas esporádicas é sintetizar um escalonamento pré-tempo de execução para um algoritmo em tempo de execução que escalone o processador[Mok84]. Através deste algoritmo *off-line*, a escalonabilidade pode ser determinada e se o conjunto de tarefas é escalonável, um algoritmo *on-line* pode ser construído para escalonar com sucesso o conjunto esporádico.

Nesta abordagem, os parâmetros da tarefa esporádica dada poderiam ser substituídos pelos parâmetros de uma tarefa periódica equivalente T_s*, tal que $c_s^* = c_s, p_s^* =$

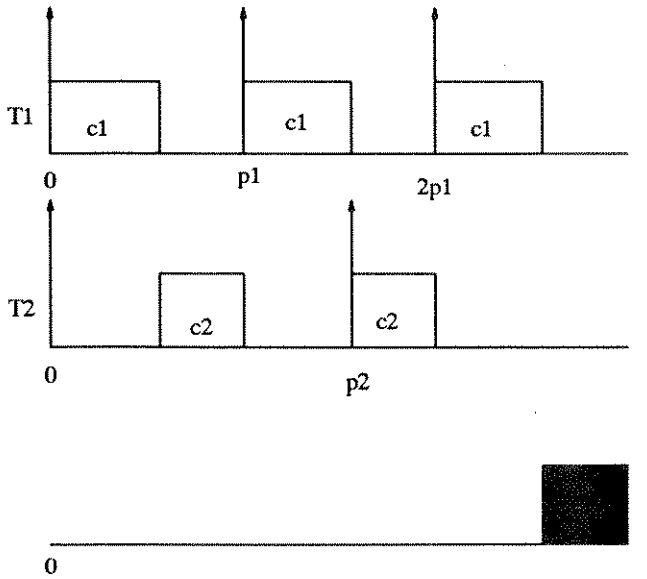

Figura 4.1: Exemplo de um escalonamento utilizando PPA

$min(d_s - c_s + 1, p_s), d_s^* = c_s$ Se o conjunto resultante de tarefas periódicas representado por $\tau \cup T_s^*$ for escalonável, então será também o conjunto $\tau \cup T_s$. O contrário, contudo, não é verdade.

Para fornecer um teste de escalonamento completo, faz-se necessária outra transformação de tarefas esporádicas em periódicas[BMR90]. Nesta abordagem, um conjunto finito legal de requisições derivado do conjunto de requisições da tarefa esporádica é construído, de tal forma que o sistema esporádico é válido se e somente se ele pode ser escalonado com respeito à este conjunto específico de requisições.

A principal diferença entre as abordagens reside no fato de que a primeira fornece uma condição suficiente para a garantia de cumprimento de prazos da tarefa esporádica, enquanto que a segunda fornece uma condição completa(exata). Em termos de ambiente, ambas se baseam em um conjunto estático de tarefas periódicas. Sabe-se que a melhor solução para estas abordagens tem complexidade PSEUDOPTIME[BMR90].

4.2.2 A Abordagem de Disponibilidade de Carga

Esta abordagem lida com o escalonamento dinâmico de tarefas de tempo real crítico, em um ambiente distribuído[SRC85]. Uma tarefa é caracterizada pelos seus parâmetros, dados em seções anteriores, podendo ser classificada em periódica ou

não-periódica. A tarefa não-periódica ocorre no sistema no máximo uma vez, com um tempo de chegada imprevisto. Isto significa que cada instância de uma tarefa esporádica deve ser encarada como uma tarefa não-periódica independente. Neste modelo, supomos que existe um conjunto inicial de tarefas periódicas em cada nó da rede, e que as execuções referentes a este conjunto devem ser garantidas. Não se discute a possibilidade de haver reconfiguração de tarefas periódicas em cada nó, embora tal possibilidade não seja explicitamente excluída.

Estamos interessados no comportamento do escalonador em um nódo particular da rede, uma vez que as decisões de escalonamento em cada nódo são locais e independentes dos outros nódos. Um conjunto de tarefas cujo escalonamento é válido é parte do sistema de cada nódo e tarefas não-periódicas podem chegar em qualquer tempo, induzindo o escalonador local a verificar se a tarefa poderá ser garantida naquele nódo, ou seja, se ela poderá executar completamente até seu prazo final. A garantia da tarefa é incondicional, e se a tarefa não for garantida em um nódo particular, ela pode tentar migrar sua execução para outro nódo, de acordo com o algoritmo distribuído.

As rotinas que analisam a escalonabilidade das tarefas fazem uso de tabelas que armazenam informações sobre as mesmas. Uma tarefa recém chegada pode ter sua execução garantida em um nódo se a disponibilidade de carga do nódo, no intervalo de validade de execução da tarefa for suficiente para conter sua computação. A disponibilidade de carga é definida como a quantidade de tempo de UCP disponível em um dado intervalo. Neste trabalho, esta disponibilidade não é calculada explicitamente, mas é levada em conta implicitamente no teste de garantia.

4.2.3 Abordagem de Máximo Tempo Livre

De acordo com a condição exata desta abordagem, deve-se avaliar o máximo de tempo livre no intervalo que T_s está habilitada à executar, no sentido de se obter uma resposta para a questão da escalonabilidade[CC89]. Esta quantidade de tempo pode ser obtida através de um procedimento que, no intervalo dado, leva em conta o tempo total que a UCP está livre, acrescido do máximo tempo de atraso que as tarefas periódicas podem sofrer, sem entretanto, invalidarem suas restrições temporais. Isto torna possível a execução válida da tarefa esporádica no intervalo, com um tempo de computação no máximo igual ao tempo livre obtido.

Resumidamente, primeiro escalonamos o conjunto τ de acordo com o algoritmo de Próximo Prazo, na variante de atraso máximo. Neste caso, as tarefas são escalonadas o mais tarde possível, mantendo-se a prioridade maior para quem tem o menor prazo corrente. Deste escalonamento, obtemos as tabelas TTC e TTLE, respectivamente, tabela de tempo de chegada e tabela de tempo-livre estático, que descrevem todas as

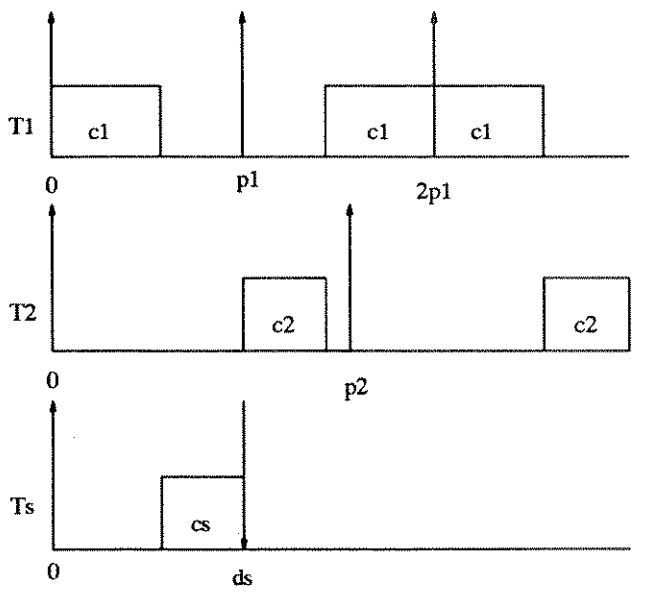

Figura 4.2: Espaço Livre Disponível

requisições de τ na sua janela periódica e os intervalos onde a UCP está livre. Através destas tabelas e dos parâmetros de T_s, podemos avaliar a quantidade $\Omega_\tau^{PP}(r_s, r_s + d_s)$, que é o máximo tempo livre disponível em $[r_s, r_s + d_s]$. Segue-se que T_s será aceita se e somente se $c_s \leq \Omega_\tau^{PP}(r_s, r_s + d_s)$. Devemos notar que r_s é qualquer um dos tempos da janela periódica P. Se aceita, T_s será escalonada juntamente com o conjunto τ através da política do Próximo Prazo. Um exemplo do teste de garantia com o escalonamento pode ser ilustrado com $T_s = (4, 0, 10, 10)$ e $\tau = \tau_{ex}$. Ver Figura 4.2, com T_1 e T_2 dados como na Figura 4.1.

4.2.4 Escalonamento por Lista de Quadros Livres

Dado um conjunto τ de tarefas periódicas, formamos uma estrutura de dados conhecida como Lista de Quadros(LQ), composta de elementos que são intervalos onde a UCP está ocupada, considerando-se o escalonamento através da política de Próximo Prazo comum, aplicada ao conjunto τ[SZ92]. Para o exemplo dado anteriormente, teremos $LQ = \{(0, 26), (30, 46), (48, 58), (60, 76)\}$.

O teste de escalonabilidade requerido é feito baseado na política de Próximo Prazo aplicada à tarefa T_s e em um subconjunto de tarefas previamente escalonadas que contém as instâncias das tarefas que são influenciadas pela presença de T_s. Isto significa

que o tamanho do subconjunto depende do intervalo de T_s. Se qualquer instância do subconjunto não for escalonável, então T_s não será aceita; em caso contrário, T_s é aceita. O subconjunto a ser reescalonado é determinado usando a LQ, que está disponível no momento de chegada de T_s.

No exemplo dado anteriormente, se $T_s = (5, 10, 20, p_s)$, devemos reescalonar os subconjuntos $\{(0, 26), (30, 46)\}$ da LQ no sentido de fornecer uma quantidade de tempo livre necessária e suficiente para escalonar T_s. Podemos ver que neste caso algumas instâncias não cumprirão seus prazos, o que significa que T_s não será aceita.

4.3 Metodologias para Escalonamento de Tarefas Esporádicas *On-line*

Lidamos com a possibilidade de termos um conjunto de tarefas periódicas que podem mudar seus parâmetros como resultado de uma aplicação particular. Neste caso, temos que encontrar um procedimento que levará em conta tais mudanças, representadas matematicamente por novos valores dos tempos de execução e período para as tarefas, bem como por novas tarefas. Nosso objetivo é encontrar um teste que seja rápido o suficiente para ser utilizado em tempo de execução e que responda se uma dada tarefa esporádica pode ser escalonada com sucesso em um sistema como o descrito. Este teste deve ser utilizável, no sentido de que uma grande parte das tarefas que sejam consideradas não escalonáveis, realmente o sejam, ou seja, se o teste de escalonabilidade pudesse ser feito previamente ao tempo de execução, as tarefas não seriam escalonáveis.

4.3.1 Teste Suficiente

No caso de todas as tarefas periódicas em um sistema τ fazerem uma requisição simultânea, sabemos que o escalonamento válido para τ depende apenas do valor de U_τ, uma vez que estamos supondo que as tarefas em τ são independentes[LL73]. Nós podemos então propor uma condição suficiente para o escalonamento de T_s, que se mostrará útil para uma grande classe de aplicações. Este teste é baseado no princípio de que qualquer tempo livre não ocupado pelas tarefas periódicas pode ser utilizado como um servidor esporádico[SSL89]. Vamos colocar a condição dada no teorema abaixo:

Teorema 4.3.1 *Seja $T_s = (c_s, r_s, d_s, p_s)$ uma tarefa esporádica e τ um conjunto de tarefas periódicas onde o prazo de cada tarefa é igual ao valor do respectivo período.*

Uma condição para a escalonabilidade de $\tau \cup T_s$ *é dada por* $c_s \leq (1 - U_\tau)d_s$, *onde* U_τ *é o fator de utilização de* τ.

Prova. Inicialmente, considere qualquer conjunto τ' de tarefas periódicas e independentes, com $d_i = p_i$ para todas as tarefas em τ'. Cada tarefa em τ' tem um tempo de pronto s_i, ou seja, a primeira instância de T_i requisita a UCP em s_i. No sentido de demonstrar a escalonabilidade de τ', devemos mostrar que[BRH91] a)$U_{\tau'} \leq 1$ and b)$\sum_{i=1}^{n} \eta_i(t_1, t_2)c_i \leq t_2 - t_1$ para $0 \leq t_1 < t_2 \leq max\{s_i\} + 2P$

onde s_i é o tempo de pronto da tarefa T_i. Supondo que a) seja verdadeira, vamos mostrar b) notando que $\eta_i(t_1, t_2)$ é dado pelo número de k's diferentes que satisfazem simultaneamente as equações $t_1 \leq s_i + kp_i$ e $s_i + (k + 1)p_i \leq t_2$. Nós temos três possibilidades:

1.$t_2 \leq s_i$: chegamos à $\eta_i(t_1, t_2) = 0$

2.$t_1 \leq s_i$: temos que $\eta_i(t_1, t_2) = \lfloor (t_2 - t_1)/p_i \rfloor$

3.$s_i \leq t_1$: temos dois casos:

a.$\lceil (t_1 - s_i)/p_i \rceil > \lfloor (t_2 - s_i)/p_i \rfloor$ o que torna $\eta_i(t_1, t_2) = 0$.

b.$\lceil (t_1 - s_i)/p_i \rceil \leq \lfloor (t_2 - s_i)/p_i \rfloor$ o que torna $\eta_i(t_1, t_2) = \lfloor (t_2 - s_i)/p_i \rfloor - \lceil (t_1 - s_i)/p_i \rceil$.

Disto, temos que, baseados no caso 1 e 3a., $\eta_i(t_1, t_2) \leq (t_2 - t_1)/p_i$. Ainda, do caso 2 temos que $\eta_i(t_1, t_2) \leq (t_2 - s_i)/p_i \leq (t_2 - s_i)/p_i$.

Do caso 3b, chegamos à $\eta_i(t_1, t_2) = \lfloor (t_2 - s_i)/p_i \rfloor - \lceil (t_1 - s_i)/p_i \rceil \leq (t_2 - s_i)/p_i - (t_1 - s_i)/p_i = (t_2 - t_1)/p_i$.

Portanto, obtemos $\sum_{i=1}^{n} (t_2 - t_1)c_i/p_i = U_{\tau'}(t_2 - t_1) \leq (t_2 - t_1)$

Então, τ' é escalonável e vemos que se $U_{\tau'} < 1$, o conjunto $\tau \cup T_s$ também é escalonável. Agora tomemos $T^* = (c^*, p^*)$, de tal forma que $c^* = C_s$ e $p^* = d_s$. Disto vêm que $U_{\tau \cup T^*} = U_\tau + c^*/p^* = U_\tau + C_s/d_s \leq U_\tau \cup (1 - U_\tau) = 1$, por hipótese. Logo, $\tau \cup T^*$ é escalonável e se considerarmos T_s como a primeira instância de T^*, então $\tau \cup T_s$ é escalonável.

Vemos que a condição suficiente para o escalonamento de T_s lida com uma comparação simples entre o tempo de computação de T_s e o mínimo tempo livre que é garantido existir no intervalo de T_s de tal forma a não prejudicar as tarefas periódicas previamente garantidas. Ver Figura 4.3, onde teremos $c_s = 1.5$.

Embora um princípio simples para escalonar uma tarefa esporádica, veremos que esta abordagem pode ser muito interessante. Podemos observar que a condição fornecida no Teorema 4.3.1 lida apenas com a parte suficiente da garantia, ou seja, podemos encontrar tarefas esporádicas que seriam escalonáveis, mas que não podem ser garantidas pelo nosso algoritmo. Será visto que, para uma grande classe de aplicações, a

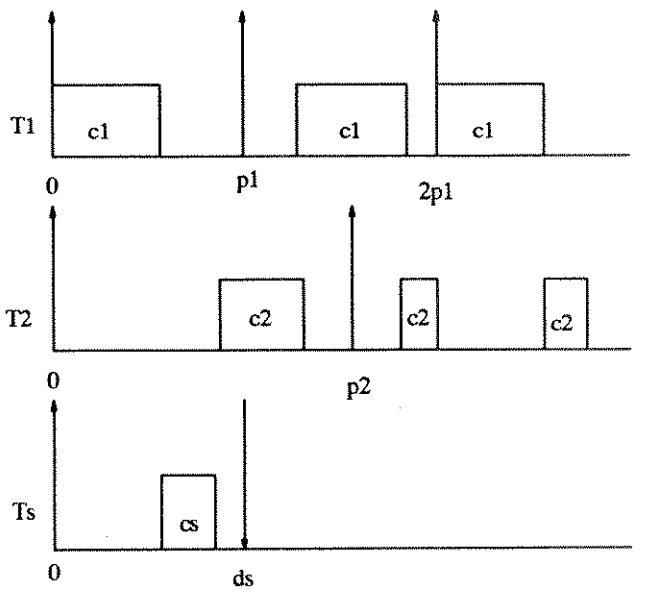

Figura 4.3: Espaço Livre da Condição Suficiente

condição suficiente fornece resultados muito próximos aos que seriam dados por uma condição exata. As similaridades entre os dois testes podem ser observadas nas equações abaixo. Se denominarmos $\mathcal{I}(d)$ como a quantidade $(1 - U_\tau)d$, a condição suficiente afirma que T_s é escalonável se $c_s \leq \mathcal{I}(d_s)$. No sentido de comparar ambas abordagens, definimos $\Omega(d)$ como uma média de $\Omega_\tau^{PP}(r_s, r_s + d_s)$ para todo r_s em P. Tomemos R como:

$$R = \Omega(d)/\mathcal{I}(d)$$

R é uma medida do quanto nossa aproximação está eficiente no processo de garantir as tarefas a serem escalonadas. Podemos ver que R é maior que 1, indicado pelo fato de que $\Omega(d) > \mathcal{I}(d)$. Vemos que a medida que R se aproxima da unidade, a nossa aproximação torna-se melhor. Vamos avaliar posteriormente os valores típicos de R para vários conjuntos de tarefas periódicas.

4.3.2 Condição Exata

Para alguns sistemas periódicos, pode-se ter que a condição suficiente descrita anteriormente não seja eficiente, devido à uma possível sobrecarga no sistema. Outra possibilidade é que uma dada tarefa esporádica possa ter um prazo muito pequeno, tornando a utilização da condição suficiente inoportuna, pois R é alto nestes casos.

Por outro lado, a condição exata pode também não ser aplicável, uma vez que estamos tratando de um sistema periódico dinâmico. Portanto, a condição suficiente não é viável e a condição exata de máximo tempo livre não está disponível.

Uma forma de permitir um teste de escalonabilidade exata para T_s vem do exame do escalonamento gerado por alguma política de atribuição de prioridades sobre as tarefas do sistema. Por exemplo, a chegada de uma tarefa esporádica T_s em um tempo r_s pode ser analisada com respeito à garantia de validade do escalonamento, desde que se construa um escalonamento para τ e T_s e se verifique se os prazos de todas as instâncias foram cumpridos. Infelizmente, tal escalonamento leva à um procedimento intratável computacionalmente, a menos que tenhamos garantias de parada que diminuam o número de instâncias que devem ser avaliadas, sem invalidar o resultado. Os passos para o teste da escalonabilidade são:

- Testar a escalonabilidade de τ através do fator de utilização.
- Construir um escalonamento tal que todas as instâncias de τ e T_s satisfaçam seus requerimentos temporais.

Deste procedimento, devemos estar aptos a responder que quantidade mínima de computação T_s pode executar no intervalo $[r_s, r_s + d_s]$ sem o prejuízo de outras instâncias. Basicamente, o procedimento irá escalonar instâncias através da política de Próximo Prazo, incluindo T_s e supondo o maior valor possível para a computação de T_s, que é o máximo tempo livre no intervalo $[r_s, r_s + d_s]$. O valor inicial do máximo tempo livre é tomado como sendo o tempo livre em $[r_s, r_s + d_s]$ restante após a execução de todas as instâncias das tarefas periódicas com prazo não maior que $r_s + d_s$. Todas as instâncias das tarefas periódicas com prazo maior que $r_s + d_s$ serão escalonadas após $r_s + d_s$. Portanto, inicialmente a computação de T_s, c_s tomará o valor do máximo tempo livre(TL) em $[r_s, r_s + d_s]$, levando em conta apenas as instâncias com prazo não superior à $r_s + d_s$.

Para cada instância j de uma tarefa periódica, definimos o atraso $L(j)$ como sendo a quantidade de atraso máximo que tal instância pode sofrer sem perder seu prazo. Sendo $D(j)$ o prazo da instância j e $E(j)$ seu tempo de término, de acordo com alguma política, então definimos $L(j) = D(j) - C(j)$. Se o atraso de j é não-negativo, a instância pode ser atrasada no máximo pelo valor do atraso. Contudo, um valor negativo do atraso indica que a instância não está cumprindo seus requisitos temporais. É nosso objetivo fundamental manter todos os atrasos com valores não-negativos.

No processo de encontrar o valor inicial do máximo tempo livre, é possível que algumas instâncias com prazo após $r_s + d_s$ tenham um valor negativo do atraso. Isto significa que a estimativa inicial para o valor máximo do tempo livre foi superestimada.

Devemos proceder à um ajuste de valores de atrasos para as instâncias com prazo maior que $r_s + d_s$ de tal forma que nenhuma delas tenha um valor negativo. No processo de ajustar os valores de atraso, devemos decrementar o valor inicial do máximo tempo livre do valor de atraso, sempre que encontramos um valor negativo do atraso. Procedendo desta forma, garantimos que todas as instâncias periódicas irão cumprir seus prazos, bem como encontramos o máximo valor possível para c_s.

O algoritmo é útil como um teste exato desde que a construção do escalonamento não se torne intratável. No sentido de provar formalmente que o procedimento é correto e útil, devemos mostrar que as propriedades seguintes são válidas:

- O tempo livre(TL) fornecido por este algoritmo é a quantidade máxima de espaço livre em $[r_s, r_s + d_s s]$ de tal forma que todas as instâncias sejam cumpridas.
- O tempo livre(TL) é calculado de tal forma que o atraso de cada instância é não-negativo. Se alguma instância com prazo maior que $r_s + d_s$ tem um atraso nulo, uma computação $c_s = TL + 1$ torna este atraso negativo.

Estas propriedades são provadas no teorema abaixo:

Teorema 4.3.2 *Dada uma tarefa esporádica $T_s = (c_s, r_s, d_s, p_s)$, com um tempo de pronto r_s em um sistema periódico τ, uma condição necessária e suficiente para a escalonabilidade de $\tau \cup T_s$ é dada por $c_s \leq TL$, onde TL é fornecida pelo algoritmo abaixo:*

a.Inicialmente, TL é o valor total de tempo livre no intervalo $[r_s, r_s + d_s]$, quando as instâncias de τ com prazo não-superior à $r_s + d_s$ são escalonadas através da política de próximo prazo.

b.Escalone cada instância cujo prazo seja maior que $r_s + d_s$ após $r_s + d_s$ através da política de próximo prazo, avaliando seu atraso $L = D - E$. Para cada valor negativo de atraso, atualize TL de tal forma que o atraso seja zerado, ou seja, $TL = TL + L$ e $L = 0$.

c.Este procedimento termina quando a UCP se torna não-ocupada.

Prova. Suponha que $c_s > TL$. Temos duas possibilidades: a primeira assume que TL não alterou seu valor depois de $r_s + d_s$, o que significa que T_s não é escalonável. A segunda possibilidade é que TL alterou seu valor devido à uma instância cujo atraso era negativo. Isto implica em uma instância com atraso nulo, que se tornaria negativo se $c_s > TL$, e portanto, T_s não seria escalonável. Para provar a suficiência, supomos que $c_s \leq TL$ e que alguma instância perde seu prazo depois de $r_s + d_s$. Por hipótese,

qualquer requisição antes da condição de terminação tem um atraso positivo, o que significa que o prazo é satisfeito. Portanto, a instância citada deve estar depois do próximo intervalo onde a UCP está não-ocupada, o que é uma contradição, uma vez que não existe um intervalo onde a UCP fica desocupada previamente à uma perda de prazo[LL73].

4.3.3 Avaliação dos Testes Suficiente e Exato

Procuraremos aqui traçar uma série de experimentos que possibilitem avaliar a desempenho dos algoritmos apresentadas anteriormente, no que diz respeito à complexidade e eficiência.

O algoritmo para o teste de suficiência tem complexidade linear, e portanto, nos interessa saber sua eficiência, quando comparado com um algoritmo ótimo. Observamos anteriormente que esta eficiência pode ser avaliada em função do fator R, que estabelece o limite da aproximação. Valores de R próximos à unidade indicam que praticamente todas as tarefas esporádicas garantidas pela condição exata também o serão pela condição suficiente. Valores de R muito altos indicam que uma grande parte das tarefas esporádicas que chegarem ao sistema podem não ser atendidas pela condição suficiente, mas seriam se a condição exata pudesse ser aplicada. Objetivamente, a métrica para se avaliar a eficiência da escalonabilidade através de R dependeria da distribuição das tarefas esporádicas no sistema.

A geração de valores representativos do fator R, por outro lado, está relacionada com o conjunto periódico particular, e não com as possíveis tarefas esporádicas. Para cada conjunto periódico dado, podemos calcular $\mathcal{I}$ em tempo linear, e calcular Ω, de tal forma a obtermos R. Portanto, estamos interessados em parâmetros de sistemas periódicos gerados aleatoriamente, de forma que tais parâmetros possam fornecer evidências de como o fator R se comporta.

A geração dos conjuntos periódicos para os experimentos foi feita utilizando um conjunto de períodos harmônicos, e os tempos de computação gerados aleatoriamente, distribuídos com uma média de tal forma a obtermos um determinado valor médio para o fator de utilização. Os valores médios para o fator de utilização escolhidos foram:

$U_1 = 0.5979$
$U_2 = 0.6167$
$U_3 = 0.6249$
$U_4 = 0.7080$
$U_5 = 0.8750$

Apresentamos os valores de R para dez conjuntos de tarefas geradas para cada valor de U acima, representativas dos sistema de controle de tráfego aéreo. Vemos que R é uma função decrescente do prazo e para muitos sistemas R cresce com o fator de utilização do conjunto periódico. Os valores de prazo se referem a possíveis prazos para as tarefas esporádicas. Nas Figuras 4.4 à 4.8, ilustramos o valor de R corresponte aos conjuntos gerados.

Com relação ao teste exato, a eficiência é conhecida de antemão, mas o importante aqui é avaliar a complexidade. Esta complexidade é PSEUDOPTIME, uma vez que o intervalo de reescalonamento se estende desde o momento da chegada da tarefa esporádica até o primeiro momento em que a UCP se torna livre. Em geral, quanto maior a utilização do sistema, mais demorado será o procedimento. Nós ilustramos também nas Figuras de 4.4 à 4.8 algumas simulações, mostrando a complexidade no cálculo de TL para várias tarefas, comparando com o número de tarefas e o tamanho do prazo da tarefa esporádica. A complexidade é dada como o número de decisões de escalonamento até o primeiro intervalo de UCP não-ocupada, dividido pelo valor do prazo. Pode ser visto que a complexidade cresce com o prazo, mas para prazos pequenos a complexidade é baixa, mesmo quando a utilização da UCP é alta. A geração das tarefas periódicas foi análoga ao caso anterior.

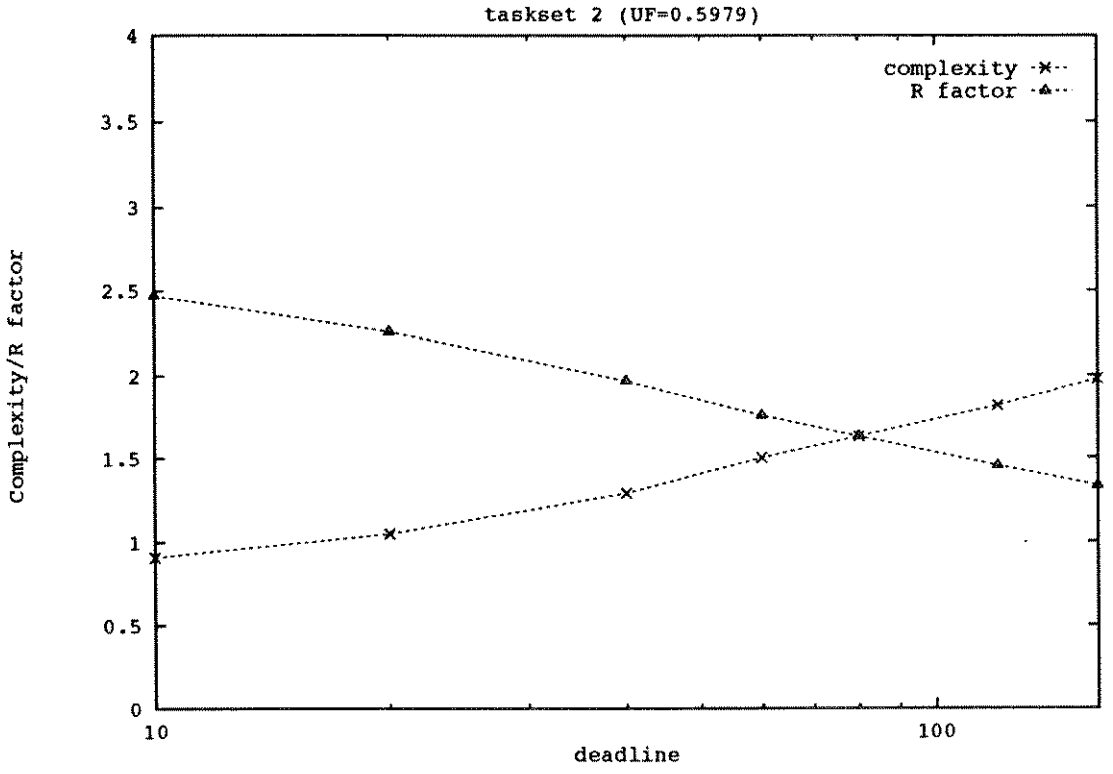

Figura 4.4: Valores de Eficiência e Complexidade para U=0.5979

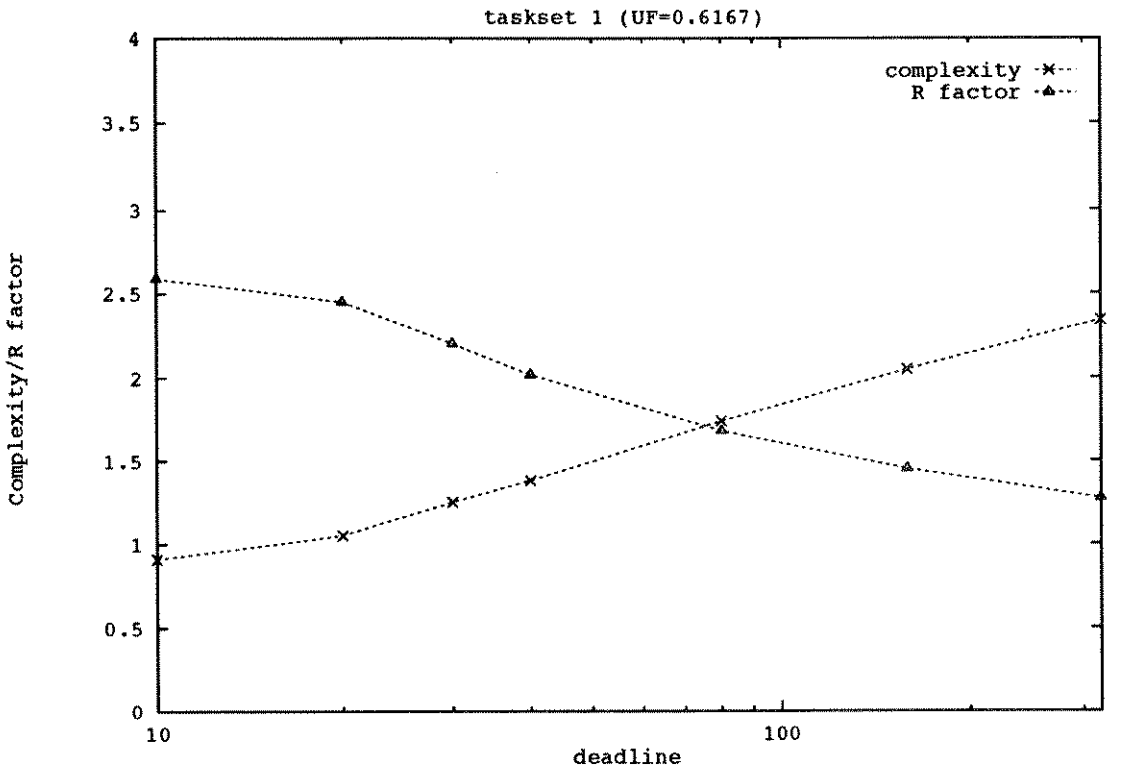

Figura 4.5: Valores de Eficiência e Complexidade para U=0.6167

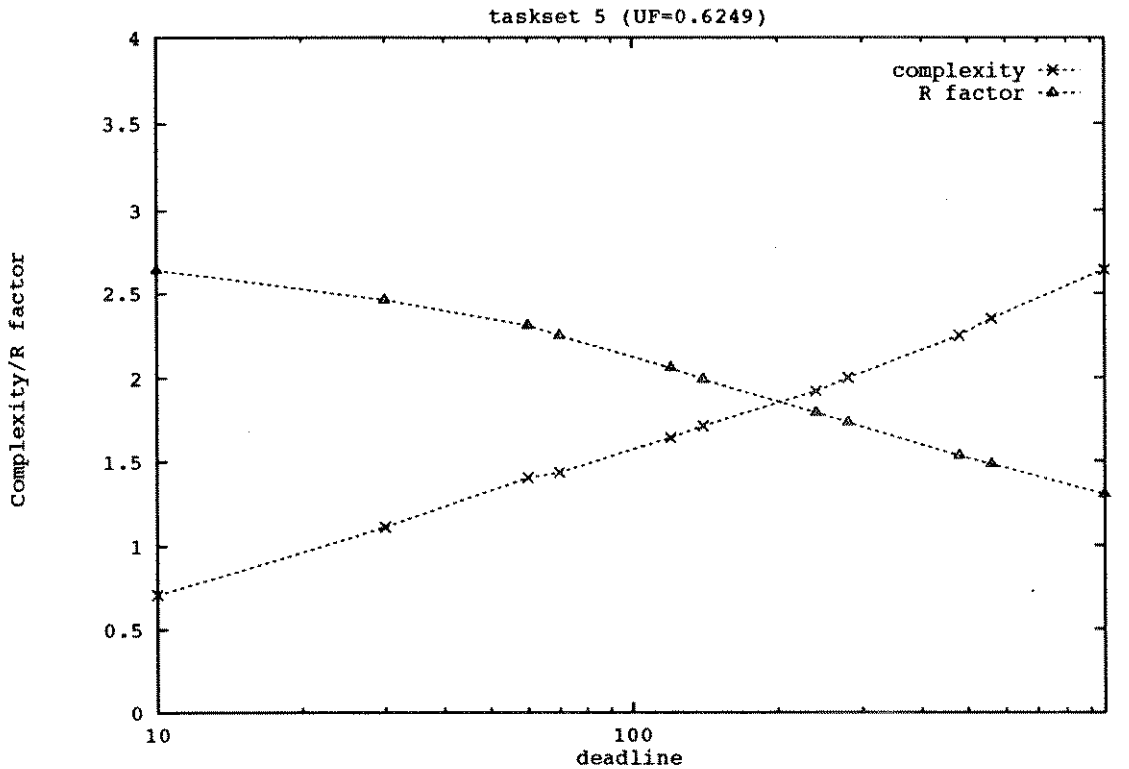

Figura 4.6: Valores de Eficiência e Complexidade para U=0.6249

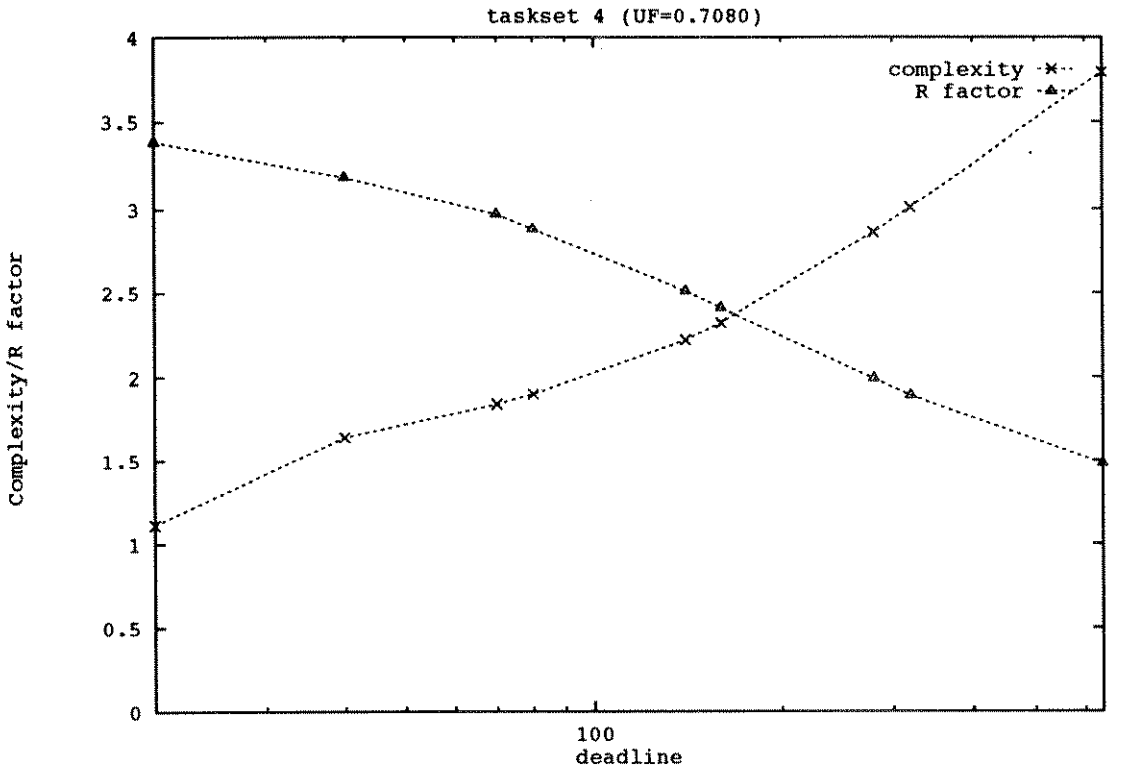

Figura 4.7: Valores de Eficiência e Complexidade para U=0.7080

4.4 Generalização para um Sistema Esporádico de n Tarefas

Vamos denominar o fator $IT[x,y]$ como a quantidade de espaço livre no intervalo $[x,y]$. Note que este fator pode representar tanto o mínimo espaço livre como o espaço livre exato. Esta representação está relacionada ao fato do escalonamento ser decidido on-line ou off-line, respectivamente.

Inicialmente, vamos estabelecer alguns conceitos. Uma tarefa esporádica T_s é dita ativa em um instante t se e somente se $t \in [r_s, r_s + d_s]$, com $c_s(t) > 0$, onde $[r_s, r_s + d_s]$ é o intervalo de tempo do tempo de pronto até o prazo de T_s e $c_s(t)$ é a computação remanescente de T_s em t. Para que T_s cumpra seu prazo, temos que ter $c_s(r_s + d_s) = 0$, onde $c_s(r) = c_s$. Como as tarefas esporádicas são dinâmicas por natureza, o número delas e quais estão ativas em qualquer tempo no sistema não é conhecido de antemão. Cada vez que uma tarefa esporádica chega e está pronta, devemos testar sua escalonabilidade, e se a resposta do teste for positiva, a tarefa se torna ativa. Em caso contrário, a tarefa é descartada.

Supomos que temos um conjunto de tarefas esporádicas ativas ST em um sistema no tempo x. Sendo $ST = \{T_{si}\}$, onde $T_{si} = (c_{si}, r_{si}, d_{si}, p_{si})$, com $i = 1..n$ e suponha que as tarefas estão ordenadas por prazo crescente. Como estas tarefas estão ativas em x por hipótese, temos para qualquer tarefa T_{si} que $r_{si} \leq x < r_{si} + d_{si}$ e $c_{si}(x) > 0$. Provavelmente, algumas tarefas já executaram algum código antes de x. Tomemos duas tarefas quaisquer tais que T_{sk} e T_{sj} têm a propriedade seguinte: $r_{sk} + d_{sk} < r_{sj} + d_{sj}$. Sendo c a quantidade total de computação das outras tarefas em $[x, r_{sk} + d_{sk}]$, temos que $c + c_{sk}(x) + c_{sj}(x) \leq IT[x, r_{sk} + d_{sk}]$, se T_{sj} tem uma prioridade maior que T_{sk},uma vez que o sistema é válido. Isto significa que se T_{sk} tiver uma prioridade maior que T_{sj}, o sistema continua válido. Logo, a escolha ótima para o escalonamento é a política de Próximo Prazo[DM89].

O procedimento para o escalonamento de tarefas esporádicas em um sistema é então baseado na política de próximo prazo. Cada vez que uma nova tarefa esporádica chega ao sistema, no tempo y, por exemplo, testamos se a nova tarefa pode ser escalonada. Sabemos que tarefas que são ativas podem ser afetadas por esta nova tarefa se seus prazos forem maiores que o prazo da nova tarefa. Suponha que temos o conjunto como descrito acima e uma tarefa esporádica $T_{new} = \{c_{snew}, r_{snew}, d_{snew}, p_{snew}\}$. Seja $r_{sk} + d_{sk} \leq r_{snew} + d_{snew} \leq r_{sk+1} + d_{sk+1}$. Então as tarefas $T_{s1}..T_{sk}$ não são afetadas por T_{new}, que pode ser escalonada se:

$$\sum_{i=1}^{k} c_{si}(y) + c_{snew} \leq IT[y, y + d_{snew}]$$

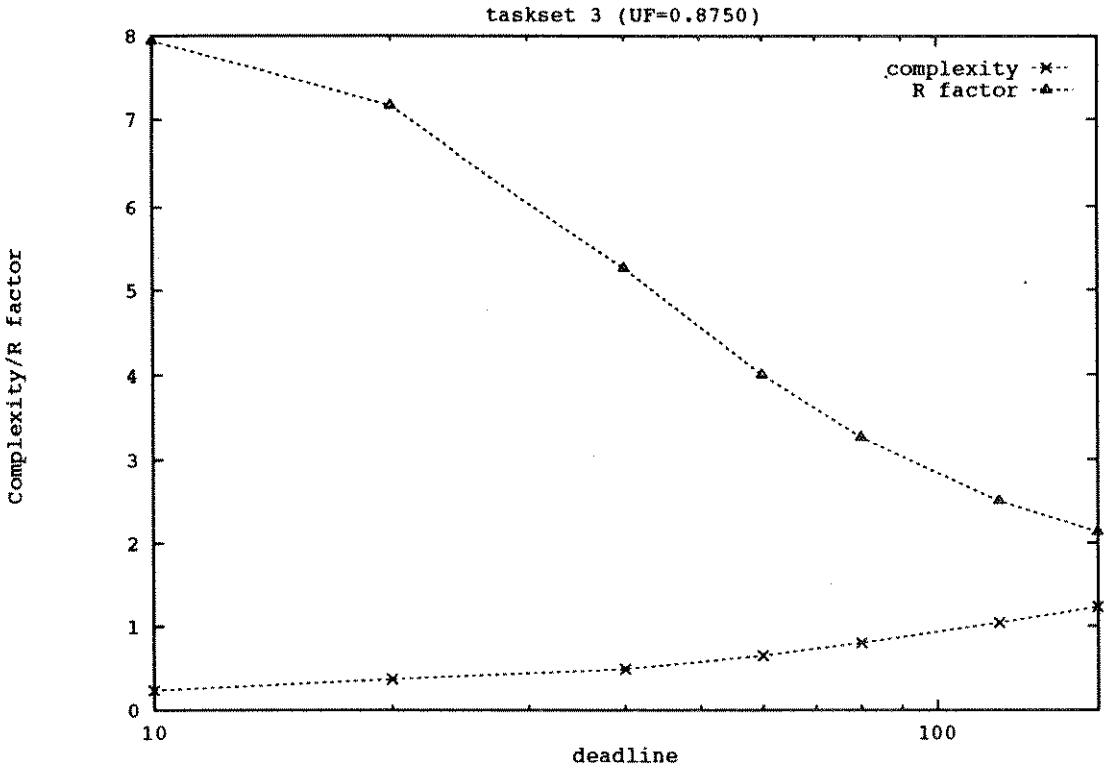

Figura 4.8: Valores de Eficiência e Complexidade para U=0.8750

As Tarefas $T_{sk+1}..T_{sn}$ devem ter suas equações reavaliadas no sentido de se saber se ainda são garantidas.

A regra geral para garantia condicional de uma tarefa esporádica é dada pela política do próximo prazo, desde que a tarefa esporádica não afete a garantia de escalonamento de tarefas periódicas já garantidas. Todas as tarefas periódicas que chegam ao sistema são classificadas de ativas ou inativas. Se a tarefa periódica é escalonável, ela compete pela UCP e é ativa. Caso contrário, ela fica em uma fila de tarefas inativas esperando para se tornar ativa quando um evento que indique que a UCP se tornou mais livre ocorre. Uma tarefa periódica torna-se ativa se todas suas instâncias satisfazem seus prazos sem que isto afete as tarefas periódicas e esporádicas previamente garantidas. Neste caso, o evento que habilita uma tarefa periódica inativa a se tornar ativa pode ocorrer a partida de tarefas periódicas ativas ou o término de uma tarefa esporádica.

Destas regras, podemos ver que uma vez que uma tarefa periódica se torna ativa, ela permanece neste estado até sua partida. Por outro lado, tarefas esporádicas não têm esta garantia incondicional, o que significa que qualquer tarefa esporádica pode ser descartada depois de ser garantida. Observamos que, através da aplicação destas regras em sistemas na condição de sobrecarga, algumas tarefas esporádicas poderão não satisfazer os seus prazos e caso isto ocorra, não é possível se prever qual tarefa continuará garantida e qual não, dentre o conjunto de tarefas esporádicas.

Existem algumas alternativas para as políticas acima[RS89]. Por exemplo, uma vez que uma tarefa esporádica seja considerada garantida, ela continua garantida[BSR88]. Outra possibilidade é permitir que tarefas esporádicas que tenham executado devam ser garantidas incondicionalmente. Neste sentido, para aplicar corretamente a política de próximo prazo, devemos levar em conta os tempos de execução destas tarefas e o teste é aplicado apenas às tarefas esporádicas que não executaram.

Alguns comentários sobre a condição exata são importantes aqui. No sentido de aplicar o procedimento de teste de garantia para o conjunto de tarefas esporádicas, primeiro ordenamos o conjunto por prazo crescente. Então pegamos, no tempo x, por exemplo, as tarefas do conjunto na ordem e fazemos $c_s = \sum_{i=1}^{j} c_{si}(x)$ e $d_s = x + d_{sj}$. Procedemos desta forma, empregando o procedimento do Teorema 4.3.2 para cada tarefa ativa.

4.5 Implementação do Sistema

Nosso sistema consiste de um conjunto de tarefas periódicas que podem ser criadas ou destruídas de acordo com as condições do ambiente, bem como de tarefas esporádicas

que chegam em um tempo imprevisível. As condições para o escalonamento válido destas tarefas foram inicialmente desenvolvidas em [LL73];[CC89]. De acordo com estas referências, a condição de pior caso para o escalonamento de um conjunto de tarefas periódicas ocorre quando todas as tarefas fazem uma requisição simultaneamente, formando um sistema síncrono. Para tal sistema, a escalonabilidade de uma tarefa deve obedecer a algumas restrições impostas pelo tempo de chegada ou criação da tarefa, e a carga do sistema, esta última especificada pelo fator de utilização das tarefas periódicas e pelas equações de escalonabilidade das tarefas esporádicas. Por exemplo, em um sistema cuja presença de tarefas esporádicas ainda não ocorreu, se uma tarefa periódica é criada, ela será aceita se e somente se o fator de utilização do sistema for limitado pela unidade, no caso de monoprocessamento.

A programação do sistema pode ser descrita inicialmente pelo Algoritmo Global, visualizado na Figura 4.9. Este algoritmo é orientado a eventos, consistindo de um laço infinito, cujo corpo consiste de rotinas que gerenciam as mudanças nos conjuntos de tarefas. Qualquer alteração em um destes conjuntos causará um evento, na forma de uma interrupção, que habilitará uma chamada a uma rotina de tratamento de conjunto esporádico ou periódico. Uma vez tratado o evento, o sistema se torna apto a tratar o próximo, retornando ao estado de espera. Esta espera é interrompida através de uma sinalização representada por **INT**.

Quando uma tarefa periódica é criada e requisita serviço, ou quando é destruída, há uma indicação de alteração do conjunto periódico, e o algoritmo Controle_Periodica é chamado. Basicamente, ele verifica se uma tarefa periódica nova pode ser ativada ou se deve esperar na fila de inativas(τ_{pi}). Esta fila é mantida ordenada pelo fator de utilização crescente. Uma tarefa periódica nova pode se tornar ativa se seu escalonamento não afetar as tarefas periódicas ativas(τ_{pa}), bem como as tarefas esporádicas garantidas(τ_{sa}). Desde que o fator de utilização total(incluindo a nova tarefa) não seja maior que a unidade, e a função PP, que verifica se as tarefas esporádicas são garantidas em um sistema periódico corrente, é válida, a nova tarefa se torna ativa. A função PP retorna verdadeiro se o conjunto periódico e o conjunto esporádico foram escalonáveis simultâneamente, e falso em caso contrário. Caso contrário, ela permanece inativa. Nós fornecemos um teste suficiente que é rápido, no sentido de verificar se as tarefas esporádicas não serão afetadas pela nova tarefa periódica através da função PP. O algoritmo ESPExato1 é ativado a cada mudança do conjunto periódico ativo e será explicado adiante. Ver Figuras 4.10,4.11 e 4.12.

Quando uma tarefa periódica é destruída, é necessário testar se alguma(s) tarefa(s) na fila de tarefas inativas pode ser escalonável e, portanto, tornar-se ativa. Estes testes são baseados no fator de utilização, uma vez que devem ser rápidos(PTIME).

No sentido de fornecer garantia que todas as tarefas farão uma requisição si-

```
Algoritmo Global;
 começo
   Exato=verdadeiro
   τ_pa = conjunto de tarefas periodicas ativas
   τ_pi = conjunto de tarefas periodicas inativas
   τ_sa = conjunto de tarefas esporadicas ativas
   laço
     INT int
     caso (int)
       Periodico : chama Controle-Periodico
       Esporadico : chama Controle-Esporadico
     fimcaso
   fimlaço
 fim
```

Figura 4.9: Algoritmo Global

multânea, seja P_a o período da nova tarefa periódica, e x o instante de tempo, depois da chegada da tarefa em r, tal que x/P_a seja inteiro. Então, x será o tempo de pronto desta tarefa periódica. Desta forma, podemos garantir que todas as tarefas periódicas se comportam como se elas formassem um sistema síncrono. Observamos que, desde a chegada até o tempo de pronto, uma tarefa é atrasada por, no máximo, $P_a - 1$.

Para lidarmos com a chegada e término de tarefas esporádicas, sinalizadas por eventos, o algoritmo Controle_Esporádico é invocado. Quando uma tarefa esporádica chega, ela se torna parte do conjunto de tarefas esporádicas(τ_{sa}) e o conjunto total é testado com respeito à sua escalonabilidade. De acordo com a seção anterior, qualquer tarefa esporádica garantida pode ser descartada se não for escalonável de acordo com a configuração corrente. Como será descrito adiante, podemos usar várias abordagens para o cálculo do tempo livre, no sentido de testar a escalonabilidade de τ_{sa}. Quando a tarefa esporádica alcança seu prazo, ela deve ser retirada do sistema e uma nova tarefa periódica que porventura seja parte da fila de tarefas inativas pode ser testada. Ver Figura 4.12.

Vamos explicitar a terminologia utilizada na descrição dos algoritmos. O termo $L(\tau_{pa})$ significa a última tarefa adicionada ao conjunto τ_{pa}. A cabeça da fila τ_{pi} é dada por $H(\tau_{pi})$. Os algoritmos são ativados por comandos **chama**. O comando **ativa** ativa a execução em *background*, levando em conta que apenas um processo é executado por

Algoritmo Controle-Periodico;
começo
τ_{pa}=conjunto de tarefas periodicas ativas
τ_{pi}=conjunto de tarefas periodicas inativas
τ_{sa}=conjunto de tarefas esporadicas ativas
se (chegada)
$T_{ch} = (C_{ch}, P_{ch})$
se $(U_{\tau_{pa}} + C_{ch}/P_{ch} \leq 1)$
se $u_{T_{ch}} < u_{F(\tau_{pi})}$
$\tau_{pa} = \tau_{pa} \cup T_{ch}$
se PP(τ_{pa},τ_{sa})
ativa ESPExato1
senão
$\tau_{pa} = \tau_{pa} - T_{ch}$
$\tau_{pi} = \tau_{pi} \cup T_{ch}$
ordene τ_{pi} por utilização crescente
fimse
senão
$\tau_{pi} = \tau_{pi} \cup T_{ch}$
ordene τ_{pi} por utilização crescente
fimse
fimse
fimse
se (partida)
se $(T_{pt} \in \tau_{pa})$
$\tau_{pt} \subset \tau_{pi}$ tal que $u_{\tau_{pt}} \leq u_{T_{pt}}$
$\tau_{pa} = \tau_{pa} - T_{pt} \cup \tau_{pt}$
senão
$\tau_{pi} = \tau_{pi} - T_{pt}$
fimse
fimse
fim

Figura 4.10: Algoritmo Controle-Periódico

Algoritmo função PP(τ_p,τ_s)
começo
$PP = verdadeiro$
para cada T_{si} em τ_s
se $(\sum_{k=0}^{i} c_{sk} > \mathcal{I}[x, x + d_{si}])$
$PP = falso$
fimse
fimpara
fim

Figura 4.11: Algoritmo PP

vez. Quando o comando **ativa** é chamado, checa-se o processo responsável por este algoritmo. Se tal processo existe, ele é morto. Depois, um novo processo é criado para a execução do algoritmo.

Os testes de escalonabilidade para tarefas esporádicas são baseados ou na condição exata ou na suficiente, juntamente com a política de próximo prazo. Também, como visto anteriormente, a condição suficiente pode comportar-se bem, dependendo do grau de utilização e do valor do prazo da tarefa esporádica. Logo, um parâmetro importante na implementação do sistema é permitir que ambas os testes possam ser feitos, dependendo da conveniência. Caso a condição exata esteja disponível, ela será usada. Senão, a conveniência de se usar a condição suficiente é verificada e, se não aprovada, podemos utilizar a condição exata através do algoritmo ESPExato2, que implementa os resultados do Teorema 4.3.2. Na terminção da tarefa esporádica, testa-se a inclusão de novas tarefas periódica ativa dentre as inativas. Discutiremos estes aspectos de integração e as políticas gerais que orientam as decisões do escalonamento abaixo.

4.5.1 Implementação Integrada

Como colocado anteriormente, estamos lidando com um sistema τ de tarefas periódicas que podem ser criadas ou canceladas a qualquer tempo. Isto impõe uma restrição séria no escalonamento de uma tarefa esporádica emergente. No sentido de tornar a implementação conveniente e integrada, apresentaremos um projeto que permitirá flexibilidade de decisão de escalonamento.

Inicialmente, vamos descrever um método que permite que os algoritmos propostos

Algoritmo Controle-Esporadico;
começo
τ_{pa}=conjunto de tarefas periodicas ativas
τ_{pi}=conjunto de tarefas periodicas inativas
τ_{sa}=conjunto de tarefas esporadicas
se (chegada)
$T_s = (c_s, d_s)$
se (Exato)
$IT = \Omega$
$\tau_{sa} = \tau_{sa} \cup T_s$
$c = 0$
senão
se($U = grande$ e $d_s = pequeno$)
se ESPExato2(τ_{pa},T_s)
$c = c_s$
fimse
senão
$IT = \mathcal{I}$
$\tau_{sa} = \tau_{sa} \cup T_s$
$c = 0$
fimse
fimse
para cada T_{si} em τ_{sa}
se $(\sum_{k=1}^{i} c + c_{sk} > IT[r_{sk}, r_{sk} + d_{sk}])$
$\tau_{sa} = \tau_{sa} - T_{si}$
fimse
fimpara
se ESPExato2(τ_{pa},T_s)
$\tau_{sa} = \tau_{sa} \cup T_s$
fimse
fimse
se (completude)
$\tau_{sa} = \tau_{sa} - T_{comp}$
$\tau_{pa} = \tau_{pa} \cup F(\tau_{pi})$
se PP(τ_{pa},τ_{sa})
ativa ESPExato1
$\tau_{pi} = \tau_{pi} - F(\tau_{pi})$
senão
$\tau_{pa} = \tau_{pa} - F(\tau_{pi})$
fimse
fimse
fim

Figura 4.12: Algoritmo Controle Esporádico

aqui sejam úteis para uso em tempo de execução. Como visto na seção 4.2.3, para cada tarefa esporádica T_s com seus parâmetros correspondentes, precisamos avaliar a quantidade $\Omega_\tau^{EDL}(r_s, r_s + d_s)$. Devido a natureza de T_s, não sabemos de antemão o valor de r_s. Para um τ particular, podemos calcular e armazenar os valores de ATT e SIT, de tal forma que quando T_s chega, recuperamos o valor de Ω indicado e fazemos o teste. Contudo, devemos estar aptos a fazer o teste mesmo quando τ se altera.

Uma possibilidade é considerar o que acontece quando as tarefas em τ alteram seus parâmetros de maneira lenta e gradual. Neste caso, tentaremos calcular o valor de $\Omega_\tau^{EDL}(r_s, r_s + d_s)$ para cada valor de r_s em P. Este cálculo pode ser feito em *background* e cada valor Ω pode ser armazenado para eventual comparação com o valor de c_s. Na prática, os sistemas geralmente consistem de dezenas de tarefas periódicas, mas o número de cadeias harmônicas não é grande[KM91], o que torna o valor de P da ordem de milhares de quanta[Car84]; [SG90b]. Logo, desde que o sistema esteja estacionário, precisamos apenas avaliar os valores de Ω e armazená-los para futuro teste com c_s, quando a tarefa esporádica chega[JaAC93]. O armazenamento dos valores de Ω requer, nestes casos, memória de alguns Kbytes.

Manter-se estacionário por um tempo longo é uma questão imprevisível. Portanto, necessitamos de um teste rápido, para o caso de o conjunto τ ser alterado e os valores de Ω não terem sido calculados quando uma tarefa esporádica chega. Resumidamente, para cada tarefa esporádica T_s, precisamos inicialmente verificar se a condição exata pode ser utilizada, o que pode ser obtido em uma variável que deve ser atualizada sempre que tal condição for verdadeira. Se a condição exata puder ser utilizada, procedemos como no parágrafo anterior. Caso ela não possa ser utilizada, a condição suficiente deve ser empregada. Se esta última falhar em responder afirmativamente à questão da escalonabilidade, então devemos fornecer uma alternativa para escalonar a tarefa esporádica.

O Algoritmo ESPExato1 descreve o procedimento pré-tempo de execução em detalhes, considerando todas as tarefas esporádicas que podem chegar no sistema. Algoritmo ESPExato1 é ativado em *background* e é responsável pelo cálculo dos valores de Ω. A variável ExactFlat é reiniciada para indicar que para o conjunto corrente de tarefas periódicas, os valores de Ω estão disponíveis para uso. Como parte dos aspectos de integração, cada vez que o conjunto de tarefas periódicas ativas muda, este algoritmo deve ser reinicializado, como parte das funções do comando **ativa**. Ver Figura 4.13.

4.5.2 Escolhas Flexíveis

Neste momento, voltamos a nossa atenção para resultados anteriores, que descrevem as curvas que ilustram como R e TL são afetados pelo conjunto de tarefas periódicas

Algoritmo ESPExato1;
começo
τ_{pa}=conjunto de tarefas periodicas ativas
D=conjunto de prazosτ_{sa}
Exato=falso
calcule tabelas CTP and TLS
para cada r_s em P
para cada d_s em D
compute $\Omega^{PPA}_{\tau_{pa}}(r_s, r_s + d_s)$
armazene $\Omega^{PPA}_{\tau_{pa}}(r_s, r_s + d_s)$
fimpara
fimpara
Exato=verdadeiro
fim

Figura 4.13: Algoritmo ESPExato1

através do fator de utilização U e pelo conjunto de tarefas esporádicas através de seus valores de prazo possíveis. O algoritmo de escalonamento esporádico deve decidir se é possível escalonar uma tarefa particular T_s a partir de um instante r_s, por exemplo, em um sistema com tarefas periódicas e esporádicas. Se a garantia de escalonamento de T_s através de I falha, temos que decidir se deveríamos usar a condição exata dada pela LQ ou se podemos introduzir a noção de versões múltiplas.

Em discussão precedente, mencionamos que R poderia significar que se a condição exata fornece uma quantidade C de tempo livre, na média, então a condição suficiente daria a quantidade C/R. Logo, mesmo para o caso onde a condição suficiente não garante o escalonamento de uma tarefa esporádica com sua computação original, poderíamos escalonar uma computação alternativa, com tempo de execução menor que a computação original, desde que representasse a mesma tarefa, em termos de funcionalidade. Este é o ponto central da abordagem de múltiplas versões, onde uma tarefa pode ter várias versões, cada qual com um tempo de execução diferente, de tal forma que uma das versões pudesse sempre ser escalonada.

O problema com a abordagem de múltiplas versões é que, em muitos casos, a mudança da versão não é desejável ou não trará resultados benéficos para o sistema, porque a melhor versão aceita pelo teste é muito diferente da versão original. Neste caso, há razões para se empregar a condição exata dada pela lista de ocupação.

A condição exata da lista de ocupação é conveniente quando temos um pequeno valor de prazo conjungado com um sistema de alto valor de U. Neste caso, provavelmente, a versão aceita pelo teste suficiente será muito pior do que a versão original. Um alto valor de U pode ser considerado em torno de 70% ou pode ser considerado com relação a R, por exemplo, quando $R > 2$. Isto também se aplica com relação a definição do que é um valor baixo para o prazo. Para se decidir qual abordagem usar, recorremos à uma heurística que deve indicar, quando aplicada a um determinado conjunto de tarefas periódicas e esporádicas, qual metodologia fornece o teste mais conveniente, com uma grande probabilidade. Como um exemplo, no esquema do algoritmo Controle_Esporádico, adicionamos um comando de decisão que escolhe qual abordagem será usada. Idealmente, espera-se que esta decisão possa ser apoiada por estatísticas sobre decisões passadas, bem como projeções sobre o estado corrente. Na Figura 4.14, apresentamos o algoritmo ESPExato2, que retorna verdadeiro se uma tarefa esporádica pode ser escalonada com a condição exata em um conjunto de tarefas periódicas ativas. Na verdade, supomos implicitamente a existência de um conjunto de tarefas esporádicas através de ajustes nos tempos de execução.

4.6 Formalização da Consistencia da Política de Escalonamento

Vamos demonstrar que a política global de escalonamento proposta é consistente com a sua especificação, ou seja, que a implementação dos algoritmos está em conformidade com a forma com a qual as tarefas serão garantidas. Inicialmente, vamos descrever formalmente o sistema global, através de uma especificação baseada em máquina de estados extendida[VTB83]. Uma máquina de estados consiste de um conjunto de estados em que o sistema pode se encontrar, um conjunto de eventos, uma série de predicados de habilitação e variáveis de estado.

O nosso sistema consiste, em qualquer momento, de um conjunto de tarefas periódicas ativas, τ_{pa}, um conjunto de tarefas esporádicas ativas, τ_{sa} e um conjunto de tarefas periódicas inativas, τ_{pi}, representando as variáveis de estado. Caso as tarefas esporádicas tenham sido garantidas por um procedimento suficiente, o sistema está no estado S. Se, por outro lado, a garantia foi obtida através de Ω, então o sistema está no estado E. Existe ainda a possibilidade de que, estando no estado S, uma tarefa esporádica que obedeça determinadas condições seja garantida na forma exata, através do algoritmo ESPExato2. Contudo, as outras tarefas esporádicas continuam garantidas, se escalonáveis, pela condição suficiente. Neste caso, temos a presença do sistema no estado M. Logo, o sistema estará em um dos tres estados, E, S ou M.

Algoritmo Função ESPExato2(τ_p,T_s);
começo
ESPExato2=falso
para $t = r_s$ ate $t = r_s + d_s$
escalone instancias com prazo menor
ou igual a $r_s + d_s$ atraves de PP
calcule TL - Tempo Livre em $[r_s, r_s + d_s]$
fimpara
enquanto(não(novotempolivre)
para cada instancia k de τ_p com prazo $> r_s + d_s$
escalone k atraves da politica PP
calcule folga $L(k) = D(k) - E(k)$
se $(L(k) < 0)$
$TL = TL - L(k)$
$L(k) = 0$
fimse
fimpara
fimenquanto
se $(c_s \leq TL)$
ESPExato2=verdadeiro
fimse
fim

Figura 4.14: Algoritmo ESPExato2

Em qualquer estado, uma série de eventos pode ocorrer, habilitando a mudança de estados, dependendo dos predicados de habilitação. Os eventos são:

- CTP. Criação de tarefa periódica
- CTS. Criação de tarefa esporádica
- FTPA. Fim de tarefa periódica ativa
- FTPI. Fim de tarefa periódica inativa
- FTS. Fim de tarefa esporádica
- EF. Fim da função ESPExato1(evento interno)

Os predicados de habilitação são baseados nos valores de retorno da função PP, que tem como entrada um conjunto de tarefas periódicas e esporádicas e retorna verdadeiro se as tarefas periódicas e esporádicas são escalonáveis de acordo com a condição suficiente, e falso, em caso contrário.

As transições possíveis entre os estados ocorrem gatilhadas pelos eventos, desde que o predicado seja verdadeiro, e tem como ação principal a alteração de variáveis de estado.

4.6.1 Especificação Formal do Sistema

A especificação informal do sistema foi descrita em seções anteriores e baseia-se na garantia incondicional de tarefas periódicas ativas, bem como na garantia condicional de tarefas esporádicas ativas, que pode ser anulada apenas por outra tarefa esporádica. O nosso objetivo final é mostrar que a implementação proposta mantém a concepção de alteração de garantia baseada nas regras informais citadas.

A espeficicação formal do sistema segue abaixo, baseada em uma linguagem de especificação formal do tipo máquina de estados extendidos. Seja T_p a tarefa periódica criada ou destruída, e portanto, subproduto de eventos periódicos. Analogamente, seja T_s a tarefa esporádica criada ou destruída.

A sintaxe das construções pode ser entendida como a passagem de um estado à outro(transição), através da ocorrência de um evento, desde que o predicado seja verdadeiro, em cujo caso um conjunto de ações vai ocorrer, dentre elas a alteração de variáveis de estado e a ocorrência de um evento. Ver na Figura 4.15 a especificação do escalonamento proposto.

01. **FROM** E **TO** E **WHEN** ITT(T_p) **provided**verdadeiro
$\{\tau_{pi} = \tau_{pi} - T_p\}$
02. **FROM** E **TO** E **WHEN** STC(T_s) **provided** verdadeiro
$\{\tau_{sa} = \tau_{sa} | \tau_{sa} \cup T_s\}$
03. **FROM** E **TO** E **WHEN** PTC(T_p) **provided** $not(PP(\tau_{pa} \cup T_p, \tau_{sa}))$
$\{\}$
04. **FROM** E **TO** E **WHEN** ATT(T_p) **provided** verdadeiro
$\{\tau_{pa} = \tau_{pa} - T_p; T_p = H(\tau_{pi}); PTC(T_p)\}$
05. **FROM** E **TO** E **WHEN** STT(T_s) **provided** verdadeiro
$\{\tau_{sa} = \tau_{sa} - T_s; T_p = H(\tau_{pi}); PTC(T_p)\}$
06. **FROM** E **TO** S **WHEN** PTC(T_p) **provided** $PP(\tau_{pa} \cup T_p, \tau_{sa})$
$\{\tau_{pa} = \tau_{pa} \cup T_p\}$
07. **FROM** S **TO** S **WHEN** ITT(T_p) **provided** verdadeiro
$\{\tau_{pi} = \tau_{pi} - T_p\}$
08. **FROM** S **TO** S **WHEN** STC(T_s) **provided** verdadeiro
$\{\tau_{sa} = \tau_{sa} | \tau_{sa} \cup T_s\}$
09. **FROM** S **TO** S **WHEN** PTC(T_p) **provided** $not(PP(\tau_{pa} \cup T_p, \tau_{sa}))$
$\{\}$
10. **FROM** S **TO** S **WHEN** ATT(T_p) **provided** verdadeiro
$\{\tau_{pa} = \tau_{pa} - T_p; T_p = H(\tau_{pi}); PTC(T_p)\}$
11. **FROM** S **TO** S **WHEN** STT(T_s) **provided** verdadeiro
$\{\tau_{sa} = \tau_{sa} - T_s; T_p = H(\tau_{pi}); PTC(T_p)\}$
12. **FROM** S **TO** S **WHEN** PTC(T_p) **provided** $PP(\tau_{pa} \cup T_p, \tau_{sa})$
$\{\tau_{pa} = \tau_{pa} \cup T_p\}$
13. **FROM** S **TP** E **WHEN** EFT **provided** verdadeiro
$\{\}$
14. **FROM** S **TO** M **WHEN** STC(T_s) **provided** $ESPExato2(\tau_p a, T_s)$
$\{\tau_{sa} = \tau_{sa} | \tau_{sa} \cup T_s\}$
15. **FROM** M **TO** M **WHEN** ITT(T_p) **provided** verdadeiro
$\{\tau_{pi} = \tau_{pi} - T_p\}$
16. **FROM** M **TO** M **WHEN** STC(T_s) **provided** verdadeiro
$\{\tau_{sa} = \tau_{sa} | \tau_{sa} \cup T_s\}$
17. **FROM** M **TO** M **WHEN** PTC(T_p) **provided** verdadeiro
$\{\tau_{pi} = \tau_{pi} \cup T_p\}$
18. **FROM** M **TO** S **WHEN** ATT(T_p) **provided** $PP(\tau_{pa} - T_p, \tau_{sa})$
$\{\tau_{pa} = \tau_{pa} - T_p\}$
19. **FROM** M **TO** S **WHEN** STT(T_s) **provided** verdadeiro
$\{\tau_{sa} = \tau_{sa} - T_s\}$
20. **FROM** M **TO** E **WHEN** EFT **provided** verdadeiro
$\{\}$

Figura 4.15: Especificação Formal do Escalonamento

Devemos mostrar que a especificação acima mantém o sistema em um estado consistente com suas definições. Provaremos que uma tarefa periódica garantida continuará garantida, independente das mudanças no sistema. Também devemos mostrar que uma tarefa esporádica garantida mantém-se garantida, a menos que outra tarefa esporádica seja garantida em seu lugar.

Os algoritmos estão corretos se eles representam as especificações informais do sistema dadas anteriormente, sumarizadas abaixo:

- a. Qualquer tarefa periódica recém-criada será garantida se seu escalonamento não comprometer o escalonamento das tarefas garantidas previamente. Caso isto ocorra, a tarefa periódica será mantida inativa, até o momento em que seu escalonamento possa satisfazer a condição citada. Uma vez ativa, a tarefa periódica permanecerá neste estado até sua partida.

- b. Qualquer tarefa esporádica será garantida após sua chegada se seu escalonamento não comprometer tarefas periódicas ativas previamente garantidas, e se isto não for possível, a tarefa não será aceita. Tarefas esporádicas são garantidas condicionalmente.

Teorema 4.6.1 *Qualquer tarefa periódica ativa T_{pa} permanecerá ativa até a ocorrência do evento $ATT(T_{pa})$.*

Prova. Suponhamos que T_{pa} é uma tarefa periódica ativa, ou seja, pertence ao conjunto τ_{pa}. De acordo com a especificação do sistema dada, as regras que habilitam a retirada de um elemento de τ_{pa} são as regras 4,10 e 18, que são ativadas se o evento $ATT(T_{pa})$ ocorrer. Logo, a ação $\tau_{pa} = \tau_{pa} - T_{pa}$ é um resultado da ocorrência do evento $ATT(T_{pa})$.

Teorema 4.6.2 *Qualquer tarefa periódica inativa T_{pi} permanecerá inativa ou até a ocorrência do evento $ITT(T_{pi})$, ocasionando a destruição da tarefa, ou como resultado da terminação de tarefas previamente garantidas, periódicas ou esporádicas.*

Prova. De acordo com a especificação do sistema, o conjunto τ_{pi} mantém seus elementos a menos que uma das regras 1,4,5,7,10,11,ou 15 ocorra. As regras 1,7 ou 15 ocorrem somente se o evento $ITT(T_{pi})$ ocorre. Por outro lado, os eventos ATT e STT podem levar a mudanças em τ_{pi}, fornecidas pelas regras 4,5,10,11. No primeiro caso, a tarefa periódica inativa será destruída e no último, ela se tornará ativa.

Teorema 4.6.3 *Qualquer tarefa esporádica T_{sa} permanecerá como tal até a ocorrência do evento $STT(T_{sa})$, representando que o prazo da tarefa foi alcançado, ou o evento $STC(T_{sa}^{new})$, a chegada de uma nova tarefa esporádica que pode comprometer sua execução.*

Prova. O conjunto τ_{sa} pode ser alterado como resultado dos eventos STT e STC, sendo que no último caso, a alteração não ocorre necessariamente. No primeiro caso, a tarefa T_{sa} é retirada de τ_{sa}. No último caso, qualquer tarefa esporádica recém-chegada pode comprometer o prazo das tarefas esporádicas correntes, como ilustrado na ação das regras 2,8,14 e 16.

4.7 Comentários Finais

As simulações conduzidas na primeira parte deste capítulo demonstram a utilidade do teste suficiente no sentido de viabilizar a garantia de escalonabilidade de tarefas esporádicas. Em grande parte das aplicações, as tarefas esporádicas tem tempos de execução bem menores que seus prazos, de tal forma que seu fator de utilização é baixo e a aplicação de teste suficiente é mais adequada e satisfatória.

As políticas de escalonamento de tarefas esporádicas procuram maximizar o uso da UCP, através da política de próximo prazo, mas tem o inconveniente de não privilegiar tarefas que chegaram primeiro ou que seriam mais importantes. Dentre as possibilidades de alteração destas políticas, destacamos uma alternativa que possa distinguir as tarefas por grau de importância e dirigir o escalonamento em favor das tarefas mais importantes. Além disto, ilustramos formalmente os requisitos que as políticas de escalonamento devem satisfazer. Isto certamente contribui para o projeto de sistemas confiáveis, e portanto previsíveis.

Capítulo 5

Tarefas com Nível de Importância

5.1 Introdução

Neste capítulo, abordaremos a questão da escalonabilidade em sistemas dinâmicos, seguindo a metodologia apresentada no capítulo precedente, procurando estabelecer uma previsibilidade na garantia de determinadas tarefas esporádicas. Gostaríamos de garantir em tempo de execução, que todas as tarefas pudessem satisfazer seus prazos. Contudo, em algumas condições, como sobrecarga inesperada no sistema, algumas tarefas podem não satisfazer suas requisições temporais. Sobrecarga transiente de tarefas esporádicas são possíveis devido à natureza aleatória do tempo de chegada e consequentemente do tempo de pronto e tem uma grande probabilidade de ocorrência em sistemas consistindo de um grande número de tarefas[JLT85]. De acordo com a política do próximo prazo, que é uma política ótima de associação de prioridades [DM89],[LL73], não é possível saber de antemão qual tarefa irá perder o seu prazo. Neste sentido, devemos aplicar uma metodologia de distinguir as tarefas por grau de importância e priorizar o escalonamento com sucesso das tarefas mais importantes. O grau de importância deve estar diretamente relacionado com a necessidade do sistema de completar uma determinada tarefa[RS91].

Várias abordagens tem sido propostas considerando a questão das tarefas importantes e funções de benefício, que são representacões matemáticas da importância das várias tarefas para o sistema[CM91]. Basicamente, as metodologias buscam um benefício global máximo que permita a um grande número de tarefas mais importantes o cumprimento de seus prazos. Neste trabalho, propomos dois algoritmos que procuram resolver esta questão seguindo uma filosofia de manter a complexidade do algoritmo em níveis aceitáveis para um STRC e ao mesmo tempo maximizar a função benefício. Mostramos por simulação que cada algoritmo proposto é conveniente dentro de cer-

tos padrões de comportamento do sistema esporádico e comparamos nossos resultados com alguns algoritmos propostos na literatura.

Na seção seguinte, vamos rever algumas abordagens de escalonamento propostas previamente, que levam em conta os aspectos de importância e prazo. Posteriormente, forneceremos dois algoritmos que tentam garantir o maior número de tarefas importantes sem excluir as outras tarefas menos prioritárias ou importantes e comparamos as propostas deste trabalho com as metodologias conhecidas através de simulações sobre um conjunto aleatório de tarefas esporádicas.

5.2 Algoritmos Críticos: Uma Revisão

Até agora trabalhamos implicitamente com a suposição de que as tarefas são escalonadas baseando-se apenas em seus prazos. Contudo, este não é o caso quando tarefas tem uma importância ou criticalidade estática, que não é geralmente relacionada com seus prazos. No sentido de se obter um desempenho adequado para as tarefas mais importantes no processo de escalonamento, temos que procurar por algoritmos que, além de se basear em prazos também levem em conta o grau de importância.

Como mencionado anteriormente, o grau de importância entre tarefas em um sistema é derivado do fato de que não é possível saber quais tarefas irão perder seus prazos em condições de sobrecarga. No sentido de garantir que as tarefas mais importantes satisfarão seus prazos, nós associamos a estas tarefas um grau predefinido de importância e dirigimos o algoritmo de escalonamento no sentido de priorizar a garantia de tais tarefas.

O grau de importância das tarefas vem de duas fontes[JLT85]: os requisitos do ambiente interno afetando a tarefa e a relação da tarefa com seu sistema circundante. A ubckysão do grau de importância de cada tarefa na política de escalonamento global dada pela função benefício é o objetivo fundamental perseguido. Vamos aplicar o conceito de importância apenas para tarefas esporádicas, uma vez que sua semântica pode estar diretamente ligada a uma exceção que o sistema deve executar com certa prioridade. Contudo, as tarefas periódicas continuarão tendo prioridade sobre as tarefas esporádicas.

Nós descreveremos nesta seção três abordagens para se lidar com importância e prazos. A segunda metodologia é baseada no aspecto de cumprimento absoluto dos prazos, o qual nomeamos criticalidade absoluta. A primeira e a terceira metodologia permitem vários tipos de prazos. Ao final do capítulo, estas metodologias serão comparadas com as metodologias propostas aqui.

Antes de tratar o problema em si, nós descreveremos a forma de garantir a esca-

lonabilidade das tarefas. Supomos que as tarefas periódicas têm prioridade sobre as tarefas esporádicas. Isto implica em permitir que as tarefas esporádicas possam perder seus prazos. No sentido de estabelecer-se o teste de escalonabilidade para tarefas esporádicas, nós aplicamos a política de próximo prazo em conjunto com o conceito de tempo livre(disponibilidade de carga) em um dado intervalo. As tarefas periódicas são descritas pelo conjunto $\tau_p = \{T_{p1}, .., T_{pn}\}$, onde $T_{pk} = (c_{pk}, r_{pk}, d_{pk}, p_{pk})$, como visto anteriormente. Para a tarefa esporádica, o conjunto τ_p implica na existência de $I_{\tau_p}[r_s, r_s + d_s]$, a quantidade de tempo livre em um intervalo $[x, x + d_s]$. Uma tarefa esporádica $T_s = (c_s, r_s, d_s, p_s)$ é escalonável se $c_s \leq I[r_s, r_s + d_s]$. A condição pode ou não ser necessária. Para um conjunto de tarefas esporádicas, nós aplicamos o conceito de tempo livre em um intervalo para o conjunto, considerando as computações das tarefas já escalonáveis, no sentido de se determinar quais as tarefas do conjunto são escalonáveis.

5.2.1 Função Densidade

Este modelo baseia-se na existência de um sistema operando em condições de carga variável, de tal forma que em um momento corrente é possível que todas as tarefas ativas no sistema possam satisfazer seus prazos. Entretando, a condição de sobrecarga requer um diferencial em relação aos algoritmos clássicos de prazo. Basicamente, dois algoritmos são propostos [JLT85], observando-se as condições abaixo:

- Dado um conjunto de processos, o escalonamento gerado priorizando-se os processos com maior valor de densidade produzirá um escalonamento com um alto valor para a função benefício.
- Dado um conjunto de processos tais que existe um escalonamento onde todos os processos podem satisfazer seus prazos, o algoritmo de próximo prazo produzirá a melhor função benefício, quando estamos lidando com uma função degrau como função de valor temporal.

As implicações destas observações são importantes no sentido de se estabelecer bons algoritmos. A primeira é que, na ausência de sobrecarga, o algoritmo de menor prazo deveria ser empregado. A segunda é que na presença de sobrecarga, o algoritmo de densidade produz um alto valor para a função benefício, embora não seja ótimo. Neste sentido, são propostos dois algoritmos, descritos abaixo.

- Algoritmo BEValue1 - Algoritmo de densidade puro. Como observações, podemos afirmar que tal algoritmo é razoável para funções degrau. A falha mais

notada se dá em condições de subcarga, principalmente em situações de baixa folga.

- Algoritmo BEValue2 - Algoritmo que combina menor prazo com densidade. As tarefas são ordenadas pelo prazo, e sequencialmente checadas em relação a uma probabilidade de sobrecarga. Se o limiar definido de sobrecarga for atingido, as tarefas até este ponto são candidatas a remoção, por ordem de menor densidade. Após a remoção de um número de tarefas, tal que a probabilidade de sobrecarga decresça, a sequência continua na ordem de menor prazo, até o fim da sequência. Nota-se experimentalmente que este algoritmo tem um comportamento superior ao anterior para o caso em que há subcarga, e degenera-se rapidamente para BEValue1 quando há sobrecarga, combinando ambos os aspectos.

O algoritmo BEValue1 é simples, mas apresenta desvantagens acentuadas em subcarga. O algoritmo BEValue2 procura sanar esta falha, aplicando o algoritmo de prazo quando houver indicações que o sistema está trabalhando em subcarga. Deve-se notar que o ponto importante deste algoritmo é a definição de quando se considera que o sistema está em subcarga. Em termos de complexidade, o algoritmo BEValue2 se baseia em um teste de escalonabilidade baseado no algoritmo de próximo prazo, aplicado sobre o conjunto de tarefas que não foram removidas. A remoção de tarefas é feita através de um procedimento rápido, uma vez que envolve apenas uma ordenação das tarefas por densidade. Portanto, pode ser considerado um algoritmo $O(n^2)$.

5.2.2 Criticalidade Absoluta e Urgência

Até este trabalho[BSR88], pouco tinha sido feito objetivando a integração de aspectos de urgência e criticalidade. A maior parte destes trabalhos foram desenvolvidos supondo níveis de importância diferentes, mas permitindo importância relativa e por outro lado, permitindo a escolha do prazo a partir dos valores de importância, e portanto, ajustando os valores de prazo aos de importância. Este foi o primeiro trabalho a lidar com importância e prazo de forma independente e integrada nas questões de escalonamento.

Basicamente, cada tarefa esporádica chega dinamicamente e independentemente umas das outras. Em tempo de execução, não existe um conhecimento prévio do tempo de chegada de uma tarefa. Além de seu tempo de chegada aleatório, uma tarefa tem um pior caso para o tempo de execução, um nível de importância e um prazo. Então, uma tarefa T_s é dada como: $T_s = (c_s, r_s, d_s, p_s, m_s)$, significando que T_s tem um pior caso de tempo de execução de c_s, um nível de importância m_s e chega ao

sistema no tempo r_s, devendo completar sua execução até $r_s + d_s$. Vamos supor que estes parametros dados para a tarefa são constantes.

No sentido de descrever o algoritmo, nós caracterizamos o sistema esporádico como um sistema onde as tarefas chegam aleatoriamente e são escalonadas através de alguma política de prioridade. Supondo que um conjunto τ_s é escalonado com sucesso no tempo t e uma tarefa T_{new} chega, procedemos como se segue:

1 Determine a escalonabilidade de T_{new} baseado na Política de Prioridade Próximo Prazo. Se $T_{new} \cup \tau_s$ é escalonável, atualize τ_s. Se não, procêda o processo de remoção no passo 2.

2 Tente garantir T_{new} removendo tarefas previamente guarantidas que têm prazo menor que d_{new}.

ALG1 Remova temporariamente as tarefas com o menor nível de importância, prioridade para o maior prazo.

ALG2 Remova temporariamente a primeira tarefa tal que sua importância é menor que m_{new} e seu prazo é o maior prazo menor que d_{new}.

3 Se qualquer tarefa for removida, procure garantir $T_{new} \cup \tau$ novamente e repita o processo até que $T_{new} \cup \tau_s$ seja garantida ou não existam mais tarefas de menor importância a serem removidas.

Vemos que nesta abordagem a sobrecarga é leve, uma vez que temos que proceder a remoção de tarefas logo depois que a primeira tarefa esporádica que torna o sistema não escalonável chega. Sobre o algoritmo ilustrado, devemos esclarecer que a escalonabilidade citada em 1) é baseada na política de Próximo Prazo mesmo na presença de tarefas periódicas. Também, a remoção de tarefas é feita de maneira cumulativa, e finalmente, a remoção é baseada no prazo da tarefa. Notamos que não existem restrições no processo de remoção, considerando-se a execução parcial ou não de tarefas. Vamos nos referir à estes algoritmos como ALG1 e ALG2, respectivamente.

5.2.3 Nível de Importância Geral

Em Sistemas de Tempo-Real Crítico, as tarefas devem terminar sua execução até o seu prazo. No sentido de se estabelecer alguma função matemática ao conceito

de importância, podemos dizer que a tarefa T_{si} pode ser caracterizada como $T_{si} = (c_{si}, r_{si}, d_{si}, p_{si}, m_{si})$ com c_{si} e d_{si} dados como anteriormente e m_{si} é um inteiro que vai de 0 à M e indica o nível de importância da tarefa. Contudo, para Sistemas de Tempo-Real Não-Crítico, as tarefas são estimuladas a cumprir seus prazos, embora isto não seja essencial para o sistema. Neste caso, é mais importante que todas as tarefas executem até o final, mesmo que o prazo seja vencido. Portanto, m_{si} pode ser considerada como uma função contínua no tempo.

Um número de abordagens tem sido propostas para escalonar tarefas de tal forma que o grau de importância seja considerado. Em qualquer caso, o objetivo é escalonar com sucesso um grande número de tarefas que sejam mais importantes. Matematicamente, isto significa maximizar algumas variáveis que levam em conta o nível de importância das tarefas satisfazendo seus requerimentos temporais. Aqui adotaremos o conceito de função benefício B, onde B é definido como a função benefício do sistema[CM91] e incorpora a contribuição de tarefas individuais.

Vamos supor que o sistema τ_s seja composto de n tarefas como $\tau_s = \{T_{s1}, .., T_{sn}\}$ e para cada tarefa T_{sk} nós temos $m_{sk} = TVF_k(t)$. A função $TVF_k(t)$ é definida como o benefício individual da tarefa T_{sk} para o sistema quando ela completa sua execução no tempo t. Ilustra-se a forma de $TVF_k(t)$ para uma tarefa T_{sk} com prazo crítico. O tempo t_i abaixo é o tempo de término para a tarefa T_{si}, de acordo com a ordem de escalonamento $\sigma = (1, 2, 3, .., n)$. Na Figura 5.1, apresentamos uma função degrau para prazo rígido.

O benefício global B é dado como:

$$B = \sum_{i=1}^{n} TVF_i(t_i)$$

$$t_i = \sum_{j=1}^{i} c_{sj}$$

Portanto, $TVF_i(t_i)$ é a contribuição para B quando T_{si} completa sua execução no tempo t_i. Nós escolhemos como objetivo de desempenho maximizar B dado para um conjunto de tarefas τ_s.

O problema de maximizar B sob as restrições acima é reconhecidamente NP-completo[GJ79]. Algumas soluções, contudo, têm sido fornecidas para várias funções especiais [LM69]. No caso de uma função de prazo crítico(degrau), o problema continua intratável. As metodologias que devem ser aplicadas para resolver este problema em Sistemas de Tempo-Real Crítico geralmente buscam soluções sub-ótimas que podem ser encontradas polinomialmente. Este é o objetivo desta abordagem. Nós temos que considerar para cada metodologia o nível de eficiência da solução sub-ótima e o quão rápida ela pode ser encontrada(complexidade).

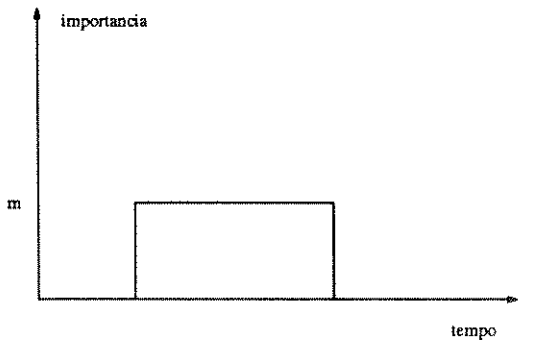

Figura 5.1: Função Degrau para Prazo Rígido

Nós apresentamos o algoritmo de sequenciamento na forma genérica. Seja $R(.)$ uma função de seleção retornando um número único da tarefa, $G(t,i)$ uma função real mapeando de $R \times \{1,..,n\}$ em R, e σ a sequência de escalonamento. O seguinte algoritmo seleciona sequencialmente as n tarefas. A cada passo de seleção, a tarefa escolhida, denotada por s, maximiza uma função calculada em t, que é o tempo de término da última tarefa escalonada. Seja τ_r o conjunto de tarefas ainda não escalonadas no tempo t.

1 $\tau_r = \tau \quad t = 0$

2 `While` $\tau_r = 0$ `Do`
$s = R(\{i | i \in \tau_r \ and \ G(t,i) = max\{G(t,j)/j \in \tau_r\}\})$
$\tau_r = \tau_r - \{s\}$
$\sigma(n - Card(\tau_r)) = s$
$t = t + C_s$
`EndWhile`

3 `Output` σ

A função $G(t,i)$ escolhida para estes experimentos foi a mais adequada para maximizar B quando usamos TVF do tipo degrau. Toma-se $G(t,i)$ como a tarefa que no tempo t tem o maior número de relações de precedência entre todas as tarefas de τ_r. Denotando $P(t,i)$ como o número de tarefas que a tarefa i precede no tempo t, podemos tomar $G(t,i) = P(t,i)$ no sentido de melhorar o valor de B [CM91]. Referimos à este algoritmo como ALG3. Podemos observar que este algoritmo estabelece a ordem de escalonamento baseado em testes de relações de precedência de tarefas, duas a duas. Para n tarefas, estes testes consomem uma complexidade $O(n^2)$. A seguir,

apresentaremos dois algoritmos para escalonar tarefas mais importantes e simulações no sentido de comparar nossa abordagem com outras propostas da literatura.

5.3 Algoritmos de Importância

Vamos propor aqui dois algoritmos para o escalonamento de tarefas observando o princípio de que as tarefas mais importantes devem ser priorizadas no sentido de satisfazer seus prazos. Isto pode ser conseguido através de uma estratégia que busca a maximização da função benefício B, mantendo-se a complexidade em níveis aceitáveis. O primeiro algoritmo é baseado na idéia de que, no sentido de permitir as tarefas mais importantes o cumprimento de seus prazos, elas devem ser garantidas em primeiro lugar, de acordo com a ordem de maior importância. Portanto, separamos o conceito de ordem de garantia com o de ordem de escalonamento. A prioridade de escalonamento continua baseada na política do Próximo Prazo. O segundo algoritmo, embora também baseado na política de Próximo Prazo, visa estabelecer uma busca completa no espaço de soluções, restringindo algumas possibilidades através de uma avaliação de corte. Descrevemos os algoritmos nas próximas subseções.

5.3.1 Algoritmo de Garantia por Ordem de Importância

A primeira metodologia para escalonamento é baseada na regra que, se T_{sk} falha em escalonar com sucesso, então ela irá falhar mesmo se não houver nenhuma tarefa menos importante escalonada na sua frente. Isto impõe uma direção no sentido de garantir as tarefas de acordo com o seu grau de importância. Contudo, a ordem de prioridade seguirá a política de Próximo Prazo. Nós descreveremos o algoritmo abaixo, dado um conjunto de tarefas $\tau_s = \{T_{s1}, .., T_{sn}\}$, onde $T_{si} = (c_{si}, r_{si}, d_{si}, p_{si}, m_{si})$ ordenado através de importância decrescente e o mesmo conjunto $\tau'_s = \{T_s^1, .., T_s^n\}$ ordenado por prazo crescente, com $T_{sk} = (c_s^k, r_s^k, d_s^k, p_s^k, m_s^k)$.

Primeiramente, tentamos garantir a tarefa T_{s1}, a mais importante das tarefas. Sendo $I[x, x + d_{s1}]$ a quantidade de espaço livre no intervalo $[x, x + d_{s1}]$, a tarefa T_{s1} é escalonável se:
$c_{s1} \leq I[x, x + d_{s1}]$
A condição é necessária caso a quantidade $I[x, x+d_{s1}]$ seja exata. Se a equação acima é válida, T_{s1} é garantida e será escalonada com sucesso, independente das outras tarefas. Passamos a investigar a próxima tarefa remanescente por ordem de maior importância, ou seja, a tarefa T_{s2}, que poderá ser garantida apenas se seu escalonamento não afetar as tarefas garantidas previamente, a saber, a tarefa T_{s1}. Isto será válido se:

Algoritmo Importancia;
começo
τ_s=conjunto de tarefas esporadicas
ordene τ_s por importancia crescente
para cada $T_{si} \subset \tau_s$
$\tau_{si} = \{T_{s1}, .., T_{si}\}$
ordene τ_{si} por prazo crescente
$escalonavel = 0$
para cada $T_s^j \subset \tau_{si}$
se $(\sum_{k=1}^{j} C_s^k > I[x, x + d^j])$
$escalonavel = 1$
fimse
fimpara
se (escalonavel=1)
$\tau_s = \tau_s - T_{si}$
fimse
fimpara
fim

Figura 5.2: Algoritmo de Garantia de Importância

$c_s^1 \leq I[x, x + d^{s1}]$
$c_s^1 + c_s^2 \leq I[x, x + d^{s2}]$
É importante notar que $d^{s1} \leq d^{s2}$, o que significa que se $d_{s1} \leq d_{s2}$ então a primeira equação acima já foi avaliada. Caso contrário ($d_{s1} > d_{s2}$), ela deverá ser avaliada.

No sentido de generalizar esta abordagem, tomamos a tarefa T_{si}, supondo que todas as tarefas avaliadas previamente, ou seja, o conjunto $\{T_{s1}, .., T_{si-1}\}$, são garantidas. A tarefa T_{si} será garantida se:
$c_s^1 \leq I[x, x + d^{s1}]$
$c_s^1 + c_s^2 \leq I[x, x + d^{s2}]$
$\vdots$
$c_s^1 + \ldots + c_s^i \leq I[x, x + d^{si}]$

Se alguma destas equações falhar, T_{si} não será garantida e, portanto, deve ser descartada. Passamos, então, para a avaliação da próxima tarefa.

Uma vez garantidas, as tarefas serão escalonadas através do próximo prazo. Nes-

ta abordagem são geradas um número máximo de $n(n-1)/2$ termos que devem ser somados e n comparações, levando a um procedimento $O(n^2)$. Na Figura 5.2, ilustramos o algoritmo de garantia de importância, que também denominaremos ALG4. Basicamente, o conjunto de tarefas esporádicas será ordenado por importância crescente, de tal modo formamos os subconjuntos por ordem de importância e testarmos se estes subconjuntos são escalonáveis. Em cada situação onde a escalonabilidade não é garantida, a tarefa corrente de menor importância é retirada.

5.3.2 Busca Exaustiva

Basicamente, esta abordagem consiste na remoção de tarefas de um conjunto não escalonável, em ordem crescente de importância, até que se atinja um conjunto de tarefas remanescentes escalonáveis. A diferença desta abordagem com as abordagens de remoção anteriores se dá no processo de remoção: remove-se sempre um subconjunto de tarefas do conjunto não escalonável, e a formação deste subconjunto é baseada em uma ordem crescente de importância. Portanto, os subconjuntos são gerados em ordem crescente de importância e o primeiro conjunto remanescente escalonável é o que tem o maior benefício possível. Embora a primeira vista esta abordagem possa parecer complexa, mostra-se por simulação que o algoritmo pode ter um desempenho bem adequado.

Supomos que estamos lidando com um conjunto de n tarefas $\tau_s = \{T_{s1}, .., T_{sn}\}$ ordenadas por importância decrescente, de tal forma que queremos remover subconjuntos em ordem de importância crescente. O conjunto potência de um conjunto de n elementos é dado como uma combinação de todos os subconjuntos possíveis do conjunto original. A cardinalidade do conjunto potência é exponencial em n e é dada abaixo:
$Card(P) = \sum_{k=1}^{n} \binom{n}{k}$
$Card(P) = 2^n$
A geração do conjunto P é feita através do procedimento que parte de um conjunto τ_s escalonável e soma uma tarefa T_{new} à este conjunto, tornando o não escalonável. Este novo conjunto τ'_s é ordenado por ordem decrescente de importância. Computamos uma lista L que será composta de subconjuntos de τ'_s em ordem crescente de importância. A importância de um conjunto de tarefas é a soma das importâncias das tarefas que compõe este conjunto. A cabeça de L é subconjunto corrente de τ'_s, ou seja, o subconjunto de menor importância tal que removendo-o de τ'_s este último continue não escalonável. Caso isto ocorra, geramos dois novos elementos para L e reordenamos L, testando a escalonabilidade do conjunto τ'_s, removendo-se a cabeça de L. Na Figura 5.3, ilustramos o algoritmo de busca exaustiva, ALG5. O conjunto de tarefas esporádicas é ordenado por importância crescente, e uma lista de subconjuntos

é formada por ordem de importância crescente. Os subconjuntos do conjunto total são avaliados, portanto, até que o primeiro seja escalonável, representado pelo subconjunto τ_{ch}.

Este procedimento é claramente complexo em termos temporais. Contudo, veremos que para algumas condições de sobrecarga, este algoritmo pode ser muito competitivo, e ainda ter a vantagem de produzir um valor ótimo para o benefício, como mostrado no teorema abaixo.

Teorema 5.3.1 *Seja τ_s um conjunto de tarefas esporádicas escalonáveis e T_{new} uma tarefa esporádica tal que $\tau_s \cup T_{new}$ não seja escalonável. Sendo L o conjunto obtido de $\tau_s \cup T_{new}$, ordenado de acordo com a maior importância. Se E é o primeiro elemento de L tal que E é escalonável através da política de próximo prazo, então o benefício ótimo para $\tau \cup T_{new}$ é dado por $B_{opt} = B_E$.*

Prova. Seja o conjunto $\tau_s \cup T_{new} = \{T_{s1}, .., T_{sm}\}$. Supomos que este conjunto esteja ordenado segundo a importância decrescente. A geração do conjunto L ocorre a partir do elemento corrente de L, definido como o elemento(subconjunto) com a menor importância. Inicialmente, $L = \{\ \}$. Seja $Curr$ este elemento. Testamos a escalonabilidade de $(\tau_s \cup T_{new}) - Curr$. Caso o teste seja positivo, o algoritmo termina. Senão, dois novos elementos são gerados a partir de $Curr$ e adicionados à L, ao mesmo tempo de o elemento $Curr$ é retirado de L. Em seguida, $Curr$ é atualizado como o elemento de L com menor importância. A geração de dois elementos a partir de $Curr$ é feita tomando-se a tarefa com maior índice em $Curr$ e gerando o elemento $No1$ como sendo $Curr$ retirando-se T_{MAXIND} e adicionando-se a tarefa de $\tau_s \cup T_{new}$ que se segue à T_{MAXIND}. O elemento $No2$ é gerado como $No1$, sem haver a retirada de T_{MAXIND}. Portanto, $No1$ tem sempre importância menor que $No2$. Ambos os elementos são acrescentados à L e um novo nódo $Curr$ é escolhido, continuando com o procedimento até que o primeiro conjunto $(\tau_s \cup T_{new}) - Curr$ seja escalonável. Chamamos de E este elemento. Do processo de formação de L verificamos que como a cada geração temos a formação de 2 novos nódos que ainda não foram examinados, o procedimento deve formar um total de $1+2+4+8+..+2^{n-1}$ nódos, ou seja, $2^n - 1$ nódos diferentes. Logo, todos os nódos do conjunto potência serão potencialmente formados. Além disto, o nódo $Curr$ é o que tem a menor importância dos nódos em L. Logo, $Curr$ é o subconjunto com menor importância que ainda não foi examinado, uma vez que todos os outros terão importância maior que seus geradores em L. Logo, E é o subconjunto de $\tau_s \cup T_{new}$ escalonável com maior importância. Sabemos que os elementos de PS anteriores à E não são escalonáveis, por hipótese. Como PS é ordenado por importância decrescente, o valor máximo do benefício é B_E, ou seja, $B_{opt} \leq B_E$. Como por hipótese, $B_{opt} \geq B_E$, uma vez que opt é óptimo, então $B_{opt} = B_E$.

Algorithm BuscaExaustiva;
começo
τ_s =conjunto de tarefas esporadicas escalonavel
$\tau'_s = \tau_s \cup T_{ch}$ conjunto nao-escalonavel
ordene $\tau'_s = \{T_{s1}, .., T_{sm}\}$ por importancia crescente
$M = m_{ch}$
$L = \{T_{s1}\}$
$escalonavel = 0$
enquanto $((escalonavel = 0)\mathbf{e}(L = not(vazio)))$
ordene L por importancia crescente
$Nocorrente = Primeiro(L)$
$L = Resto(L)$
$SNo1, SNo2$ nos derivados de $Nocorrente$
para cada $SNoi$
se $(m_{SNoi} \geq M)$
$L = L \cup Nocorrente$
fimse
fimpara
$\tau_{ch} = \tau_s - Nocorrente$
se (τ_{ch} e escalonavel)
$escalonavel = 1$
fimse
fimenquanto
se $(escalonavel = 1)$
$B = B_{\tau_{ch}}$
senão
$B = B_{\tau_s}$
fimse
fim

Figura 5.3: Algoritmo de Busca Exaustiva

5.4 Modelagem dos Experimentos

No sentido de avaliarmos o comportamento dos algoritmos propostos ALG4 e ALG5, juntamente com os algoritmos conhecidos ALG1,ALG2,e ALG3, procuramos conduzir uma série de experimentos que reflitam o desempenho em termos de benefício e complexidade de cada algoritmo. Basicamente, os experimentos devem tomar como entrada um conjunto de tarefas periódicas representativo e um possível conjunto de tarefas esporádicas. Cada entrada individual é um par (conjunto de tarefas periódicas, conjunto de tarefas esporádicas) tomado destes conjuntos, de tal modo que o conjunto de tarefas periódicas é escalonável e pelo menos parte do conjunto de tarefas esporádicas tambem o é. Se o conjunto de tarefas periódicas não for escalonável, esta entrada não será avaliada. Cada algoritmo a ser avaliado vai escalonar com sucesso parte do conjunto de tarefas esporádicas de acordo com suas regras de importância e prioridade. Em resumo, cada algoritmo vai tentar escalonar com sucesso as tarefas da entrada seguindo a regra de satisfazer o escalonamento para todas as tarefas periódicas e garantir as tarefas esporádicas de acordo com suas importâncias individuais. A função de benefício é avaliada para cada conjunto de entrada, bem como a complexidade, esta última tomada como a quantidade de iteração envolvida na construção do conjunto de tarefas esporádicas remanescente.

5.4.1 Satisfação do Conjunto Periódico

Estamos supondo que o conjunto de tarefas periódicas é escalonável de antemão. Além disto, requeremos que a escalonabilidade das tarefas esporádicas não devem afetar a do conjunto de tarefas periódicas.

Nos algoritmos analisados, existem duas abordagens no que concerne ao escalonamento de tarefas esporádicas. A primeira se baseia no princípio de que a remoção de tarefas de um conjunto correntemente não-escalonável pode tornar tal conjunto escalonável, utilizando uma ordem de execução baseada no menor prazo. A escalonabilidade do conjunto remanescente é garantida através do fator I, que leva em conta o conjunto de tarefas periódicas. Portanto, esta metodologia assegura que o conjunto de tarefas periódicas não é afetado pela escalonabilidade das tarefas esporádicas.

A segunda abordagem não utiliza a ordem de menor prazo, pois assume que tarefas esporádicas são ordenadas de acordo com uma função, avaliada em momentos específicos, que indica qual deve ser a próxima tarefa na ordem de execução. Se supomos que um conjunto de tarefas periódicas τ_p é escalonável, no tempo x, e consideramos $\tau_s(x)$ o conjunto esporádico sob análise no tempo x, seja a tarefa T_{sa_1} a primeira a executar, depois da decisão do algoritmo. Esta tarefa vai completar sua

execução não no tempo $x + c_{sa_1}(x)$, mas em algum tempo depois, uma vez que as tarefas periódicas estão presentes e ocupam tempo de UCP. De fato, a tarefa T_{sa_1} deverá terminar sua execução em $x + c'_{sa_1}(x)$, onde $c_{sa_1}(x) = I[x, x + c_{sa_1}(x)']$. Este procedimento garante uma prioridade para o conjunto de tarefas periódicas. De acordo, no tempo $x + c_{sa_1}(x)'$, uma nova avaliação vai determinar a próxima tarefa esporádica, na ordem avaliada pelo algoritmo. Sendo T_{sa_2} esta tarefa, seu tempo de término será $c_{sa_2}(x)'$, onde $c_{sa_1}(x) + c_{sa_2}(x) = I[x, x + c_{sa_2}(x)']$. Este procedimento deve se aplicar para todas as tarefas do conjunto esporádico.

5.4.2 Metodologia de Geração de Tarefas

Cada rodada da simulação deve ter como entrada um par de conjuntos de tarefas periódica e esporádica, respectivamente, τ_p e τ_s, no sentido de avaliar os valores de I e realizar os algoritmos. Inicialmente, podemos considerar a geração destes conjuntos aleatoriamente. Por exemplo, sendo $\mathcal{TP}$ o conjunto de conjuntos de tarefas periódicas gerado através de uma distribuição de probabilidade para o fator de utilização e $\mathcal{TS}$ o conjunto de conjuntos de tarefas esporádicas, para cada elemento de $\mathcal{TP}$ nós avaliamos a escalonabilidade de cada elemento de $\mathcal{TS}$. Portanto, para cada conjunto de tarefas periódicas, nós geramos um conjunto de tarefas esporádicas, de acordo com as distribuições de probabilidade escolhidas, e avaliamos o benefício e complexidade do conjunto de tarefas esporádicas remanescente. Contudo, o que realmente é necessário na entrada é um conjunto de tarefas esporádicas gerado levando se em conta que existe um conjunto de tarefas periódicas. Se geramos um conjunto de tarefas esporádicas supondo a existência de uma carga periódica ativa, não existe a necessidade de geração de um conjunto de tarefas periódicas em separado.

5.4.3 Aplicando a Abordagem do Menor Prazo

Para a aplicação da abordagem de menor prazo usada nos algoritmos ALG1,ALG2,ALG4 e ALG5, decidimos a escalonabilidade de cada tarefa esporádica no tempo x através das seguintes equações:

$\sum_{i=1}^{k} c_{si}(x) \leq I[x, x + d_{sk}]$

e $k = 1..n$. Isto é equivalente à n equações do tipo:

$\sum_{i=1}^{k} c'_{si}(x) \leq d_{sk}(x)$

onde $c'_{s1}(x) = c_{s1}(x) - \delta[x, x + d_{s1}]$, e $d_{sk}(x)$ é o prazo remanescente em x para T_{sk}, e $\delta[x, x + d_{s1}] = d_{s1}(x) - I[x, x + d_{s1}]$. Para qualquer outro k, temos que $c'_{sk} = c_{sk} + \delta[x, x + d_{sk}] - \delta[x, x + d_{sk-1}]$ e $\delta[x, x + d_{sk}] = d_{sk} - I[x, x + d_{sk}]$. De acordo com as propriedades acima, $c_{sk} \geq c'_{sk}$, uma vez que $\delta[x, x + d_{sk}]$ é monotonicamente crescente com d_{sk}. Isto significa que a presença de um conjunto de tarefas periódicas é suposto nas simulações, uma vez que o conjunto de tarefas esporádicas gerado pelas distribuições é parametrizado por valores médios de fator de carga.

5.4.4 Aplicando a Ordem Arbitrária

Para o algoritmo ALG4, a ordem de execução das tarefas pode ser diferente da ordem de menor prazo, uma vez que tal ordem depende de relações de precedência entre tarefas. Quando um conjunto de tarefas periódicas está presente, o algoritmo que gera σ deve levar em conta o novo momento de avaliação das relações de precedência. Por exemplo, na ausência de tarefas periódicas, nós atualizamos $t(k)$ como $t(k) = t(k) + c_{sa_k}$, onde $t(1) = x$ e $t(k)$ é dado como $\sum_{j=1}^{k} c_{sa_j}$, onde c_{sa_j} é o tempo de execução da j-ésima tarefa esporádica em σ, ou seja, $T_{sa_j} = \sigma(j)$. Com um conjunto de tarefas periódicas τ_p, nós teremos que adiar o tempo de execução da tarefa esporádica. Neste caso, a tarefa T_{sa_1} teria um novo tempo de execução dado por c'_{sa_1}, onde $c_{sa_1} = I[x, x + c'_{sa_1}]$. Portanto, ao invés da geração de c_{sa_i}, para todas as tarefas esporádicas, nós geramos c'_{sa_i}, como na abordagem de menor prazo.

5.4.5 Geração de Tarefas Esporádicas

Aqui descreveremos o método para a geração da carga esporádica, levando em conta a carga periódica. A carga esporádica será gerada baseando-se em dois princípios diferentes:

- Geração de tarefas usando uma distribuição uniforme até que a sobrecarga seja alcançada[BSR88].
- Geração de tarefas de forma aleatória.

O primeiro método é adequado para os algoritmos ALG1 e ALG2[BSR88]. Foram gerados 200 conjuntos de tarefas através de uma distribuição uniforme de tempos de execução, de folga e de valores de importância. Cada ponto representa uma amostra de 10 conjuntos de tarefas. Os valores médios para execução, folga(prazo menos tempo de computação) e importância foram 5,20 e 10, respectivamente. Cada conjunto foi

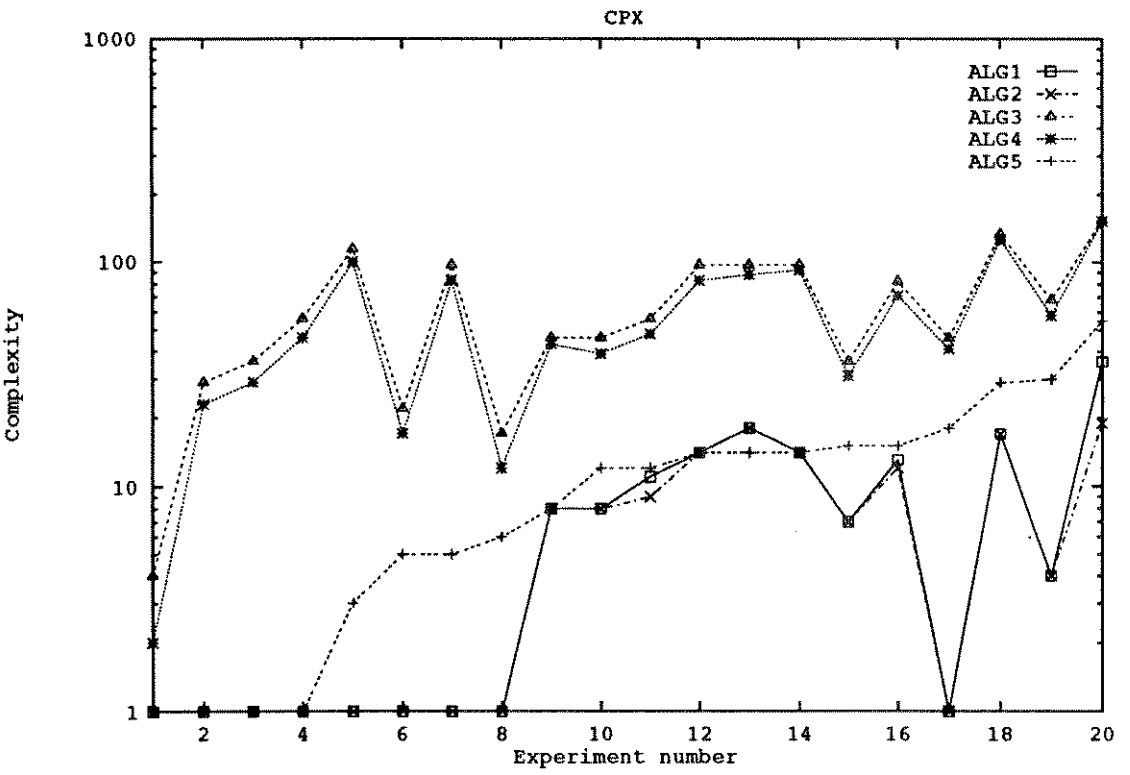

Figura 5.4: Valores de Complexidade de ALG1 até ALG5

gerado de tal forma que através da remoção da última tarefa gerada, o conjunto de tornaria escalonável[BSR88]. Ilustramos na Figura 5.4 os resultados para os algoritmos ALG1 até ALG5 em termos de complexidade. Ilustramos na Figura 5.5 os valores de benefício para os algoritmos ALG1 até ALG4. Constatou-se nesta rodada de simulação que o benefício de ALG4 é quase idêntido ao fornecido por ALG5.

O segundo método gera conjuntos de tarefas esporádicas usando uma distribuição de Poisson para tempos de execução, folga e níveis de importância. Os valores médios para execução e importância foram 5 e 10, respectivamente. Foram geradas 5 distribuções, cada uma correspondendo uma valor médio de folga tomado entre os valores de 10,20,30,40 e 50. O número médio de tarefas por conjunto é 10, o que significa que cada distribuição gerou 1000 conjuntos de tarefas, totalizando 10000 tarefas. Dependendo do conjunto, houve sobrecarga e os algoritmos de importância puderam ser avaliados. Como esta sobrecarga não se baseia na remoção de uma tarefa, apenas os algoritmos ALG3 e ALG4 são ilustrados, pois os algoritmos ALG1 e ALG2 necessitam que o sistema seja inicialmente escalonável e que a chegada da última tarefa torne o

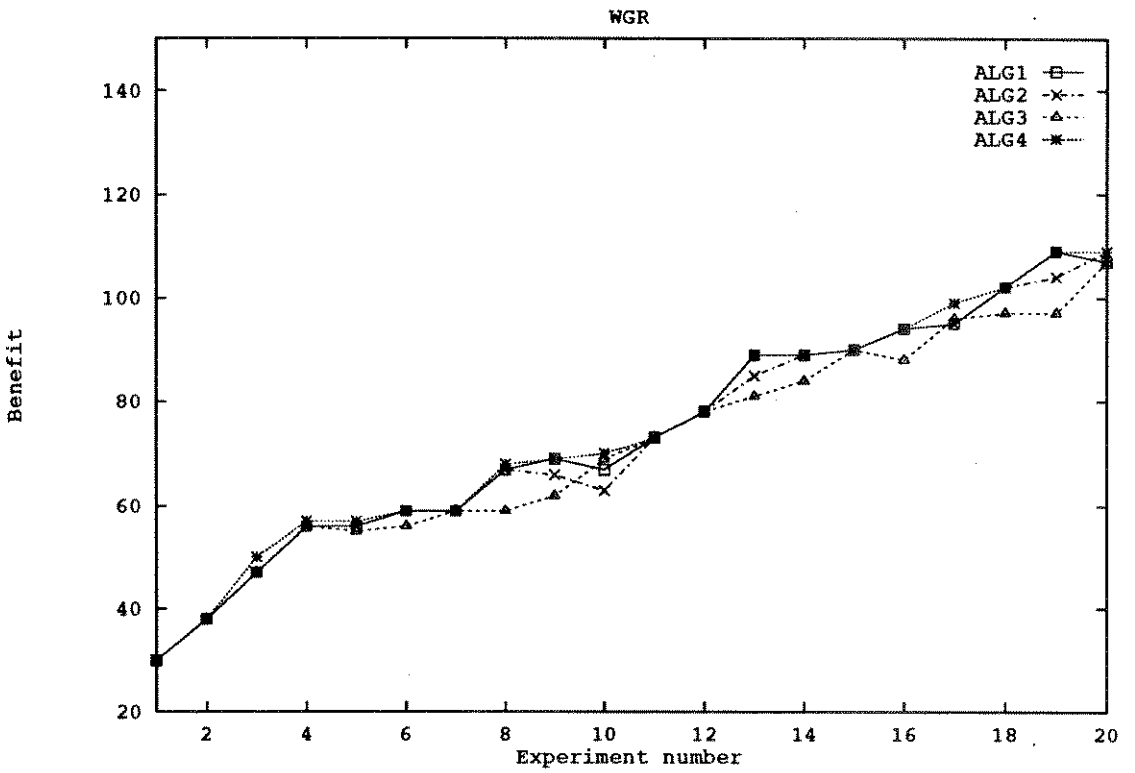

Figura 5.5: Valores de Benefício de ALG1 até ALG4

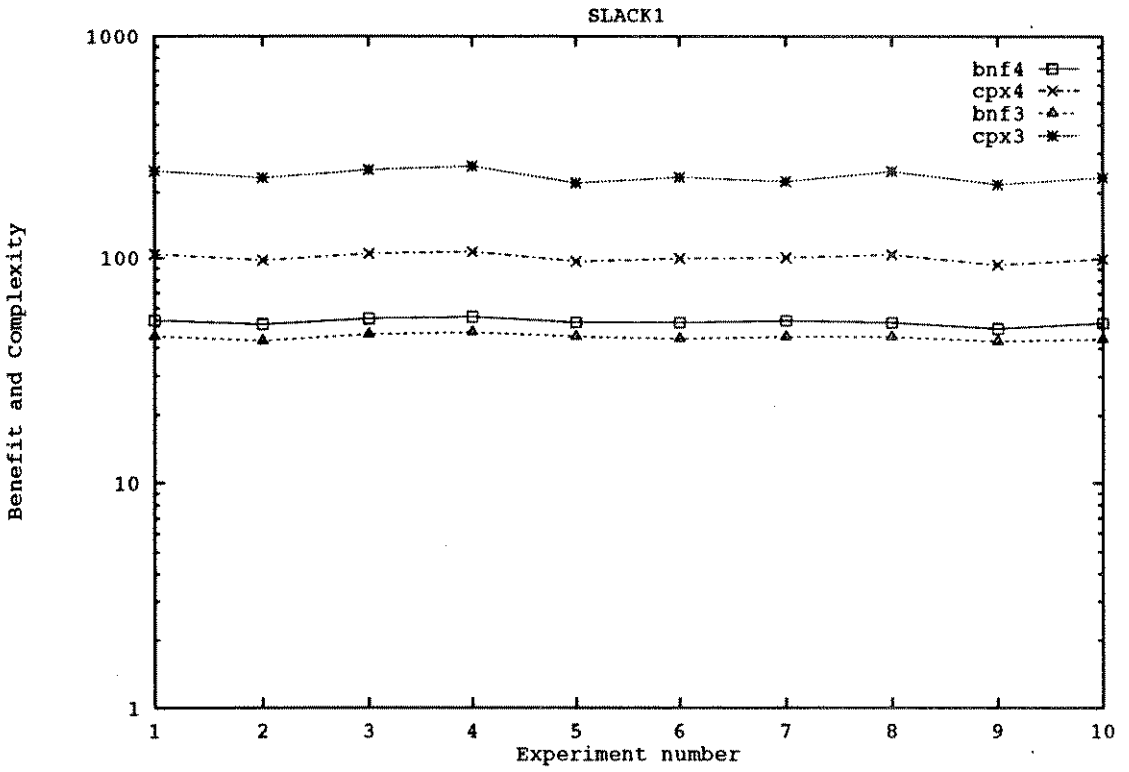

Figura 5.6: Valores de Eficiência e Complexidade para Folga 1

sistema não-escalonável[BSR88]. O benefício dado pelo algoritmo ALG5 é, novamente, quase idêntido ao de ALG4, para todas as distribuições, enquanto sua complexidade é, teoricamente, exponencial. Mostramos o comportamento dos algoritmos ALG3 e ALG4, de acordo com os valores de benefício e complexidade. Cada ponto representa uma total de 10% dos conjuntos em sobrecarga. Por exemplo, para a folga de 40, dos 1000 conjuntos, 191 apresentaram sobrecarga. Nas Figuras 5.6 à 5.10, apresentamos os valores de benefício e complexidade para os valores da folga.

5.5 Resultados e Comentários Finais

Como foi ilustrado nas figuras, o algoritmo ALG4 tem um desempenho melhor, na média, que seus similares. O benefício de ALG5, que é ótimo, é praticamente igual ao de ALG4. Os custos computacionais associados com os cinco algoritmos mostram que ALG4 é melhor ALG3, e que ALG5 tem um desempenho aceitável em baixa

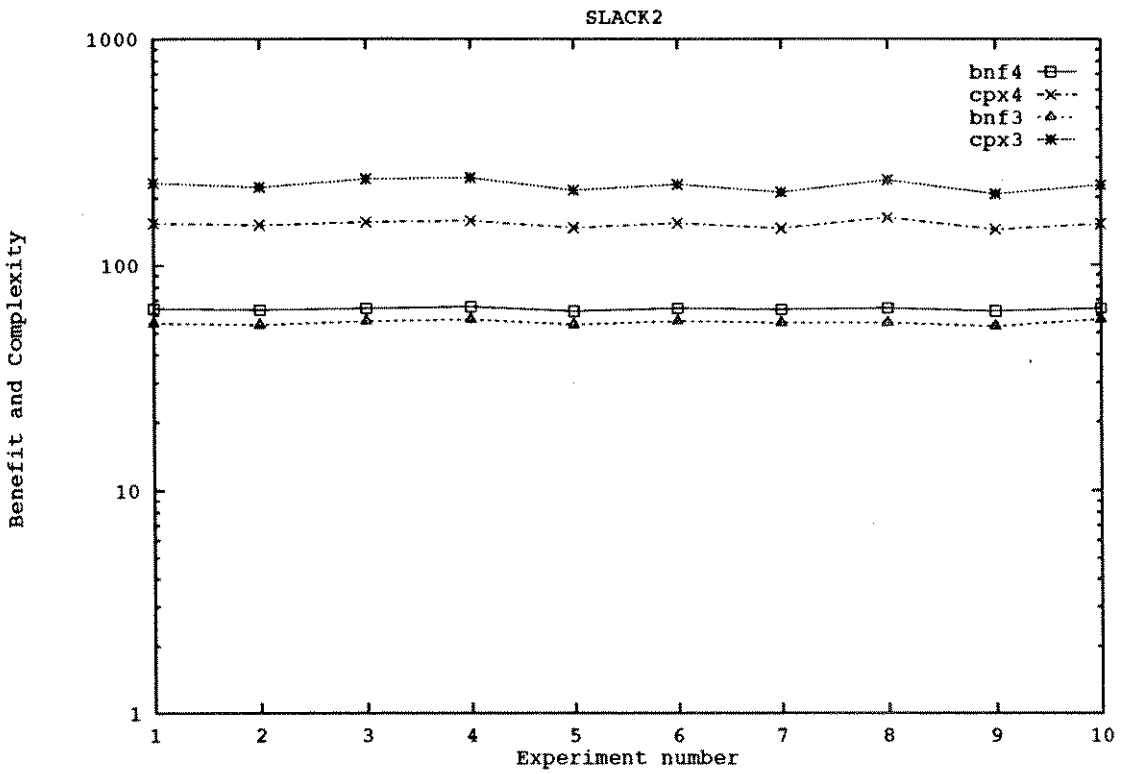

Figura 5.7: Valores de Eficiência e Complexidade para Folga 2

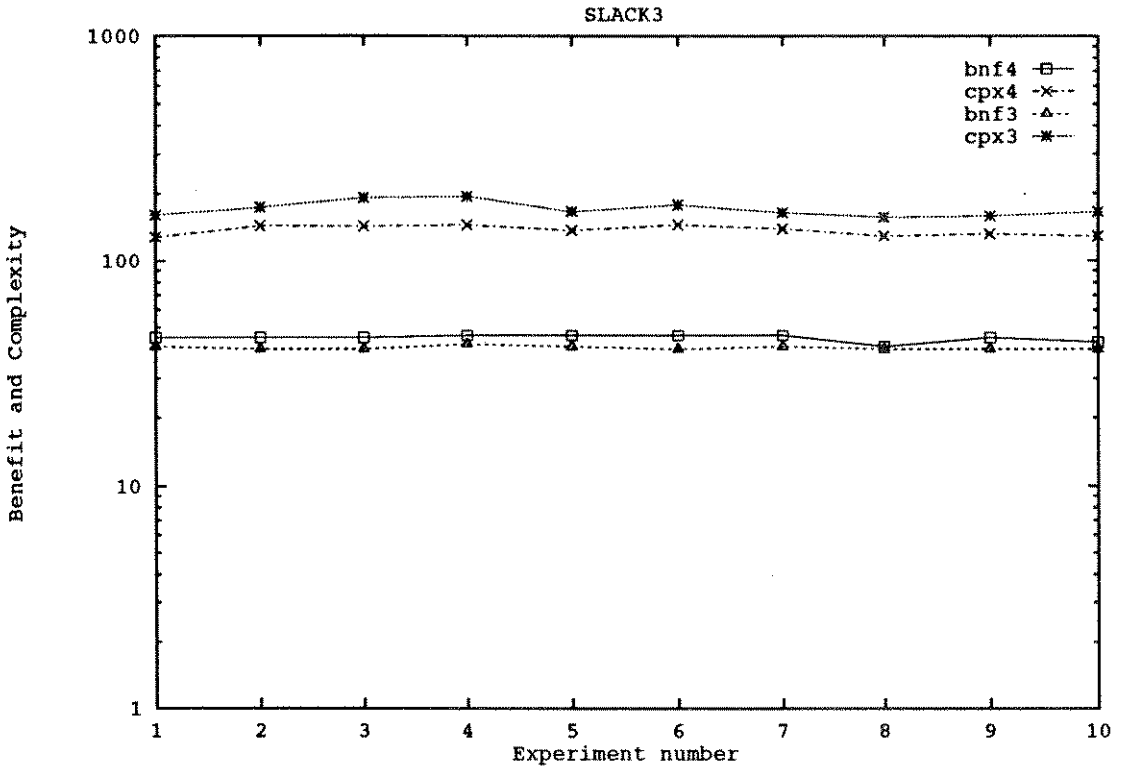

Figura 5.8: Valores de Eficiência e Complexidade para Folga 3

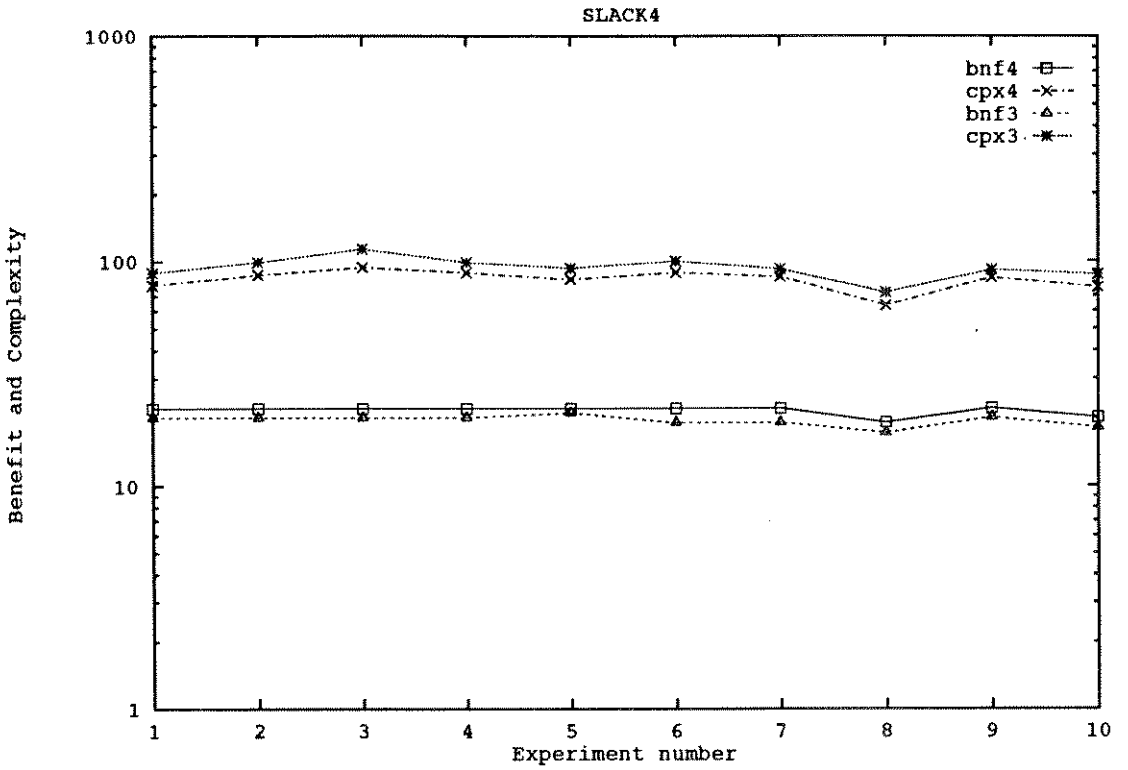

Figura 5.9: Valores de Eficiência e Complexidade para Folga 4

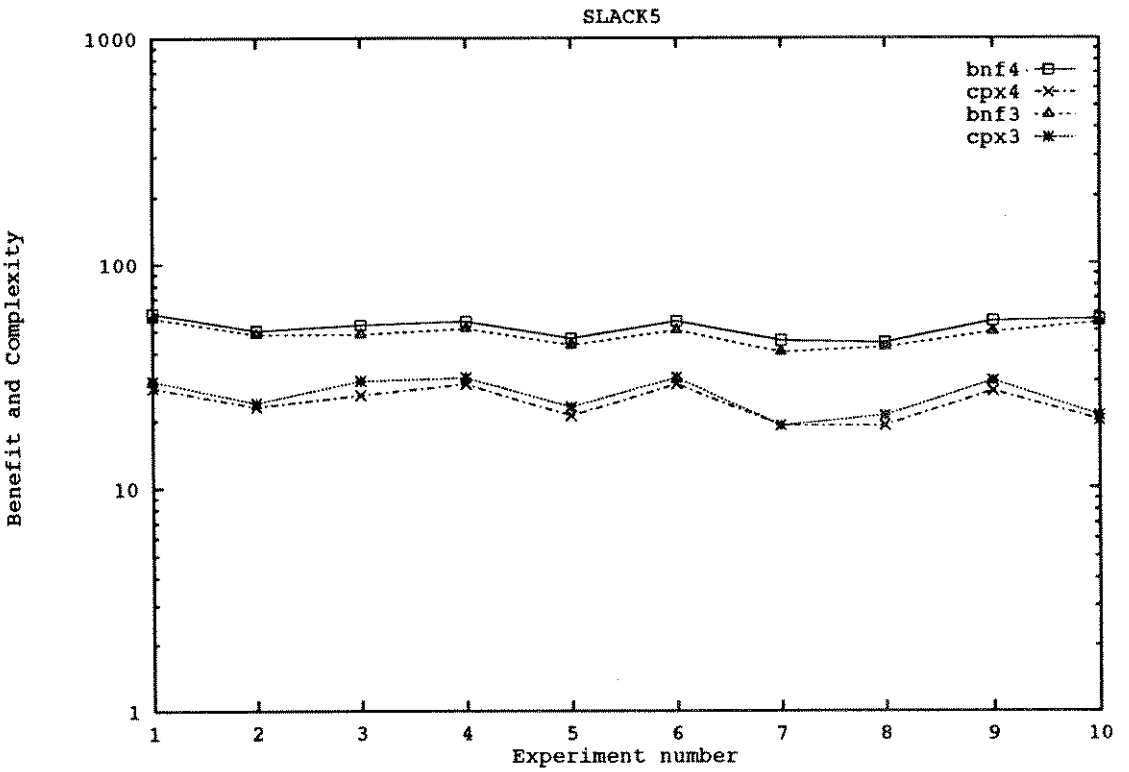

Figura 5.10: Valores de Eficiência e Complexidade para Folga 5

sobrecarga.

A segunda fase das simulações são também ilustradas. Aqui, vemos que ALG4 tem um desempenho superior estável em relação à ALG3, tanto em complexidade como em benefício, independente da carga imposta por valores médios de folga diferentes. Podemos observar que, na média, o benefício dado por ALG4 é próximo do ótimo e, sem dúvida, fornece a melhor opção de uso para prazos rígidos.

Um aspecto fundamental para a plena realização deste modelo é a escolha adequada dos níveis de importância das tarefas esporádicas e do nível de utilização de UCP das tarefas periódicas. O primeiro item deve ser definido no projeto das funções a serem implementadas pelos projetistas, procurando satisfazer os requisitos de segurança. A segunda questão é também função do algoritmo de escalonamento geral, como abordado no capítulo anterior, e pode ser incluída como um teste do nível de utilização dos algoritmo dados neste capítulo.

Capítulo 6

Tarefas com Relação de Precedência

6.1 Introdução

Até o momento, procuramos tratar de tarefas periódicas e esporádicas, dinâmicas e independentes. Tarefas esporádicas são, devido à sua funcionalidade e natureza, independentes das outras, de modo que seu escalonamento é completamente preemptível. Por outro lado, as aplicações com tarefas periódicas independentes são mais restristas. De fato, em vários ambientes de controle e supervisão de processos físicos, as tarefas computacionais estão interligadas de tal modo que suas execuções não podem ocorrer em qualquer ordem, mas devem obedecer à determinadas restrições de precedência e de exclusão mútua[FP88].

As restrições de precedência tem origem na necessidade de compartilhamento de informações por parte das tarefas. Por exemplo, um conjunto de tarefas periódicas atualizam e verificam dados processados a partir de determinadas entradas, de tal modo a gerar suas saídas de forma consistente. Portanto, as tarefas deveriam ser representadas como subtarefas com segmentos independentes e , a princípio, preemptíveis. Outras formas de restrições de precedência surgem como resultado da especialização de funções que estão temporalmente ordenadas[Hua85], de tal modo que o seu mapeamento em programas computacionais impõe uma certa serialização na execução de tais programas[Hu62]. Além disto, o uso compartilhado de recursos impõe uma ordenação no acesso aos mesmos[Jef89].

Por outro lado, o uso de recursos compartilhados em intervalos de tempo não desprezíveis pode gerar regiões em que as tarefas não poderiam ser preemptadas. En-

tretanto, vamos considerar que as seções críticas sejam suficientemente pequenas, de tal modo a permitir a preempção de qualquer segmento de tarefa por qualquer outro, desde que a relação de precedência seja obedecida. Vários sistemas reais apresentam problemas com relações de precedência, para aplicações que exijam atualização de dados[SG90b] e desconsideram a questão da exclusão mútua, ou tratam a exclusão mútua dentro do *overhead* do tempo de execução de cada tarefa[MAC+87].

Na abordagem de tarefas concorrentes emprega-se primitivas da linguagem de especificação das tarefas que permitam a sincronização explícita das mesmas, de tal modo a tornar o processo de escalonamento independente do processo de decisão de quais tarefas devem ser bloqueadas e quais devem estar aptas a executar, em função dos mecanismos de bloqueio da sincronização. Isto permite que a fila de tarefas prontas seja atualizada independentemente do processo de escalonamento. As tarefas bloqueadas em filas de condições são manipuladas pelo sistema através de eventos que são ativados por outras tarefas, quando o motivo que originou o bloqueio cessa.

Existem várias abordagens para se expressar comunicação e sincronização entre tarefas. Uma destas formas se faz através de semáforos[Dij71]. Já foi demonstrado, contudo, que o uso de semáforos para tratar exclusão mútua, mesmo em casos especiais, constitui-se em um problema intratável[Mok83]. Outra forma de indicar relações de precedência se dá através de troca de mensagens, de tal forma que mensagens possam habilitar a execução de tarefas ou parte destas. Monitores[Hoa74] também podem ser empregados, em versões simplificadas em relação à idéia original[Mok83], para lidar com a comunicação.

A base da comunicação empregada neste trabalho tem como princípio a comunicação por troca de mensagens[Hoa78], uma vez que esta comunicação não faz suposições sobre a preemptividade das tarefas, e tem uma grande abrangência. Contudo, em termos de sistema operacional, cada ponto de comunicação/sincronização pode ser substituído por uma indicação de fim de segmento, de tal forma que uma tarefa consiste de um conjunto de segmentos em sequência, e o escalonador tem agora conhecimento desta relação de precedência. Este procedimento minimiza o *overhead* causado pelo bloqueio de liberação de tarefas e permite que as tarefas possam eventualmente ser divididas em segmentos paralelos e sequenciais.

Neste capítulo, procuraremos abordar a questão da escalonabilidade de um conjunto de tarefas periódicas cujas execuções mantem relações de precedência entre si[PS87]. O objetivo é fornecer uma metodologia que permita resolver o teste de escalonabilidade de uma forma conveniente para Sistemas de Tempo Real Crítico, ou seja, com algoritmos rápidos, e ao mesmo tempo indicar a abrangência das soluções adotadas. Neste contexto, como resultado do trabalho, é feito um diagnóstico de como a programação de tarefas, em relação ao tempo de execução e pontos de comunicação, pode ser

alterada, no sentido de fornecer uma solução otimizada. Propõe-se ainda a utilização de metodologias de múltiplas versões para tempos de computação das tarefas como forma de viabilizar a escalonabilidade de um sistema de tarefas periódicas e garantir a previsibilidade.

O estudo consiste em determinar a escalonabilidade de um conjunto de tarefas T, o qual possui um subconjunto de tarefas periódicas sujeito à relações de precedência. A relação de precedência indica regiões de exclusão mútua e sincronização, e é implementada através de uma abordagem que segmenta as tarefas em subtarefas. O problema da escalonabilidade neste caso é, em geral, intratável. Isto torna a análise da escalonabilidade inconveniente para ser tratada em tempo de execução, impedindo a análise de sistemas dinâmicos, onde as tarefas podem ser criadas e destruídas. Desta forma, necessitaríamos de soluções mais rápidas para a garantia da escalonabilidade das tarefas, o que pode acarretar soluções sub-ótimas. Para que tal decisão seja possível, haverá a necessidade de estabelecermos novos parâmetros de escalonabilidade, bem como modificar as características dos processos.

Os trabalhos na área de exclusão mútua e relação de precedência lidam, em geral, com algoritmos intratáveis e só podem ser aplicados pré-tempo de execução. Entre estes, citamos o prazo revisado[Mok83] e o escalonamento por menor atraso[XP90]. Embora sejam garantidamente soluções ótimas, o esforço computacional não pode ser aplicado para o caso de tarefas que se alteram dinamicamente.

O nosso modelo de relação de precedência é adequado para aplicações onde um conjunto de tarefas se comunica, alterando e/ou recuperando valores de um conjunto de variáveis globais que controlam o processo de controle. A leitura e escrita destas variáveis indicam parte do estado global do sistema. Portanto, cada tarefa que faz referência a estas variáveis deve obedecer dinamicamente às restrições de precedência, de tal forma que tarefas comunicantes possam ser dinamicamente criadas e destruídas, dependendo do grau de controle que deve ser imposto ao sistema, e do número de objetos controlados.

6.2 Análise de Escalonabilidade para 2 Tarefas Cooperantes

Vamos analisar as condições para garantir a escalonabilidade de um sistema de tarefas periódicas síncronas, ou seja, $r_i = 0$, no qual duas tarefas estão presentes. Seja $T = \{T_1, T_2\}$ o sistema e T_1, T_2 as tarefas periódicas que se comunicam. Vamos supor que os parâmetros de cada uma das tarefas sejam dados por $T_i = (c_{i1}, .., c_{ia_i}, d_i, p_i)$, onde c_{ij} é o j-ésimo segmento da tarefa T_i(a tarefa T_i possui a_i segmentos), d_i é o prazo da tarefa

T_i e p_i seu período. Supomos que $d_i = p_i$, e vamos indicar as relações de precedência entre segmentos da seguinte forma: se o segmento a deve preceder o segmento b, então $a \rightarrow b$. Assim, em uma mesma instância da tarefa T_i, temos $c_{ik} \rightarrow c_{ik+1}$. A semântica associada ao segmento é a de que devemos executar um segmento, do ínicio ao fim, independentemente da execução de outros segmentos. Contudo, os segmentos têm relações de precedência entre si. Logo, o segmento é a unidade de escalonamento. Vamos proceder à analise de escalonabilidade, quando duas tarefas têm segmentos que mantém uma relação de precedência, indicado por $T_1 \leftrightarrow T_2$, ou seja, as tarefas são cooperantes.

A precedência entre duas tarefas T_1, T_2 é caracterizada pelas relações de precedências entre os segmentos de suas instâncias. As tarefas comunicantes devem ter períodos múltiplos entre si[Mok83]. Seja $p_2 = k_2^1 p_1$, com k_2^1 inteiro, maior que a unidade. Como a tarefa T_1 tem o menor período, ela terá um número de segmentos por instância menor do que o da tarefa T_2. Em geral, a tarefa com menor período terá apenas um segmento, por tratar-se de uma única sequência de computação a serviço de outras tarefas através das relações de precedência. Neste caso, teremos as tarefas $T_2 = (c_{21}^j, .., c_{2k_2^1}^j, p_2)$ e $T_1 = (c_{11}^k, p_1)$, onde j é a instância de T_2 e k é a instância de T_1. Na Figura 6.1, ilustramos o escalonamento onde duas tarefas estão se comunicando. As relações de precedência podem ser descritas como

$c_{2j}^k \rightarrow c_{2j+1}^k$
$c_{2j}^l \rightarrow c_{11}^{k_2^1 l+j}$
$c_{11}^l \rightarrow c_{2x}^y$,
onde $x = l - \lfloor l/k_2^1 \rfloor k_2^1$ e $y = \lceil l/k_2^1 \rceil$.

Sendo c_2 a computação total de uma instância de T_2, se T_1 e T_2 fossem tarefas independentes, uma condição necessária e suficiente para o escalonamento do sistema T é a de que o fator de utilização seja menor que a unidade, sendo que desta forma, o sistema pode ser garantido, se for escalonado através de um algoritmo ótimo. Vamos analisar um algoritmo ótimo, a política de próximo prazo. Através desta política, os prazos dos segmentos são atualizados de acordo com as relações de precedência, de tal forma que a obediência dos prazos implica na observação destas relações. Como temos tarefas comunicantes, a condição acima deixa de ser suficiente para garantir o escalonamento, embora continue sendo necessária. Por exemplo, seja T o conjunto tal que:

$T_1 = (c_{11}, p_1)$ e $T_2 = (c_{21}, c_{22}, c_{23}, p_2)$

onde $c_{11} = 2$, $p_1 = 5$, $c_{21} = c_{22} = 1$, $p_2 = 15$. Dadas as relações de precedência,

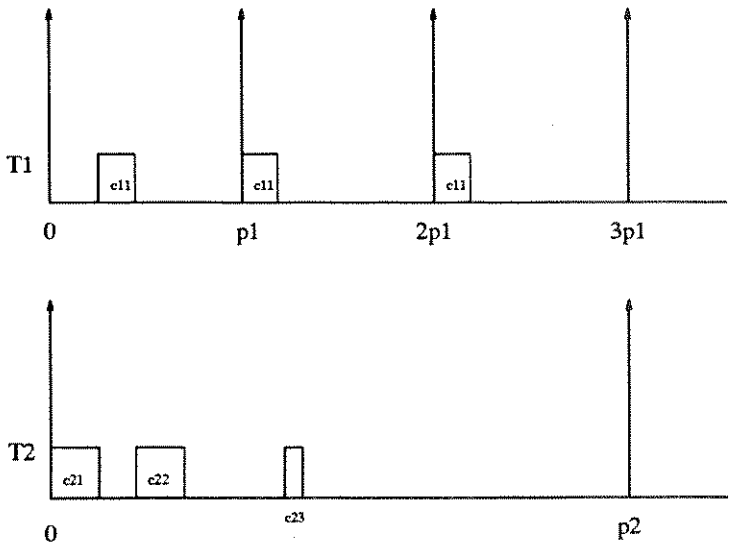

Figura 6.1: Comunicação entre Duas Tarefas

o conjunto $\{T_1, T_2\}$ pode ser escalonado através do algoritmo de próximo prazo, obedecendo a precedência dos segmentos. Podemos observar que se $c_{23} \leq 6$, o conjunto será escalonável. Contudo, se $c_{23} = 7$, o conjunto não será escalonável, apesar de termos $U \leq 1$. Portanto, mesmo que $U \leq 1$, o sistema pode não ser escalonável. Desta forma, precisamos estabelecer uma nova condição para o teste de escalonabilidade, uma vez que o teste de escalonabilidade geral é inconveniente para uso *on-line*(EXPTIME). Na Figura 6.2, ilustramos o escalonamento de $\{T_1, T_2\}$ de acordo com a política de próximo prazo, satisfazendo as restrições de precedência.

A abordagem proposta aqui se baseia na hipótese de que os segmentos dentro de uma mesma tarefa podem ter tempos de computações dependentes de versões. Em tal caso, seria possível alterar-se, quando necessário, as versões, no sentido de fornecer uma condição adequada ao teste de escalonabilidade.

Seja $T = \{T_1, T_2\}$ dados como acima. Seja $c_{2max} = max(c_{2j})$, para $j = 1..k_2^1$. Logo, c_{2max} é o segmento com maior tempo de execução de T_2. O teorema abaixo estabelece uma condição suficiente para o escalonamento de T_1 e T_2, de acordo com as premissas acima.

Teorema 6.2.1 *Seja $T = \{T_1, T_2\}$ um sistema de tarefas periódicas como dado acima. A condição suficiente para o escalonamento de T é dada por:*
$U \leq 1 - u_2(c_{2max}/c_{2avg} - 1)$

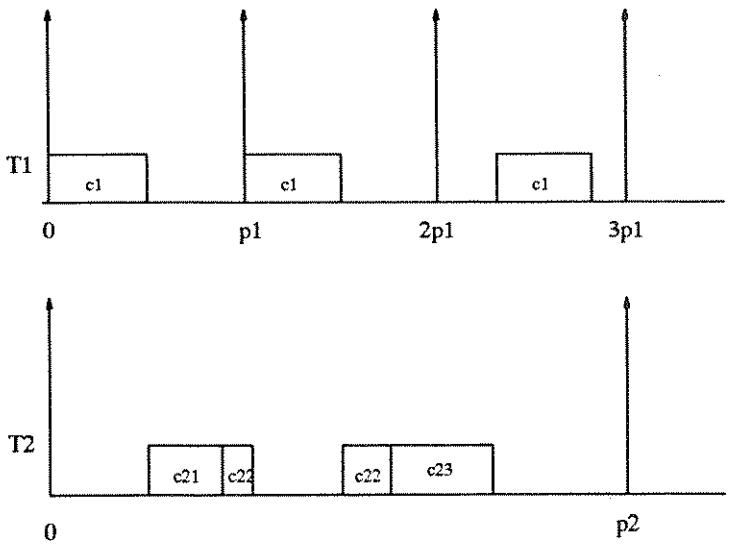

Figura 6.2: Comunicação entre Duas Tarefas através do Algoritmo Ótimo

onde $u_i = c_i/p_i$, c_{2max} *é o maior segmento de* T_2 *e* c_{iavg} *é a média dos tempos de computação de* T_i.

Prova. Seja U o fator de utilização que obedece a condição acima e suponhamos que o sistema T não seja escalonável. Logo, a primeira instância de T_1 ou T_2 deve perder seu prazo. Seja T_2 a tarefa que perde o prazo na sua primeira instância. Isto significa que o segmento k_2^1 de T_2 executa após $p_2 = k_2^1 p_1$. Como estamos lidando com o algoritmo de próximo prazo, isto significa que o segmento $k_2^1 - 1$ de T_2 executou depois de $(k_2^1 - 1)p_1$, uma vez que T_1 cumpre o seu prazo e que U segue a condição acima, o que implica em $c_1 + c_{2max} \leq p_1$. Desta forma, chegaremos à conclusão que o primeiro segmento de T_2 deveria ter terminado sua execução após p_1 o que segundo o algoritmo de próximo prazo é impossível, a menos que $c_1 + c_{21} > p_1$. Portanto, temos uma contradição, e T_2 não perde o prazo. Pelo mesmo motivo, T_1 não perde o prazo na primeira instância, e portanto, em nenhuma outra instância. Logo, a condição dada no teorema é suficiente para garantir a escalonabilidade de T.

Verificamos que a condição suficiente é melhor quanto mais próximo de 1 for a razão c_{imax}/c_{iavg}, principalmente para os valores de u_i maiores. O ideal seria se a condição suficiente fosse dada quando:
$\sum_{j=1}^{2} u_j(c_{jmax}/c_{javg} - 1) \rightarrow 0)$
Isto acontece quando $c_{jmax} = c_{javg}$, ou seja, todos os segmentos são iguais. Neste caso, mostraremos que $U \leq 1$ é condição necessária e suficiente. Observe que para duas

tarefas, $c_{1max}/c_{1avg} = 1$.

Teorema 6.2.2 *Seja $T = \{T_1, T_2\}$ um sistema de tarefas periódicas como dado. A condição necessária e suficiente para o escalonamento de T é dada por $U \leq 1$, caso $c_{jmax} = c_{javg}$, para $j = 1, 2$.*

Prova. Seja $T' = \{T'_1, T'_2\}$ o sistema de tarefas periódicas tal que $T'_2 = (c_{2avg}, p1)$ e $T'_1 = T_1$. Vamos mostrar que T e T' são equivalentes do ponto de vista de escalonabilidade. Inicialmente, suponhamos que T seja escalonável. Logo, todas as instâncias cumprem seus prazos dentro do mínimo múltiplo comum, que é p_2. Portanto, independentemente das relações de precedência, teremos que c_2 e $k_2^1 c_1$ devem executar em p_2, ou seja, $c_2 + k_2^1 c_1 \leq p_2$. Isto implica em $c_{2avg} + c_1 \leq p_1$, ou ainda $U' \leq 1$. Como as tarefas em T' são independentes, o conjunto T' é escalonável. Vamos agora supor que T' é escalonável, ou seja, $U' \leq 1$. Portanto, $c_{2avg} + c_1 \leq p_1$. Logo, se usarmos o próximo prazo para escalonar as tarefas em T, podemos escalonar c_{21} e c_1 antes de p_1, c_{21} ,c_1 entre $[p_1, 2p_1]$ e assim por diante. Logo, T também é escalonável, uma vez que cumpre seus requisitos temporais.

De maneira geral, podemos observar que a condição necessária para o escalonamento impõe que o fator de utilização não seja maior que a unidade, e a condição suficiente é válida desde que $U \leq U_{min}$, onde U_{min} foi dado no Teorema 6.2.1. Entre 1 e U_{min} não se pode afirmar nada, ou seja, a região em que o fator de utilização for tal que $1 \geq U \geq U_{min}$ não é determinada. Logo, procuraremos por valores de U_{min} mais próximos da unidade. Na Figura 6.3, apresentamos o escalonamento relativo às tarefas da Figura 6.2, utilizando a reserva de tempo. Verificamos que a escalonabilidade não foi cumprida.

Vemos que é prioritário a minimização da quantidade $\sum_{j=1}^{2} u_j(c_{jmax}/c_{javg} - 1)$, obtido através de um ajuste de tal forma que c_{jmax} se aproxime de c_{javg}. Veremos posteriormente de que forma isto pode ser conseguido.

6.3 Análise de Escalonabilidade para k Tarefas Cooperantes

Nesta seção, vamos avaliar o teste de escalonabilidade para k tarefas comunicantes. Seja T um conjunto de tarefas periódicas tais que existem relações de precedência entre todas as tarefas no sistema. Basicamente, podemos ter duass diferentes formas de comunicação:

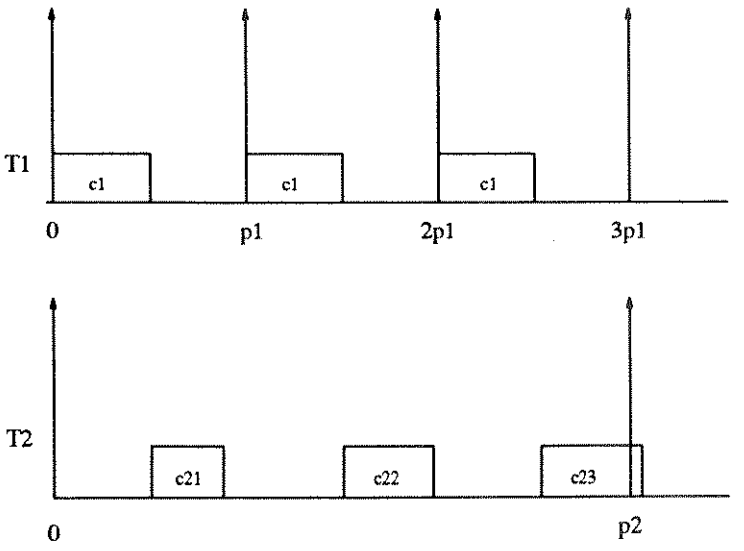

Figura 6.3: Escalonamento com Reserva de Tempo

- a. Comunicação direta de todas as tarefas com uma tarefa servidora simples, de maior frequência, ilustrada na Figura 6.4.

- b. Comunicação de tarefas com a tarefa de maior frequência através de tarefas com frequências intermediárias, ilustrada na Figura 6.5.

Seja T o conjunto de tarefas tal que $T = \{T_1, .., T_l\}$, sendo que $T_i = (c_{i1}, .., c_{ik_i^l}, p_i)$, ou seja, T_i possui k_i^l blocos em cada instância, aonde $k_i^l = p_i/p_l$. A tarefa T_i se comunica com T_l através das relações de precedência, de tal forma que a cada instância de T_i corresponda um segmento de T_l. As relações de precedência podem ser especificadas como $T_i \leftrightarrow T_j$, quando T_i mantém relação de precedência com T_j, e $T_i \rightarrow T_j$ quando nesta relação, $p_j < p_i$. Seja $u_j = c_j/p_j$ o fator de utilização da tarefa T_j e $\delta_j = c_{imax}/c_{iavg}$ o fator de distribuição de execução da tarefa T_j, como visto. Um valor de δ_j próximo da unidade indica que os segmentos da tarefa T_j apresentam uma distribuição uniforme.

6.3.1 Comunicação com Servidor Simples

Neste caso, vamos supor que para todo $i \neq 1$ teremos a relação $T_1 \rightarrow T_i$. O seguinte teorema fornece a condição suficiente para o escalonamento das tarefas em T.

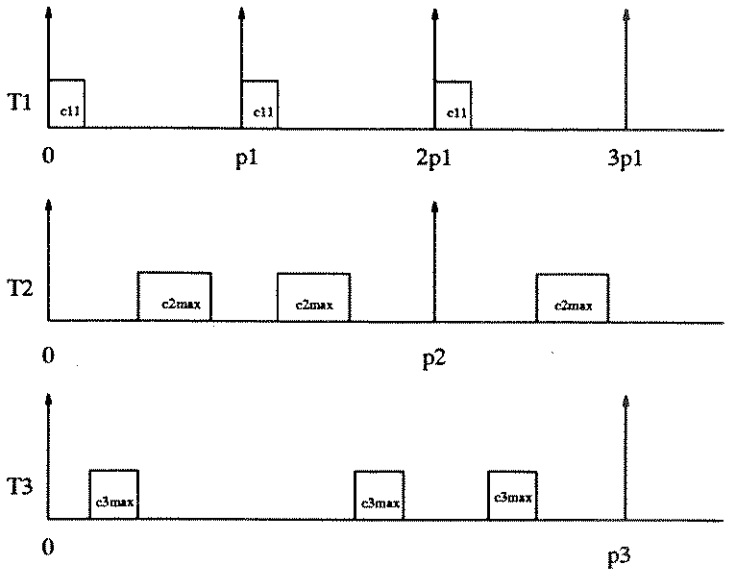

Figura 6.4: Comunicação Direta com Servidora

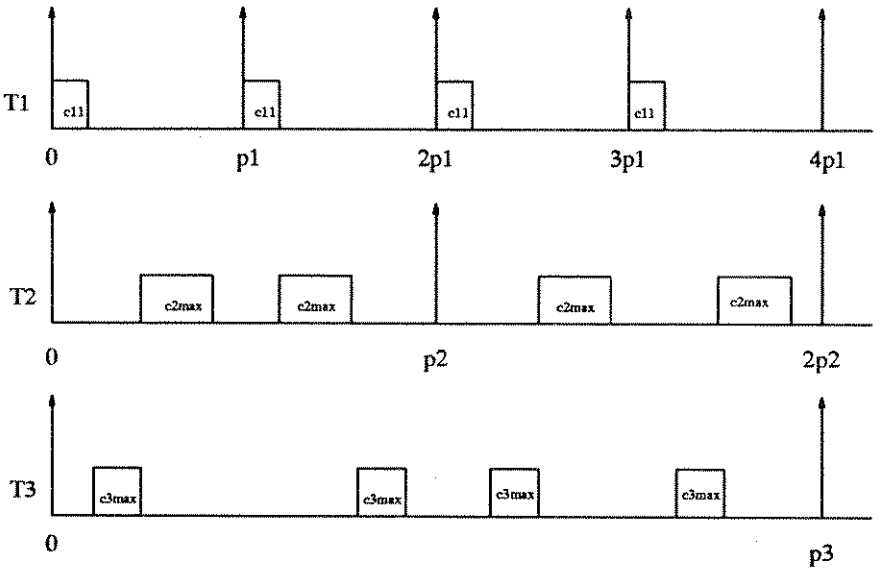

Figura 6.5: Comunicação com Tarefas Intermediárias

Teorema 6.3.1 *Seja T um sistema de l tarefas periódicas tais que todas se comunicam com a tarefa de maior frequência, denominada servidora. Uma condição suficiente para o escalonamento de T é dada por:*
$\sum_{j=1}^{l} \delta_j u_j \leq 1$

Prova. Segue-se como uma consequência do teorema 6.2.1. A condição dada acima pode ser transformada até implicar na seguinte equação:
$\sum_{j=1}^{l} c_{jmax} \leq p_1$
Suponha que alguma tarefa T_i perca o prazo. Isto significa que em p_i não podemos executar todos os segmentos que deveriam executar, que são todos os segmentos até p_i. Uma vez que não teremos tempo ocioso no intervalo $[0, p_i]$, estas requisições ultrapassam o valor de p_i. Estas requisições são dadas por $\sum_{j=1}^{l} \sum_{k=1}^{i} c_{ji}$. Este valor deve ser superior à p_i, o que implicará em uma contradição com a equação original. Portanto, a tarefa T_i não perderá seu prazo, e como T_i é qualquer, nenhuma tarefa perderá seu prazo.

6.3.2 Comunicação Geral entre Tarefas

Neste grafo, todas as tarefas mantem relação de precedência direta ou indiretamente com uma tarefa de maior frequência, ou seja, se $T_i \rightarrow T_j$, para $j \neq i$, então ou $j = 1$ ou $T_j \rightarrow T_k$, e assim por diante, até que alguma tarefa intermediária se comunique diretamente com a tarefa de maior frequência. Portanto, o grafo de comunicação geral é uma árvore do tipo *intree*.

Teorema 6.3.2 *Seja T um sistema de l tarefas periódicas tais que todas se comunicam com a tarefa de maior frequência através de tarefas intermediárias. Uma condição suficiente para o escalonamento de T é dada por:*
$\sum_{j=1}^{l} \delta_j u_j \leq 1$

Prova. Suponha que $\sum_{j=1}^{l} \delta_j u_j \leq 1$. Portanto, de acordo com o teorema anterior, se todas as tarefas se comunicassem diretamente com o servidor, sua escalonabilidade seria garantida, bem como as restrições de precedência. Portanto, as relações de precedência continuam valendo, e a escalonabilidade pode ser garantida.

6.3.3 Análise de Escalonabilidade para n Tarefas Gerais

Seja um conjunto de tarefas periódicas T, com n tarefas, das quais m são tarefas independentes e k são tarefas com relação de precedência. As tarefas que apresentam

relação de precedência podem apresentar uma comunicação geral. O teorema abaixo apresenta a condição para o escalonamento do sistema T.

Teorema 6.3.3 *Seja T um sistema com n tarefas periódicas, tais que m são tarefas independentes e k apresentam relações de precedência entre si. Suponhamos que T seja ordenado de tal forma que as m primeiras tarefas sejam independentes. Uma condição suficiente para o escalonamento de T é dada por:*
$\sum_{j=1}^{n} \delta_j u_j \leq 1$
onde $\delta_j = 1$ para $j = 1..m$ e tem seu valor convencional, para as tarefas comunicantes.

Prova. A equação acima implica em:
$\sum_{j=1}^{k} \delta_j' u_j' + \sum_{j=1}^{m} \delta_j^* u_j^* \leq 1$
onde $\delta_j^* = 1$, pois as tarefas são independentes e $\delta_j' \geq 1$. Portanto, $\sum_{j=1}^{n} u_j \leq 1$, ou seja, se todas as tarefas fossem independentes, elas seriam escalonáveis.

Um sistema com tarefas independentes e cooperantes pode ser analisado em tempo de execução, inclusive com respeito à escalonabilidade das tarefas esporádicas. Na Figura 6.6 apresentamos um esquema do algoritmo de controle periódico para tarefas cooperantes e independentes.

6.4 Complexidade do Modelo de Bloqueio Máximo

O modelo apresentado neste capítulo procura obter um procedimento que possa ser executado *on-line*, quando necessário, de modo que uma de suas características é a rapidez na resolução da escalonabilidade. Mesmo quando o resultado deve ser calculado pré-tempo de execução, o modelo apresenta a vantagem de ser tratável. Vamos avaliar a complexidade apresentada para a solução de um teste de escalonabilidade de um sistema T de n tarefas, algumas apresentando relações de comunicação.

Se considerarmos um conjunto de tarefas T com relações de precedência dadas em uma das formas anteriores, teríamos como entrada, para cada tarefa, o tempo de computação de cada um de seus segmentos e o seu período. O número de segmentos por tarefa dependerá da razão entre o período da tarefa e o período mínimo do sistema. Portanto, o número de segmentos da tarefa T_k será p_k/p_1, onde p_1 é o período mínimo. Logo, o número total de segmentos é limitado por $n(p_n/p_1)$, onde p_n é o maior período, sem perda de generalidade. Chamemos N o número de segmentos, e supondo um esquema de codificação como ilustrado acima para a entrada, este número será PSEUDOPTIME, uma vez que depende do valor do período máximo. Mesmo se

Algoritmo Controle-Periodico-Cooperante;
começo
τ_{pa}=conjunto de tarefas periodicas ativas
τ_{pi}=conjunto de tarefas periodicas inativas
τ_{sa}=conjunto de tarefas esporadicas ativas
se (chegada)
$T_{ch} = (C_{ch}, P_{ch})$
se $(U_{\tau_{pa}} + C_{ch}/P_{ch} \leq 1)$
se $u_{T_{ch}} < u_{F(\tau_{pi})}$
$\tau_{pa} = \tau_{pa} \cup T_{ch}$
se PP(τ_{pa},τ_{sa})
ativa ESPExato1
senão
$\tau_{pa} = \tau_{pa} - T_{ch}$
$\tau_{pi} = \tau_{pi} \cup T_{ch}$
ordene τ_{pi} por utilização crescente
fimse
senão
$\tau_{pi} = \tau_{pi} \cup T_{ch}$
ordene τ_{pi} por utilização crescente
fimse
fimse
fimse
se (partida)
se $(T_{pt} \in \tau_{pa})$
$\tau_{pt} \subset \tau_{pi}$ tal que $u_{\tau_{pt}} \leq u_{T_{pt}}$
$\tau_{pa} = \tau_{pa} - T_{pt} \cup \tau_{pt}$
senão
$\tau_{pi} = \tau_{pi} - T_{pt}$
fimse
fimse
fim

Figura 6.6: Algoritmo Controle-Periódico-Cooperante

supormos que existe um esquema de codificação conveniente para a entrada, ela pode ser, ainda assim, uma função exponencial do número de tarefas.

De acordo com o modelo proposto, devemos avaliar, para cada tarefa, o valor do seu maior segmento e tomarmos este valor como base para a condição de suficiência. Portanto, o algoritmo é linear em N, o número de segmentos. Se pudermos limitar N como uma função polinomial em n, o algoritmo proposto seria PTIME. Verificamos, portanto, que no caso geral, o nosso algoritmo é PSEUDO-PTIME, uma vez que depende do tamanho da entrada.

Um fator pode ser usado para indicar que o algoritmo é conveniente para uso em tempo de execução: uma entrada pré-compilada, de forma que os valores de c_{imax} e c_{iavg} sejam conhecidos e disponíveis para uso imediato, ou seja, existe a possibilidade de modificarmos a forma de codificação de maneira à fazermos uso da propriedade do nosso modelo, que é estabelecer uma condição suficiente baseada apenas nos valores máximo e médio do tempo de execução dos segmentos. Portanto, cada tarefa comunicante deve apresentar uma entrada contendo apenas os valores c_{imax}, c_{iavg} e p_i, calculados em tempo anterior à execução. Desta forma, a entrada é PTIME, e o algoritmo é completamente tratável. Veremos a seguir que tal pré-compilação poderá, inclusive, direcionar determinados valores para os segmentos, de forma a minimizar o valor de δ_i. Para que esta redução de complexidade seja factível, é necessário se conhecer, em tempo anterior à execução, os parâmetros de todas as tarefas, inclusive as inativas.

6.5 Análise dos Tempos de Computação e Pontos de Comunicação

Vimos anteriormente que, dado um conjunto T de tarefas periódicas, contendo tarefas independentes e comunicantes, sendo U o fator de utilização do conjunto T, podemos afirmar que:

- Se $U > 1$, o sistema T não é escalonável.
- Se $U \leq 1 - \sum_{j=1}^{l} u_j(c_{jmax}/c_{javg} - 1)$ então o sistema T é escalonável.
- Se U estiver entre estes dois extremos, ou seja, $1 > U > 1 - \sum_{j=1}^{l} u_j(c_{jmax}/c_{javg} - 1)$, nada se pode afirmar.

Disto, vemos que é importante minimizarmos a quantidade c_{jmax}/c_{javg}, também conhecida por δ_j. Neste sentido, as nossas condições seriam aplicáveis em mais casos.

O nosso objetivo nesta seção é procurar alternativas para fornecer um valor de δ_j como desejado, através da uniformização dos tempos de execução dos segmentos das instâncias. Basicamente, isto pode ser feito aplicando-se uma abordagem que permita variar as versões dos segmentos, de tal forma a adaptarmos a melhor escolha dos segmentos que deverão compor a tarefa dinamicamente, em função da carga corrente do sistema. Por outro lado, existe também a possibilidade de alterarmos os pontos de comunicação, no sentido de modificarmos os valores dos segmentos.

6.5.1 Múltiplos Pontos de Comunicação

Adotamos o conceito de ponto de comunicação como sendo o local onde as tarefas se sincronizam e/ou trocam dados, de tal forma que sua ocorrência se dê entre dois segmentos da tarefa. Desta forma, a determinação dos pontos de comunicação e sua escolha estão implicitamente relacionados com o tamanho dos segmentos, podendo até influir na sua determinação.

Se as partições das tarefas em segmentos provêm do fato de que as tarefas necessitam de se comunicar em determinados pontos, é possível melhorar-se a relação de δ_i através de uma programação adequada das tarefas, de tal forma que os segmentos apresentem uma uniformidade aceitável. Isto pode ser conseguido através de uma escolha adequada da localização dos pontos de comunicação.

A semântica associada aos pontos de comunicação estabelece que toda computação anterior à um determinado ponto de comunicação deve preceder a computação posterior àquele ponto de comunicação, em todas as tarefas. Esta interpretação sugere que, de uma forma geral, os pontos de comunicação não são independentes, e são na verdade, determinados pela necessidade de sincronização/comunicação entre as tarefas. Desta forma, sua posição seria fixa e determinada independentemente da quantidade de tempo de execução dos códigos que o precedem e o sucedem. Entretanto, a programação pode ser dirigida para uma uniformização, dentro de uma mesma tarefa, dos tempos de execução de seus segmentos, sem que haja uma alteração na semântica original dos pontos de comunicação, através de uma análise dos tempos de computação entre cada ponto de comunicação, das escolhas adequadas para a localização destes pontos e da possibilidade de remanejamento de código que seja independente dos aspectos de sincronização.

6.5.2 Múltiplas Versões de Segmentos

Verificamos que δ_j deve ser próximo da unidade, para que a condição suficiente seja uma boa aproximação. Neste sentido, devemos procurar por tarefas que forneçam

$c_{jmax} = c_{javg}$. Isto poderia ser conseguido se, para cada tarefa, pudermos reduzir ou aumentar o tempo de execução de cada um de seus segmentos dinamicamente. Isto poderia ser conseguido através da escolha de uma versão para cada segmento, dentre as várias versões possíveis. Desta forma, poder-se-ia montar uma tarefa com versões independentes entre os segmentos de maneira a minimizarmos a diferença entre o maior segmento(tempo de execução) e a média do tempo de execução dos segmentos. Esta técnica permitiria uma melhoria na utilização da condição suficiente.

Seja uma tarefa T_j, com a_j segmentos, de tal forma que cada segmento tenha um conjunto de versões. No sentido de simplificar a programação e escolha de versões, vamos permitir a existência de tres versões por segmento, representando a versão original(0), a versão aceitável(1) e a pior versão aceitável(2). Logo, a tarefa T_j pode, agora, ser descrita como $T_j = (c_{j1}(l), .., c_{ja_j}(l), p_j)$, onde $l = 0, 1, 2$ representa a versão do segmento. Os valores dos tempos de execução alternativos dos segmentos são escolhidos *off-line*, de forma a adaptar os valores à uniformidade desejada.

Dado um conjunto T, verificamos sua escalonabilidade inicialmente testando se $\sum_{j=1}^{n} \delta_j u_j \leq 1$, onde $\delta_j = c_{jmax}/c_{javg}$ para tarefas comunicantes e $\delta_j = 1$ para tarefas independentes. Caso positivo, o conjunto é escalonável. Senão, verificamos se o conjunto tem possibilidades de ser escalonado através do teste de necessidade, ou seja, verificamos se $\sum_{j=1}^{n} u_i \leq 1$. Caso positivo, procuraremos reformular o subconjunto de tarefas comunicantes, no sentido de minimizarmos δ_j.

O problema que descreveremos a seguir será conhecido como o problema da seleção de configuração de versão. Podemos descrevê-lo da seguinte forma: dado um conjunto de tarefas periódicas $T = \{T_j\}$, com T_j dado como acima, queremos uma configuração deste conjunto de tal forma que a soma dos valores $\sum_{j=1}^{n} \delta_j u_j$ seja minimizada. Isto é possível desde que examinemos todas as configurações possíveis e verifiquemos qual delas vai fornecer uma uniformidade maior nos valores dos segmentos, de tal forma que o valor de δ_j seja minimizado. Portanto, devemos determinar, para cada tarefa T_j uma seleção de valores de segmentos de tal forma que a contribuição de T_j seja mínima. Este problema de otimização pode ser colocado como um problema de decisão, desde que expresso como uma questão que procura responder afirmativamente ou negativamente à pergunta se existe uma configuração de T_j tal que a diferença entre o valor mínimo e o valor máximo entre quaisquer dois segmentos da configuração é menor ou igual à um valor inteiro k.

Definição do problema de uniformidade de segmentos: dado uma tarefa periódica $T_j = (c_{j1}(l)..c_{ja_j}(l), p_j)$, com a_j segmentos, cada qual com $l = 3$ versões, e um inteiro positivo K, existe uma configuração, ou seja, uma escolha de versão para cada segmento de tal forma que a diferença entre o valor máximo e mínimo dos segmentos escolhidos seja, no máximo K?

Teorema 6.5.1 *O problema de uniformidade de segmentos é NP-completo.*

Prova. O problema de uniformidade de segmentos pode ser transformado do problema de escolha de períodos[KM91]. É fácil notar que a escolha da configuração é idêntica em ambos os casos. O problema de uniformidade está em NP, uma vez que uma solução apresentada por um algoritmo não-determinístico é checada em tempo polinomial.

Portanto, do que foi visto acima, a solução da escolha das versões para tarefas de tal forma que δ_j seja minimizado deverá ser aproximada por um algoritmo sub-ótimo. Como se trata de uma escolha de configuração, podemos usar os mesmos métodos de aproximação para tratar o problema de escolha de períodos.

6.6 Comentários Finais

É importante que um algoritmo que se pretenda útil em um ambiente de Tempo-Real Crítico seja rápido na decisão de escalonamento, e ao mesmo tempo, adaptativo, inclusive tendo a possibilidade desta adaptação ser feita em tempo de execução. Desta forma, a previsibilidade seria garantida, uma vez que sempre se pode garantir algum nível de execução de tarefas, e com um tempo de resolução bem curto. Seguindo estes princípios, algumas decisões úteis podem ser tomadas pré-tempo de execução, de forma à auxiliar as decisões essencialmente dinâmicas.

No que se refere a este trabalho, podemos estipular os valores de δ pré-tempo de execução, de tal forma que a escalonabilidade possa ser decidida rapidamente. O conhecimento de todos os parâmetros das tarefas em tempo anterior à execução é determinante para se escolher um valor apropriado para as possíveis versões de forma à minimizar o valor de δ.

Capítulo 7

Conclusão

7.1 Introdução

Este capítulo está estruturado em tres seções. Na seção 7.1, revisamos os objetivos deste trabalho. Na seção 7.2, são discutidos os resultados obtidos e na seção 7.3, propomos sugestões para a continuidade desta linha de pesquisa.

7.2 Revisão dos Objetivos

O Objetivo geral deste trabalho é propiciar um conjunto de avaliações que indiquem a escalonabilidade de um conjunto tarefas de tempo real que operam em um ambiente dinâmico. Portanto, este trabalho contribui para o projeto de sistemas operacionais para STRCs em termos de minimização do tempo de decisão da escalonabilidade, uma vez que esta decisão deve ser tomada em tempo de execução. O modelo de tarefas de um sistema típico engloba tarefas periódicas e esporádicas dinâmicas, e seu teste de escalonabilidade em tempo de execução não pode degradar a utilização da UCP. Os modelos de associação de prioridades se basearam nos princípios de atribuição fixa e atribuição dinâmica de prioridades com servidor esporádico[Va94].

7.3 Resultados Obtidos

Inicialmente, procuramos mapear o modelo de tarefas como um conjunto de tarefas periódicas representando tanto o conjunto original de tarefas periódicas, como o conjunto de tarefas esporádicas. Por ser mais simples de implementar, procuramos avaliar

a escalonabilidade deste conjunto utilizando uma atribuição fixa de prioridades, neste caso, o algoritmo de prazo monotônico. Verificamos que, apesar de termos obtido uma condição suficiente para a escalonabilidade de um conjunto de tarefas independentes com prazo arbitrário não disponível na literatura, este tipo de mapeamento subutiliza enormemente a UCP, em função do tipo de atribuição de prioridades. Entretanto, como um teste de escalonabilidade para tarefas periódicas em tempo de execucão, com prazos arbitrários, verificamos que esta condição pode ser muito útil, inclusive podendo ser integrada com outros métodos mais complexos.

A seguir, escolhemos um mapeamento de tarefas periódicas utilizando o algoritmo do próximo prazo associado à idéia de tempo livre em um intervalo, de tal forma que as tarefas esporádicas pudessem ter sua escalonabilidade testada em tempo de execução, dependendo da disponibilidade deste tempo livre. Através desta atribuição variável de prioridades, verificamos que a utilização da UCP aumenta e ao mesmo tempo, uma grande porcentagem de tarefas esporádicas pode cumprir suas restrições temporais. Apesar de contemplar bem o escalonamento de tarefas periódicas dinâmicas, este método não tem eficácia quando procuramos escalonar tarefas esporádicas essenciais, pois em situações de sobrecarga, não há garantia de que as tarefas essenciais vão cumprir seus prazos.

A possibilidade de sobrecarga necessita de uma abordagem que faça a distinção entre as tarefas esporádicas essenciais e as críticas. Esta necessidade foi comtemplada com a introdução de nível de importância entre tarefas esporádicas, de tal forma que tarefas mais importantes deveriam ter alguma forma de prioridade sobre as menos importantes. Como a importância não tem, em princípio, relação com o prazo, esta solução não é trivial. Foi proposto um algoritmo que tem uma complexidade polinomial para a solução deste problema, e ilustrado que este algoritmo tem soluções melhores que os encontrados na literatura[Va94a].

Finalmente, consideramos a escalonabilidade com tarefas periódicas que compartilham recursos, de tal forma que sua execução deve seguir regras de precedência. Em geral, tais recursos são compartilhados de tal forma que a seção crítica é muito pequena, possibilitando que a única relação entre as tarefas seja através das relações de precedência. Desenvolvemos um método que pode ser aplicado para teste em tempo de execução, quando um subconjunto de tarefas periódicas compartilham recursos sem exclusão mútua.

7.4 Sugestões para Continuidade do Trabalho

Como continuidade do trabalho, estamos investigando de que forma podemos estender o mecanismo de importância. A extensão do mecanismo de importância para toda tarefa do sistema, seja periódica ou esporádica, deve ser considerada. Em geral, associa-se um nível de importância apenas para tarefas esporádicas, considerando que as tarefas periódicas tem uma importância uniforme e superior à das tarefas esporádicas. Seria interessante investigar aplicações onde determinadas tarefas esporádicas tivessem uma necessidade de garantia maior, de tal forma que um número maior de tarefas essenciais pudessem ser ativadas e satisfatoriamente tratadas ao mesmo tempo. Evidentemente, isto acarretaria um ganho na utilização da UCP, uma vez que não haveria reserva de tempo, sobre as metodologias que empregam a reserva de tempo. Por outro lado, esta abordagem garantiria a previsibilidade de execução sem prejuízo para tarefas periódicas importantes.

Bibliografia

[ABRW92] Neil C. Audsley, Alan Burns, M. F. Richardson and A. J. Wellings. Deadline Monotonic Scheduling Theory. In *IFAC Workshop on Real-Time Programming*, pp. 55–60, 1992.

[AC93] Juan Manuel Adán-Coello. *Uma Contribuição à Programação e Escalonamento de Sistemas Distribuídos de Tempo-Real.* PhD thesis, Universidade de Campinas, Campinas-SP, 1993.

[ACa92] Juan Manuel Adán-Coello and Maurício F. Magalhães. A Scheduling Strategy for Distributed Hard Real-Time Programming Environment. In *IFAC Workshop on Real-Time Programming*, 1992.

[Aud90] Neil Audsley. Deadline Monotonic Scheduling. Technical Report YCS 146, University of York, 1990.

[BMR90] Sanjoy K. Baruah, Aloysius K. Mok and Louis E. Rosier. Preemptively Scheduling Hard Real-Time Sporadic Tasks on One Processor. In *Proceedings of the 11st. IEEE Real-Time Systems Symposium*, pp. 182–190. IEEE, December 1990.

[BRH91] Sanjoy K. Baruah, Louis E. Rosier and R. R. Howell. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. *Real-Time Systems Journal*, 1991.

[BSR88] Sara R. Biyabani, John A. Stankovic and Krithivasan Ramamrithan. The Integration of Deadline and Criticalness in Hard Real-Time Scheduling. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, pp. 152–159. IEEE, 1988.

[Car84] Carlow. Architecture of the Space Shuttle Primary Avionics Software System. *CACM*, 27:926–936, September 1984.

[CC89] Houssine Chetto and Maryline Chetto. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Trans. on Soft. Engineering*, 15(10):1261–1269, October 1989.

[CM91] Ken Chen and Paul Muhlethaler. A Family of Scheduling Algorithms for Real-Time Systems using Time Value Functions. Technical report, 1991.

[Cof76] E. G. Coffman. *Computer and Job-Shop Scheduling.* Introduction to Deterministic Scheduling Theory. John Wiley & Sons, 1976.

[CSR87] Shengchang Cheng, John A. Stankovic and Krithivasan Ramamrithan. Scheduling Algorithms for Hard Real-Time Systems: A Brief Survey. *IEEE Real-Time Systems Newsletter*, 3(2):1–23, Summer 1987.

[Dij71] Edsger W. Dijkstra. Hierarquical Ordering of Sequential Processes. *Acta Informatica 1*, pp. 115–138, 1971.

[DM89] Michael L. Dertouzos and Aloysius K. Mok. Multiprocessor On-line Scheduling of Hard Real-Time Systems. *IEEE Trans. on Soft. Engineering*, 1989.

[FP88] S. R. Faulk and David L. Parnas. On Synchronization in Hard-Real-Time Systems. *CACM*, 31(3):274–287, 1988.

[Gar81] J. R. Garman. The Bug Heard Round the World. *ACM Software Engineering Notes*, 6(5):3–10, October 1981.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[Hen80] Kathryn L. Heninger. Specifying Software Requirements for Complex Systems: New Techniques and their Application. *IEEE Trans. on Soft. Eng.*, SE-6(1):1–12, January 1980.

[Hoa74] C. A. R. Hoare. Monitors: An Operating System Structuring Concept. *CACM*, 17(10):549–557, October 1974.

[Hoa78] C. A. R. Hoare. Communicating Sequential Processes. *CACM*, 21(8):666–677, August 1978.

[Hu62] T. C. Hu. Parallel Sequencing and Assembly Line Problems. *Operation Research*, 1962.

[Hua85] James P. Huang. Modeling of Software Partition for Distributed Real-Time Applications. *IEEE Trans. on Soft. Engineering*, SE-11(10):1113–1125, October 1985.

[JaAC93] Alencar Melo Jr, Maurício F. Magalhães e Juan Manuel Adán-Cuello. Escalonamento On-line de Processos Esporádicos em Sistemas de Tempo-Real Crítico. Em *Congresso da Sociedade Brasileira de Automática*, 1993.

[Jef89] Kevin Jeffay. Analysis of a Synchronization and Scheduling Discipline for Real-Time Tasks with Preemption Constraints. In *IEEE 10th Real-Time Systems Symposium*, pp. 295–305, December 1989.

[JLT85] E. Douglas Jensen, C. Douglass Locke and Hideyuki Tokuda. A Time-driven Scheduling Model for Real-Time Operating Systems. In *IEEE 6th Real-Time Systems Symposium*, pp. 112–122, December 1985.

[Jr77] Mario J. Gonzalez Jr. Deterministic Processor Scheduling. *Computing Surveys*, 9(3):173–204, September 1977.

[KM91] Tei-Wei Kuo and Aloysius K. Mok. Load Adjustment in Adaptive Real-Time Systems. In *Proceedings of the 12nd. IEEE Real-Time Systems Symposium*. IEEE, December 1991.

[Kuo91] Tek-Wei Kuo. Research proposal, 1991.

[LC89] Liestman and Campbell. A Fault-Tolerant Scheduling Problem. 1989.

[Leh90] John P. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *IEEE 11st Real-Time Systems Symposium*, pp. 201–209. IEEE, December 1990.

[Lev84] Nancy G. Leveson. Software Safety: Why, What and How. *ACM Trans. on Computer Science*, 1984.

[LL73] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[LM69] E. L. Lawler and J. M. Moore. A Functional Equations and its Application to Resource Allocation. *Management Science*, 16(1):77–84, 1969.

[LM80] Joseph Y.-T. Leung and M. L. Merril. A Note on Preemptive Scheduling of Periodic, Real-Time Tasks. *Information Processing Letters*, 11(3), November 1980.

[LS86] John Lehoczky and Lui Sha. Performance of Real-Time Bus Scheduling Algorithms. *ACM Performance Evaluation Review*, (14), 1986.

[LSD89] John P. Lehoczky, Lui Sha and Ye Ding. The Rate Monotonic Scheduling Algorithms: Exact Characterization and Average Case Behaviour. In *IEEE 10th Real-Time Systems Symposium*, pp. 166–171. IEEE, December 1989.

[LW82] Joseph Y.-T. Leung and Jennifer Whitehead. On the Complexity of Fixed-priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation*, (2):237–250, 1982.

[MAC+87] Aloysius K. Mok, Prasanna Amerasinghe, Moyer Chen, Supoj Sutanthavibul and Kamtorn Tantisirivat. Synthesis of a Real-Time Message Processing System with Data-driven Timing Constraints. In *IEEE 8th Real-Time Systems Symposium*, pp. 133–143. IEEE, December 1987.

[MF75] G. McMahon and M. Florian. On Scheduling with Ready Time and Due Dates to Minimize Maximum Lateness. *Operation Research*, 23, 1975.

[Mok83] Aloysius K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment.* PhD thesis, MIT, Cambridge, MA, 1983.

[Mok84] Aloysius K. Mok. The Design of Real-Time Programming Systems based on Process Models. In *IEEE 5th Real-Time Systems Symposium*, 1984.

[PS87] Dartzen Peng and Kang G. Shin. Modeling of Concurrent Task Execution in Distributed System for Real-Time Control. *IEEE Trans. on Computers*, 36(4):500–516, April 1987.

[RS89] Krithivasan Ramamrithan and John A. Stankovic. Overview of the Spring Project. *IEEE Real-Time Systems Newsletter*, 5(1):18–44, winter 1989.

[RS91] Krithivasan Ramamrithan and John A. Stankovic. *Foundations of Real-Time Computing: Scheduling and Resource Management*, Scheduling Strategies Adopted in Spring: An Overview, pp. 277–305. Klewer Academics, 1991.

[SG90a] Lui Sha and John B. Goodenough. Real-Time Scheduling Theory and Ada. *Computer*, 23(4):53–62, 1990.

[SG90b] Terry Sheppard and Martin Gagné. A Model of the f18 Mission Computer Software for Pre-Run-Time Scheduling. In *10th Int. Conf. on Distributed Computing Systems*, pp. 62–69, 1990.

[SKL85] Kang G. Shin, C. M. Krishna and Y.-H. Lee. A Unified Method for Evaluating Real-Time Computer Controllers and its Application. *IEEE Trans. on Automatic Control*, AC-30(4):357–366, April 1985.

[SLR86] Lui Sha, John P. Lehoczky and Ragunathan Rajkumar. Solutions for Some Practical Problems in Prioritized Preemptive Scheduling. In *IEEE 7th Real-Time Systems Symposium*, pp. 181–191. IEEE, December 1986.

[SR88] John A. Stankovic and Krithivasan Ramamrithan. *Hard Real-Time Systems - Tutorial Text.* What is a Real-Time System. IEEE Press, 1988.

[SR91] John A. Stankovic and Krithivasan Ramamrithan. Editorial: What is Predictability for Real-Time Systems. *Real-Time Systems Journal*, 1991.

[SRC85] John. A. Stankovic, Krithivasan Ramamrithan and Shengchang Cheng. Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems. *IEEE Trans. on Computers*, C-34(12):1130–1143, December 1985.

[SRL90] Lui Sha, Ragunathan Rajkumar and John Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Trans. on Computers*, 39:1175–1185, 1990.

[SSL89] Brinkley Sprunt, Lui Sha and John Lehoczky. Aperiodic Task Scheduling for Hard-Real-Time Systems. *Journal of the Real-Time Systems*, (1):27–60, 1989.

[Sta88a] John A. Stankovic. *Hard Real-Time Systems - Tutorial Text.* Real-Time Computing Systems: The Next Generation. IEEE Press, 1988.

[Sta88b] John A. Stankovic. Misconceptions about Real-Time Computing. *IEEE Computer*, October 1988.

[SZ92] Karsten Schwan e Hongyi Zhou. Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads. *IEEE Trans. on Soft. Engineering*, 18(8):736–747, August 1992.

[Tan92] Andrew Tanembaum. *Modern Operating Systems.* Introduction to Processes. Prentice-Hall, 1992.

[Va94] Sibelius L. Vieira and Maurício F. Magalhães. On-line Sporadic Task Scheduling in Hard Real-Time Systems. In *6th Euromicro Workshop on Real-Time Systems*, Vasteras, Sweden, 1994.

[Va94a] Sibelius L. Vieira and Maurício F. Magalhães. Sporadic Task Scheduling with Overload Conditions, 1994. Submitted for Publication.

[Va94b] Sibelius L. Vieira and Maurício F. Magalhães. Dynamic Scheduling of Periodic and Sporadic Hard Real-Time Tasks, 1994. Submitted for Publication.

[VTB83] C. A. Vissers, R. L. Tenney and G. V. Bochmann. Formal Description Techniques. *Proceedings of the IEEE*, 71(12):1356–1364, December 1983.

[Wan91] Farn Wang. Research proposal, 1991. Private Communication.

[Wir77] Nicolaus Wirth. Towards a Discipline of Real-Time Programming. *CACM*, 20(8):577–583, August 1977.

[XP90] Jia Xu and David L. Parnas. Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations. *IEEE Trans. on Soft. Engineering*, 16(3):360–369, march 1990.

[XP91] Jia Xu and David L. Parnas. On Satisfying Timing Constraints in Hard-Real-Time Systems. *Software Engineering Notes*, 16(5):137–144, December 1991.

[YTA87] Xiaoping Yuan, Satish Tripath and Ashok Agrawala. Scheduling in Real-Time Distributed Systems - A Review. Technical Report UMIACS-TR-87-62, University of Maryland, 1987.

[ZR87] W. Zhao and K. Ramamrithan. Virtual Time CSMA protocols for Hard-Real-Time Communications. *IEEE Transactions on Software Engineering*, 1987.

[ZRS87] W. Zhao, K. Ramamrithan and J. Stankovic. Preemptive Scheduling under Time and Resource Constraints. *IEEE Transactions on Computers*, 1987.