Escalonamento de Tarefas Imprecisas em Ambiente Tempo-Real Distribuído

Rômulo Silva Oliveira

Instituto de Informática
Univ. Fed. do Rio Grande do Sul
Caixa Postal 15064
Porto Alegre-RS, 91501-970
romulo@inf.ufrgs.br

Joni da Silva Fraga

Lab. de Controle e Microinformática
Univ. Fed. de Santa Catarina
Caixa Postal 476
Florianópolis-SC, CEP 88040-900
fraga@lcmi.ufsc.br

Resumo

A técnica de Computação Imprecisa tem sido proposta como uma abordagem para o problema do escalonamento de sistemas de tempo-real capaz de prover garantia e flexibilidade. Este trabalho analisa o emprego de Computação Imprecisa no escalonamento de aplicações tempo-real distribuidas. Inicialmente são discutidos os problemas associados com este objetivo. Um modelo de tarefas para ambientes distribuídos é definido. É apresentada uma abordagem para o escalonamento de tarefas imprecisas distribuidas. Finalmente, algoritmos de escalonamento são apresentados.

Abstract

The Imprecise Computation technique has been proposed as an approach to the scheduling of Real-Time Systems that is able to provide both guarantee and flexibility. This paper analyzes the use of Imprecise Computation in the scheduling of distributed real-time applications. We first discuss problems associated with that goal. It is defined a task model that is suitable to distributed environments. An approach to the scheduling of distributed imprecise tasks is showed. Finally, scheduling algorithms are presented.

1. Introdução

Sistemas computacionais usados em aplicações de tempo real, além dos aspectos funcionais, estão submetidos a requisitos de natureza temporal. Nestes sistemas os resultados fornecidos devem ser corretos não somente do ponto de vista lógico, mas também devem ser gerados de modo a atender os requisitos de tempo. O não atendimento de deadlines provoca falhas de natureza temporal nesses sistemas que, em alguns casos, são consideradas críticas no que diz respeito às suas consequências.

Este trabalho trata de Sistemas Distribuídos de Tempo Real. Um problema básico encontrado na construção deste tipo de sistema é a alocação e o escalonamento das tarefas nos recursos computacionais disponíveis. Em sistemas de tempo real existe uma dificuldade intrínsica 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.

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 que todas as tarefas serão concluídas no momento correto. Entretanto, aplicações construidas desta forma resultam em sistemas pouco flexíveis e na subutilização dos recursos computacionais. Exemplos desta abordagem são encontrados em [XU 93]. 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 a classe de aplicações com requisitos temporais críticos. Exemplos dessa última abordagem podem ser encontrados em [RAM 89].

Na Computação Imprecisa ([LIU 94]) cada tarefa da aplicação 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 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. Esta técnica procura, de certa forma, conciliar os dois objetivos fundamentais citados antes.

Computação Imprecisa pode ainda ser vista como um mecanismo de tolerância a faltas temporais. Mais exatamente, como um mecanismo capaz de tolerar sobrecargas no sistema sem comprometer seus requisitos temporais críticos. A não conclusão de uma tarefa completa antes do seu respectivo deadline é uma falta tolerada, na medida que sua parte obrigatória é sempre executada antes do deadline. Neste caso, temos uma degradação graciosa do sistema na medida em que sua qualidade é reduzida.

Poucos trabalhos existentes na literatura tratam de Computação Imprecisa em ambiente distribuído. Quando o fazem ([YU 92], [HUA 95], [FEN 96]), abordam questões localizadas e casos particulares. Não existe na literatura um estudo amplo sobre a questão de "como resolver o problema do escalonamento quando sistemas de tempo real distribuídos são construidos a partir do conceito de Computação Imprecisa".

O objetivo geral deste artigo é mostrar como aplicações de tempo real, construídas a partir do conceito de Computação Imprecisa, podem ser escalonadas em ambiente distribuído. Em outras palavras, mostrar que o conceito de Computação Iimprecisa pode ser adaptado para um ambiente onde tarefas executam em diferentes processadores e a comunicação entre elas é implementada através de mensagens.

Neste artigo é inicialmente discutida a problemática do escalonamento de tarefas imprecisas em ambiente tempo real distribuído. Em seguida, é descrito o modelo de tarefas adotado. Uma abordagem apropriada para o problema é apresentada. Finalmente, são indicados algoritmos de escalonamento apropriados para o contexto em questão.

2. Discussão da Problemática

A problemática do escalonamento tempo real de tarefas imprecisas em ambiente distribuído envolve diversos aspectos. A discussão está dividida em quatro aspectos.

Alocação

No contexto de sistemas distribuídos, o escalonamento tempo real é geralmente resolvido em duas etapas. Na primeira etapa é feita a alocação das tarefas aos processadores. Na segunda etapa é feito o escalonamento local de cada processador, tomado individualmente. Este escalonamento local considera como carga as tarefas alocadas na primeira etapa. Em aplicações de tempo real geralmente não existe migração de tarefas em tempo de execução, o que demandaria recursos tanto de processamento como de comunicação. Como cada tarefa é alocada permanentemente a um processador, aumenta a importância de uma alocação adequada.

Em função da complexidade do problema, as soluções propostas na literatura tendem a ser subótimas. É possível dividir as soluções existentes em duas linhas gerais: pesquisa heurística e pesquisa aleatória (ou algoritmos de otimização). Determinados problemas permitem a construção de boas heurísticas, que conduzem rapidamente a uma boa solução. Nas situações onde o problema de alocação se torna mais complexo, fica difícil criar heurísticas que capturem todas as sutilezas e implicações das decisões tomadas. Pesquisa aleatória passa a ser atrativa com os seus algoritmos de otimização.

O objetivo primário da alocação é garantir que todas as tarefas sempre poderão concluir suas respectivas partes obrigatórias antes do deadline. Para isto, o algoritmo de alocação propõe diversas soluções de alocação, verificando para cada uma delas a escalonabilidade das partes obrigatórias. Com respeito ao objetivo primário, qualquer solução de alocação que permita uma garantia em projeto para as partes obrigatórias das tarefas imprecisas é igualmente satisfatória.

A alocação possui como objetivo secundário aumentar as chances das partes opcionais das tarefas serem executadas. Considere duas soluções de alocação X e Y, igualmente satisfatórias do ponto de vista do objetivo primário, isto é, partes obrigatórias podem ser garantidas tanto em X como em Y. É possível que a alocação X seja melhor balanceada com respeito a alocação Y, de forma que partes opcionais possam ser mais executadas em X do que em Y. Logo, do ponto de vista do objetivo secundário, a alocação X será superior à alocação Y.

Garantia para as Partes Obrigatórias

Dentro do contexto da Computação Imprecisa, é necessário ainda em tempo de projeto garantir que cada parte obrigatória será concluída dentro do seu respectivo deadline. Ao tratar deste aspecto, é possível ignorar a existência das partes opcionais e considerar cada tarefa formada exclusivamente por sua parte obrigatória.

Existem na literatura inúmeras soluções de escalonamento para oferecer garantia quando um conjunto de tarefas executa em um único processador. Para sistemas distribuídos os trabalhos existentes são em muito menor número, em função dos problemas adicionais encontrados. Uma previsibilidade determinista para as partes obrigatórias pode ser obtida, entre outras formas, através de escalonamentos baseados em prioridades fixas ([LIU 73]). Nesta abordagem, tarefas recebem prioridades segundo alguma política específica e um teste de escalonabilidade é executado em tempo de projeto, determinando se existe a garantia de que todas as tarefas serão executadas dentro dos deadlines. O teste de escalonabilidade deve ser compatível com a política de

atribuição de prioridades adotada. Em tempo de execução, um escalonador preemptivo produz a escala de execução usando as prioridades das tarefas.

Em sistemas distribuídos também é comum o surgimento de relações de precedência entre tarefas. Uma relação de precedência tem origem em uma necessidade de sincronização e/ou passagem de dados entre duas tarefas da aplicação. Uma mensagem cria uma relação de dependência entre a tarefa remetente e a tarefa destinatária. A tarefa destinatária somente pode iniciar sua execução após receber a mensagem. Embora isto ocorra também em sistemas centralizados, o fato das tarefas estarem distribuídas por diferentes processadores dificulta o escalonamento no sistema. Qualquer análise de escalonabilidade para ambiente distribuído deve ser capaz de lidar com relações de precedência.

É possível empregar deslocamentos temporais ("offsets", [AUD 93a]) para implementar relações de precedência entre tarefas. Ao estabelecer um deslocamento temporal ("offset") entre as liberações de duas tarefas, é possível garantir que a tarefa sucessora somente iniciará sua execução após o término da tarefa predecessora. Basta que este deslocamento temporal seja maior ou igual ao tempo máximo de resposta da tarefa predecessora. No caso de haver passagem de dados, no instante que a tarefa sucessora é liberada é garantido que os dados já foram colocados no lugar apropriado pela tarefa predecessora. Esta técnica é algumas vezes chamada de liberação estática de tarefas ("statically released tasks", [SUN 95]), pois o instante de liberação das tarefas é previamente fixado. O sistema original é transformado em um sistema equivalente onde as tarefas são independentes mas apresentam um deslocamento entre si. Testes de escalonabilidade são então aplicados a este sistema equivalente.

Também é possível implementar relações de precedência através do envio explícito de uma mensagem da tarefa predecessora para a tarefa sucessora. Esta mensagem informa que a tarefa predecessora foi concluída e a tarefa sucessora pode ser liberada. A mensagem pode também conter os dados a serem transmitidos de uma tarefa para a outra. Esta técnica é algumas vezes chamada de liberação dinâmica de tarefas ("dynamically released tasks", [SUN 95]), pois o instante de liberação da tarefa sucessora depende do instante de conclusão da tarefa predecessora, o qual somente é conhecido em tempo de execução. A incerteza a respeito do instante de liberação da tarefa sucessora pode ser modelada como um jitter de liberação ("release jitter", [TIN 94]). Para fins de análise, o sistema original é transformado em um sistema equivalente onde as tarefas são independentes mas apresentam uma variação no instante de liberação. Novamente, testes de escalonabilidade são aplicados ao sistema equivalente.

Uma necessidade básica neste contexto são protocolos de comunicação com atraso limitado. Protocolos de comunicação tempo real fogem do escopo deste trabalho e não serão abordados. Neste trabalho é suposta a existência de um protocolo capaz de garantir o envio de uma mensagem entre dois processadores com um atraso limitado.

Identificação de Folgas na Escala de Execução

Garantir, em tempo de projeto, que todas as partes obrigatórias serão concluidas antes do respectivo deadline implica em reservar recursos para o pior caso. Pior caso aqui se refere tanto aos tempos de execução quanto à pior combinação possível de liberações de tarefas. Em tempo de execução, existem vários fatores capazes de gerar folgas na utilização dos recursos. Estas folgas são usadas para executar partes opcionais.

O tempo de processador que sobra após a reserva de recursos, para o pior caso de todas as partes obrigatórias, é chamado de capacidade restante. Esta capacidade restante é normalmente medida em termos de utilização do processador. Ela pode ser dedicada à execução de partes opcionais sem prejuizo para as partes obrigatórias. Sua grandeza, em termos de utilização do processador, já é conhecida em tempo de projeto.

A probabilidade do pior caso acontecer é, em geral, muito pequena. Na maior parte das vezes, as partes obrigatórias podem ser concluidas antes do deadline com menos recursos do que foi reservado em tempo de projeto. Isto acontece quando uma determinada ativação de tarefa ocupa menos tempo de processador do que o pior caso previsto em projeto. O tempo de processador que foi reservado em tempo de projeto mas não utilizado na execução é chamado de tempo ganho ("gain time"). Sua grandeza e instante de ocorrência somente é conhecida em tempo de execução.

É também possível que uma dada tarefa seja ativada com uma frequência menor do que a frequência máxima estabelecida em projeto. Desta forma, a parte obrigatória desta tarefa vai consumir menos recursos do que no pior caso. Este tempo de processador reservado mas não utilizado em função de uma menor frequência de ativação também pode ser usado na execução de partes opcionais. Sua grandeza e instante de ocorrência somente é conhecida em tempo de execução.

Os testes de escalonabilidade aplicados em tempo de projeto sempre consideram, na sua análise, a pior combinação possível para a liberação das tarefas. Por exemplo, os testes baseados no conceito do instante crítico reservam recursos para quando todas as tarefas são liberadas simultaneamente. Sempre que as tarefas não forem liberadas simultaneamente, haverá uma sobra de recursos em tempo de execução. Isto acontece com frequência devido às tarefas esporádicas, às tarefas periódicas com períodos diferentes entre si, ao jitter de liberação e às relações de precedência. Nestas situações, mais recursos estarão sendo liberados para a execução de partes opcionais.

O somatório de todas estas fontes de recurso resulta na capacidade ociosa do sistema ("spare capacity"). Esta capacidade ociosa deve ser detectada e usada para a execução das partes opcionais. É necessário que as partes opcionais executadas ocupem somente a capacidade ociosa, para não comprometer a execução das partes obrigatórias previamente garantidas. Os algoritmos para a detecção e o aproveitamento da capacidade ociosa tendem a ser simples e pouco eficientes ou então eficientes porém complexos. Este problema é agravado pelo fato dos algoritmos serem empregados em tempo de execução, disputando recursos (processador) com as próprias partes opcionais.

Escolha das Partes Opcionais para Execução

Muitas vezes a folga identificada na escala de execução não é suficiente para executar todas as partes opcionais do sistema. Desta forma, é necessário escolher quais das partes opcionais disponíveis para execução em um dado momento deverão ser efetivamente executadas e quais deverão ser descartadas. Esta escolha deve levar em consideração a importância de cada tarefa para o sistema.

Em muitos sistemas a importância de executar precisamente uma determinada tarefa depende do comportamento passado do sistema. Uma situação comum são tarefas periódicas onde a importância de uma execução precisa aumenta na medida que a tarefa é executada imprecisamente. Neste tipo de tarefa a imprecisão é, de certa maneira,

cumulativa. É desejável que a tarefa seja algumas vezes executada de forma precisa para interromper o acumulo de imprecisão. Este tipo de dependência é chamada de intra-tarefa ([OLI 96]). Exemplos de tarefas com dependências intra-tarefa aparecem em aplicações tais como sistemas de radar e sistemas de controle ([CHU 90]).

Tarefas que apresentam uma relação de produtor e consumidor também apresentam uma dependência com respeito a importância de uma execução precisa. A tarefa produtora gera como saída dados que serão usados como entrada pela tarefa consumidora. Muitos algoritmos apresentam um menor tempo de execução quando os dados recebidos são de melhor qualidade. Desta forma, ao executar precisamente a tarefa produtora estaremos diminuindo o tempo de execução da tarefa consumidora. Este "tempo extra de processador" que sobra poderá ser utilizado então para executar outras partes opcionais, aumentando a utilidade do sistema. Este tipo de dependência é chamada de inter-tarefa ([OLI 96]). Exemplos de tarefas com dependências inter-tarefa aparecem em aplicações tais como reconhecimento de voz, sistemas de radar e processamento de imagem para navegação autônoma ([FEN 94]).

A maioria dos estudos de computação imprecisa considera que o valor/erro de uma tarefa é definido no seu intante de chegada. É suposto que ela não é afetada pelo que acontece a outras tarefas ou a liberações prévias da mesma tarefa. Os trabalhos [FEN 94] e [CHU 90] constituem uma exceção. Em [FEN 94] o modelo de tarefas inclui dependências inter-tarefa. Erros nos dados de entrada de uma tarefa podem estender o tempo máximo de execução de suas próprias partes obrigatórias e/ou opcionais. Em [CHU 90] o modelo de tarefas inclui dependências intra-tarefa; é suposto que todas as tarefas devem ser executadas precisamente pelo menos uma vez a cada certo número de liberações. Nos dois trabalhos citados o modelo de erro é tal que uma parte opcional poderia eventualmente tornar-se obrigatória ou aumentar o tempo global de execução obrigatória do sistema, perdendo assim seu carater "opcional".

Algoritmos para escolha de partes opcionais devem contemplar os aspectos descritos acima. Em particular, as dependências intra-tarefa e inter-tarefa devem ser consideradas no momento de comparar a importância para o sistema da execução precisa de diferentes tarefas. Como tais algoritmos são executados on-line, é importante que seu custo computacional seja baixo.

3. Modelo Básico de Tarefas Adotado

Esta seção descreve o modelo básico de tarefas que foi adotado no trabalho. São descritas as propriedades usuais associadas com tarefas em sistemas de tempo real. A apresentação neste ítem procura clareza e simplicidade na discussão das características do modelo. Um tratamento mais formal pode ser encontrado em [OLI 97].

O modelo de tarefas empregado supõe a execução da aplicação em um sistema distribuído formado por um conjunto **H** composto por **h** processadores, conectados através de um meio de comunicação. O envio de mensagens entre dois processadores quaisquer (comunicação remota) é delimitado por um tempo máximo Δ. É suposto que o protocolo de comunicação será executado por um processador auxiliar, não competindo portanto com as tarefas da aplicação. O tempo para o envio de uma mensagem entre duas tarefas no mesmo processador será considerado zero para efeito de escalonamento.

Uma aplicação distribuída neste modelo é expressa na forma de um conjunto de atividades. Cada atividade pode ser representada por um grafo orientado acíclico, correspondendo a um conjunto de tarefas. Os nodos desde grafo representam tarefas e os arcos são relações de precedência. Uma relação de precedência direta não será satisfeita de forma implícita, através da passagem de tempo. Ela implica no envio de uma mensagem explícita da tarefa predecessora para a tarefa sucessora.

Uma aplicação é formada por um conjunto **A** de atividades contendo **m** atividades que podem ser periódicas ou esporádicas. Cada atividade A_X, $1 \leq x \leq m$, pertencente ao conjunto **A**, está sujeita a uma sequência infinita de ativações. Uma atividade periódica A_X qualquer é caracterizada por um período P_X, ou seja, um intervalo regular de tempo entre ativações sucessivas. Uma atividade esporádica A_X qualquer é caracterizada por um intervalo mínimo de tempo entre ativações P_X.

Toda aplicação está associada a um conjunto **T** contendo **n** tarefas, ou seja, o conjunto contendo todas as tarefas pertencentes a todas as atividades do conjunto **A**. As tarefas não possuem relações de exclusão mútua entre si. Qualquer tarefa pode ser preemptada, ou seja, ter sua execução suspensa e retomada mais tarde. Cada tarefa T_i é caracterizada por um jitter de liberação máximo J_i, uma prioridade global $\rho(T_i)$ e um deadline D_i, relativo ao início da sua atividade. Para toda tarefa T_i, $1 \leq i \leq n$, $T_i \in Ax$, temos que $D_i \leq P_X$. As tarefas recebem individualmente uma prioridade única na aplicação.

Cada tarefa T_i é composta por uma parte obrigatória e uma parte opcional. O tempo de execução no pior caso de ambas as partes é conhecido em tempo de projeto e denotado por M_i e O_i, respectivamente. É também suposto que existe uma restrição 0/1 para as partes opcionais. Isto é, cada parte opcional deve ser executada completamente ou não iniciada. Esta decisão deve ser tomada antes de iniciar a execução da tarefa. A tarefa T_i não inicia sua execução até receber uma mensagem de cada uma de suas predecessoras diretas. Este estudo se limita a técnica de múltiplas versões onde duas versões são usadas na programação de tarefas imprecisas. Múltiplas versões é a forma mais flexível para a programação de tarefas imprecisas. Funções monotônicas e funções melhoramento podem ser consideradas formas especiais de múltiplas versões ([LIU 94]).

É suposto que cada tarefa T_i está associada com um valor nominal V_i. O valor nominal V_i representa a utilidade da tarefa T_i no sistema. Entretanto, o valor nominal não considera as dependências entre tarefas. Esta tese introduz o conceito de valor efetivo. Em tempo de execução, a cada liberação $T_{i,k}$ da tarefa T_i é associado um valor efetivo $V_{i,k}$. O valor efetivo é calculado a partir do valor nominal e considera as dependências entre tarefas. O valor adicionado por $T_{i,k}$ ao sistema será zero no caso de uma execução imprecisa ou $V_{i,k}$ quando a parte opcional for executada. O objetivo do escalonamento das partes opcionais é maximizar o valor total do sistema.

A dependência intra-tarefa está associada com a importância de ocasionalmente executar uma dada tarefa precisamente. Sendo assim, a dependência intra-tarefa entre as liberações $T_{i,k}$ e $T_{i,k+1}$, de uma dada tarefa T_i, é modelada assumindo-se que uma execução imprecisa de $T_{i,k}$ vai aumentar o valor efetivo de uma execução precisa da liberação $T_{i,k+1}$. O valor efetivo da liberação $T_{i,k+1}$, denotado por $V_{i,k+1}$, será:

V_i quando a execução de $T_{i,k}$ é precisa;

$V_i + (\alpha_i . V_{i,k})$ quando a execução de $T_{i,k}$ é imprecisa;

onde α_i é chamada de taxa de recuperação da tarefa T_i. Embora o conceito de dependência intra-tarefa já estivesse implícito em [CHU 90], a formulação apresentada neste trabalho, baseada em taxa de recuperação, é original.

A dependência inter-tarefa está associada ao fato de dados de entrada com melhor qualidade reduzirem o tempo de execução de algumas tarefas. Estes dados de entrada são, muitas vezes, o produto da execução de uma tarefa predecessora direta. Desta forma, vamos supor que as dependências inter-tarefa surgem entre tarefas com relações de precedência direta. A dependência inter-tarefa entre as liberações $T_{j,k}$ e $T_{i,k}$ é modelada assumindo-se que uma execução precisa de $T_{j,k}$ reduzirá o tempo de execução no pior caso das partes obrigatória e opcional de $T_{i,k}$. Em outras palavras, o tempo de execução de $T_{i,k}$ no pior caso será:

$M_i + O_i$ quando a execução de $T_{j,k}$ é imprecisa;

$\beta_{j,i} . M_i + \gamma_{j,i} . O_i$ quando a execução de $T_{j,k}$ é precisa;

onde $\beta_{j,i}$ e $\gamma_{j,i}$, $0<\beta_{j,i}\leq 1$, $0<\gamma_{j,i}\leq 1$, são fatores de redução ligando T_j a T_i. Esta formulação para a dependência inter-tarefa é uma adaptação da usada em [FEN 94].

É importante observar que nenhum dos dois tipos de dependências definidos é capaz de aumentar a carga obrigatória do sistema. A dependência intra-tarefa resulta em aumento no valor efetivo de algumas partes opcionais. A dependência inter-tarefas resulta na redução do tempo máximo de execução de algumas tarefas. De qualquer modo, a carga obrigatória do sistema não aumenta. Esta propriedade é importante pois, caso contrário, o teste de escalonabilidade realizado em tempo de projeto perderia sua validade no momento em que a carga obrigatória do sistema aumentasse.

4. Abordagem de Escalonamento Distribuído

Nesta seção será descrita a abordagem proposta neste trabalho para escalonar tarefas imprecisas em ambiente distribuído. Não é possível comparar esta abordagem com propostas alternativas uma vez que inexistem na literatura outras abordagens completas para o escalonamento de tarefas imprecisas em ambiente distribuído. Entretanto, os algoritmos propostos para a abordagem adotada podem ser, estes sim, avaliados ou comparados com soluções alternativas.

A figura 1 ilustra a abordagem adotada, composta por quatro algoritmos. O algoritmo 1 realiza a alocação das tarefas aos processadores; o algoritmo 2 é capaz de verificar se as partes obrigatórias serão sempre concluídas antes do respectivo deadline; o algoritmo 3 realiza a escolha de quais partes opcionais, em um dado momento, devem ser consideradas para execução; o algoritmo 4 é capaz de garantir que uma dada parte opcional pode ser executada sem prejuízo para as partes previamente garantidas.

O primeiro algoritmo corresponde ao processo de alocação das tarefas aos processadores. O conjunto inicial de tarefas é particionado em **h** subconjuntos, onde **h** representa o número de processadores. Como uma alocação de sucesso exige que todas as partes obrigatórias sejam garantidas, o algoritmo de alocação deve usar o algoritmo 2

para verificar se este objetivo foi atendido. O algoritmo de alocação procura também obter um balanceamento de carga que amplie as chances de uma parte opcional executar.

O segundo algoritmo deve ser capaz de verificar se todos os deadlines serão cumpridos quando somente as partes obrigatórias são executadas. Uma vez que prioridades fixas serão empregadas, é necessário determinar uma política para a atribuição de prioridades e então um teste de escalonabilidade apropriado. O problema é complicado pelo fato de tarefas em diferentes processadores se comunicarem através de mensagens. Logo, a escala de execução de um processador pode ser afetada pelos atrasos decorrentes de mensagens que são enviadas a partir de outro processador.

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. Obviamente, o nível mínimo de importância para uma parte opcional ser admitida depende da carga do processador a cada instante.

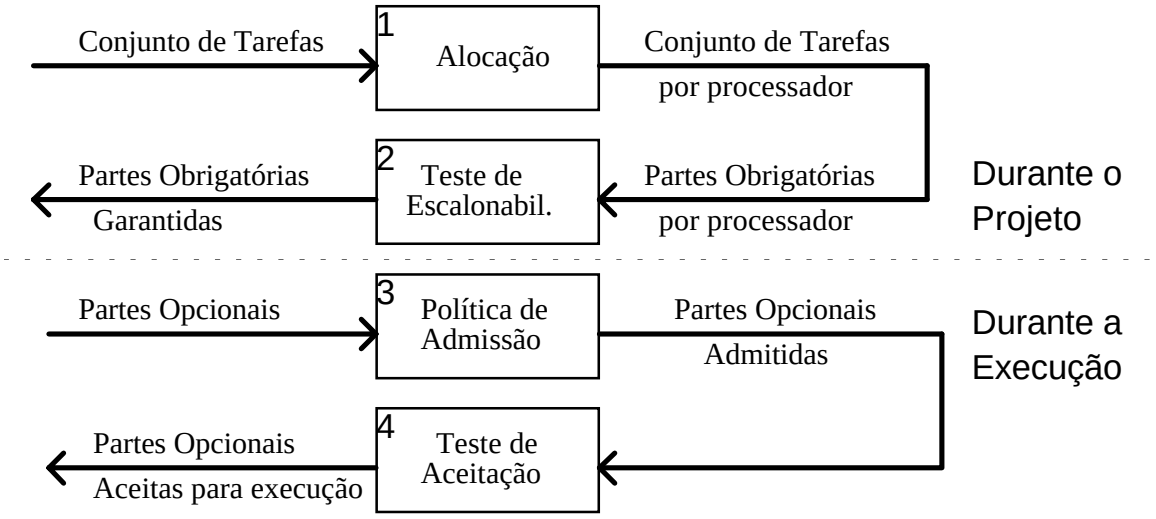

Figura 1 - Abordagem geral adotada.

O quarto algoritmo avalia se uma dada parte opcional, previamente admitida pelo algoritmo 3, pode ser aceita para execução. Uma parte opcional somente poderá ser aceita para execução se não comprometer as garantias fornecidas anteriormente. O algoritmo 4 deve levar em conta as condições supostas pelo algoritmo 2. Em outras palavras, uma parte opcional pode ser aceita se ela não alterar as condições supostas pelo algoritmo 2 de tal forma a invalidar o teste de escalonabilidade feito no projeto.

O objetivo geral deste trabalho é mostrar como aplicações de tempo real, construídas a partir do conceito de Computação Imprecisa, podem ser escalonadas em ambiente distribuído. Uma vez definida a abordagem que será empregada, resta resolver os quatro problemas de escalonamento.

5. Soluções Algorítmicas para a Abordagem Adotada

A abordagem descrita na seção anterior exige uma solução algorítmica para cada um dos quatro problemas específicos. Esta seção indica como estes quatro problemas

podem ser resolvidos. Devido a limitação de espaço, as soluções algorítmicas serão indicadas em seus princípios básicos. Uma formalização mais rigorosa, assim como análise quantitativa dessas soluções, podem ser encontradas em [OLI 96] e [OLI 97].

5.1 Alocação

O problema de alocação de tarefas em ambiente distribuído é bastante complexo. Na maioria dos casos, obter uma solução ótima implica em um custo de processamento proibitivo. Desta forma, são utilizadas abordagens subótimas. Neste trabalho é usado um método de pesquisa aleatória bastante difundido, o recozimento simulado ("simulated annealing", [KIR 83], [TIN 92a], [BUR 93]). A justificativa para o uso de técnicas de otimização vem da dificuldade de construir boas heurísticas, considerando a complexidade do problema tratado.

O objetivo primário da alocação é garantir os deadlines das tarefas, considerando só as partes obrigatórias. O objetivo secundário será prover um razoável balançeamento de carga, de modo a evitar sobrecargas localizadas e aumentar as chances das partes opcionais executarem. Para isto, é necessário definir a função que avalia a qualidade de uma solução. No recozimento simulado, a energia **E** associada com a qualidade de uma solução pode ser calculada por: $E = K_e \times E_e + K_b \times E_b$; onde E_e e E_b representam as energias associadas com a escalonabilidade das tarefas e o balanceamento da carga.

O emprego dos valores K_e e K_b permite que o aspecto escalonabilidade receba uma importância maior do que o aspecto balanceamento da carga. Jamais uma solução bem balanceada porém não escalonável terá uma energia menor do que uma solução com péssimo balanceamento porém com todos os deadlines garantidos. Ao escolhermos os valores de K_e e K_b estaremos garantindo que a escalonabilidade será o objetivo primário da alocação, enquanto o balanceamento é apenas um objetivo secundário. É possível afirmar que o algoritmo de alocação busca, entre todas as soluções que garantem a escalonabilidade do sistema, aquela solução com o melhor balanceamento de carga.

O algoritmo de alocação procura aumentar as chances de uma parte opcional ser escalonada em tempo de execução. Uma primeira forma de fazer isto é balancear a carga no sistema, considerando a duração total das tarefas. A partir desta abordagem simples, melhorias podem ser feitas. Para efeito de balanceamento de carga, é mais realista considerar o tempo médio de execução das tarefas e não o tempo máximo de execução. Esporádicas são tratadas, no pior caso, como periódicas com período igual ao intervalo mínimo entre ativações. No caso do balanceamento da carga, este tratamento é pessimista. Para efeitos de balanceamento de carga, as esporádicas devem ser tratadas como periódicas com período igual ao intervalo médio entre ativações. Quando as tarefas possuem valores nominais fixos, é possível utilizar estes valores nominais para julgar o balanceamento de carga obtido. Duas situações são indesejáveis: termos opcionais demais em um mesmo processador e termos tarefas de muito valor concentradas em um mesmo processador. Em aplicações onde o valor nominal de uma tarefa varia ao longo do tempo não é possível este tipo de análise.

5.2 Teste de Escalonabilidade

No teste de escalonabilidade a preocupação é com a garantia da parte obrigatória de cada tarefa. As partes opcionais serão executadas com os recursos (tempo de

processador) que sobrarem após esta garantia ter sido obtida. Desta forma, para efeito do teste de escalonabilidade, as partes opcionais das tarefas podem ser ignoradas.

A maior parte dos trabalhos publicados empregam prioridades fixas e liberação estática de tarefas para implementar relações de precedência. Relações de precedência arbitrárias em monoprocessadores são consideradas em [AUD 93a] e [GER 94]. O trabalho [TIN 94] trata de relações de precedência arbitrárias em sistemas distribuídos. Os trabalhos [SHA 94] e [SUN 95] consideram apenas relações de precedência lineares ("pipelines"), onde cada tarefa possui no máximo um predecessor e um sucessor.

Liberação dinâmica de tarefas é empregada em [HAR 94] para implementar relações de precedência lineares em monoprocessador. Liberação dinâmica também é empregada em [TIN 94] e [SUN 96] na análise de sistemas distribuídos onde existem apenas relações de precedência lineares. O único trabalho conhecido dos autores que emprega liberação dinâmica de tarefas para suportar relações de precedência arbitrárias em ambiente distribuído é apresentado em [BUR 93]. O cálculo empregado é tal que as relações de precedência são transformadas em jitter de liberação e todas as tarefas passam a ser tratadas como tarefas independentes.

Neste trabalho é proposto o emprego de prioridades fixas e liberação dinâmica de tarefas para escalonar sistemas tempo real críticos em ambiente distribuído. A teoria de escalonamento relacionada com prioridades fixas evoluiu bastante nos últimos anos, passando a suportar modelos de tarefas bem mais complexos ([SHA 94], [AUD 93b]) que aqueles empregados nos primeiros trabalhos ([LIU 73]). Ao mesmo tempo, a liberação dinâmica de tarefas é mais flexível que a liberação estática. Por exemplo, a liberação dinâmica de tarefas facilita o escalonamento, em ambiente distribuído, de conjuntos de tarefas esporádicas com relações de precedência entre si. Além disto, o emprego de liberação estática de tarefas fragmenta as sobras de processador, tornando-as menos úteis para o escalonamento de partes opcionais.

Neste trabalho vamos considerar que tarefas recebem prioridades seguindo a política do deadline monotônico (DM, "deadline monotonic", [LEU 82]). O deadline monotônico é uma forma simples e rápida de atribuir prioridades. Em sistemas de tarefas com relações de precedência o DM não é ótimo. Entretanto, DM é ótimo na classe das políticas que geram prioridades decrescentes dentro dos grafos de precedência ([HAR 94]). Além disto, DM pode ser considerado uma boa heurística de propósito geral.

O teste de escalonabilidade descrito em [OLI 97] utiliza prioridades fixas e liberação dinâmica de tarefas para implementar relações arbitrárias de precedência em ambiente distribuido. Ela supõe deadline menor ou igual ao período para toda a atividade e emprega jitter de liberação para modelar em parte o efeito das relações de precedência. Diferentemente de [BUR 93], a análise descrita em [OLI 97] considera também o efeito positivo das relações de precedência sobre o escalonamento, pois elas limitam os possíveis padrões de interferência. Desta forma, conjuntos de tarefas que seriam considerados inviáveis pela análise daquele trabalho, são mostrados viáveis pela análise proposta aqui. Exemplos neste sentido são apresentados em [OLI 97].

5.3 Política de Admissão

A abordagem adotada neste trabalho inclui a execução de partes opcionais quando o processador não estiver executando partes obrigatórias. Em [DAV 95] foram

descritas duas heurísticas para políticas de admissão em sistemas com componentes opcionais. Na política FCFS todas os componentes opcionais do sistema são sempre admitidos. Na política AVDT somente componentes opcionais com uma densidade de valor superior a densidade média de valor do sistema são considerados. A densidade de valor é obtida a partir da divisão do valor de uma tarefa pelo seu tempo máximo de execução. O modelo de tarefas usado em [DAV 95] supõe independência entre tarefas.

Em [OLI 96] são apresentadas duas novas heurísticas para serem usadas como política de admissão quando tarefas imprecisas são empregadas. As duas políticas propostas em [OLI 96], CVDT e INTER, consideram a existência de dependências intra-tarefa e inter-tarefa no sistema. Elas são analisadas através de simulação e os resultados comparados com os obtidos pelas duas heurísticas apresentadas antes em [DAV 95]. As políticas CVDT e INTER são adotadas neste trabalho por apresentarem melhores resultados na presença de dependência entre tarefas.

5.4 Teste de Aceitação

O teste de aceitação é usado em tempo de execução. Logo, seu custo computacional ("overhead") deve ser baixo. Uma solução ótima para o teste de aceitação é definida como aquela capaz de detectar toda a capacidade ociosa existente tão logo ela seja gerada. Soluções ótimas para o teste de aceitação são inviáveis na prática, em função da complexidade de seus algoritmos.

Muitos trabalhos na literatura procuram garantir, em tempo de execução, tarefas aperiódicas. Embora tais algoritmos não tenham sido criados especificamente para Computação Imprecisa, eles podem ser utilizados como testes de aceitação para partes opcionais. Isto é possível pois, do ponto de vista do escalonador, uma parte opcional pode ser considerada como uma tarefa aperiódica que não foi garantida em tempo de projeto e que necessita de uma garantia dinâmica.

Em [RAM 90] são apresentados algoritmos capazes de oferecer uma garantia dinâmica para tarefas aperiódicas a partir da manipulação explícita da escala de execução. Eles mantém uma tabela onde está anotado quando cada tarefa vai ocupar o processador. A tabela é construida dinamicamente, em tempo de execução. Muitas vezes este tipo de algoritmo é chamado de baseado em planejamento ("planning based").

Em [SPR 89] são apresentados servidores capazes de oferecer garantia dinâmica para tarefas aperiódicas no contexto de escalonamento baseado em prioridades fixas. Em tempo de projeto, após todas as tarefas periódicas terem sido garantidas, estes servidores reservam para si a capacidade restante no sistema. Durante a execução, os servidores utilizam sua própria capacidade para executar tarefas aperiódicas.

Em [THU 94] é apresentada a tomada estática de folga ("static slack stealing") como um esquema capaz de identificar capacidade ociosa no contexto de prioridades fixas. Em tempo de projeto é construida uma tabela de folgas que indica quando o sistema pode admitir a execução de tarefas aperiódicas sem comprometer as tarefas previamente garantidas. Na medida que as tarefas são executadas a tabela é atualizada.

Em [DAV 93a] é apresentada a tomada dinâmica de folga ("dynamic slack stealing"). Naquele trabalho o princípio da tomada estática de folga é adaptado para um modelo de tarefas que inclui sincronização entre tarefas, jitter na liberação e tarefas esporádicas garantidas. É incluída na folga todo o tempo ganho quando uma tarefa não

usa todo o tempo de processador que havia sido reservado para ela. Este algoritmo calcula as folgas em tempo de execução. O algoritmo de tomada dinâmica de folga resulta em excelente eficiência com relação à identificação das folgas no sistema. Porém, o seu custo computacional ("overhead") é muito elevado, o que inviabiliza sua utilização.

Em [DAV 93b] e [AUD 94] é proposta uma solução aproximada para o algoritmo de tomada dinâmica de folga (DASS, "dynamic approximate slack stealing"). Esta aproximação é tal que, toda capacidade ociosa identificada realmente existe. Entretanto, este algoritmo apresenta uma eficiência inferior ao algoritmo anterior, pois ele não é capaz de identificar todas as folgas existentes no sistema em um dado momento. As simulações apresentadas em [DAV 93b] mostram que, mesmo não sendo uma solução ótima, a tomada dinâmica de folga aproximada resulta em eficiência superior às soluções baseadas em servidores. Em [OLI 97] é mostrado que DASS pode ser facilmente adaptado para o ambiente distribuído.

Neste trabalho é proposto o emprego do DASS como teste de aceitação. O DASS é um compromisso entre eficiência na identificação de folgas e custo computacional. Além disto, o modelo de tarefas suportado pelo DASS é o mais próximo do modelo de tarefas adotado neste trabalho.

6. Conclusões

Neste trabalho foi feito inicialmente um levantamento dos problemas relacionados com o escalonamento tempo real de tarefas imprecisas em ambiente distribuído. A discussão da problemática foi dividida em quatro aspectos: alocação das tarefas aos processadores, garantia em tempo de projeto para as partes obrigatórias, identificação de folgas em tempo de execução e escolha de quais partes opcionais devem ser executadas quando não for possível executar todas. Uma vez definido o modelo básico de tarefas a ser usado, foi descrita a abordagem adotada no sentido de atingir o objetivo geral, isto é, escalonar tarefas imprecisas em ambiente distribuído. Nossas soluções algorítmicas para os quatro problemas são indicadas e discutidas no contexto da respectiva bibliografia.

A abordagem proposta neste trabalho compatibiliza a garantia necessária em tempo de projeto para sistemas críticos com uma maior utilidade para o sistema através da execução de partes opcionais. O custo de um sistema crítico é reduzido na medida em que apenas as funções e propriedades realmente críticas do sistema recebem uma garantia para o seu comportamento temporal no pior caso. Partes opcionais são escalonadas dentro de uma política de melhor esforço. A técnica empregada, Computação Imprecisa, pode ser vista como um mecanismo capaz de tolerar sobrecargas no sistema sem comprometer seus requisitos temporais críticos. Neste caso, temos uma degradação graciosa do sistema na medida em que sua qualidade é reduzida.

Foram realizadas simulações que proporcionaram uma oportunidade para empregar a solução completa proposta neste trabalho. Inicialmente, aplicações foram geradas aleatoriamente, dentro de determinados padrões de utilização obrigatória e opcional. Em seguida, as tarefas foram alocadas conforme a solução baseada em recozimento simulado proposta na seção 5.1. Como parte do problema de alocação, a escalonabilidade das tarefas foi testada através do algoritmo indicado na seção 5.2. Uma vez feita a alocação, foi simulada a execução da aplicação. Durante a simulação da execução foi usada a solução indicada na seção 5.4 para identificar folgas no sistema e,

com base nelas, estabelecer um teste de aceitação. Finalmente, as políticas de admissão citadas na seção 5.3 foram incluidas na simulação.

A contribuição deste texto deve ser entendida como a apresentação de uma abordagem completa para o escalonamento de tarefas imprecisas em ambientes distribuídos. Os autores desconhecem na literatura um outro tratamento completo para este problema. As soluções de escalonamento indicadas, embora neste texto apareçam de forma resumida, foram formalizadas, avaliadas e implementadas. Os teoremas e as simulações utilizadas para este propósito podem ser encontrados em [OLI 97].

7. Referências

[AUD 93a] N. Audsley, K. Tindell, A. Burns. The End of the Line for Static Cyclic Scheduling? Proceedings of the Fifth Euromicro Workshop on Real-Time Systems, IEEE Computer Society Press, pp. 36-41, june 1993.

[AUD 93b] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, A. J. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. Software Engineering Journal, Vol. 8, No. 5, pp.284-292, 1993.

[AUD 94] N. C. Audsley, R. I. Davis, A. Burns. Mechanisms for Enhancing the Flexibility and Utility of Hard Real-Time Systems. Proceedings of the IEEE Real-Time Systems Symposium, pp. 12-21, San Juan, Puerto Rico, december 1994.

[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.

[BUR 93] A. Burns, M. Nicholson, K. Tindell, N. Zhang. Allocating and Scheduling Hard Real-Time Tasks on a Point-to-Point Distributed System. Proceedings of the Workshop on Parallel and Distributed Real-Time Systems, pp. 11-20, 1993.

[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, september 1990.

[DAV 93a] R. I. Davis, K. W. Tindell, A. Burns. Scheduling Slack Time in Fixed Priority Pre-emptive Systems. Proc. IEEE Real-Time Syst. Symp., pp.222-231, 1993.

[DAV 93b] R. I. Davis. Approximate Slack Stealing Algorithms for Fixed Priority Pre-emptive Systems. Department of Computer Science report, Univ. of York, 1993.

[DAV 95] R. Davis, S. Punnekkat, N. Audsley, A. Burns. Flexible Scheduling for Adaptable Real-Time Systems. Proceedings 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. Univ. of Illinois at Urbana-Champaign, DCS, Technical Report #1888, 1994.

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

[GER 94] R. Gerber, S. Hong, M. Saksena. Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes. Proceedings of the IEEE Real-Time Systems Symposium, december 1994.

[HAR 94] M. G. Harbour, M. H. Klein, J. P. Lehoczky. Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems. IEEE Transactions on Software Engineering, Vol. 20, No. 1, pp. 13-28, january 1994.

[HUA 95] X. Huang, A. Cheng. Applying Imprecise Algorithms to Real-Time Image and Video Transmission. Proc. IEEE Workshop on Real-Time App. Chicago, may 1995.

[KIR 83] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. Optimization by Simulated Annealing. Science (220), pp. 671-680, 1983.

[LEU 82] J. Y. T. Leung, J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. Perf. Evaluation, 2(4), pp.237-250, dec. 1982.

[LIU 73] C. L. Liu, J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM, Vol. 20, No.1, pp.46-61, january 1973.

[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.

[OLI 96] R. S. Oliveira, J. S. Fraga. Scheduling Imprecise Computation Tasks with Intra-Task / Inter-Task Dependence. Proceedings of the 21st IFAC/IFIP Workshop on Real Time Programming, Gramado-RS-Brasil, pp. 51-56, november 1996.

[OLI 97] R. S. Oliveira. Escalonamento de Tarefas Imprecisas em Ambiente Distribuído. Tese de Doutorado, Dep. de Eng. Elét., Univ. Fed. de Santa Catarina, 1997.

[RAM 89] K. Ramamritham, J. A. Stankovic, W. Zhao. Distributed Scheduling of Tasks with Deadlines and Resource Requirements. IEEE Transactions on Computers, Vol. 38, No. 8, pp. 1110-1123, august 1989.

[RAM 90] K. Ramamritham, J. A. Stankovic, P.-F. Shiah. Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 1, No. 2, pp. 184-194, april 1990.

[SHA 94] L. Sha, R. Rajkumar, S. S. Sathaye. Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems. Proceedings of the IEEE, Vol. 82, No. 1, pp. 68-82, january 1994.

[SPR 89] B. Sprunt, L. Sha, J. Lehoczky. Aperiodic Task Scheduling for Hard-Real-Time Systems. The Journal of Real-Time Systems, Vol. 1, pp. 27-60, 1989.

[SUN 95] J. Sun, J. W.-S. Liu. Bounding the End-to-End Response Time in Multiprocessor Real-Time Systems. Proc. Workshop on Parallel and Distributed Real-Time Systems, pp.91-98, Santa Barbara, CA, april 1995

[SUN 96] J. Sun, J. W.-S. Liu. Synchronization Protocols in Distributed Real-Time Systems. Proc. of 16th Int. Conference on Distributed Computing Systems, may 1996.

[THU 94] S. R. Ramos-Thuel, J. P. Lehoczky. Algorithms for Scheduling Hard Aperiodic Tasks in Fixed-Priority Systems Using Slack Stealing. Proceedings of the 15th IEEE Real-Time Systems Symposium, pp.22-33, December 1994.

[TIN 92] K. W. Tindell, A. Burns, A. J. Wellings. Allocating Hard Real-Time Tasks: An NP-Hard Problem Made Easy. Real-Time Syst., Vol.4, No.2, jun 1992.

[TIN 94] K. Tindell, J. Clark. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. Microprocessors and Microprogramming, 40, 1994.

[XU 93] J. Xu, D. L. Parnas. On Satisfying Timing Constraints in Hard-Real-Time Systems. IEEE Trans. on Software Engineering, Vol.19, No.1, pp.70-84, Jan 1993.

[YU 92] A. C. Yu, K.-J. Lin. A Scheduling Algorithm for Replicated Real-Time Tasks. IEEE IPCCC'92, pp. 395-402, 1992.