Uma Solução Mista para o Escalonamento Baseado em Prioridades de Aplicações Tempo Real Críticas

Rômulo Silva de Oliveira* e Joni da Silva Fraga

Laboratório de Controle e Microinformática - LCMI-EEL-UFSC
Campus Universitário - Trindade - Florianópolis - SC
Caixa Postal 476 - CEP 88040-900
E-mail: romulo@lcmi.ufsc.br e fraga@lcmi.ufsc.br

Resumo

Sistemas computacionais de tempo real críticos são identificados como aqueles submetidos a requisitos de natureza temporal onde uma previsibilidade determinista é necessária. Neste tipo de sistema é muitas vezes empregado escalonamento baseado em prioridades. Este artigo apresenta uma solução de escalonamento tempo real baseada em prioridades que emprega, simultaneamente, prioridades fixas e variáveis. Tarefas periódicas ou esporádicas, com release jitter e deadline menor ou igual ao período, recebem prioridades fixas. Tarefas periódicas, com deadline igual ao período, recebem uma prioridade menor que a do primeiro conjunto e são escalonadas segundo EDF. Foi desenvolvido um teste de escalonabilidade suficiente mas não necessário para o modelo. Um exemplo numérico foi usado para ilustrar sua aplicação.

Abstract

Hard Real-Time Computing Systems are systems that have timing constraints and require a deterministic predictability. Priority-based scheduling is often used in this kind of system. This paper presents a priority-based real-time scheduling solution that applies, at the same time, fixed and variable priorities. Periodic or sporadic tasks, with release jitter and deadline less than or equal to the period, receive fixed priorities. Periodic tasks, with deadline equal to the period, receive a lower priority compared to the first set and are scheduled by EDF. It was developed a sufficient but not necessary schedulability test for this model. A numeric example was used to show its application.

1. Introdução

Sistemas computacionais de tempo real (STR) são identificados como aqueles submetidos a requisitos de natureza temporal. Em geral, requisitos temporais são expressos através de deadlines (prazo máximo para execução) associados com as reações do sistema à estímulos externos. A dificuldade de escalonar tarefas com requisitos de tempo real é bastante conhecida, constituindo uma área de pesquisa intensa.

O termo previsibilidade ("predictability") é utilizado para descrever a capacidade de se conhecer o comportamento temporal de um sistema antes de sua execução, em função do escalonamento empregado. Na literatura, a noção de

*Professor do Instituto de Informática da UFRGS, em doutoramento no LCMI-UFSC.

previsibilidade é associada com uma antecipação determinista (todos os deadlines serão cumpridos) ou com uma antecipação probabilista (qual a probabilidade de um deadline ser cumprido) do comportamento temporal.

Aplicações tempo real críticas são aquelas onde falhas de natureza temporal são consideradas críticas no que diz respeito às suas consequências. Nestes sistemas uma falha temporal, ou seja, a perda de um deadline, pode representar a perda de vidas ou enormes danos materiais ou ao ambiente.

Aplicações tempo real críticas demandam uma previsibilidade determinista. Este tipo de previsibilidade pode ser obtido, entre outras formas, através do escalonamento baseado em prioridades ([LIU 73]). 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.

As soluções baseadas em prioridades podem ser divididas em prioridades fixas e prioridades variáveis. Quando prioridades fixas são usadas, cada tarefa recebe em projeto uma prioridade que vale durante toda a vida do sistema. Políticas clássicas para atribuir prioridades fixas são o taxa monotônica ("rate monotonic", [SHA 94]) e o deadline monotônico ("deadline monotonic", [AUD 93a]). Também existem algorítmos para atribuir prioridades em modelos de tarefas mais complexos ([AUD 93c]).

A política mais usada para atribuir prioridades variáveis é o EDF ("earliest deadline first", [LIU 73], [CHE 89]). Em geral, EDF prove um resultado melhor que prioridade fixa, no sentido que aplicações inviáveis usando prioridades fixas podem ser escalonadas com EDF. Entretanto, os testes de escalonabilidade existentes para prioridade fixa admitem modelos de tarefas muito mais flexíveis que os existentes para EDF. O resultado é que nenhuma abordagem consegue ser absolutamente melhor que a outra.

Neste trabalho é apresentada uma solução de escalonamento tempo real também baseada em prioridades premptivas. Ao contrário das abordagens descritas antes, vamos empregar prioridades fixas e variáveis simultaneamente no sistema. A solução proposta apresenta vantagens tanto de prioridades fixas como de prioridades variáveis. Este trabalho considera apenas aplicações tempo real críticas, onde a garantia em tempo de projeto é uma necessidade imprescindível.

O restante do texto está organizado da seguinte maneira: a seção 2 descreve a motivação para este trabalho através de um exemplo numérico; a seção 3 faz um esboço da solução proposta; a seção 4 formaliza o problema; a seção 5 desenvolve um teste de escalonabilidade apropriado e ilustra sua aplicação; a seção 6 contém as considerações finais.

2. Motivação

Muitas aplicações práticas apresentam uma mistura de tarefas periódicas com tarefas esporádicas. Tarefas periódicas são liberadas uma vez a cada período. Tarefas esporádicas não possuem uma frequência fixa de liberação, mas sim um intervalo mínimo de tempo entre liberações. Tanto tarefas esporádicas quanto tarefas periódicas caracterizam uma carga estática, o que permite uma previsibilidade determinista para o conjunto.

Por exemplo, em sistemas de controle é normal usar tarefas periódicas para manter laços de controle e outras funções contínuas do sistema, enquanto tarefas esporádicas são responsáveis por funções de emergência ou comandos do operador.

Outro aspecto importante é o *release jitter**. Ele acontece sempre que existem tarefas esporádicas e o sistema somente amostra eventos externos em momentos pré-determinados (por exemplo, a cada tick do relógio). Suponha uma tarefa esporádica **T** com intervalo mínimo **I** entre liberações. Suponha ainda que o sistema somente amostre os eventos externos a cada **J** unidades de tempo. Se o evento associado com a liberação da tarefa **T** ocorrer imediatamente após um tick do relógio, ele somente será reconhecido, para efeitos de escalonamento, no próximo tick. Este atraso torna possível que dois eventos consecutivos sejam reconhecidos pelo sistema em um intervalo de tempo **I-J**, menor que o valor suposto **I**. Nesta situação, é dito que esta tarefa possui um jitter na liberação de **J** unidades de tempo. Uma discussão sobre o conceito de *release jitter* pode ser encontrada em [TIN 94].

Considere uma aplicação composta por uma tarefa esporádica T_1, responsável por uma função de emergência, e duas tarefas periódicas T_2 e T_3, responsáveis pelas funções normais do sistema. Suas características temporais aparecem abaixo, seguindo a notação:

C: tempo máximo de computação;
P: período ou intervalo mínimo entre liberações;
D: deadline relativo ao início da liberação;
J: release jitter máximo.

Tarefa T_1, esporádica:
$C_1=1,\ P_1=100,\ D_1=2,\ J_1=1$
Tarefa T_2, periódica:
$C_2=5,\ P_2=10,\ D_2=10,\ J_2=0$
Tarefa T_3, periódica:
$C_3=6,\ P_3=15,\ D_3=15,\ J_3=0$

Vamos inicialmente tentar escalonar usando prioridades fixas. Podemos usar o teste de escalonabilidade exato proposto em [AUD 93b] para calcular o tempo máximo de resposta R_i de cada tarefa T_i. Inicialmente o valor W_i é calculado iterativamente, a partir de um valor inicial $W_i^0=C_i$, pela equação:

$$W_i^{k+1} = C_i + \sum_{\forall T_j \in hp(T_i)} \left\lceil (J_j + W_i^k) \div P_j \right\rceil \times C_j \ ,$$

* Para efeito de clareza, alguns termos em ingles, de uso corrente na comunidade científica brasileira, não serão traduzidos.

onde hp(T_i) é o conjunto de tarefas com prioridades maiores que T_i. Em seguida, o valor R_i é calculado por: $R_i = J_i + W_i$. Como obviamente a tarefa T_1 será a de maior prioridade, vamos tentar as duas soluções possíveis.

Ordem das prioridades: T_1, T_2, T_3

$R_1 = 2$, $R_1 \leq D_1$, T_1 é escalonável;
$R_2 = 6$, $R_2 \leq D_2$, T_2 é escalonável;
$R_3 = 17$, $R_3 > D_3$, T_3 NÃO é escalonável.

Ordem das prioridades: T_1, T_3, T_2

$R_1 = 2$, $R_1 \leq D_1$, T_1 é escalonável;
$R_3 = 7$, $R_3 \leq D_3$, T_3 é escalonável;
$R_2 = 12$, $R_2 > D_2$, T_2 NÃO é escalonável.

Como o teste aplicado é exato e esgotamos as possibilidades de atribuição de prioridades, não existe solução com prioridades fixas para esta aplicação.

Vamos agora tentar escalonar usando EDF. Podemos usar o teste de escalonabilidade apresentado em [CHE 90]. Este teste calcula a utilização total do processador. O problema dos testes de escalonabilidade para EDF é que eles consideram a utilização de cada tarefa no pior caso. A tarefa T_1 acaba tendo uma utilização de 100% em função do release jitter e do deadline muito menor que seu período. No final temos o seguinte somatório de utilizações:

$1 \div 1 + 5 \div 10 + 6 \div 15 = 1 + 0.5 + 0.4 = 1.9$

A conclusão é que o sistema não é escalonável. Entretanto, isto deve-se ao fato de que o teste para o EDF não é apropriado para tarefas esporádicas. Na próxima seção será apresentada uma proposta de escalonamento misto para estas situações.

3. Solução Proposta

No exemplo numérico da seção 2, podemos observar que as tarefas T_2 e T_3 são escalonáveis pelo EDF. A soma de suas utilizações é apenas 0.9. O problema é a tarefa T_1, que é melhor atendida pela política de prioridades fixas.

Vamos agora propor uma solução mista, baseada em prioridades, para sistemas como o apresentado na seção anterior. A idéia básica é criar dois conjuntos de tarefas, **H** e **L**. As tarefas do conjunto **H** são escalonadas segundo prioridades fixas. As tarefas do conjunto **L** somente executam quando não existe nenhuma tarefa de **H** pronta para executar. Em outras palavras, todas as tarefas do conjunto **L** possuem prioridade inferior às tarefas de **H**. Entre as tarefas do conjunto **L** é usado EDF para decidir quem executa antes.

A escalonabilidade das tarefas do conjunto **H** pode ser verificada de forma completamente independente do conjunto **L**. Isto decorre do fato das tarefas de **L** jamais interferirem com as tarefas de **H**. Desta forma, qualquer método de atribuição de

prioridades fixa e seu respectivo teste podem ser usados para mostrar a escalonabilidade das tarefas de **H**. A seção 5 descreve um teste de escalonabilidade apropriado para as tarefas de **L**.

No exemplo numérico da seção 2, obviamente a tarefa T1 receberia uma prioridade fixa, executando tão logo chegue. As tarefas T2 e T3 formariam o conjunto **L** e seriam escalonadas com EDF. Desta forma temos um tratamento privilegiado para a tarefa de emergência enquanto as tarefas periódicas são escalonadas de forma mais eficiente com o EDF.

Uma vantagem adicional do esquema misto é prover um certo controle no caso de falhas do sistema. Especificamente, pode-se escolher as tarefas críticas do sistema para compor o conjunto de prioridades fixas, garantindo-se assim que elas serão as últimas a perder um deadline, caso isto venha a acontecer. Alguns autores ([LEH 89], [AUD 93a]) colocam esta característica como uma das principais vantagens de prioridade fixa sobre prioridades variáveis.

Liu e Layland em [LIU 73] já propuseram uma abordagem mista, mas para quando todas as tarefas são periódicas, com deadline igual ao período, e as tarefas com prioridade fixa possuem os períodos mais curtos. Nosso problema difere no sentido que as tarefas com prioridade fixa podem ser esporádicas, possuir release jitter e deadline menor ou igual ao período. Também em nosso modelo as tarefas de prioridade fixa não precisam possuir período menor.

Em outro trabalho ([JEF 93]), Jeffay e Stone empregam uma abordagem mista para compatibilizar tratadores de interrupções de hardware com a execução de tarefas normais. Enquanto os tratadores de interrupções são acionados conforme uma política de prioridades fixas, tarefas normais empregam EDF. Naquele trabalho é suposto que todas as interrupções são periódicas e ocorrem em fase. Também é exigido que a ocupação do processador seja inferior a 100%. Novamente, nosso problema difere no sentido que as tarefas com prioridade fixa podem ser esporádicas e possuir release jitter. Também em nosso modelo a ocupação do processador pode chegar a 100%. É importante observar ainda que nosso modelo não impede que as tarefas de prioridade fixa sejam, na verdade, tratadores de interrupções.

Não é do conhecimento dos autores outros trabalhos onde uma abordagem mista seja empregada para resolver a classe de problemas que é apresentada aqui.

4. Formulação do Problema

Esta seção descreve as suposições feitas para o desenvolvimento do teste de escalonabilidade, na próxima seção.

Uma aplicação **A** é formada por um conjunto de **n+m** tarefas, independentes entre si, executando em um único processador. Qualquer tarefa pode ser preemptada, ou seja, ter sua execução suspensa e retomada mais tarde.

As tarefas T_1 a T_n são executadas segundo uma política de prioridades fixas. Cada tarefa T_i, $1 \leq i \leq n$, é caracterizada por um release jitter máximo J_i, um tempo de execução no pior caso C_i, um período (tarefa periódica) ou um intervalo mínimo entre liberações (tarefa esporádica) P_i e um deadline relativo ao instante de liberação D_i. Para todas as tarefas T_i, $1 \leq i \leq n$, temos que $D_i \leq P_i$.

As tarefas T_{n+1} a T_{n+m} são executadas segundo o algorítmo EDF, somente quando o processador não está ocupado com as tarefas T_1 a T_n. Cada tarefa T_j, onde $n+1 \leq j \leq n+m$, é caracterizada por um tempo de execução no pior caso C_j, um período (tarefa periódica) P_j e um deadline relativo ao instante de liberação D_j. Para todas as tarefas T_j, $n+1 \leq j \leq n+m$, temos que $D_j = P_j$. A primeira liberação de cada tarefa deste grupo acontece no instante zero de tempo. Liberações subsequentes de cada tarefa acontecem a seguir no início dos respectivos períodos.

Apenas para efeito de notação, vamos definir os dois subconjuntos das tarefas pertencentes à aplicação. Por definição,

$H = \{ T_i \mid 1 \leq i \leq n \}$ e

$L = \{ T_j \mid n+1 \leq j \leq n+m \}$. Obviamente,

$A = H \cup L$.

Com respeito a escalonabilidade da aplicação, é suposto ainda que o conjunto **H** é escalonável quando executado isoladamente. Por exemplo, a análise de escalonabilidade descrita em [AUD 93b] pode ser usada para mostrar quando, para qualquer tarefa T_i, $T_i \in H$, todas as liberações serão concluídas antes do respectivo deadline.

Também é suposto que o conjunto **L** é escalonável quando executado isoladamente. Por exemplo, a análise descrita em [CHE 90] é capaz de mostrar quando todas as liberações serão concluídas antes do respectivo deadline.

5. Análise da Escalonabilidade

Nesta seção será desenvolvido um teste de escalonabilidade para o modelo de tarefas proposto. Inicialmente é mostrado que, a partir das suposições feitas, o conjunto **H** é escalonável dentro da aplicação **A**. Em seguida é desenvolvido um teste especificamente para tarefas do conjunto **L**.

Teorema 1

Qualquer tarefa $T_i \in H$ é escalonável em **A**.

Prova

Foi suposto que o conjunto **H** é escalonável quando executado isoladamente. Sabemos também que as tarefas do conjunto **L** não geram interferência sobre as tarefas do conjunto **H**. Logo, as tarefas do conjunto **H** continuam escalonáveis dentro da aplicação **A**.

□

Teorema 2

Sejam $T_i \in H$ e $T_j \in L$ tarefas. Seja T_j^k uma liberação qualquer da tarefa T_j. A interferência máxima de T_i sobre T_j^k acontece quando ocorre uma liberação de T_i simultaneamente com a liberação de T_j^k.

Prova

Existe uma relação de prioridade fixa entre T_i e T_j, onde T_j possui a menor prioridade. Desta forma, podemos usar um raciocínio idêntico ao do teorema do instante crítico [LIU 73] para afirmar* que a interferência máxima de T_i sobre T_j^k acontece quando ocorre uma liberação de T_i simultânea com T_j^k.

□

Neste ponto vamos introduzir a notação f_j^k para representar o tempo máximo de resposta de T_j^k, em relação ao seu instante de liberação. Segundo esta notação, o sistema será escalonável se e somente se $\forall T_x, \forall T_x^y,\ f_x^y \le D_x$.

Existe uma realimentação positiva entre a interferência de T_i sobre T_j^k e o tempo de resposta de T_j^k. Quanto maior for a interferência de T_i sobre T_j^k, maior será o tempo de resposta f_j^k. Ao mesmo tempo, quanto maior for o tempo de resposta f_j^k, devido em parte às outras tarefas da aplicação, maior o número de liberações de T_i que ocorrerão antes da conclusão de T_j^k, aumentando sua interferência sobre T_j^k.

Podemos calcular um limite ("upper bound") para a interferência de T_i sobre T_j^k supondo $f_j^k = D_j$, ou seja, supondo que a liberação T_j^k é concluída exatamente no instante do seu deadline. É preciso ressaltar que esta é uma suposição pessimista, pois é bem possível que $f_j^k < D_j$ mesmo no pior caso. Esta suposição pessimista vai gerar um teste suficiente mas não necessário para a escalonabilidade do sistema.

Teorema 3

Um limite superior para a interferência máxima de T_i sobre T_j^k, denotado por $I_j^k(i)$, é dado por

$$I_j^k(i) = \left\lfloor (J_i + D_j) \div P_i \right\rfloor \times C_i + \min(C_i, D_j - \left\lfloor (J_i + D_j) \div P_i \right\rfloor \times P_i) \ . \qquad \text{[Equação 1]}$$

Prova

Esta equação foi adaptada diretamente de [AUD 93d], com a inclusão do efeito do release jitter como descrito em [AUD 93b]. □

A equação 1 considera o número máximo de liberações de T_i que podem ocorrer durante a execução de T_j^k. A última liberação de T_i pode não caber inteira dentro de T_j^k. É importante observar que o valor $I_j^k(i)$ depende apenas das características temporais das tarefas T_i e T_j. Ele não depende das demais tarefas da aplicação. Esta propriedade é importante pois vai permitir que o conjunto **H** e **L** possam ser alterados livremente sem mudar o valor $I_j^k(i)$.

Teorema 4

---

* Devido a limitação de espaço não vamos reproduzir aqui a prova do teorema do instante crítico. O leitor é convidado a ler esta prova em [LIU 73].

Seja T_i uma tarefa tal que $T_i \in H$. Seja T_j uma tarefa tal que $T_j \in L$. Seja T_x uma tarefa tal que $T_x \in L$, com $P_x=D_x=P_j=D_j$ e $C_x = I_j^k(i)$. Nestas condições, a interferência máxima de T_x sobre T_j^k é igual ao limite superior da interferência máxima de T_i sobre T_j^k, ou seja, igual a $I_j^k(i)$.

Prova

Como $T_x \in L$ e $T_j \in L$, as tarefas T_x e T_j são escalonadas segundo o EDF. Como $P_x = P_j$, teremos uma liberação T_x^k simultânea com T_j^k. Como as tarefas T_x e T_j possuem deadlines iguais, a decisão sobre a ordem de execução é arbitrária. No pior caso, T_x^k executa antes, gerando sobre T_j^k uma interferência C_x, ou seja, uma interferência igual a $I_j^k(i)$.

□

Neste ponto vamos introduzir a notação $T_{j,i}$ para representar a tarefa periódica que, como mostrado no teorema 4, causaria sobre T_j uma interferência igual ao limite superior da interferência máxima causada por T_i, ou seja, $I_j^k(i)$. Vamos ainda definir o conjunto H_x da seguinte forma:

$H_0 = H$,
$H_x = H - \{ T_1, T_2, ..., T_x\}$, onde $1 \leq x \leq n$,

ou seja, H_x representa o conjunto H após serem retiradas as tarefas de T_1 a T_x. De forma semelhante, definimos o conjunto $L_{j,x}$ como:

$L_{j,0} = L$,
$L_{j,x} = L \cup \{ T_{j,1}, T_{j,2}, ..., T_{j,x}\}$, onde $1 \leq x \leq n$,

ou seja, $L_{j,x}$ representa o conjunto L acrescido de tarefas que vão gerar uma interferência sobre T_j igual ou maior do que a interferência máxima que as tarefas T_1 a T_x , do conjunto H, geram sobre Tj.

Podemos agora enunciar os dois teoremas a respeito da escalonabilidade do conjunto modificado de tarefas.

Teorema 5

Se a tarefa T_j é escalonável dentro do conjunto $H_x \cup L_{j,x}$, então ela será escalonável dentro do conjunto $H_{x-1} \cup L_{j,x-1}$.

Prova

Temos $H_x \cup L_{j,x} = \{ T_{x+1}, ..., T_n \} \cup L \cup \{ T_{j,1}, ..., T_{j,x-1}, T_{j,x} \}$
e $H_{x-1} \cup L_{j,x-1} = \{ T_x, T_{x+1}, ..., T_n \} \cup L \cup \{ T_{j,1}, ..., T_{j,x-1} \}$.

A única diferença está na presença de $T_{j,x}$ no primeiro conjunto e T_x no segundo conjunto. Sabemos que a interferência sobre qualquer T_j^k causada por $T_{j,x}$ é igual ao limite superior para a interferência causada por T_x, ou seja, $I_j^k(x)$.

Como a interferência máxima sobre qualquer T_j^k causada por $T_{j,x}$ é sempre maior ou igual a interferência máxima causada por T_x, temos que se T_j for escalonável

no primeiro conjunto (inclui $T_{j,x}$ com interferência $I_j^k(x)$) também o será no segundo conjunto (inclui T_x com interferência menor ou igual a $I_j^k(x)$). □

Teorema 6

Se a tarefa T_j é escalonável dentro do conjunto $L_{j,n}$, então ela será escalonável dentro do conjunto $H \cup L$.

Prova

Consequência direta da aplicação consecutiva do teorema 5, para **x** variando de **n** até **1**.

□

Como consequência do teorema 6, podemos aplicar o teste de escalonabilidade para EDF descrito em [LIU 73] ou [CHE 89]. A tarefa Tj será escalonável se

$$\sum_{T_x \in L_{j,n}} (C_x \div P_x) \leq 1. \qquad \text{[equação 2]}$$

Este teste é suficiente mas não necessário porque a interferência causada pelas tarefas de maior prioridade, representada por C_x, é superestimada pela equação 1.

O teste acima verifica apenas a escalonabilidade de T_j. É necessário repetir agora todo o procedimento para as outras **m-1** tarefas de **L**. O algorítmo a seguir mostra como a escalonabilidade de todas as tarefas de **L** pode ser testada.

[1] Para cada tarefa T_j, $T_j \in L$, faça
[2] Para cada tarefa T_i, $T_i \in H$, faça
[3] Calcule $I_j^k(i)$ conforme a equação 1
[4] Calcule a utilização da tarefa $T_{j,i}$, dada por $I_j^k(i) \div P_j$
[5] Aplica a equação 2 sobre o conjunto $L_{j,n} = L \cup \{T_{j,1},...,T_{j,n}\}$

Basicamente, este algorítmo analisa a escalonabilidade das tarefas $T_j \in L$, uma de cada vez (linha 1). Para cada T_j em particular, percorre todo o conjunto **H** (linha 2), calculando o limite superior para a interferência máxima de $T_i \in H$ sobre o T_j em questão (linha 3). Este valor permite determinar a utilização da tarefa $T_{j,i}$ correspondente (linha 4). Uma vez que todas as tarefas T_i foram analisadas com respeito ao T_j em questão, as utilizações de todas as tarefas $T_{j,i}$ são conhecidas e a equação 2 pode ser aplicada sobre o conjunto $L_{j,n}$ (linha 5) para determinar se este T_j em particular é escalonável. O segmento da linha 2 até a linha 5 será repetido para cada tarefa de **L**.

Como existem **m** tarefas em **L** e existem **n** tarefas em **H**, a complexidade para testar a escalonabilidade de todo o conjunto **L** será **O(n×m)**.

5.1 Simplificação para Alguns Sistemas

Em muito sistemas temos que o intervalo mínimo entre liberações das tarefas esporádicas é muito maior que o período das tarefas periódicas. Isto é, $\forall T_i \in H$, $\forall T_j \in L$, temos $P_i >> P_j$. Como $P_j = D_j$, teremos também $P_i >> D_j$. Um exemplo extremo é a

tarefa de liberação de um air-bag, que é executada uma única vez durante a vida do sistema e, portanto, possui intervalo entre liberações infinito.

Nos sistemas onde esta condição é satisfeita, teremos
$I_j^k(i) = \min(C_i, D_j)$.

Como $C_i < D_j$, caso contrário o sistema seria inviável, temos,
$I_j^k(i) = C_i$.

5.2 Exemplo Numérico

Nesta seção o exemplo numérico da seção 2 é retomado e sua escalonabilidade analisada segundo o método proposto. Temos que

H = { T_1 } e
L = { T_2, T_3 } .

Podemos novamente aplicar o teste proposto em [AUD 93b] e determinar que o tempo máximo de resposta de T_1 será 2. Logo, T1 é escalonável. Vamos agora verificar a escalonabilidade de cada tarefa do conjunto **L**.

Tarefa T_2

$$I_2^k(1) = \lfloor (1+10) \div 100 \rfloor \times 1 + \min(1, 10 - \lfloor (1+10) \div 100 \rfloor \times 100)$$
$$I_2^k(1) = \lfloor 11 \div 100 \rfloor \times 1 + \min(1, 10 - \lfloor 11 \div 100 \rfloor \times 100)$$
$$I_2^k(1) = 0 \times 1 + \min(1, 10 - 0 \times 100)$$
$$I_2^k(1) = 0 + \min(1, 10 - 0) = 0 + 1 = 1$$
$$\sum_{T_x \in L_{j,n}} (C_x \div P_x) = 1 \div 10 + 5 \div 10 + 6 \div 15 = 0.1 + 0.5 + 0.4 = 1$$

A tarefa T_1 gera sobre T_2 uma interferência equivalente a uma utilização de 0.1, o que somado a utilização de 0.9 do conjunto **L**, mostra que T_2 é escalonável.

Tarefa T_3

$$I_3^k(1) = \lfloor (1+15) \div 100 \rfloor \times 1 + \min(1, 15 - \lfloor (1+15) \div 100 \rfloor \times 100)$$
$$I_3^k(1) = \lfloor 16 \div 100 \rfloor \times 1 + \min(1, 15 - \lfloor 16 \div 100 \rfloor \times 100)$$
$$I_3^k(1) = 0 \times 1 + \min(1, 15 - 0 \times 100)$$
$$I_3^k(1) = 0 + \min(1, 15 - 0) = 0 + 1 = 1$$
$$\sum_{T_x \in L_{j,n}} (C_x \div P_x) = 1 \div 15 + 5 \div 10 + 6 \div 15 = 0.067 + 0.5 + 0.4 = 0.967$$

A tarefa T_1 gera sobre T_3 uma interferência equivalente a uma utilização de 0.067, o que somado a utilização de 0.9 do conjunto **L**, mostra que T_3 é escalonável.

Como cada tarefa de **L** foi individualmente mostrada escalonável, temos que o conjunto **L** é escalonável na aplicação. Como o conjunto **H**, isto é T_1, é escalonável, temos que a aplicação **A** é escalonável.

6. Conclusões

Neste artigo foi proposta uma solução de escalonamento tempo real baseada em prioridades que emprega, simultaneamente, prioridades fixas e variáveis. Tarefas periódicas ou esporádicas, com release jitter e deadline menor ou igual ao período, recebem prioridades fixas. Tarefas periódicas, com deadline igual ao período, recebem em conjunto uma prioridade menor e são escalonadas segundo EDF. Foi desenvolvido um teste de escalonabilidade suficiente mas não necessário para o modelo. Um exemplo numérico foi usado para ilustrar sua aplicação.

A solução de escalonamento proposta neste trabalho expande o leque de opções que os projetistas dispõem para solucionar o escalonamento de sistemas tempo real críticos.

Uma questão ainda em aberto é a utilização de protocolos para compartilhamento de recursos, tais como o *priority ceiling protocol* ([SHA 90]) e o *stack resource policy* ([BAK 91]), no modelo de tarefas aqui proposto.

7. Agradecimentos

Os autores gostariam de agradecer à prof. Keiko Ono Fonseca, por suas valiosas sugestões e comentários.

8. Referências

[AUD 93a] N. C. Audsley, A. Burns, A. J. Wellings. Deadline Monotonic Scheduling Theory and Application. Control Engineering Practice, Vol. 1, No. 1, pp. 71-78, february 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 93c] 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 93d] N. C. Audsley. Flexible Scheduling of Hard Real-Time Systems. Department of Computer Science thesis, University of York, 1993.

[BAK 91] T. P. Baker. Stack-Based Scheduling of Realtime Processes. The Journal of Real-Time Systems, Vol. 3, pp. 67-90, 1991.

[CHE 89] H. Chetto, M. Chetto. Some Results of the Earliest Deadline Scheduling Algorithm. IEEE Transactions on Software Engineering, Vol.15, No.10, pp.1261-1269, 1989.

[CHE 90] H. Chetto, M. Silly, T. Bouchentouf. Dynamic Scheduling of Real-Time Tasks under Precedence Constraints. The Journal of Real-Time Systems, Vol. 2, pp. 181-194, 1990.

[JEF 93] K. Jeffay, D. L. Stone. Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems. Proceedings of the IEEE Real-Time Systems Symposium, pp. 212-221, december 1993.

[LEH 89] J. P. Lehoczky, L. Sha, Y. Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. Proceedings of the IEEE Real-Time Systems Symposium, pp.166-171, december 1989.

[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.

[SHA 90] L. Sha, R. Rajkumar, J. P. Lehoczky. Priority Inheritance Protocols: An approach to Real-Time Synchronization. IEEE Transactions on Computers, Vol. 39, No. 9, pp. 1175-1185, september 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.

[TIN 94] K. W. Tindell, A. Burns, A. J. Wellings. An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks. Real-Time Systems, pp. 133-151, 1994.