Escalonamento de Aplicações Tempo Real Mistas com Relações de Precedência Arbitrárias em Ambiente Distribuído

Rômulo Silva de Oliveira

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

Resumo

Este artigo considera uma abordagem mista para o escalonamento tempo real. Em tempo de projeto uma análise de escalonabilidade verifica se todas as tarefas críticas estão garantidas. Em tempo de execução, ativações individuais de tarefas aperiódicas são executadas na medida do possível. O objetivo deste artigo é desenvolver um teste capaz de aceitar para execução tarefas aperiódicas independentes, com deadline soft, sem comprometer um conjunto de tarefas críticas garantido anteriormente, as quais possuem relações arbitrárias de precedência em um ambiente distribuído. O artigo também discute o relacionamento entre relações de precedência e identificação de folgas através de simulações da solução proposta e análise dos resultados.

Abstract

This paper assumes a mixed approach to real-time scheduling. At design-time a schedulability test determines whether all critical tasks are guaranteed. At run-time, individual activations of aperiodic tasks are executed whenever possible. The objective of this paper is to develop a test to accept for execution independent aperiodic tasks with soft deadline, without jeopardizing a set of previously guaranteed critical tasks that may present arbitrary precedence relations in a distributed environment. This paper also discusses the relationship between precedence relations and slack identification by simulating the proposed solution and analysing the results.

Palavras-chave: Escalonamento, tempo real, distribuído.

1 Introdução

Na medida em que o uso de sistemas computacionais prolifera-se 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. Um problema básico encontrado na construção de sistemas distribuídos de 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 a sua utilidade.

Uma das técnicas possíveis para resolver o problema de escalonamento tempo real é empregar uma abordagem mista [AUD 94a]. Em tempo de projeto uma análise de escalonabilidade verifica se todas as *tarefas críticas* (deadline hard) estão garantidas, isto é,

jamais perderão um deadline. Em tempo de execução, ativações individuais de *tarefas aperiódicas* (deadline soft), não garantidas em tempo de projeto, são executadas na medida do possível ("best-effort"). Esta execução acontece desde que não comprometa os deadlines das tarefas críticas previamente garantidas.

Garantia em projeto para tarefas com deadlines críticos pode ser obtida, entre outras formas, através de escalonamento baseado em prioridades fixas [LIU 73]. Tarefas recebem prioridades de acordo com alguma política e um teste de escalonabilidade é usado em tempo de projeto para verificar se todas as tarefas cumprirão seus deadlines em um cenário de pior caso. Durante a execução, o escalonador apenas seleciona a próxima tarefa baseado em suas prioridades fixas.

Relações de precedência entre tarefas surgem quando uma tarefa recebe e/ou envia mensagens para outras tarefas. As relações de precedência entre tarefas podem ser classificadas como lineares ou arbitrárias. Em um sistema com apenas relações de precedência lineares cada tarefa pode ter, no máximo, uma tarefa predecessora e uma tarefa sucessora, formando o que é normalmente chamado de "pipeline". Sistemas com relações de precedência arbitrárias permitem que cada tarefa possua várias tarefas predecessoras e várias tarefas sucessoras. Embora a precedência arbitrária seja mais flexível com respeito ao projeto do sistema, sua análise é mais complexa do que quando existe apenas precedência linear.

Em [OLI 98] foi apresentado um teste de escalonabilidade para aplicações tempo real distribuídas onde tarefas podem apresentar relações arbitrárias de precedência. O modelo adotado supõe que as tarefas são periódicas ou esporádicas, possuem prioridade fixa e deadline hard fim-a-fim sempre igual ou menor que o respectivo período. O teste apresentado naquele artigo emprega um método para transformar relações arbitrárias de precedência em *jitter* de liberação. Após a eliminação de todas as relações de precedência presentes no conjunto de tarefas é possível aplicar qualquer teste de escalonabilidade disponível para tarefas independentes. Embora a análise apresentada em [OLI 98] resolva o problema da garantia para tarefas críticas, aquele artigo não aborda a questão do escalonamento simultâneo de tarefas aperiódicas independentes com deadline soft.

O escalonamento de aplicações mistas, compostas por tarefas críticas e tarefas não críticas, exige que, em tempo de execução, a capacidade ociosa ("spare capacity") dos processadores seja detectada. Esta capacidade ociosa será utilizada para executar tarefas aperiódicas com deadline soft que, como não são críticas, não foram garantidas em projeto. É necessário um teste em tempo de execução para determinar quando e por quanto tempo uma tarefa aperiódica pode ser executada sem comprometer as garantias já fornecidas.

Uma solução de escalonamento mista, onde as tarefas críticas são garantidas durante o projeto e tarefas aperiódicas não críticas são executadas na medida do possível, pode ser obtida através do emprego em projeto da análise descrita em [OLI 98], acompanhada de um teste em aceitação usado em tempo de execução. É necessário portanto definir um algoritmo que, durante a execução da aplicação ("on-line"), determina se a execução de uma tarefa aperiódica independente não irá comprometer a execução das tarefas previamente garantidas. Ao resolver esta questão, estaremos oferecendo uma solução para o escalonamento de aplicações mistas com relações de precedência arbitrárias em ambiente distribuído.

O objetivo deste artigo é desenvolver um teste capaz de aceitar tarefas independentes aperiódicas para execução sem comprometer um conjunto de tarefas garantido anteriormente, as quais possuem relações arbitrárias de precedência em um ambiente distribuído. Em outras

palavras, desenvolver um teste de aceitação para ser usado em conjunto com a análise de escalonabilidade apresentada em [OLI 98]. Ao mesmo tempo, a solução apresentada deverá oferecer um compromisso razoável entre o custo computacional do algoritmo e a capacidade ociosa que ela consegue detectar.

Duas questões paralelas são: em que processador alocar cada tarefa e como escolher, entre as tarefas aperiódicas esperando pelo processador, qual executar aproveitando a capacidade ociosa detectada. Estas duas questões não serão abordadas neste trabalho.

O restante deste artigo está organizado da seguinte forma: a seção 2 faz uma revisão da literatura com respeito a execução de tarefas aperiódicas. A seção 3 descreve a abordagem adotada neste trabalho; na seção 4 são apresentadas as adaptações necessárias em função do modelo de tarefas adotado neste trabalho; a seção 5 trata das ligações entre relações de precedência e identificação das folgas; na seção 6 são feitos comentários gerais sobre o método descrito neste artigo; finalmente, as conclusões aparecem na seção 7.

2 Trabalhos Correlatos

O escalonamento de tarefas aperiódicas é feito em tempo de execução. Logo, seu custo computacional ("overhead") deve ser baixo. De outra forma, a capacidade ociosa dos processadores seria consumida pelo próprio algoritmo de escalonamento, ao invés de ser usada pelas tarefas da aplicação. A solução de menor custo computacional é executar tarefas aperiódicas somente quando nenhuma tarefa crítica estiver liberada para execução, isto é, somente quando todas as tarefas críticas estiverem suspensas a espera de uma mensagem ou de uma nova ativação. Neste caso, o algoritmo fica reduzido a simplesmente determinar quando o processador está livre ("idle"). Esta solução, chamada de *execução em background* [SPR 89], possui como desvantagem a demora em detectar a capacidade ociosa. Uma solução ótima para o escalonamento de tarefas aperiódicas é definida como aquela capaz de detectar toda a capacidade ociosa existente tão logo ela seja gerada. Soluções ótimas neste caso são inviáveis na prática, em função da complexidade de seus algoritmos. Existe na literatura um conjunto de soluções de compromisso para o problema da identificação de capacidade ociosa.

Em [LEH 87] e [SPR 89] são apresentados *servidores* capazes de determinar quando uma tarefa aperiódica pode ser executada no contexto de escalonamento baseado em prioridades fixas. Após todas as tarefas periódicas terem sido garantidas, ainda em tempo de projeto estes servidores reservam para si a capacidade restante no sistema. Em tempo de execução, estes servidores utilizam esta capacidade restante para executar tarefas aperiódicas.

Os servidores falham em capturar outras fontes de recurso do sistema, tais como o tempo ganho na execução ("gain time"). Em situações práticas os processadores são dimensionados de forma que, quando consideramos o comportamento no pior caso das tarefas críticas, a capacidade restante é pequena. Nestes casos, espera-se executar as tarefas aperiódicas com os recursos liberados na medida que as tarefas críticas não ocupam todo o recurso reservado, ou seja, apresentam um comportamento melhor do que o pior caso. Isto é possível pois o tempo de execução no pior caso de uma tarefa é, em geral, várias vezes superior ao seu tempo médio de execução. Além do tempo ganho na execução, as diferenças de fase entre as tarefas geram uma capacidade ociosa temporária que também não é capturada pelos servidores ([BUR 96]).

Em [LEH 92], [THU 93] e [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. Esta capacidade ociosa pode ser usada para escalonar tarefas aperiódicas.

Esses trabalhos supõem uma aplicação composta por um conjunto de tarefas periódicas garantidas em tempo de projeto mais um conjunto de tarefas aperiódicas que devem ser executadas na medida do possível. Em tempo de projeto é construida uma tabela de folgas que indica em que momentos o sistema aceita a execução de tarefas aperiódicas sem comprometer as tarefas previamente garantidas. Na medida em que as tarefas são executadas, esta tabela é mantida atualizada. Desta forma, quando uma tarefa apresenta um comportamento melhor do que o pior caso, as folgas descritas pela tabela são ampliadas. De forma semelhante, a execução de uma tarefa aperiódica implica no consumo de folga e, portanto, na atualização da tabela. Uma variação do algoritmo básico é apresentada em [TIA 94].

A tomada estática de folga é capaz de apresentar excelente eficiência no que diz respeito à identificação de folgas. Entretanto, a aplicação de tomada estática de folga está restrita a conjuntos de tarefas estritamente periódicas. Além disto, o tamanho da tabela de folgas construida em tempo de projeto e mantida durante a execução é proporcional ao mínimo múltiplo comum dos períodos das tarefas. Esta tabela possuirá um tamanho muito grande quando os períodos forem números primos entre si.

Em [DAV 93a] é apresentada a *tomada dinâmica de folga* ("dynamic slack stealing"). Neste 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. O tempo ganho quando uma tarefa não usa todo o tempo de processador que havia sido reservado para ela ("gain time") é incluído na folga. Esta ampliação do modelo de tarefas suportado, em relação à tomada estática de folga, é possível na medida que o algoritmo proposto calcula as folgas em tempo de execução. A tabela mantida em tempo de execução é proporcional ao número de tarefas e não ao MMC (mínimo múltiplo comum) de seus períodos. O algoritmo de tomada dinâmica de folga também resulta em excelente eficiência com relação à identificação das folgas no sistema. Entretanto, seu custo computacional ("overhead") é muito elevado, o que inviabiliza sua utilização na prática.

Em [DAV 93b] e [AUD 94b] é proposta uma solução aproximada para o algoritmo de tomada dinâmica de folga. A aproximação realizada garante a existência de toda capacidade ociosa identificada. 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, no que diz respeito à execução de tarefas aperiódicas. Como no caso anterior, o modelo de tarefas suportado inclui tarefas esporádicas e *jitter* na liberação. Entretanto, o custo computacional da solução aproximada é muito menor do que o custo da tomada dinâmica de folga.

3 Descrição da Abordagem

Neste trabalho será empregada a técnica de tomada dinâmica de folga aproximada (DASS, "dynamic approximate slack stealing"), proposta em [DAV 93b]. Através da discussão apresentada na seção anterior é possível observar que o DASS é um compromisso entre eficiência na identificação de folgas e custo computacional. Ele resolve de forma satisfatória a questão de como determinar se a execução de uma tarefa aperiódica não irá comprometer a execução das tarefas previamente garantidas. Além disto, o modelo de tarefas suportado pelo DASS é o mais próximo do modelo de tarefas adotado neste trabalho.

No nosso modelo, sempre que existem tarefas aperiódicas para executar, ocorre uma solicitação automática de tempo do processador. O algoritmo DASS mantém contadores de folga $S_i(t)$ para cada nível **i** de prioridade. No momento de decidir executar ou não uma tarefa aperiódica, os contadores de folga são consultados. A seção 3.1 descreve como os valores $S_i(t)$ são calculados pelo DASS.

Suponha que, no momento em que é solicitado tempo de processador para executar a tarefa aperiódica T_a, a tarefa crítica liberada para execução com maior prioridade seja a tarefa T_h. Para que uma tarefa aperiódica possa executar antes de T_h é necessário consumir folgas de todos os níveis de prioridade iguais ou inferiores ao nível **h**. Considerando que números maiores indicam prioridade inferior, uma tarefa aperiódica poderá executar durante C_a unidades de tempo somente se $\forall j \geq h \; . \; C_a \leq S_j(t)$, onde **t** é o instante no qual o teste é executado. Logo, o tempo de processador que pode ser fornecido para a tarefa T_a a partir do instante **t** é dado por:

$$C_a = \min_{\forall j \geq h} S_j(t) \, .$$

3.1 Cálculo das Folgas pelo DASS

Esta seção descreve como o DASS calcula e mantém os contadores de folga para cada nível de prioridade. Em tempo de execução o suporte de execução mantém para cada tarefa os seguintes valores:

P_i: Período ou intervalo mínimo entre chegadas da tarefa T_i;

C_i: Tempo máximo de computação da tarefa T_i;

$x_i(t)$: Instante, em valores absolutos, da próxima liberação de T_i, supondo que ela ocorra o mais cedo possível;

$d_i(t)$: Instante, em valores absolutos, do próximo deadline de T_i a ser cumprido;

$c_i(t)$: Tempo máximo de execução restante na atual invocação de T_i.

A partir destes valores é possível calcular a folga existente no nível **i** em um dado instante **t**, denotada por $S_i(t)$. Este valor representa a quantidade de interferência extra que T_i pode receber e ainda cumprir o próximo deadline $d_i(t)$. Como o algoritmo de cálculo usado é aproximado, $S_i(t)$ é na verdade um limite mínimo ("lower bound") para este valor.

O valor de $S_i(t)$ é dado por:

$$S_i(t) = \left[d_i(t) - t - \sum_{\forall j \leq i} I_j(t, d_i(t)) \right]_0$$

onde $I_j(t,d_i(t))$ representa o tempo máximo de execução da tarefa T_j dentro do intervalo $[t,t+d_i(t))$. A notação $[\ a\ ]_0$ é definida como:

a quando $a \geq 0$;
0 quando $a < 0$.

O valor $I_j(t,d_i(t))$ é dado por:

$$I_j(t, d_i(t)) = c_j(t) + f_j(t, d_i(t)) \times C_j + \mathrm{Min}\left(C_j, \left(d_i(t) - x_j(t) - f_j(t, d_i(t)) \times P_j\right)_0\right)$$

onde $f_j(t,d_i(t))$ é o número de invocações completas de T_j dentro do intervalo $[t,d_i(t))$, dado por:

$$f_j(t,d_i(t)) = \left\lfloor (d_i(t) - x_j(t))/P_j \right\rfloor_0 .$$

O valor $S_i(t)$ é calculado sempre que a invocação corrente de T_i é concluida. Neste momento, qualquer tarefa com prioridade superior a T_i já terá concluido e estará esperando a próxima chegada ou liberação. Observe que se uma tarefa com prioridade superior estivesse liberada, ela estaria executando no lugar de T_i. Logo, no cálculo, o valor $c_j(t)$ será sempre zero (T_j aguarda nova chegada) ou C_j (T_j já chegou mas aguarda a liberação).

Entre um cálculo e outro, o valor das folgas de cada tarefa deve ser mantido atualizado, em função das folgas que são geradas ou consumidas. A cada chaveamento de contexto a tabela de folgas deverá ser atualizada. Suponha que ocorre um chaveamento de contexto após a tarefa T_i executar por **E** unidades de tempo. Neste caso, a folga de todas as tarefas com prioridade superior à tarefa T_i deve ser diminuida de **E** unidades. Em outras palavras, $\forall j<i$, $S_j(t) = S_j(t) - E$. Suponha ainda que a tarefa T_i executou **F** unidades de tempo a menos que o seu tempo de execução no pior caso. Neste caso, a folga de todas as tarefas com prioridade igual ou inferior à T_i deve ser aumentada de **F** unidades: $\forall j \geq i$, $S_j(t)=S_j(t)+F$.

Uma tarefa aperiódica poderá executar no nível de prioridade **i** enquanto a folga existente no nível **i** for maior que zero. Além disto, caso a tarefa aperiódica seja executada, ela vai gerar interferência sobre os níveis de **i+1** até **n**. Logo, é necessário que a folga existente nestes níveis também seja maior ou igual ao tempo que a tarefa aperiódica executar. Em outras palavras, uma tarefa aperiódica T_a poderá executar durante C_a unidades de tempo no nível de prioridade **i** quando:

$$C_a \leq \operatorname*{Min}_{\forall j \geq i} S_j(t) .$$

Caso a tarefa aperiódica T_a seja executada, é necessário corrigir a tabela de folgas conforme o tempo que ela executar, fazendo:

$$\forall j \geq i, S_j(t) = S_j(t) - C_a .$$

4 Adaptação do DASS para o Modelo de Tarefas Adotado

O modelo de tarefas adotado neste trabalho utiliza, para as tarefas críticas, o mesmo modelo descrito em [OLI 98]. É suposto um sistema distribuído composto por **h** processadores. O atraso na comunicação remota é limitado e possui um valor máximo **Δ**. O atraso associado com uma comunicação local é suposto zero. Relações de precedência são implementadas por uma mensagem enviada pela tarefa predecessora para a tarefa sucessora.

Uma aplicação é definida por um conjunto **A** de **m** atividades periódicas ou esporádicas. Cada atividade A_X, $A_X \in A$, gera um conjunto possivelmente infinito de chegadas. Uma atividade periódica A_X é caracterizada pelo seu período P_X, isto é, o intervalo de tempo entre duas chegadas sucessivas. Uma atividade esporádica A_X é caracterizada pelo seu intervalo de tempo mínimo P_X entre ativações.

Cada aplicação é associada com um conjunto **T** de **n** tarefas, isto é, o conjunto de todas as tarefas que pertencem a alguma atividade do conjunto **A**. Cada tarefa T_i é caracterizada por

uma prioridade global $\rho(T_i)$, um tempo máximo de execução C_i e um deadline D_i relativo ao instante de chegada da sua atividade. Para todas as tarefas T_i, $1 \leq i \leq n$, $T_i \in A_x$, temos que $D_i \leq P_x$. Se a tarefa T_i é uma tarefa inicial de sua atividade ela pode ter um *jitter* de liberação máximo J_i maior que zero. É suposto que se as tarefas T_i e T_k são ambas tarefas iniciais da atividade A_x, então $J_i = J_k$. Cada tarefa T_i é também associada a um conjunto $dPred(T_i)$. Este conjunto inclui todas as tarefas que são predecessoras diretas de T_i. A tarefa T_i não é liberada até receber uma mensagem de cada uma das suas predecessoras diretas.

Todas as tarefas T_i foram previamente alocadas aos processadores. Como resultado do algoritmo de alocação, é possível que tarefas de uma mesma atividade fiquem espalhadas por vários processadores. Uma tarefa não pode migrar durante a execução. O símbolo R_i será usado para denotar o tempo máximo de resposta da tarefa T_i, isto é, o intervalo de tempo entre o instante de chegada da sua atividade e o instante da conclusão de T_i no pior caso. O símbolo $H(T_i)$ será usado para denotar o processador onde T_i executa.

O modelo de tarefas adotado neste trabalho difere em parte do modelo de tarefas usado para definir o DASS em [DAV 93b]. Em particular, no modelo de tarefas usado em [DAV 93b] não existem relações de precedência e a aplicação executa em um único processador. Ambos os modelos possuem em comum o fato de trabalharem com prioridades fixas e tarefas que podem apresentar *jitter* na liberação. Esta seção mostra como o algoritmo DASS pode ser adaptado para o modelo de tarefas empregado neste trabalho.

Basicamente, o algoritmo DASS calcula qual a quantidade de interferência extra que cada tarefa pode receber sem aumentar o tempo máximo de resposta além do seu deadline. Isto significa que, ao aplicar o DASS original, o tempo máximo de resposta das tarefas poderá aumentar, mas nunca passar do respectivo deadline.

Na análise de escalonabilidade proposta em [OLI 98], a eliminação das relações de precedência entre diferentes processadores somente foi possível porque as mesmas foram substituidas por *jitter* na liberação da tarefa sucessora. *Jitter* de liberação foi adicionado em uma quantidade igual ao tempo máximo de resposta da tarefa predecessora.

Uma vez que o emprego do DASS, por aceitar algumas partes opcionais, aumenta o tempo máximo de resposta de uma tarefa, torna-se necessário aumentar também os *jitter* de liberação associados com estes tempos máximos de resposta e recalcular a escalonabilidade de todo o sistema. Isto é inviável na prática pois o fenômeno ocorre em tempo de execução e de forma distribuída. A aceitação de uma parte opcional no processador **p** vai aumentar os tempos de resposta das tarefas que executam em **p**. Por sua vez, isto vai aumentar o *jitter* na liberação de suas sucessoras remotas, dificultando o escalonamento das tarefas em um outro processador qualquer.

Por exemplo, considere a aplicação hipotética descrita pela figura 1. Caso a execução da parte opcional de T_6 aumente o tempo máximo de resposta de T_6, então seria aumentado o *jitter* na liberação de T_{15}. Com um *jitter* na liberação aumentado, o tempo máximo de resposta de T_{15} também seria maior. Em um efeito cascata, o mesmo acontece com T_{16}, tornando-se possível que o novo tempo máximo de resposta de T_{15} ou T_{16} venha a ser maior que o respectivo deadline. Uma aplicação do DASS considera o efeito local da interferência extra gerada, mas não considera o efeito propagado através das relações de precedência. Através de relações de precedência entre diferentes processadores teremos este efeito espalhado pelo sistema.

Este problema não ocorreria caso o *jitter* de liberação das tarefas sucessoras fosse definido em função do deadline da tarefa predecessora e não de seu tempo máximo de resposta. Como a aplicação do DASS não invalida os deadlines, o cálculo realizado em tempo de projeto continuaria válido. Entretanto, como o deadline de uma tarefa pode ser muito maior que o seu tempo máximo de resposta, estariamos aumentando o pessimismo da análise e possivelmente tornando um sistema escalonável em não escalonável.

Uma forma de contornar o problema é usar o tempo máximo de resposta no lugar do deadline nas equações que calculam a folga de uma tarefa que possui sucessores remotos. Em outras palavras, se $\exists T_j \in Suc(T_i).H(T_j) \neq H(T_i)$ então R_i será usado no lugar de D_i para o cálculo de $S_i(t)$. Esta informação (usar R_i ou D_i) depende apenas das propriedades estáticas da tarefa (suas relações de precedência) e pode ser anotada no descritor da tarefa, no momento que ela é criada.

Considerando a aplicação hipotética apresentada na figura 1, uma tarefa aperiódica somente poderá ser executada no processador 1 caso o tempo máximo de resposta de T_6 não seja afetado. Com isto estaremos garantindo que a tarefa T_{15} receberá a mensagem de T_6, no pior caso, como previsto na análise em tempo de projeto.

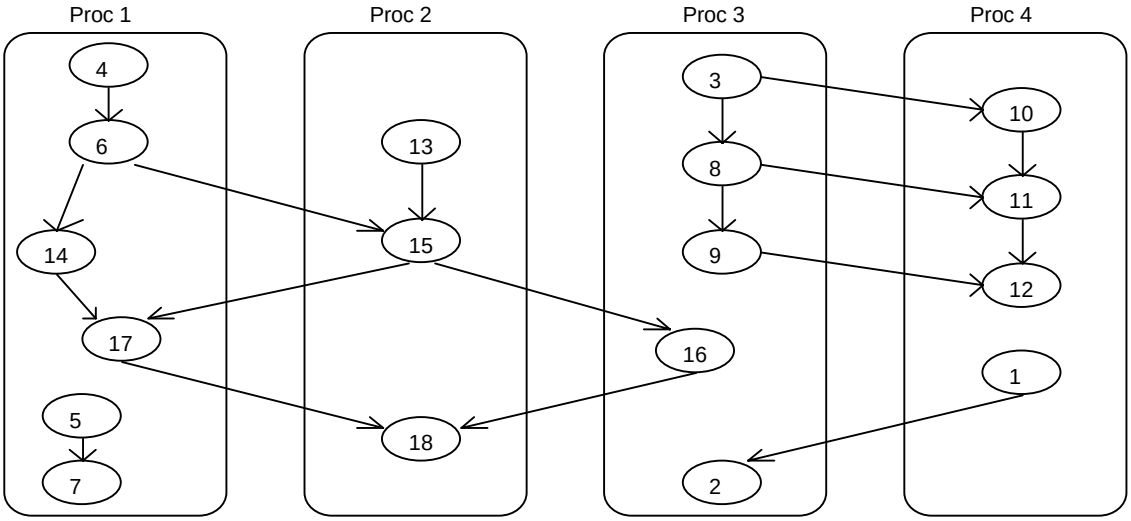

Figura 1 - Aplicação hipotética executada em 4 processadores.

Em resumo, no DASS modificado (MDASS), tarefas aperiódicas somente poderão executar caso a sua execução não altere o tempo máximo de resposta das tarefas locais com sucessores remotos, nem viole o deadline das demais tarefas críticas locais. Este comportamento difere do DASS normal, que aceita tarefas aperiódicas caso os tempos máximos de resposta sejam alterados, mas não acima do respectivo deadline.

Caso, no instante **t**, uma tarefa aperiódica T_a seja aceita pelo teste do MDASS para executar durante C_a unidades de tempo, isto significa que é possível executar uma computação com tempo máximo de execução C_a na prioridade **i**. Neste caso, é garantido que nenhum tempo máximo de resposta de tarefa com sucessor remoto será alterado e, por consequência, nenhum deadline será perdido em outros processadores.

A literatura que descreve o DASS também não discute as alterações necessárias para quando a aplicação inclui tarefas esporádicas. Neste caso, o único cuidado adicional refere-se ao cálculo do momento futuro no qual deverá ocorrer a próxima liberação da tarefa esporádica. Neste aspecto, a diferença básica entre tarefas periódicas e tarefas esporádicas é

que as tarefas periódicas possuem todos os seus momentos de chegada previamente fixados, ainda em tempo de projeto. Por exemplo, suponha que T_i é uma tarefa periódica com período P_i. Em qualquer instante **t** é possível determinar com certeza o próximo instante de chegada de T_i. A sua liberação ocorrerá antes caso seu *jitter* de liberação seja zero. Logo, $x_i(t)$ corresponde exatamente ao instante da próxima chegada de T_i, isto é, $\lceil t / P_i \rceil \times P_i$.

Suponha agora que T_j é uma tarefa esporádica. Neste caso, $x_j(t)$ deve indicar este momento de liberação futura sem que os instantes de chegada sejam conhecidos. Apenas os instantes de liberação passados são conhecidos. Suponha que, no instante **t**, o valor **y** representa o momento da última liberação de T_j. Obviamente, **y** é conhecido do escalonador. No pior caso, a liberação que ocorreu em **y** apresentou um *jitter* de liberação máximo. Isto vai permitir que a tarefa T_j sejá liberada novamente em $x_j(t) = y + P_j - J_j$.

5 Relações de Precedência e Identificação de Folgas

Parte da solução de escalonamento considerada neste trabalho é usar as folgas do sistema, uma vez garantidos os deadlines das tarefas críticas, para executar tarefas aperiódicas com deadline soft. É sempre possível utilizar estas folgas através da execução em *background*. Neste esquema, as tarefas aperiódicas recebem uma prioridade mais baixa que as prioridades de todas as tarefas críticas do sistema. Desta forma, as tarefas aperiódicas executam somente quando nenhuma tarefa crítica está liberada para executar.

Embora a execução em *background* eventualmente utilize toda folga que aparece no sistema, ela demora em fazê-lo [SPR 89]. Por exemplo, quando uma tarefa crítica é concluida com menos tempo de processador que o seu pior caso, a parte não usada poderia ser imediatamente aproveitada. Entretanto, na execução em *background*, este tempo ganho somente seria aproveitado quando a fila do processador não incluisse nenhuma tarefa crítica.

O objetivo dos vários métodos descritos na seção 2 é exatamente identificar as folgas no sistema tão logo elas apareçam. Desta forma, a execução das tarefas aperiódicas é antecipada. Em termos subjetivos é possível afirmar que a "quantidade de folga" é determinada pelas características das tarefas, enquanto o "instante da identificação da folga" depende do método utilizado. Quando as tarefas aperiódicas possuem deadline soft, o resultado de uma identificação antecipada de folgas resulta em tempo médio de resposta menor. Quando as tarefas aperiódicas possuem deadline firme, o resultado é uma maior taxa de tarefas aperiódicas com deadline atendido.

Considerando o modelo de tarefas empregado neste trabalho, temos que relações de precedência em ambiente distribuído geram *jitter* de liberação nas tarefas sucessoras. A existência de *jitter* de liberação, associada com o método descrito na seção 3, resulta na detecção antecipada de folgas no processador que hospeda uma tarefa sucessora, cuja predecessora executa em outro processador. Na equação que calcula a folga no nível **i** de prioridade, o *jitter* de liberação da tarefa T_i está incluído em $d_i(t)$, pois o mesmo deve ser tolerado. Entretanto, ele não aparece em $I_j(t,d_i(t))$, pois não é uma interferência. Isto resulta na imediata identificação do *jitter* de liberação como folga, isto é, um atraso na conclusão da tarefa T_i que, entretanto, não ocupa tempo de processador.

A transformação do *jitter* de liberação em folga pode ser ilustrada pelas tarefas T_1 e T_2 da aplicação descrita na figura 1. No instante zero é possível aplicar as equações de cálculo de folga e chegar ao valor para a folga de T_1. Como T_1 possui sucessor remoto, o seu tempo máximo de resposta é usado no lugar do deadline, como explicado na seção 4. Observe que o *jitter* de liberação intrínseco de T_1 será identificado como folga. Já a tarefa T_2 possui um *jitter*

de liberação maior, em função do tempo máximo de resposta de T_1. Ao mesmo tempo, T_2 não possui sucessores remotos, o que significa que é necessário preservar apenas o seu deadline e não o seu tempo máximo de resposta.

A alocação das tarefas aos processadores pode ser usada para influenciar a detecção de folgas em tempo de execução. A concentração de tarefas mais terminais em um processador faz com que as folgas neste processador sejam detectadas antes. Tarefa mais terminal significa uma tarefa mais próxima do final da atividade, ou seja, sujeita a um *jitter* de liberação maior. Caso seja interesse do projetista tornar a detecção de folgas homogênea no sistema, este deverá fazer uma distribuição homogênea das tarefas terminais pelos processadores.

Com o objetivo de explorar as propriedades que vinculam relações de precedência, *jitter* de liberação e detecção de folgas, foram realizadas uma série de experiências. Em cada experiência uma aplicação hipotética é simulada e a detecção de folgas é medida. As aplicações hipotéticas foram criadas com o objetivo de destacar determinados comportamentos. A próxima seção descreve algumas destas aplicações hipotéticas e os respectivos resultados medidos.

5.1 Descrição das Experiências

O objetivo das experiências é identificar os mecanismos que vinculam as relações de precedência à identificação de folgas. Foram testadas várias configurações neste sentido. Um dos principais objetivos é determinar se a concentração de *jitter* de liberação nas tarefas de um único processador é capaz de melhorar a identificação de folgas naquele processador.

As figuras 2 e 3 descrevem duas aplicações simuladas. A aplicação ESPALHADA, figura 2, é composta por 3 atividades lineares, com 3 tarefas cada uma, mais 3 tarefas críticas independentes. As atividades estão distribuídas de tal forma que cada processador possui uma tarefa inicial de atividade, uma tarefa intermediária de atividade, uma tarefa final de atividade e uma tarefa crítica independente. A tabela 1 contém os dados relativos às tarefas, onde C_i representa o tempo máximo de computação, P_i representa o período, D_i o deadline e R_i o tempo máximo de resposta calculado pelo algoritmo apresentado em [OLI 98]. Todas as tarefas são periódicas e a única fonte de *jitter* de liberação são as relações de precedência.

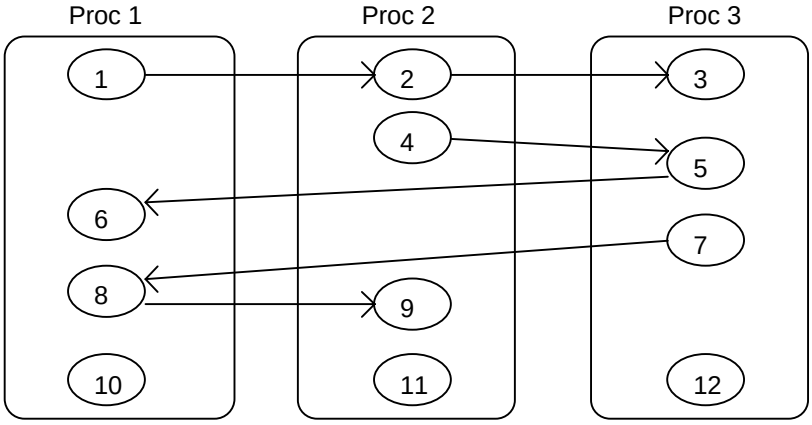

Figura 2 - Aplicação hipotética ESPALHADA, tarefas terminais espalhadas.

A aplicação CONCENTRADA, figura 3, é composta pelas mesmas atividades e tarefas da aplicação ESPALHADA, inclusive com os mesmos tempos de computação, deadline e período. A única diferença está na alocação das tarefas, a qual concentra no processador 1 todas as tarefas iniciais das atividades, no processador 2 todas as tarefas intermediárias e no

processador 3 todas as tarefas finais. Cada processador também deve executar uma tarefa crítica independente, como na aplicação anterior. A tabela 2 contém os dados relativos às tarefas e os cálculos realizados em tempo de projeto.

Tarefa	C_i	P_i	D_i	R_i
T_1	2	12	12	2
T_2	2	12	12	7
T_3	2	12	12	12
T_4	2	20	20	4
T_5	2	20	20	13
T_6	2	20	20	20
T_7	2	30	30	8
T_8	2	30	30	19
T_9	2	30	30	28
T_{10}	6	30	30	16
T_{11}	6	30	30	16
T_{12}	6	30	30	18

Tabela 1 - Características da aplicação hipotética ESPALHADA, Δ=3.

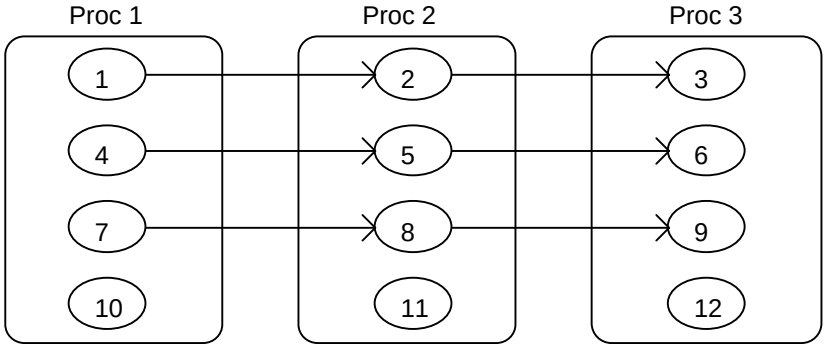

Figura 3 - Aplicação hipotética CONCENTRADA, tarefas terminais concentradas.

Tarefa	C_i	P_i	D_i	R_i
T_1	2	12	12	2
T_2	2	12	12	7
T_3	2	12	12	12
T_4	2	20	20	4
T_5	2	20	20	11
T_6	2	20	20	20
T_7	2	30	30	6
T_8	2	30	30	15
T_9	2	30	30	28
T_{10}	2	30	30	12
T_{11}	2	30	30	16
T_{12}	2	30	30	20

Tabela 2 - Características da aplicação hipotética CONCENTRADA, Δ=3.

A ferramenta utilizada para simular as aplicações mostra, passo a passo, qual tarefa ocupa o processador e quais são os valores na tabela de folgas. É suposto que todas as tarefas ocupam o processador pelo seu tempo máximo de execução e as tarefas esporádicas chegam com a freqüência máxima. A partir dos rastros ("traces") gerados pela simulação é possível construir gráficos que mostram a folga detectada em cada processador ao longo do tempo.

O resultado da execução da aplicação ESPALHADA durante um mínimo múltiplo comum dos períodos das tarefas é mostrado na figura 4. É possível observar que não existe diferença significativa entre os processadores, com respeito a folga detectada. Os patamares que surgem antes do instante 30 e do instante 60 são gerados porque as tarefas críticas independentes são executadas somente quando o seu deadline está próximo. Como estas tarefas possuem um tempo de execução igual a 6, cerca de 6 unidades de tempo antes dos instantes 30 e 60 estarão ocupadas com as tarefas críticas independentes e nenhuma folga será identificada nestes intervalos de tempo. Existem situações nas quais o patamar possui uma duração menor do que o esperado porque o MDASS é um método aproximado de detecção de folga. Isto é, em determinados momentos o algoritmo pode subestimar a folga existente e executar tarefas críticas antes do que seria realmente necessário.

A figura 5 mostra a folga identificada nos processadores quando a aplicação CONCENTRADA executa durante o mesmo período de tempo. É possível observar no gráfico que, no início, o processador 1 está ocupado executando as tarefas críticas iniciais das 3 atividades. Ao mesmo tempo, o *jitter* de liberação das tarefas intermediárias e finais das atividades fornecem, nos processadores 2 e 3, a possibilidade de detectar folgas mais cedo. As tarefas iniciais de atividades alocadas ao processador 1 formam um comboio de tarefas que devem ser executadas imediatamente, para preservar o seu tempo máximo de resposta, como descrito na seção 4. Depois que o processador 1 executa o comboio formado por T_1, T_4 e T_7, ele entra em um período de folga onde poderá executar tarefas aperiódicas.

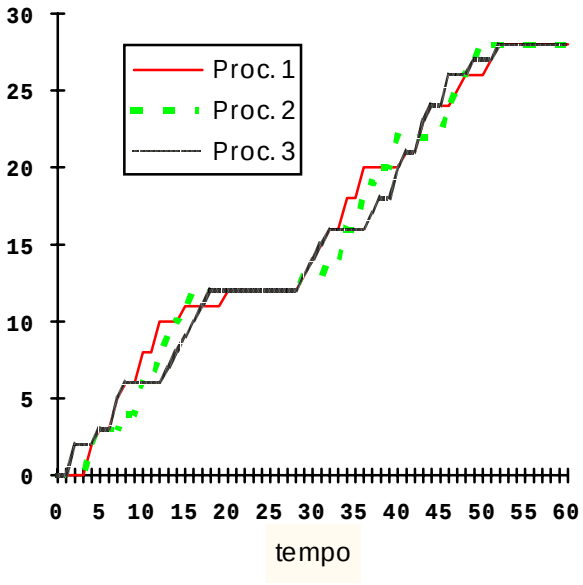

Figura 4 - Folga detectada, MDASS, na aplicação ESPALHADA.

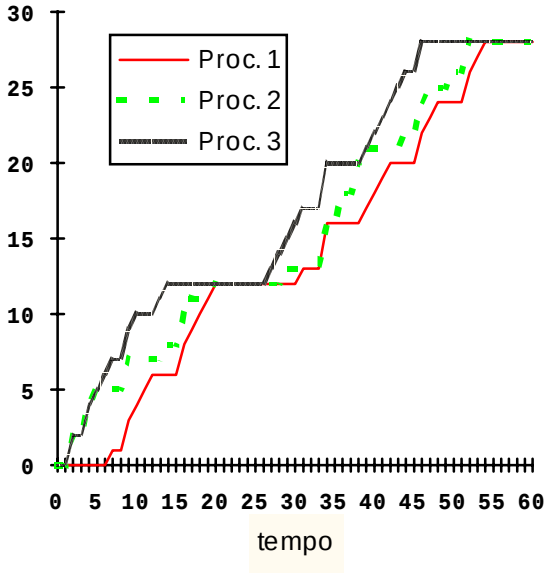

Figura 5 - Folga detectada, MDASS, na aplicação CONCENTRADA.

Este efeito comboio aparece no gráfico da figura 5, na forma de uma defasagem entre as curvas dos processadores. Esta defasagem desaparece quando aproxima-se o deadline das tarefas críticas independentes e estas precisam ser executadas. Ao final do mínimo múltiplo comum dos períodos todos os processadores terão necessariamente igualado a quantidade de folga detectada. Como dito antes, o mecanismo de detecção de folgas afeta o momento da detecção, mas a quantidade é determinada pela carga no sistema.

Em qualquer caso, os resultados do MDASS são muito melhores do que uma execução em *background* das tarefas aperiódicas. Os gráficos das figuras 6 e 7 mostram a identificação de folgas nas aplicações ESPALHADA e CONCENTRADA, respectivamente, quando execução em *background* é usada. É possível observar que folga em um determinado processador somente é identificada depois que todas as tarefas prontas para executar estiverem concluidas, isto é, quando o processador estiver livre. Neste contexto, o processador 3 na aplicação CONCENTRADA detecta folgas mais cedo. Após executar a tarefa crítica independente local, o processador 3 fica livre até que a execução das atividades evolua até as respectivas tarefas finais, ali alocadas. Enquanto isto não acontece, o processador 3 é usado para executar tarefas aperiódicas.

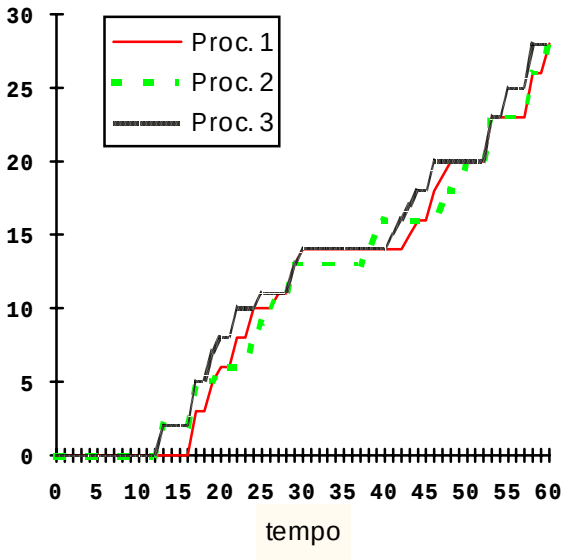

Figura 6 - Folga detectada, em *background*, na aplicação ESPALHADA.

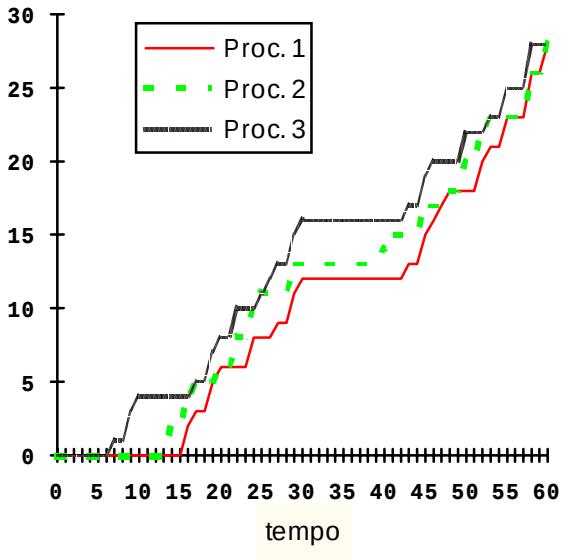

Figura 7 - Folga detectada, *background*, na aplicação CONCENTRADA.

6 Considerações Gerais Sobre o MDASS

Esta seção resume as conclusões que resultaram das experiências descritas na seção anterior. Diversas experiências foram realizadas variando o número de atividades, a forma como elas são alocadas e os tempos de computação das tarefas. Os próximos parágrafos contêm as observações que resultaram destas experiências.

Relações de precedência remotas causam *jitter* de liberação, o qual permite uma identificação antecipada de folgas. Pode-se esperar que um sistema com muitas relações de precedência remotas irá atender melhor suas tarefas aperiódicas do que um sistema onde as relações de precedência são apenas locais ou inexistam. É importante destacar que, em contrapartida, as relações de precedência remotas implicam em atrasos na rede de comunicação que acabam por aumentar o tempo máximo de resposta das atividades.

Como descrito na seção anterior, a concentração em um mesmo processador de tarefas críticas sem predecessores mas com sucessores remotos acaba por criar um efeito comboio onde o processador deve primeiro executar todas estas tarefas críticas para depois executar tarefas aperiódicas. Ao mesmo tempo, os demais processadores aproveitam o benefício do *jitter* de liberação causado por estas mesmas precedências para antecipar a execução das tarefas aperiódicas. Um desempenho mais homogêneo com respeito ao atendimento de tarefas aperiódicas pode ser obtido com uma distribuição equilibrada das tarefas iniciais das atividades pelos processadores do sistema.

Como descrito na seção 4, a folga das tarefas que possuem sucessores remotos é calculada com respeito ao seu tempo máximo de resposta, enquanto a folga das demais tarefas considera o deadline. Em um sistema escalonável o deadline é sempre maior ou igual ao tempo máximo de resposta. Logo, existe uma antecipação maior de folgas quando a tarefa não possui sucessores remotos. Um processador que executa apenas tarefas sem sucessores remotos será mais eficiente na detecção das folgas do que um processador onde existem tarefas deste tipo.

Suponha duas tarefas T_i e T_j que executam em processadores diferentes e T_i é predecessora de T_j. O cálculo em projeto do tempo máximo de resposta de T_j já inclui qualquer atraso relativo à relação de precedência com T_i. Quando o MDASS é usado, a execução de T_j será retardada até o máximo instante possível, com o objetivo de antecipar a execução das tarefas aperiódicas. É importante observar que, neste caso, o comportamento de T_i em tempo de execução não é importante para o escalonamento no processador $H(T_j)$. O escalonamento em $H(T_j)$ será feito como se T_i gerasse o maior atraso possível, independentemente do comportamento de T_i. A razão para isto é que, ao supor o maior atraso possível devido a T_i, a execução de T_j é retardada em benefício das tarefas aperiódicas.

É importante observar que várias fontes de capacidade ociosa presentes em um sistema são aproveitadas pelo MDASS. Por exemplo, sempre que uma tarefa com prioridade maior que T_i ocupa o processador por um tempo menor que o pior caso, ela gera sobre T_i uma interferência menor que aquela considerada para o cálculo do tempo máximo de resposta de T_i. Esta interferência já computada poderá ser então gerada por alguma tarefa aperiódica, sem alterar a escalonabilidade de T_i. O mesmo acontece quando uma tarefa é ativada com uma frequência menor do que a máxima prevista.

O algoritmo MDASS foi implementado para a realização dos experimentos. As estruturas de dados necessárias para implementar o MDASS são mínimas. Além das propriedades estáticas das tarefas (período, tempo máximo de execução, etc), é necessário apenas manter a quantidade de folga $S_i(t)$ de cada tarefa e o quanto cada tarefa já executou da presente liberação $c_i(t)$. No caso das tarefas esporádicas, é necessário também o instante da última liberação. Todos os demais valores podem ser calculados a partir destes.

As alterações realizadas no DASS para adaptá-lo ao modelo de tarefas deste trabalho não aumentaram o seu custo computacional. As mudanças aconteceram nas equações e não no algoritmo de cálculo. Assim, as observações feitas em [DAV 93b] sobre o custo computacional do algoritmo DASS são válidas também para o MDASS.

7 Conclusões

Neste artigo foi considerado o problema da execução de tarefas aperiódicas com deadline soft quando juntamente com tarefas críticas, conforme o modelo de tarefas descrito em [OLI 98]. Foram consideradas diversas abordagens presentes na literatura: baseadas em

servidores, tomada estática de folga e tomada dinâmica de folga. Este trabalho adota como ponto de partida a abordagem da tomada dinâmica de folga aproximada (DASS).

O modelo de tarefas original do DASS difere daquele adotado por este trabalho. Logo, foi necessário descrever como o DASS pode ser utilizado em nosso contexto. Tarefas esporádicas com *jitter* de liberação recebem tratamento especial no momento do cálculo da folga. Tarefas com sucessores remotos precisam preservar o seu tempo máximo de resposta, para evitar que o atraso propagado através das relações de precedência faça alguma tarefa remota perder o seu deadline. O método resultante da modificação do DASS foi chamado de MDASS. A quantidade de memória necessária para a implementação do algoritmo DASS modificado não é preocupante, pois as tabelas necessárias em cada processador são proporcionais ao número de tarefas alocadas ao respectivo processador.

Foram realizadas diversas experiências com o objetivo de investigar o relacionamento entre relações de precedência e a identificação de folgas. Relações de precedência entre tarefas alocadas a diferentes processadores resultam em *jitter* de liberação na tarefa sucessora. Ao mesmo tempo, a existência de *jitter* de liberação é aproveitada pelo MDASS para detectar folgas. Como resultado, processadores que hospedam tarefas com maior *jitter* de liberação conseguem detectar folgas mais cedo. A seção 5.1 apresentou alguns gráficos que ilustram este efeito. A seção 6 resume as observações realizadas.

Na medida em que, cada vez mais, aplicações tempo real envolvem aspectos críticos juntamente com diversos aspectos não críticos, as técnicas de escalonamento mistas serão cada vez mais utilizadas na construção de sistemas tempo real. Isto é válido tanto para sistemas distribuídos como para processamento paralelo e processamento centralizado.

8 Referências

[AUD 94a] N. C. Audsley, A. Burns, R. I. Davis, A. J. Wellings. Integrating Best Effort and Fixed Priority Scheduling. Proceedings of the IEEE Real-Time Systems Symposium, pp. 12-21, San Juan, Puerto Rico, december 1994.

[AUD 94b] 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 96] A. Burns, A. J. Wellings. Dual Priority Scheduling in ADA 95 and Real-Time Posix. Proceedings of the 21st IFAC/IFIP Workshop on Real Time Programming, pp. 45-50, Gramado-RS-Brazil, november 1996.

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

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

[LEH 87] J. P. Lehoczky, L. Sha, J. K. Strosnider. Enhanced Aperiodic Responsiveness in Hard-Real-Time Environments. Proceedings of the IEEE Real-Time Systems Symposium, San Jose-CA, pp.261-270, 1987.

[LEH 92] J. P. Lehoczky, S. Ramos-Thuel. An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems. Proceedings of the IEEE Real-Time Systems Symposium, pp. 110-123, 1992.

[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.

[OLI 98] R. S. Oliveira, J. S. Fraga. Escalonamento de Tarefas com Relações Arbitrárias de Precedência em Sistemas Tempo Real Distribuídos. 16o Simpósio Brasileiro de Redes de Computadores - SBRC'98. Rio de Janeiro-RJ, maio de 1998.

[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.

[THU 93] S. Ramos-Thuel, J. P. Lehoczky. On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems. Proceedings of the IEEE Real-Time Systems Symposium, pp. 160-171, 1993.

[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.

[TIA 94] T.-S. Tia, J. W. S. Liu, M. Shankar. Aperiodic Request Scheduling in Fixed-Priority Preemptive Systems. Department of Computer Science Technical Report #1859, University of Illinois at Urbana-Champaign, 1994.