Revista de
Informática
Teórica e Aplicada

Computação Imprecisa

Rômulo Silva Oliveira

UFRGS - Instituto de Informática
romulo@inf.ufrgs.br
Caixa Postal 15064 - 91501-970
Porto Alegre - RS

Abstract

Real-Time Computing Systems are defined as those systems that have timing constraints. This paper describes the Imprecise Computation technique. It has been proposed as an approach to the scheduling of Real-Time Systems. We present motivations for its use and basic programming paradigms. We also discuss the different scheduling goals associated with this approach in the literature. Finally, several scheduling algorithms are described and classified.

Keywords: Imprecise computation, real-time systems, scheduling

Resumo

Sistemas computacionais de tempo real são identificados como aqueles submetidos a requisitos de natureza temporal. Este trabalho descreve a técnica de Computação Imprecisa, uma abordagem proposta para o escalonamento de sistemas tempo real. São apresentadas motivações para o seu emprego e as formas básicas de programação. São também discutidos os diferentes objetivos de escalonamento propostos na literatura para este tipo de abordagem. Finalmente, diversos algoritmos de escalonamento são descritos e classificados.

Palavras-chave: computação imprecisa, sistemas tempo real, escalonamento.

1. Introdução

Sistemas computacionais de tempo real (STR) são identificados como aqueles sistemas computacionais submetidos a requisitos de natureza temporal. Nestes sistemas, os resultados devem estar corretos não somente do ponto de vista lógico, mas também devem ser gerados no momento correto. Os defeitos de natureza temporal nestes sistemas são, em alguns casos, considerados críticos no que diz respeito às suas conseqüências.

Na medida em que o uso de sistemas computacionais prolifera em nossa sociedade, aplicações com requisitos de tempo real tornam-se cada vez mais comuns. Estas aplicações variam muito com relação ao tamanho, complexidade e criticalidade. Entre os sistemas mais simples estão os controladores embutidos em utilidades domésticas, tais como lavadoras de roupa e videocassetes. Na outra extremidade deste espectro estão os sistemas militares de defesa e o controle de tráfego aéreo. Exemplos de aplicações críticas são os sistemas responsáveis pelo monitorização de pacientes em hospitais e os sistemas embarcados em veículos, de automóveis até aviões, e sondas espaciais. Entre aplicações não críticas estão os videogames e as aplicações multimedia.

No contexto da automação industrial são muitas as possibilidades (ou necessidades) de empregar sistemas com requisitos de tempo real ([REM 93]). Exemplos são os sistemas de controle embutidos em equipamentos industriais, os sistemas de supervisão e controle de células de manufatura e os sistemas responsáveis pela supervisão e controle de plantas industriais como um todo. Em função do baixo custo dos processadores, dos avanços na área de redes de computadores e da necessidade física de algumas aplicações, soluções distribuídas são cada vez mais empregados.

Um problema básico encontrado na construção de sistemas tempo real é a alocação e o escalonamento das tarefas nos recursos computacionais disponíveis. Existe uma dificuldade intrínseca em compatibilizar dois objetivos fundamentais ([BUR 91]): garantir que os resultados serão produzidos no momento desejado e dotar o sistema de flexibilidade para adaptar-se a um ambiente dinâmico e, assim, aumentar sua utilidade. A dificuldade de escalonar tarefas com requisitos de tempo real é bastante conhecida, constituindo uma área de pesquisa intensa atualmente.

Em um extremo existem soluções de escalonamento que supõe um conjunto fixo de tarefas a serem executadas. Estas soluções reservam recursos para o pior caso e são capazes de garantir ainda em projeto que todas as tarefas serão concluídas no momento correto. Entretanto, aplicações construídas desta forma resultam em sistemas pouco flexíveis e na subutilização dos recursos computacionais. Entre as soluções deste primeiro grupo estão aquelas baseadas em executivo cíclico ([DAM 89], [KOP 89]), onde todo o trabalho de escalonamento é realizado em tempo de projeto ("off-line"). Este primeiro grupo de soluções também inclui o escalonamento baseado em prioridade acompanhado por um teste de escalonabilidade executado em tempo de projeto ([AUD 93], [SHA 94], [TIN 94]).

No outro extremo temos as soluções de escalonamento que não garantem o comportamento temporal da aplicação. Tarefas são escalonadas na medida do possível. Embora os recursos computacionais sejam plenamente utilizados e o sistema resultante seja bastante flexível, a falta de uma garantia prévia para o seu comportamento temporal inviabiliza este tipo de solução para muitas classes de aplicações. Nas soluções de escalonamento deste segundo grupo (melhor esforço na execução), não existe garantia em tempo de projeto de que os deadlines serão cumpridos. O escalonamento tipo "melhor esforço" ("best effort") quando muito fornece uma previsibilidade probabilista sobre o comportamento temporal do sistema, a partir de uma estimativa da carga. Algumas propostas dentro desta linha oferecem uma "garantia dinâmica" ao determinar, em tempo de execução, quais prazos serão ou não atendidos.

Uma conseqüência imediata destas abordagens dinâmicas é a possibilidade de sobrecargas ("overload") no sistema. O sistema se encontra em estado de sobrecarga quando não é possível executar todas as tarefas dentro dos seus respectivos prazos. É importante observar que a sobrecarga não é um estado anormal, mas uma situação que ocorre naturalmente em sistemas que empregam uma abordagem tipo melhor esforço. Logo, é necessário um mecanismo para tratar a sobrecarga. As abordagens usuais para o tratamento de sobrecarga são descarte por completo de algumas tarefas ([RAM 89], [RAM 90]) ou execução de todas as tarefas com o sacrifício do prazo de execução de algumas delas ([JEN 85]).

Uma das técnicas existentes na literatura para resolver o problema de escalonamento tempo real é a Computação Imprecisa ([LIU 94]). Esta técnica procura, de certa forma, conciliar os dois objetivos fundamentais citados antes. Este artigo contém uma descrição geral da Computação Imprecisa, com especial ênfase ao aspecto escalonamento. O restante do artigo está organizado da seguinte forma: a seção 2 caracteriza a técnica de Computação Imprecisa; a seção 3 descreve as formas de programação normalmente empregadas; a seção 4 discute os diferentes objetivos de escalonamento possíveis na presença de tarefas compostas por parte obrigatória e parte opcional; a seção 5 descreve diversos algoritmos de escalonamento existentes na literatura; finalmente, na seção 6 são feitos os comentários finais.

2. Caracterização da Computação Imprecisa

Os sistemas em geral, que não apresentam restrições de tempo real, são caracterizados por uma abordagem do tipo "fazer o trabalho usando o tempo que for necessário". Já os sistemas tempo real possuem uma abordagem diferente, pois é preciso garantir que será possível atender prazos, geralmente impostos pelo ambiente do sistema. Logo, a preocupação é "garantir que o trabalho será concluído no tempo disponível".

A dificuldade encontrada no escalonamento tempo real levou alguns autores a proporem uma abordagem diferente, do tipo "fazer o trabalho possível dentro do tempo disponível". Isto significa sacrificar a qualidade dos resultados para poder cumprir os prazos exigidos. Esta técnica, conhecida pelo nome de Computação Imprecisa, flexibiliza o

escalonamento tempo real. As primeiras publicações a respeito desta técnica datam de 1987 ([LIN 87a], [LIN 87b]).

Computação Imprecisa está fundamentada na idéia de que cada tarefa do sistema possui uma *parte obrigatória* ("mandatory") e uma *parte opcional* ("optional"). A parte obrigatória da tarefa é capaz de gerar um resultado com a qualidade mínima, necessária para manter o sistema operando de maneira segura. A parte opcional então refina este resultado, até que ele alcançe a qualidade desejada. O resultado da parte obrigatória é dito *impreciso* ("imprecise result"), enquanto o resultado das partes obrigatória+opcional é dito *preciso* ("precise result"). Uma tarefa é chamada de *tarefa imprecisa* ("imprecise task") se for possível decompo-la em parte obrigatória e parte opcional.

Em situações normais, tanto a parte obrigatória quanto a parte opcional são executadas e o sistema gera resultados com a precisão desejada. Se, por algum motivo, não for possível executar todas as tarefas do sistema, algumas partes opcionais serão deixadas de lado. Este mecanismo permite uma degradação controlada do sistema, na medida em que pode-se determinar o que não será executado em caso de sobrecarga.

É possível dizer que Computação Imprecisa faz uma composição das abordagens tipo melhor esforço (partes opcionais) com as abordagens que oferecem garantia (partes obrigatórias). A técnica como um todo é do tipo melhor esforço, pois não oferece uma previsibilidade determinista para todas as tarefas do sistema. Entretanto, as partes obrigatórias tomadas isoladamente formam um subproblema que deve ser tratado pelas abordagens que oferecem garantias. Neste subproblema devem ser observadas as condições necessárias para uma previsibilidade determinista, ou seja, carga limitada, conhecida, e reserva de recurso para o pior caso.

As partes opcionais serão executadas quando existirem recursos disponíveis, sem garantias em tempo de projeto. No máximo ter-se-á uma previsibilidade probabilista, se a carga for definida também em termos probabilistas. Isto permite a melhor utilização dos recursos pelas partes opcionais, que podem aproveitar recursos reservados em tempo de projeto e não utilizados pelas partes obrigatórias. Como não existe reserva de recursos para as partes opcionais, podem ocorrer sobrecargas temporárias. Neste caso, a qualidade do resultado é sacrificada para que os prazos sejam mantidos. Na falta de recursos, partes opcionais podem ser até totalmente descartadas.

Computação Imprecisa diferencia-se das demais abordagens tipo melhor esforço exatamente na forma como a sobrecarga é tratada. Nas demais abordagens do tipo melhor esforço, a sobrecarga é tratada através do simples descarte de tarefas ou da flexibilização do conceito de deadline. Na Computação Imprecisa, o tempo de computação das tarefas é negociado a partir da introdução do conceito de qualidade do resultado. Tarefas não são simplesmente descartadas, mas a qualidade do resultado produzido é parcialmente sacrificada. Isto gera uma redução na demanda pelos recursos do sistema que permite o atendimento dos deadlines, no seu sentido mais rigoroso.

O modelo de tarefas normalmente associado com Computação Imprecisa não exclui a existência de tarefas somente com parte obrigatória ou somente com parte opcional. Cabe

à semântica da aplicação definir quais são as partes obrigatórias e quais são as opcionais. Este é um modelo de tarefas mais flexível do que o encontrado nas outras abordagens.

3. Motivação e Exemplos de Aplicações

Em sistemas computacionais cuja carga não é completamente conhecida em projeto, Computação Imprecisa representa um mecanismo para tratamento de sobrecarga que respeita os deadlines das tarefas. É um tratamento diferente da simples rejeição da tarefa ou da flexibilização do conceito de deadline. O escalonamento é feito no sentido de obter o melhor comportamento possível para o sistema, dadas as restrições de recursos e deadlines.

Em uma situação de carga conhecida e limitada, sempre é possível reservar recursos para o pior caso de todas as tarefas e garantir todos os deadlines. Entretanto, em sistemas grandes e complexos, o custo de um sistema com tal nível de garantia pode ser proibitivo. Uma abordagem tipo melhor esforço é justificada neste contexto pelo aspecto econômico. Através do emprego da Computação Imprecisa, o custo do sistema diminui, pois são reservados recursos apenas para as partes obrigatórias das tarefas. Ao mesmo tempo, é mantido um comportamento mínimo garantido em tempo de projeto.

Computação Imprecisa também pode ser usada para viabilizar o emprego, em sistemas de tempo real, de algoritmos cujo tempo de execução no pior caso torna o escalonamento inviável. Alguns autores acreditam que este pode ser o caminho para obter-se uma previsibilidade determinista em ambientes não deterministas, que requerem algoritmos complexos, cujo tempo de execução no pior caso é difícil de prever. Em [LIU 94] é citado o exemplo de uma tarefa que, interativamente, procura a raiz de um polinômio. Um exemplo de aplicação que emprega Computação Imprecisa com esta finalidade pode ser encontrado em [KOP 89]. Naquele artigo é descrita uma aplicação que controla a laminação de aços planos. A qualidade do aço, com respeito a sua espessura, depende da precisão no controle da velocidade de fabricação. Um algoritmo simples forma a parte obrigatória enquanto um algoritmo complexo, baseado em um modelo preciso do sistema, forma a parte opcional.

Liu e outros descrevem Computação Imprecisa em [LIU 94] como um mecanismo para tratar de sobrecargas temporárias. Desta forma, tais sistemas ficariam mais robustos e confiáveis. Como exemplos de aplicação, são citados o processamento de imagens e o rastreamento de objetos. Também é sugerido em [LIU 94] o uso de Computação Imprecisa como forma de tolerância a falhas em sistemas tempo real. A execução da parte obrigatória da tarefa pode ser vista como uma recuperação em avanço ("forward recovery"), quando uma falha qualquer impede a execução completa da parte opcional da tarefa. Resultados usáveis dentro do prazo, ainda que imprecisos, aumentam a qualidade do sistema. Por exemplo, [HUL 95] cita o sistema de alerta de tráfego e colisão ("Traffic Alert and Collision Avoidance System", TCAS) usado em aeronaves civis para alertar o piloto de uma possível colisão. Obviamente um alerta aproximado a tempo é melhor do que um alerta preciso porém tarde demais.

Uma linha de raciocínio semelhante é seguida em [YU 92] e [SHI 94]. Computação Imprecisa é utilizada em [YU 92] para que o sistema possa tolerar a perda de recursos computacionais em função de falhas no hardware. Na medida em que alguns processadores falham, uma mudança no modo de operação aumenta a carga nos processadores que continuam funcionando. Uma degradação suave acontece na medida em que resultados menos precisos são gerados em função desta sobrecarga. Também em [SHI 94], Computação Imprecisa é citada como uma técnica que pode prover tolerância a falhas em sistemas tempo real. Desta forma, a presença de elementos faltosos em um sistema tempo real resulta em uma redução na qualidade dos serviços prestados, com o objetivo de permitir ao sistema continuar atendendo os prazos das tarefas críticas.

Em [GAR 94] é feita uma retrospectiva dos trabalhos na área de inteligência artifical visando aplicações em tempo real. Garvey e Lesser citam Computação Imprecisa como a forma natural para implementar algoritmos tipo "a qualquer tempo" ("anytime algorithms"). Um algoritmo tipo "a qualquer tempo" é formado por refinamentos iterativos que, a qualquer instante, podem ser interrompidos e a melhor resposta até o momento é fornecida. É esperado que a qualidade da resposta melhore na medida em que o algoritmo tenha mais tempo para executar. Como exemplo de algoritmos deste tipo é possível citar as aproximações numéricas, pesquisas heurísticas, programação dinâmica e pesquisa em banco de dados.

O artigo [FOR 96] descreve uma arquitetura e um modelo de tarefas baseado em Computação Imprecisa que foi desenvolvido especialmente para aplicações de controle que misturam métodos convencionais com técnicas de inteligência artificial. Como exemplos de aplicações são citados controle de processos, aviônica e robótica. Em [HUA 95] o conceito de Computação Imprecisa é empregado em sistemas distribuídos como um mecanismo para controlar congestionamentos na transmissão de vídeo em tempo real. Neste caso, Computação Imprecisa permite adaptar a qualidade da imagem transmitida ao tempo disponível para a transmissão. Aplicação semelhante é descrita em [MIL 94] e [FEN 96] para a transmissão de vídeo através de chaves ATM.

Uma área promissora para o emprego de Computação Imprecisa é a de simulação em tempo real. Por exemplo, aplicações como o Simulador de Direção Iowa ("Iowa Driving Simulator", [KUH 95]). Nestes sistemas, a necessidade de simular o comportamento dinâmico do objeto em questão e simultaneamente gerar as animações gráficas pertinentes dentro de um prazo determinado representa o cenário perfeito para um compromisso entre qualidade e tempo disponível.

4. Formas de Programação

Existem três formas básicas de programar tarefas imprecisas normalmente citadas na literatura: a programação pode ser feita com funções monotônicas, funções de melhoramento ou múltiplas versões.

As *funções monotônicas* ("monotone functions") são aquelas cuja qualidade do resultado aumenta (ou pelo menos não diminui) na medida em que o tempo de execução da

função aumenta. As computações necessárias para obter-se um nível mínimo de qualidade correspondem à parte obrigatória. Qualquer computação além desta será incluída como parte opcional. O nível mínimo de qualidade deve garantir uma operação segura do sistema, enquanto a parte opcional refina progressivamente o resultado da tarefa. Segundo [LIU 91], algoritmos deste tipo podem ser encontrados nas áreas de cálculo numérico, estimativa probabilista, pesquisa heurística, ordenação e consulta a banco de dados. Este tipo de função faz com que o escalonador tenha que decidir quanto tempo de processador cada parte opcional deve receber. É a forma de programação que fornece maior flexibilidade ao escalonador.

Funções de melhoramento ("sieve functions") são aquelas cuja finalidade é produzir saídas no mínimo tão precisas quanto as correspondentes entradas. O valor de entrada é melhorado de alguma forma pela função. Se o resultado recebido como entrada por uma função de melhoramento é aceitável como saída, então a função pode ser completamente omitida (não executada). As funções de melhoramento normalmente formam partes opcionais que seguem algum cálculo obrigatório. Tipicamente, não existe benefício em executar uma função de melhoramento parcialmente. Isto significa que o escalonador deve optar, antes de iniciar a tarefa, em executá-la completamente ou não executá-la. Um exemplo citado em [LIU 91] é o processamento de sinais de radar. Nestes, o passo que computa uma nova estimativa para o nível de ruído no sinal recebido pode ser omitido e a estimativa anterior usada no lugar. Também algoritmos de processamento de imagens são capazes de produzir imagens razoáveis mesmo se algumas etapas forem descartadas.

Uma tarefa imprecisa também pode ser implementada através de *múltiplas versões* ("multiple versions"). Normalmente são empregadas duas versões. A versão primária gera um resultado preciso, porém possui um tempo de execução no pior caso desconhecido ou muito grande. A versão secundária gera um resultado impreciso, porém seguro para o sistema, em um tempo de execução menor e conhecido. A cada ativação da tarefa, cabe ao escalonador escolher qual versão será executada. No mínimo, deve ser garantido tempo de processador para a execução da versão secundária, a qual corresponde a parte obrigatória. A parte opcional é definida pela diferença entre os tempos máximos de execução das versões primária e secundária. Esta técnica, usada na aplicação descrita em [KOP 89], é a mais flexível do ponto de vista da programação.

A forma de programação empregada interfere no comportamento do escalonador. Quando uma função monotônica é empregada, o tempo de processador alocado à parte opcional pode ser qualquer valor entre zero e o tempo máximo de execução da parte opcional. Isto porque, em uma função monotônica, qualquer tempo de processador fornecido ajuda a melhorar a qualidade do resultado. Além disto, a decisão de interromper a execução da parte opcional pode ser tomada a qualquer instante, mesmo quando esta já estiver executando (figura 1-a).

Quando a parte opcional executa uma função de melhoramento, não existe benefício em executa-la parcialmente. Assim, o escalonador deve decidir se executa a função de melhoramento completamente ou a descarta. Esta característica é chamada na bibliografia de *restrição 0/1* ("0/1 constraint"). Ela restringe de certo modo a flexibilidade do

escalonador. Além disto, a decisão deverá ser tomada, no mais tardar, quando a parte obrigatória é concluida e a parte opcional deve (ou não) ser iniciada. Uma vez iniciada a execução da parte opcional, ela é executada até o final (figura 1-b).

De forma semelhante, multiplas versões criam uma restrição 0/1. Isto é, o emprego de múltiplas versões obriga o escalonador a executar completamente a parte opcional (escolhendo a versão primária) ou descartá-la completamente (escolhendo a versão secundária). Esta decisão deve ser tomada antes de iniciar a parte obrigatória da tarefa pois, uma vez escolhida a versão, a execução ou não da parte opcional já estará automaticamente definida (figura 1-c).

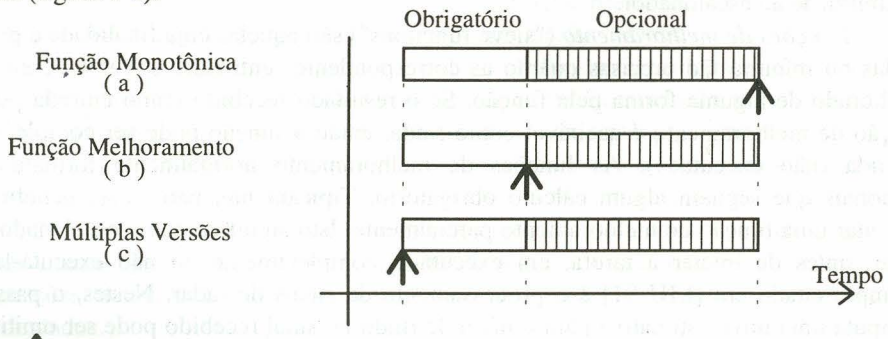

Figura 1 - Último instante para decidir sobre a execução da parte opcional.

Uma discussão sobre como programar tarefas imprecisas usando a linguagem ADA pode ser encontrada em [LIU 88] e [AUD 91]. Em [KEN 91] é descrita a linguagem Flex, com suporte específico para este tipo de programação. Em [HUL 95] e [HUL 96] a arquitetura cliente/servidor convencional é modificada para suportar servidores imprecisos. Nesta proposta, implementada sobre o sistema operacional "Real-Time Mach", o cliente negocia com o servidor impreciso a qualidade do serviço.

5. Escalonamento de Tarefas Imprecisas

Toda proposta, dentro da Computação Imprecisa, necessita explicitar qual o objetivo a ser considerado no escalonamento das partes opcionais. Este objetivo será utilizado em caso de sobrecarga. Quando não é possível executar completamente todas as partes opcionais, é necessário algum critério para escolher quais partes opcionais terão seu tempo de execução sacrificado (parcialmente ou totalmente). Normalmente, este critério é definido a partir de uma medida do erro introduzido no sistema pelas tarefas tomadas individualmente.

5.1 Função Erro

Quando a parte opcional de uma tarefa é executada completamente, ela gera um resultado com qualidade máxima. Porém, quando esta mesma parte opcional não é

executada completamente, o resultado gerado possui uma qualidade inferior. Para efeitos de escalonamento, muitas propostas associam um valor de erro a cada parte opcional não executada completamente. Este valor de erro quantifica a diferença entre a qualidade do resultado preciso e a qualidade do resultado efetivamente gerado. É suposto que os erros associados com tarefas individuais contribuem, de alguma forma, para a redução da qualidade do sistema como um todo. Na literatura também pode ser encontrado o conceito de benefício gerado pela parte opcional. O *benefício* é definido com um sentido oposto ao do erro. Em outras palavras, uma parte opcional executada completamente gera um erro zero e um benefício máximo. Uma parte opcional completamente descartada gera um erro máximo e um benefício zero. Neste texto serão usados os dois termos, conforme o momento. É preciso definir como calcular este valor de erro. Na maioria das propostas encontradas na literatura este erro é suposto proporcional ao tempo de execução que faltou para concluir a parte opcional em questão. É suposta uma linha reta para a relação "erro da parte opcional" versus "tempo de execução", como ilustra o gráfico na figura 2.

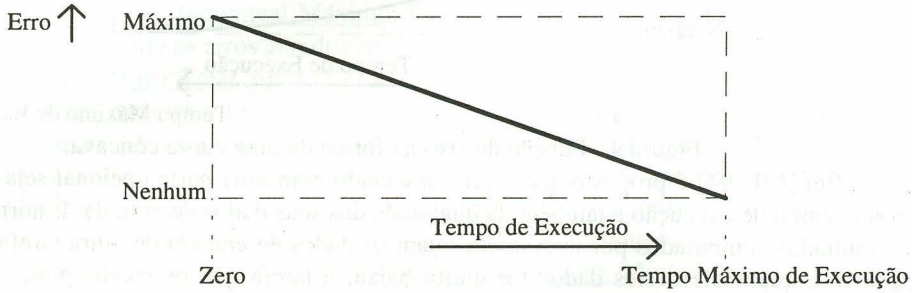

Figura 2 - Função de erro na forma de uma linha reta.

Quando a tarefa imprecisa apresenta uma restrição 0/1 (funções melhoramento e múltiplas versões) a parte opcional deve ser completamente executada ou então completamente descartada. Em outras palavras, não existe redução do erro quando a parte opcional é parcialmente executada. Neste caso, a função de erro adquire a forma ilustrada pela figura 3.

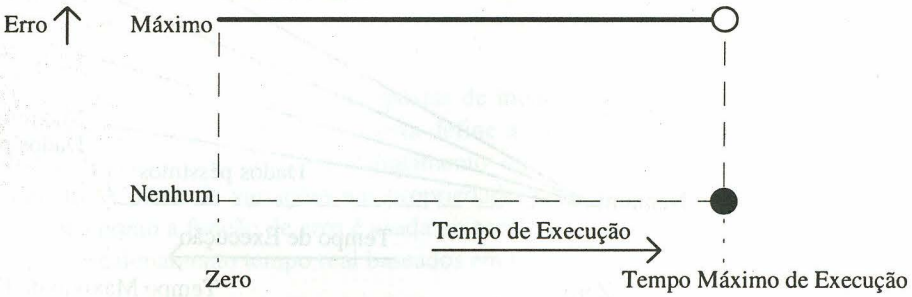

Figura 3 - Função de erro na forma de degrau.

É reconhecido que, na prática, a função de erro não tem sempre o formato de uma linha reta. Dependendo do algoritmo em questão, esta curva pode côncava, convexa ou ainda possuir um formato mais complexo. Por exemplo, um algoritmo para cálculo numérico que realiza iterações que convergem para o resultado, possivelmente possui uma curva côncava, como ilustra a figura 4. No início da execução do algoritmo, o erro do resultado diminui rapidamente, pois cada iteração representa um salto em direção ao resultado final. No final, com o resultado impreciso próximo do resultado exato, cada iteração faz apenas um pequeno refinamento.

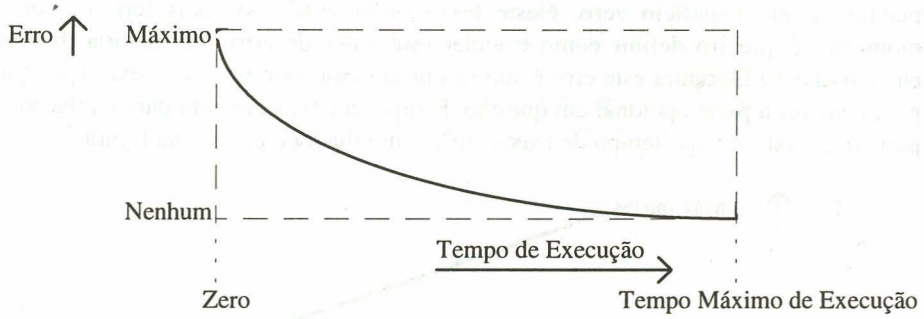

Figura 4 - Função de erro na forma de uma curva côncava.

Em [LIU 94] é proposto que o erro associado com uma parte opcional seja função do seu tempo de execução e também da qualidade dos seus dados de entrada. É normal que os resultados computados por uma tarefa sejam os dados de entrada de outra tarefa. Neste caso, se a qualidade destes dados for muito baixa, a tarefa que os recebe pouco poderá fazer. A figura 5 ilustra a combinação de tempo de execução e qualidade dos dados de entrada como determinantes do benefício gerado pela parte opcional. Uma outra possibilidade é descrita em [FEN 94], onde a qualidade dos dados de entrada não afeta diretamente a qualidade final mas sim o tempo de execução da tarefa. Neste caso, dados de entrada com melhor qualidade resultam em um tempo de execução menor.

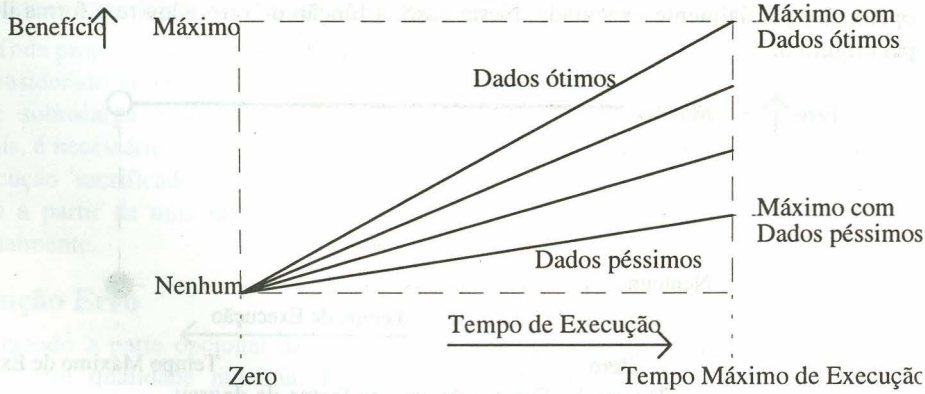

Figura 5 - Função de erro na forma de um conjunto de retas.

Na verdade, existem muito poucos trabalhos na bibliografia que não utilizam uma linha reta como função de erro.

5.2 Objetivo Global do Escalonamento

Existem na bibliografia diversas propostas para o uso da função de erro no escalonamento. Muitas vezes, os objetivos foram escolhidos mais pela sua elegância matemática do que a partir de um estudo aprofundado dos requisitos das aplicações tempo real. Não existem trabalhos na bibliografia discutindo, sob a ótica da engenharia de software, a utilidade ou aplicabilidade das funções de erro propostas e a forma como são empregadas no escalonamento das partes opcionais. Estes são os objetivos normalmente encontrados na literatura com relação ao uso da função de erro:

Minimiza Erro Total: Procura minimizar o erro total do sistema, definido como o somatório dos erros associados com as partes opcionais não executadas. Algumas propostas associam diferentes pesos às tarefas, os quais são multiplicados ao erro.

Minimiza Erro Individual Máximo: Procura minimizar o maior erro observado no sistema, considerando os erros gerados em cada ativação de tarefa individualmente.

Minimiza Erro Total Após Minimizar Erro Individual Máximo: Minimiza o erro total do sistema, após ter conseguido minimizar o erro individual máximo das tarefas.

Minimiza Erro Individual Máximo Após Minimizar Erro Total: Minimiza o erro individual máximo das tarefas do sistema, após ter conseguido minimizar o erro total.

Minimiza Número de Partes Opcionais Descartadas: Minimiza o número de partes opcionais não executadas completamente. Usada em conjunto com restrição 0/1.

Minimiza Erro Médio do Sistema (tarefas periódicas): Minimiza o erro médio do sistema, definido como o somatório dos erros médios de cada tarefa.

Minimiza Erro Individual Acumulado Máximo (tarefas periódicas): O erro associado com cada parte opcional em particular é acumulado (somado) pela tarefa até que a sua parte opcional seja completamente executada. Quando a sua parte opcional é completamente executada, o erro acumulado pela tarefa é zerado. O objetivo é minimizar o erro máximo acumulado por qualquer tarefa do sistema.

Escalona Conforme a Importância: Cada tarefa possui uma medida de importância relativa. Durante a execução, o objetivo é sacrificar antes as tarefas menos importantes.

6. Soluções de Escalonamento

Existe na bibliografia diversas propostas de modelos usando técnicas identificadas como Computação Imprecisa. Cada proposta define as propriedades do modelo de tarefas considerado, define o conceito de escalonamento ótimo e então propõe algoritmos de escalonamento. A partir de variações nas propriedades das tarefas, na definição da função erro e na forma como a função de erro é usada no escalonamento, é possível criar inúmeros problemas de escalonamento tempo real baseados em Computação Imprecisa.

A análise matemática dos algoritmos de escalonamento requer a formalização dos conceitos de tarefa, parte obrigatória, parte opcional, prazo, erro, etc. A maioria das propostas modelam cada tarefa T_i como a composição de uma parte obrigatória M_i e uma parte opcional O_i. Existe uma relação de precedência entre M_i e O_i. Cada tarefa T_i é caracterizada por um deadline e um *peso* para o sistema ("weight"). Os tempos de execução de M_i e O_i somados resultam no tempo máximo de execução de T_i. Nesta formalização é necessária uma estimativa para o tempo de execução de O_i, mesmo que esta seja exagerada. As formas básicas de programação são modeladas facilmente. No caso de multiplas versões, M_i representa a execução da versão secundária, enquanto M_i+O_i representa a execução da versão primária.

A parte obrigatória de uma tarefa deve ter seu deadline garantido. Para tanto, podem ser empregados, em tempo de projeto, algoritmos que trabalham com o pior caso e fornecem uma previsibilidade determinista, tais como o taxa monotônica e o deadline monotônico ([SHA 94]),[AUD 93]). Já as partes opcionais são escalonadas (ou não) visando maximizar a utilidade do sistema em tempo de execução ou, de forma equivalente, minimizar o erro gerado pela não execução de algumas partes opcionais.

O termo *escalonamento preciso* ("precise schedule") é usado para descrever uma solução de escalonamento onde as partes opcionais são completamente executadas. Desta forma, todas as tarefas sempre geram um resultado preciso. Já um *escalonamento satisfatório* ("feasible schedule") é aquele onde as tarefas sempre executam sua parte obrigatória, mas as partes opcionais podem ou não ser executadas completamente. Todo escalonamento preciso é também satisfatório, mas o recíproco não é verdadeiro.

6.1 Classificação das Propostas

Existem propostas que definem como *carga do sistema* um conjunto dinâmico de ativações de tarefas. Neste modelo de carga, tarefas imprecisas são dinamicamente ativadas, durante a execução do sistema. Em tempo de execução, o algoritmo de escalonamento procura aumentar o benefício gerado pelo sistema da melhor maneira possível. Como esta carga é potencialmente ilimitada, este algoritmo sozinho não consegue garantir que as partes obrigatórias de todas as tarefas serão executadas antes do respectivo deadline. Em função disto, as propostas que seguem esta linha consideram que as tarefas apresentadas ao escalonador satisfazem a restrição de que as partes obrigatórias são sempre escalonáveis ("feasible mandatory constraint"). Em outras palavras, é suposto que sempre é possível obter um escalonamento satisfatório. Existem dois caminhos para satisfazer a restrição de que partes obrigatórias são escalonáveis:

- A carga representada apenas pelas partes obrigatórias não nulas é, na realidade, limitada e conhecida antes da execução do sistema. Em tempo de projeto, um teste de escalonabilidade fornece a garantia de que todas as partes obrigatórias serão concluídas antes do respectivo deadline. Em tempo de execução, o escalonador trabalha como se a carga fosse completamente dinâmica, porém apresentando um comportamento que não compromete o resultado do teste de escalonabilidade.

• O escalonador rejeita tarefas imprecisas ativadas dinamicamente cuja parte obrigatória não pode ser executada dentro do deadline. Neste caso, a aplicação é obrigada a realizar algum tipo de tratamento de exceção em função das rejeições.

Existem propostas que consideram como carga do sistema um conjunto estático de tarefas periódicas ou esporádicas. Neste caso, a carga é limitada e conhecida, permitindo que todas as partes obrigatórias sejam garantidas em tempo de projeto. As propostas deste grupo fornecem uma solução completa para o escalonamento.

Algumas propostas mais antigas ([SHI 89], [LIU 91]), consideram que a carga do sistema é formada por um conjunto estático de ativações singulares (individuais), completamente conhecido antes da sua execução. Este grupo de propostas possui pouco valor prático. Seu mérito foi servir como base para vários outros modelos.

6.2 Propostas Limitadas às Partes Opcionais

Todas as propostas dentro desta classe trabalham a partir da suposição que partes obrigatórias são sempre escalonáveis ("feasible mandatory constraint").

Minimiza Erro Total [SHI 92]

Em [SHI 92] são apresentados três algoritmos para escalonar conjuntos dinâmicos de tarefas imprecisas de forma preemptiva em monoprocessador. Todas as tarefas possuem o mesmo peso e são independentes. Os algoritmos são ótimos no sentido que conseguem minimizar o erro total do sistema. Os algoritmos, baseados no EDF ("Earliest Deadline First"), são aplicáveis a modelos de tarefas levemente diferenciados com respeito a carga. São eles: *NORA* ("No-Off-line tasks and on-line tasks Ready upon Arrival") com complexidade O(N log N), *OAR* ("On-line tasks with Arbitrary Ready time") com complexidade $O(N \log^2 N)$ e *ORA* ("On-line tasks Ready upon Arrival") com complexidade O(N log N), onde **N** é o número de tarefas.

Minimiza Erro Total [HO 92a]

Em [HO 92a] é discutido o escalonamento preemptivo de tarefas imprecisas em monoprocessador. Tarefas são independentes com pesos diferentes e partes opcionais possuem restrição 0/1. Quando escalonamento ótimo é definido como aquele que minimiza o erro total do sistema, o problema torna-se NP-hard ([SHI 91]). Em [HO 92a] é feita uma redução deste problema a uma situação para a qual existe solução ótima na bibliografia com complexidade pseudo-polinomial $O(N^5.X^2)$, onde **N** é o número de tarefas e **X** é o somatório dos tempos de computação das partes opcionais. Também é fornecida uma heurística sub-ótima para este problema, com complexidade $O(N^2)$.

Minimiza Número de Partes Opcionais Descartadas [HO 92a]

Ainda em [HO 92a] é discutido o problema de minimizar o número de partes opcionais descartadas. Em [SHI 91] é mostrado que minimizar o descarte é polinomial se todas as partes opcionais tiverem o mesmo tempo de execução. Em [HO 92a] é feita uma redução deste problema a uma situação para a qual existe solução ótima com complexidade $O(N^7)$ para quando as partes opcionais possuem tempos de execução diferentes. Também é descrita uma heurística sub-ótima com complexidade $O(N^2)$.

Minimiza Erro Individual Máximo [HO 94]

Em [HO 94] é apresentado um algoritmo preemptivo para escalonar conjuntos de tarefas imprecisas em multiprocessadores. As tarefas são indepedentes e possuem pesos diferentes. O algoritmo apresentado é ótimo no sentido que ele minimiza o erro individual máximo das tarefas. Sua complexidade computacional é $O(N^2)$ para monoprocessadores e $O(N^3.Log^2\ N)$ para multiprocessadores homogêneos.

Minimiza Erro Individual Máximo e então o Erro Total [HO 94]

O artigo [HO 94] também apresenta um algoritmo ótimo no sentido que ele minimiza o erro total, ponderado pela importância de cada tarefa, após ter minimizado o erro individual máximo. Entre todos os escalonamentos possíveis que possuem um erro individual máximo minimizado, este algoritmo encontra um escalonamento com erro total mínimo. Ele apresenta complexidade $O(N^2)$ para monoprocessadores e complexidade $O(N^3.Log^2\ N)$ para multiprocessadores homogêneos.

Minimiza Erro Total e então o Erro Individual Máximo [HO 92b]

Em [HO 92b] é apresentado um algoritmo semelhante ao anterior. Ele é ótimo no sentido que minimiza o erro individual máximo, após ter minimizado o erro total ponderado do sistema. Sua complexidade computacional é $O(K.N^2)$ para monoprocessadores e $O(K.N^3.Log^2\ N)$ para multiprocessadores homogêneos, onde **K** representa o número de pesos diferentes existentes.

6.3 Propostas Completas

Abaixo estão algumas propostas completas para o escalonamento de tarefas imprecisas, isto é, consideram tanto partes obrigatórias quanto partes opcionais.

Minimiza Erro Médio Não Acumulativo [CHU 90]

Em [CHU 90] é discutido o problema de escalonar tarefas imprecisas independentes que são periódicas. A qualidade do resultado gerado por uma tarefa é medida em termos do erro médio da tarefa percebido ao longo de vários períodos consecutivos. Somente o efeito médio dos erros é observável e relevante. Um exemplo desta situação ocorre em tarefas que recebem, melhoram e transmitem quadros ("frames") de imagens de vídeo. Embora a tarefa seja executada periodicamente, os erros gerados em diferentes ativações são independentes entre si. Em [CHU 90] são utilizadas prioridades preemptivas no escalonamento. As partes obrigatórias possuem prioridades superiores a todas as partes opcionais. Cinco diferentes heurísticas são propostas para definir as prioridades das partes opcionais em tempo de execução. Nenhuma delas é ótima no sentido de que obtém o erro médio global mínimo. Fica claro também que o formato da função de erro é crucial na escolha da heurística a ser empregada.

Minimiza Erro Acumulado Máximo [CHU 90]

Ainda em [CHU 90] são consideradas tarefas com erro cumulativo. Um exemplo seria uma tarefa que processa o sinal de radar retornado por um alvo em movimento. Quando a tarefa não é completamente executada, ela gera uma estimativa grosseira da

posição. É necessário que, ocasionalmente, a tarefa seja completamente executada, pois uma sequência de resultados imprecisos pode fazer o sistema perder o alvo de vista. O modelo considera conjuntos de tarefas independentes periódicas em um monoprocessador. O objetivo do algoritmo é garantir os prazos das partes obrigatórias e providenciar para que cada parte opcional consiga executar completamente antes de passarem **Q** períodos. O artigo descreve uma heurística com resultados sub-ótimos.

Escalona Conforme Importância [AUD 91]

Em [AUD 91], o escalonamento é preemptivo e baseado em prioridades fixas. Um teste de escalonabilidade é usado para verificar se os prazos das partes obrigatórias serão cumpridos. Tarefas são independentes e periódicas ou esporádicas com intervalo mínimo entre ativações. As partes opcionais são ignoradas pelo teste de escalonabilidade em tempo de projeto, pois sua execução não é garantida. Entretanto, partes opcionais também recebem prioridades fixas, menores que as das partes obrigatórias. Estas prioridades são baseadas em uma estimativa da importância da tarefa para o sistema.

Minimiza Erro Total em Tempo de Projeto [YU 92]

Em [YU 92] é apresentada uma solução para o problema de alocação e escalonamento de tarefas imprecisas replicadas em um multiprocessador. O objetivo desta proposta é tolerar falhas no hardware, supondo que o software é livre de erros. Quando um processador falha, ocorre uma redução dos recursos disponíveis. O sistema reage com uma redução no tempo de execução das partes opcionais e uma redução no número de réplicas das tarefas. Em tempo de projeto, heurísticas são empregadas para alocar as réplicas aos processadores e definir qual deverá ser o tempo de processador alocado para cada uma, em função do número de processadores em funcionamento. Esta flexibilidade na determinação do tempo de execução das tarefas está ligada à existência de uma parte opcional em cada tarefa. É importante notar que, nesta proposta, o tempo de processador que cada parte opcional efetivamente recebe é definido ainda em projeto.

Maximiza Benefício Total [DAV 95]

Em [DAV 95] é descrita uma solução para escalonar partes opcionais de tarefas imprecisas com restrição 0/1. São consideradas tarefas independentes com prioridades fixas. Em tempo de projeto é verificada a escalonabilidade das partes obrigatórias. Em tempo de execução um algoritmo com complexidade $O(N^2)$ identifica folgas na demanda por processador que podem ser usadas para executar partes opcionais. Uma heurística sub-ótima com complexidade O(N) seleciona partes opcionais para execução. É suposto que cada parte opcional executada contribui com uma quantidade de valor para o sistema. O objetivo é maximizar o valor total do sistema.

Maximiza Benefício Total [OLI 96]

Em [OLI 96] é empregada a mesma abordagem de [DAV 95]. São descritas duas heurísticas sub-ótimas para selecionar partes opcionais, com complexidade O(N) e $O(N^2)$. Estas duas heurísticas foram otimizadas para a situação na qual tarefas possuem dependencias intra-tarefa e inter-tarefas, respectivamente. Dependência intra-tarefa acontece quando uma execução imprecisa aumenta o valor de uma execução precisa da

mesma tarefa no futuro. Dependência inter-tarefas acontece quando a precisão nos dados de entrada de uma tarefa afeta o seu tempo máximo de execução.

6.4 Ambiente Distribuído

Existem poucos trabalhos na literatura analisando o emprego de Computação Imprecisa em ambientes distribuídos. Em [KOP 89] uma tarefa possui versão primária e versão secundária. A versão primária fornece resultado com qualidade máxima, mas não é garantida. A versão secundária fornece resultado minimamente aceitável, mas é garantida. As duas versões executam paralelamente em processadores diferentes. Se a versão primária termina dentro do prazo, seus resultados são utilizados. Caso contrário, são utilizados os resultados da versão secundária.

Em [OLI 97] é descrita uma abordagem completa para escalonar tarefas imprecisas em ambiente distribuído. A proposta emprega prioridades fixas no escalonamento das tarefas e supõe que múltiplas versões são usadas na programação. O modelo de tarefas empregado inclui tarefas periódicas ou esporádicas, com deadline menor que o respectivo período e relações de precedência entre tarefas. A figura 6 ilustra a abordagem, composta por quatro algoritmos.

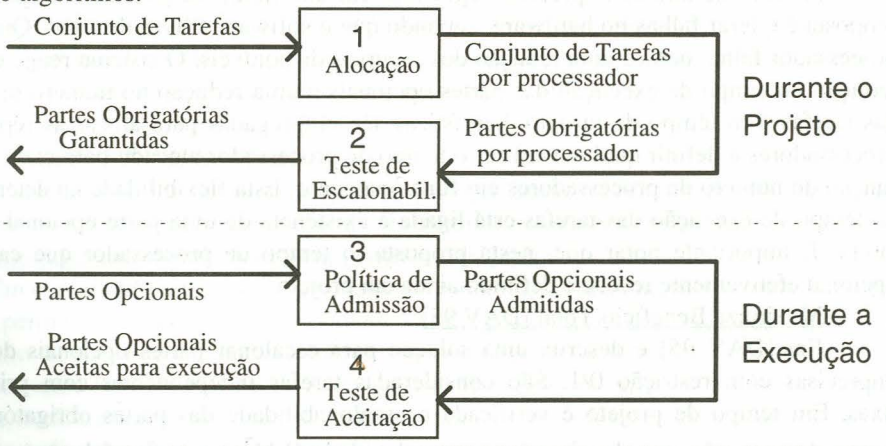

Figura 6 - Abordagem proposta em [OLI 97] para ambiente distribuído.

O primeiro algoritmo corresponde ao processo de alocação das tarefas aos processadores. Como uma alocação de sucesso exige que todas as partes obrigatórias sejam garantidas, o algoritmo de alocação deve usar o segundo algoritmo para verificar se este objetivo foi atendido. Ao mesmo tempo, o algoritmo de alocação procura obter um balanceamento de carga de tal sorte que as chances de uma parte opcional executar sejam ampliadas. O terceiro algoritmo avalia, em tempo de execução, se uma dada parte opcional deve ser ou não considerada para execução. A idéia aqui é implementar uma política de admissão que descarta partes opcionais cuja importância para o sistema é muito baixa. O quarto algoritmo avalia se uma dada parte opcional, previamente admitida pelo terceiro

algoritmo, pode ser aceita para execução. Uma parte opcional somente poderá ser aceita para execução se não comprometer as garantias fornecidas anteriormente.

7. Conclusões

Com respeito às aplicações tempo real em geral, especialmente aplicações encontradas na automação industrial, o mais razoável é esperar conjuntos mistos de tarefas. Aplicações industriais nem sempre apresentam tarefas críticas com respeito à segurança ("safety-critical"). Mas quase sempre existirão tarefas críticas à missão. O usuário da aplicação certamente gostaria de garantias, em tempo de projeto, para as tarefas críticas à missão, mesmo que esta garantia signifique um gasto adicional de recursos. Entretanto, para a maioria das tarefas do sistema, provavelmente uma abordagem tipo melhor esforço ("best-effort") seria suficiente.

Existe um dilema na área de escalonamento tempo real, entre ter garantia para o prazo das tarefas ou ter adaptabilidade e eficiência no uso dos recursos. Se o aspecto flexibilidade na carga for ignorado, sempre é possível tratar todas as tarefas no sentido "garantia". Mas esta abordagem muitas vezes é inviável, tal o volume de recursos necessários para o pior caso de todas as tarefas. Ela é viável apenas em sistemas muito pequenos ou em sistemas críticos ao ponto de justificar todos os recursos gastos. Por outro lado, se a necessidade de garantia for ignorada, é possível tratar todas as tarefas no sentido "melhor esforço". Mas isto significa não ter qualquer garantia temporal para a aplicação. Estas considerações são válidas para os contextos centralizado e distribuído.

A técnica Computação Imprecisa oferece uma das possíveis soluções de compromisso para este dilema. Na medida em que aplicações tempo real envolvem aspectos críticos juntamente com diversos aspectos não críticos, a técnica de Computação Imprecisa poderá ser um caminho atraente para a construção de sistemas tempo real. Entretanto, os trabalhos sobre Computação Imprecisa publicados tratam, em geral, apenas do problema de escalonamento local. Mesmo assim, somente empregando modelos de tarefas relativamente simples. Existem atualmente diversas questões em aberto, com respeito à Computação Imprecisa:

- Investigação de métodos para a definição da importância ("weight") das tarefas em tempo de projeto, a partir da especificação do sistema.
- Incorporação da técnica de Computação Imprecisa às técnicas de construção de software existentes hoje. Em particular, técnicas orientadas a objeto.
- Desenvolvimento de linguagens de programação apropriadas para a Computação Imprecisa, considerando-se suas diferentes formas de programação. Em particular, o emprego de Reflexão Computacional ([LIS 96]) como técnica capaz de implementar a escolha de partes opcionais em tempo de execução.
- Incorporação de exclusão mútua aos modelos de tarefas empregados e respectiva solução a nível de algoritmo de escalonamento.
- Investigação de funções de erro mais realistas, provavelmente associadas com determinadas classes de aplicações. Em muitas classes de aplicações, tais como vídeo e

gráficos para animação, os efeitos da imprecisão são subjetivos. Neste caso, o próprio objetivo do escalonamento deve ser determinado de forma empírica.

• Aproveitamento da distribuição na própria técnica, através da execução paralela das partes obrigatória e opcional. Provável integração da Computação Imprecisa com outras técnicas de tolerância a falhas.

O principal obstáculo hoje à utilização da técnica de Computação Imprecisa em aplicações de tempo real é o fato das técnicas tradicionais de engenharia de software não incorporarem o conceito de execução imprecisa. Neste sentido, existe muito trabalho a ser feito na área de Engenharia de Software. Uma das poucas ferramentas existentes hoje que suporta esté conceito é o PERTS, descrito em [LIU 93]. Como colocado em [AUD 91], "o uso disseminado de técnicas, tais como Computação Imprecisa, somente acontecerá se elas forem integradas aos métodos usuais de engenharia de software".

Referências

[AUD 91] N. C. Audsley, A. Burns, M. F. Richardson, A. J. Wellings. Incorporating Unbounded Algorithms into Predictable Real-Time Systems. Technical Report RTRG/91/102. University of York, Department of Computer Science, sep 1991.

[AUD 93] N. C. Audsley, A. Burns, A. J. Wellings. Deadline Monotonic Scheduling Theory and Application. Control Engineering Practice, Vol. 1, No.1, pp.71-78, feb 1993.

[BUR 91] A. Burns, A. J. Wellings. Criticality and Utility in the Next Generation. The Journal of Real-Time Systems, Vol. 3, correspondence, pp. 351-354, 1991.

[CHU 90] J. Y. Chung, J. W. S. Liu, K. J. Lin. Scheduling Periodic Jobs that Allow Imprecise Results. IEEE Trans. on Computers, Vol. 39, No. 9, pp.1156-1174, sep 1990.

[DAM 89] A. Damm, J. Reisinger, W. Schwabl, H. Kopetz. The Real-Time Operating System of Mars. ACM Operating Systems Review, Vol. 23, No. 3, pp.141-151, 1989.

[DAV 95] R. Davis, S. Punnekkat, N. Audsley, A. Burns. Flexible Scheduling for Adaptable Real-Time Systems. Proc. of the IEEE Real-Time Technology and Applications Symposium, pp. 230-239, may 1995.

[FEN 94] W. Feng, J. W.-S. Liu. Algorithms for Scheduling Tasks with Input Error and End-to-End Deadlines. Un. of Ill. at Urbana-Champaign, Department of Computer Science, Technical Report #1888, 1994.

[FEN 96] W. Feng, J. W.-S. Liu. Performance of a Congestion Control Scheme on an ATM Switch. Proceedings of the International Conference on Networks, Orlando, Florida, january 1996.

[FOR 96] A. Garcia-Fornes, H. Hassan, A. Crespo. Strategies for Scheduling Optional Tasks in Intelligent Real-Time Environments. Journal of Systems Architecture, 42 (1996), pp. 391-407.

[GAR 94] A. Garvey, V. Lesser. A Survey of Research in Deliberative Real-Time Artificial Intelligence. Real-Time Systems, Vol. 6, pp. 317-347, 1994.

[HO 92a] K. I.-J. Ho, J. Y.-T. Leung, W.-D. Wei. Scheduling Imprecise Computation Tasks with 0/1-Constraint. Tech. Rep., Dep. of Comp. Science and Engineering, University of Nebraska-Lincoln, 1992.

[HO 92b] K. I.-J. Ho, J. Y.-T. Leung, W.-D. Wei. Minimizing Constrained Maximum Weighted Error for Doubly Weighted Tasks. Tech. Rep., Dep. of Computer Science and Engineering, Un. of Nebraska, 1992.

[HO 94] K. I.-J. Ho, J. Y.-T. Leung, W.-D. Wei. Minimizing Maximum Weighted Error for Imprecise Computation Tasks. Journal of Algorithms, 16, pp. 431-452, 1994.

[HUA 95] X. Huang, A. M. K. Cheng. Applying Imprecise Algorithms to Real-Time Image and Video Transmission. Proceedings of the IEEE Workshop on Real-Time Applications. Chicago, may 1995.

[HUL 95] D. L. Hull, W. Feng, J. W.-S. Liu. Enhancing the Performance and Dependability of real-Time Systems. AAAI Fall Symposium on Flexible Computation, Cambridge-Massachusetts, november 1996.

[HUL 96] D. Hull, W.-C. Feng, J. W.-S. Liu. Operating System Support for Imprecise Computation. AAAI Fall Symposium on Flexible Computation, Cambridge-Massachusetts, november 1996.

[JEN 85] E. D. Jensen, C. D. Locke, H. Tokuda. A Time-Driven Scheduling Model for Real-Time Operating Systems. Proceedings of the Real-Time Systems Symposium, pp.112-122, december 1985.

[KEN 91] K. B. Kenny, K.-J. Lin. Building Flexible Real-Time Systems Using the Flex Language. IEEE Computer, pp.70-78, may 1991.

[KOP 89] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabi, C. Senft, R. Zainlinger. Distributed Fault-Tolerant Real-Time Systems: The Mars Approach. IEEE Micro, pp. 25-40, feb. 1989.

[KUH 95] J. Kuhl, D. Evans, Y. Papelis, R. Romano, G. Watson. The Iowa Driving Simulator: An Immersive Research Environment. IEEE Computer, pp. 35-41, july 1995.

[LIN 87a] K. J. Lin, S. Natarajan, J. W. S. Liu. Imprecise Results: Utilizing Partial Computations in Real-Time Systems. In Proc. IEEE 8th Real-Time Systems Symp., San Jose, California, USA, Dec. 1987.

[LIN 87b] K. J. Lin, S. Natarajan, J. W. S. Liu. Scheduling Real-Time, Periodic Jobs Using Imprecise Results. In Proc. IEEE 8th Real-Time Systems Symposium, San Jose, California, USA, Dec. 1987.

[LIS 96] M. L. B. Lisboa, C. M.F. Rubira, L. E. Buzato. Arquitetura Reflexiva para o Desenvolvimento de Software Tolerante a Falhas. XXIII SEMISH, pp.155-166, Recife-PE, 1996.

[LIU 88] J. W. S. Liu, K.-J. Lin. On Means to Provide Flexibility in Scheduling. Ada Letters Special, Vol.VIII, No.7, pp.32-34, 1988.

[LIU 91] J. W. S. Liu, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, J.-Y. Chung, W. Zhao. Algorithms for Scheduling Imprecise Computations. IEEE Computer, may 1991.

[LIU 93] J. W. S. Liu, J.-L. Redondo, Z. Deng, T.-S. Tia, R. Bettati, A. Silberman, M. Storch, R. Ha, W.-K. Shih. PERTS: A Prototyping Environment for Real-Time Systems. Technical Report, Dept. of Computer Science, University of Illinois, may 1993.

[LIU 94] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, J.-Y. Chung. Imprecise Computations. Proceedings of the IEEE, Vol. 82, No. 1, pp. 83-94, january 1994.

[MIL 94] V. Millan-Lopez, W. Feng, J. W.-S. Liu. Using the Imprecise-Computation Technique for Congestion Control on a Real-Time Traffic Switching Element. Proc. of the Int. Conf. on Parallel & Distributed Systems, Taiwan, dec. 1994.