ESCALONAMENTO BASEADO EM INTERVALO DE TEMPO

Fábio Rodrigues de la Rocha*, Rômulo Silva de Oliveira*

*Departamento de Automação e Sistemas - PGEEL - UFSC - Florianópolis,SC

Emails: frr@das.ufsc.br, romulo@das.ufsc.br

Abstract— This paper proposes a new task model for expressing timing constraints that do not naturally admit expression in terms of deadlines and periods. In our task model, jobs are divided into segments `A`, `B` and `C`. Segment `B` must execute inside a time-interval to fulfill some application constraints. The execution of `B` is valid if performed inside that time-interval, otherwise, its contribution may be considered valueless to its task. The model uses benefit functions to specify when an action should be performed for the maximum benefit. We integrate some scheduling approaches from the literature to obtain a possible scheduling solution for our model.

Keywords— Task Model, Time-Interval, Scheduling, *QoS*.

1 Introdução

O problema de escalonar tarefas que devem terminar até um determinado deadline é um assunto antigo na área de tempo real. O significado de um deadline para uma tarefa é um tempo limite para o término de sua computação. Se a computação terminar até o deadline, o resultado estará correto independente do momento em que esta efetivamente terminou. Um cenário comum é um sistema embutido composto por um único processador que periodicamente interage com o ambiente externo amostrando sensores e controlando dispositivos.

Embora muitas aplicações possam ser representadas pelo modelo de tarefas periódicas e deadlines, existem algumas situações onde tarefas possuem requisitos especial (Ravindran et al., 2005). Em algumas aplicações, tarefas necessitam que parte de seu código execute dentro de um intervalo de tempo específico. Um intervalo de tempo é uma janela de execução constituída por um início e fim. Em geral, o início do intervalo de tempo é determinado durante a execução. O seguinte caso de uso ilustra uma aplicação.

① Em sistemas embutidos, tarefas podem enviar mensagens utilizando um controlador de protocolo embutido no hardware tal como i^2c, *RS232*, *USB*, *CAN*. Em muitos microcontroladores de baixo custo, durante a transmissão de dados a *CPU* é mantida ocupada movendo dados da memória para a porta do controlador de protocolo e esperando uma resposta ou término da transmissão. Tanto a tarefa quanto a transmissão precisam ser escalonadas. Além do mais, a transmissão de dados não pode ser preemptada e as vezes deve ser realizada dentro de um intervalo de tempo.

Claramente este caso de uso não possui um limite de tempo para completar parte de sua computação, no máximo ele possui um intervalo de tempo e possivelmente um intervalo de tempo ideal com o maior benefício. Assim, o conceito de deadline é inapropriado para modelar estes tipos de aplicações as quais pela falta de embasamento teórico são implementadas com escalonadores convencionais levando a falta de previsibilidade.

Algoritmos de escalonamento bem conhecidos tomam decisões baseando-se na freqüência de chegada das tarefas (*RM*), deadine absoluto (*EDF*) e deadline relativo (*DM*). Uma melhor solução de escalonamento para o problema do intervalo de tempo seria incluir no escalonador o conhecimento do benefício da tarefa como função do tempo em que ela executa (função benefício).

Neste artigo é apresentado um novo modelo de tarefas para suportar uma classe de aplicações de tempo real. Assume-se um intervalo ideal para executar um *job* em um único processador, o qual resulta na maior contribuição para a aplicação. O valor da tarefa é reduzido antes e após o intervalo de tempo ideal. Computações realizadas antes e após este intervalo podem ser inúteis para os propósitos da aplicação. Abordagens clássicas da literatura de tempo real foram adaptadas para criar uma solução para este problema de escalonamento. O restante deste artigo é organizado da seguinte maneira. Seção 2 resume a literatura relacionada. Seção 3 apresenta o modelo do intervalo ideal. Seção 4 apresenta uma abordagem de escalonamento e seção 5 algumas avaliações experimentais. Seção 6 apresenta as conclusões e trabalho futuro.

2 Revisão da literatura

O assunto de escalonamento baseado em valor é apresentado numa visão geral em (Burns et al., 2000). Em (Buttazzo et al., 1995) é apresentado um estudo sobre situações de sobrecarga onde as tarefas são compostas por um deadline e uma métrica de qualidade. Em (Liu et al., 1994) é apresentado um modelo para escalonar tarefas compostas por uma parte obrigatória e uma parte opcional que incrementa o benefício obtido da execução da tarefa. Neste modelo, é aceitável que somente as partes obrigatórias sejam executadas, além disso, a execução da parte opcional não está relacionada com um intervalo de tempo específico, dentro do qual deve executar. O assunto de funções utilidade para associar um benefício a execução da tarefa em relação ao seu tempo de término é

apresentado em (Li, 2004). Em (Lipton and Tomkins, 1994) é apresentado um problema de escalonamento on-line de intervalos no qual um conjunto de intervalos de tempo são apresentados ao algoritmo de escalonamento. Os intervalos de tempo são não-preemptivos, possuem início e fim e não podem ser escalonados antes nem depois. Como trabalhos futuros, os autores discutem o problema similar no qual os tempos de liberação são mais gerais e onde a tarefa poderia requisitar que um dado intervalo seja escalonado dentro de "x" unidades de tempo. Em (Mazzini and Armentano, 2001) os autores apresentam um algoritmo heurístico para escalonar jobs minimizando o quanto o job esta atrasado/adiantado para o caso de jobs não-preemptivos. O assunto de tarefas com relações de offset é apresentado em (Tindell, 1992) e (Pellizzoni and Lipari, 2005).

3 Modelo de tarefas baseado em intervalo de tempo

Neste modelo, tarefas τ_i, $i \in \{1 \dots n\}$ são descritas por um tempo de execução de pior caso W_i, periodo T_i, um deadline D_i. Assume-se que $T_i = D_i$. Cada τ_i consiste em uma infinita série de *jobs* $\{\tau_{i1},\dots,\tau_{ij},\dots\}$, j^{th} tal que τ_{ij} é liberado no tempo $(j-1)\cdot T_i, j \geq 1$ e deve terminar até o tempo $(j-1)\cdot T_i + D_i$ ou uma falha temporal ocorrerá. Nos definimos como **segmento** um grupo sequencial de instruções dentro de τ_i (mostrado na figura 1). Tarefa τ_i é composta por três segmentos chamados A_i, B_i e C_i. Denotamos o primeiro índice do segmento como representativo da tarefa em questão e segundo índice como o *job*, desta forma, o primeiro *job* do segmento A_i é chamado A_{i1}, o segundo *job* é A_{i2} e assim por diante para todos os segmentos. O pior tempo de execução de A_i é W_{A_i}, de B_i é W_{B_i} e de C_i é W_{C_i}. A soma do pior tempo de execução de todos os segmentos é igual ao pior tempo de execução da tarefa τ_i (W_{A_i}+W_{B_i}+W_{C_i}=W_i). Assumimos que existe uma relação de precedência entre os segmentos $A_i \prec B_i \prec C_i$.

A execução dos segmentos A_i, B_i e C_i é sujeito ao deadline da tarefa τ_i, D_i o qual, neste caso é um deadline fim a fim. O segmento A_i é responsável por realizar suas computações e pode requerer ou não a execução do segmento B_i. Desta forma, o tempo de chegada do segmento B_i é determinado on-line pelo segmento A_i. Caso a execução do segmento B_i seja requerida, segment C_i (o que é um código de finalização da tarefa) deve ser executado. Assim, mesmo que a execução do segmento A_i seja periódica com período T_i, segmentos B_i e C_i são esporádicos. Em caso B_i e C_i são sejam requisitados para executar, segmento A_i poderá executar até o deadline D_i. Caso contrário, logo após segmento B_i concluir sua execução, segmento C_i é liberado. Como consideramos escalonamento num sistemas de apenas um processador, segmentos não podem sobrepor-se no tempo.

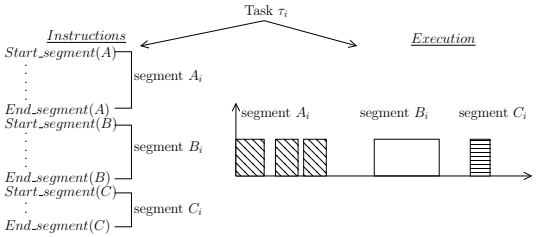


Figura 1: Tarefa τ_i com segmentos.

3.1 Métrica de qualidade de serviço

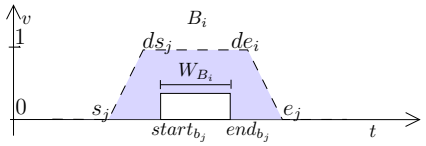
A execução do segmento B_{ij} está também sujeita a um **intervalo de tempo** $[s_{i,j}, e_{i,j}]$ o qual é definido pelo segmento A_{ij} em tempo de execução e pode mudar para cada *job* j, isto é: segmento B_{ij} deve executar dentro deste intervalo de tempo para gerar um benefício positivo. O tamanho do intervalo $[s_{i,j}, e_{i,j}]$ é constante e denominado ρ_i. Dentro do intervalo de tempo $[s_{i,j}, e_{i,j}]$, existe um **intervalo de tempo ideal** $[ds_{i,j}, de_{i,j}]$ de tamanho constante denominado ψ_i onde a execução do segmento B_{ij} resulta no maior benefício para τ_i ($\psi_i \leq \rho_i$). Figuras 2 e 3 apresentam funções de qualidade para descrever aplicações comuns para sistemas de tempo real. A escolha de uma função particular para uma tarefa é um requisito da aplicação a qual também determinará os valores de $s_{i,j}$,$e_{i,j}$,$ds_{i,j}$ e $de_{i,j}$.

Nestas figuras, o eixo y representa o benefício obtido v e o eixo x o tempo de ativação t. O segmento B_{ij} é apresentado como executando com o seu pior tempo de execução (W_{B_i}), iniciando em $start_{bj}$ e terminando em end_{bj}. A função benefício $v(t)$ em função do tempo é dada pelas equações em cada figura. Na equação 1 o QoS é mostrado como um benefício cumulativo da execução do segmento B_{ij} dentro do intervalo de tempo. A equação resulta em um valor entre $[0, 100\%]$ e representa o percentual do máximo benefício. O máximo benefício é somente alcançado quando B_i executa todo o seu código dentro do **intervalo de tempo ideal** $[ds_{i,j}, de_{i,j}]$. O objetivo é maximizar o QoS para cada execução de B_i. Como anteriormente apresentado, a execução do segmento B_i não é sempre requerida por A_i e para os casos em que não é requerida não existe valor de QoS para acumular.

$$QoS(B_{i,j}, start_{B_{i,j}}, end_{B_{i,j}}) = \frac{\int_{start_{B_{i,j}}}^{end_{B_{i,j}}} v\,(t)\,dt}{end_{B_{i,j}} - start_{B_{i,j}}} \cdot 100 \quad (1)$$

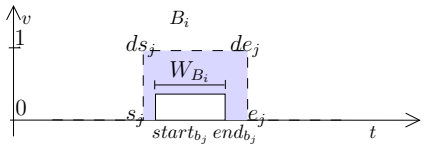
3.2 Controle de acesso

O problema do intervalo de tempo pode apresentar também requisitos de acesso exclusivo durante



$$v(t) = \begin{cases} 0, & t < s_j \text{ or } t > e_j \\ 1, & ds_j \leq t \leq de_j \\ 1 - \left(\frac{ds_j - t}{ds_j - s_j}\right), & s_j \leq t < ds_j \\ 1 - \left(\frac{t - de_j}{e_j - de_j}\right), & de_j < t \leq e_j \end{cases}$$

Figura 2: *QoS* Para benefício cumulativo.



$$v(t) = \begin{cases} -\infty, & t < s_j \text{ or } t > e_j \\ 1, & s_j \leq t \leq e_j \end{cases}$$
$$s_j = ds_j \qquad e_j = de_j$$

Figura 3: *QoS* Para benefício rígido.

a execução do segmento B dentro do intervalo ideal. A natureza dos recursos sob controle do segmento B impoẽ requisitos de acesso para assegurar a consistência durante a operação sobre o recurso. Neste artigo, assume-se o segmento B como não preemptivo garantindo desta forma o controle de acesso.

4 Abordagem de escalonamento

Para propósitos de implementação, é natural representar os segmentos de tarefas como subtarefas. Assim, mapea-se todos os segmentos de tarefas τ_i em subtarefas, mantendo os mesmos nomes A_i, B_i e C_i. Subtarefas A_i and C_i são escalonadas utilizando-se *EDF* preemptivo. Os deadlines internos são associados a subtarefas através de uma regra de atribuição de deadline. Inicialmente, enumera-se os desafios e possíveis soluções na literatura para as abordagens apresentadas.

4.1 Desafios

O primeiro desafio em nossa abordagem é a precedência entre as subtarefas. No *EDF*, a precedência correta pode ser assegurada através dos deadlines de cada subtarefa (Blazewicz, 1976). Considerando o deadline original da tarefa τ_i como D_i, é possível associar um novo deadline para cada subtarefa tal que $D_{A_i} < D_{B_i} < (D_{C_i} = D_i)$ atendendo ao requisito.

Um problema diferente emerge quando ao invés de somente uma relação de precedência entre as tarefas existe um requisito temporal, tal como no problema do intervalo de tempo. No problema do intervalo de tempo, o deadline da subtarefa A_i não é necessariamente um tempo para iniciar B_i. A definição do problema diz que o segmento B_i deve iniciar dentro de uma janela de tempo ajustada por A_i. O mínimo tempo para liberar B_i é um requisito do problema e este valor é o deadline para o segmento anterior. Sem perda de generalidade, assume-se um limite inferior e superior para a liberação do segmento B_i $[Bmin_i, Bmax_i]$ e ajusta-se o deadline $D_{A_i} = Bmin_i$, $D_{B_i} = Bmax_i + \rho_i$ (figura 4).

O segundo desafio é o tempo de liberação das subtarefas. Precisa-se assegurar no teste de escalonabilidade que uma subtarefa chegará somente depois de um tempo predeterminado. Utilizando-se offsets para controlar a chegada das subtarefas e um teste de escalonabilidade orientado a offsets pode-se atender estes requisitos. O último desafio é o aspecto não preemptivo de B_i. De forma análoga, o teste de escalonabilidade deve assegurar que as execuções não preemptivas serão escalonáveis em conjunto com as demais seções preemptivas do sistema de tarefas. Jeffay e Stone propuseram um teste de escalonabilidade para verificar a escalonabilidade de tarefas na presença de interrupções de software. Interrupções são consideradas tarefas de prioridade mais alta e assim podem preemptar as demais tarefas de aplicações.

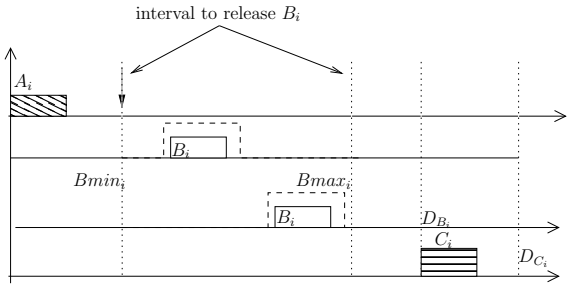


Figura 4: Limites para liberar B_i.

4.2 Teste de escalonabilidade

Nas seguintes subseções, a escalonabilidade do sistema de tarefas é verificada dividindo o problema em dois. Na primeira parte, verfica-se a escalonabilidade das subtarefas A_i e C_i na presença de subtarefas não preemptivas B_i. B_i pode chegar entre $[Bmin_i, Bmax_i]$. Na segunda parte, verifica-se a capacidade de B_i executar dentro do seu intervalo de tempo ideal, mesmo que a posição deste possa variar. Esta informação é útil para garantir a execução de tarefas com métrica rígida de QoS. Além disso, determina-se os valores mínimos e máximos que podem ser obtidos para o QoS de B_i.

4.2.1 Teste de escalonabilidade para subtarefas A e C

O teste de escalonabilidade de subtarefas A e C é realizado utilizando-se a abordagem de de-

manda de processador (k. Baruah et al., 1990). A demanda de uma tarefa no intervalo de tempo $[t_1, t_2]$ é o tempo cumulativo necessário para processar todas as k instâncias de tarefas que foram liberadas e devem terminar dentro deste intervalo de tempo. Assume-se $g_i(t_1, t_2)$ como a demanda de processamento de τ_i. Desta forma, $g_i(t_1, t_2) = \sum_{r_{i,k} \geq t_1, d_{i,k} \leq t_2} \cdot C_i$. Num sistema de tarefas $\tau = \tau_1, \tau_2, \ldots, \tau_n$ a demanda de processador em $[t_1, t_2]$ é $g(t_1, t_2) = \sum_{i=1}^{n} g_i(t_1, t_2)$.

A quantidade de tempo de processamento requerido em $[t_1, t_2]$ deve ser menor ou igual do que o tamanho do intervalo $[t_1, t_2]$. Assim, $\forall t_1, t_2$ $g(t_1, t_2) \leq (t_2 - t_1)$.

Assume-se uma função $\eta_i(t_1, t_2)$ que fornece o número de ativações da tarefa τ_i com liberação e deadline dentro de $[t_1, t_2]$. $\eta_i(t_1, t_2) = max\{0, \lfloor \frac{t_2 + T_i - D_i - \Phi_i}{T_i} \rfloor - \lceil \frac{t1 - \Phi_i}{T_i} \rceil\}$. Na figura 5 as únicas ativações contabilizadas por η_i são $\tau_{i,2}$ e $\tau_{i,3}$. A ativação $\tau_{i,1}$ possui um tempo de liberação antes de t_1 e $\tau_{i,4}$ possui um deadline após t_2.

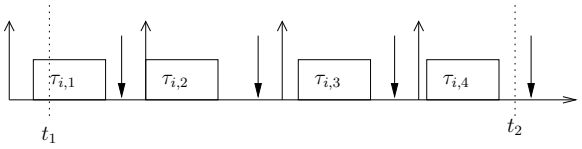


Figura 5: Ativações de τ_i

A demanda de processador dentro do intervalo de tempo é igual ao número de ativações que foram liberadas e terminaram dentro do intervalo de tempo multiplicado pelo tempo de computação C_i. Assim, $g_i(t_1, t_2) = max\{0, \lfloor \frac{t_2 + T_i - D_i - \Phi_i}{T_i} \rfloor - \lceil \frac{t1 - \Phi_i}{T_i} \rceil\} \cdot C_i$ e a demanda de processamento para todo o sistemas de tarefa é :

$$g(t_1, t_2) = \sum_{i=1}^{n} max\{0, \lfloor \frac{t_2 + T_i - D_i - \Phi_i}{T_i} \rfloor - \lceil \frac{t_1 - \Phi_i}{T_i} \rceil\} \cdot W_i \quad (2)$$

A escalonabilidade de um sistemas de tarefas assíncrono com deadline menor ou igual ao período pode ser verificado pela equação 3. Em sistemas de tarefas assincronos, a escala de execução repete-se a cada $[2 \cdot H + \Phi]$ onde H é o hiperperíodo ($H = mmc\{T_1, T_2, \ldots, T_n\}$) e Φ é o maior offset entre tarefas ($\Phi = max\{\Phi_1, \Phi_2, \ldots, \Phi_n\}$).

$$\forall t_1, t_2 \qquad g(t_1, t_2) \leq (t_2 - t_1) \quad (3)$$

4.2.2 Tarefas não preemptivas

Um importante passo para verificar a escalonabilidade de tarefas preemptivas e tarefas não preemptivas foi dado por Jeffay e Stone em (Jeffay and Stone, 1993). Os autores mostraram uma condição de escalonabilidade em um modelo para assegurar a escalonabilidade utilizando *EDF* na presença de interrupções. Basicamente, o autor assume interrupções com tarefas de mais alta prioridade que preemptam qualquer tarefa de aplicação. Desta forma, eles modelam o gerenciador de interrupções como um tempo que é roubado das tarefas de aplicação. Caso as tarefas consigam terminar antes de seus deadlines mesmo sofrendo a interferência de interrupções, o conjunto de tarefas é escalonável. O conjunto de tarefas é composto por n tarefas de aplicação e m gerenciadores de interrupções. Interrupções são descritas por um tempo de computação CH e um tempo mínimo entre ativações TH. O tempo de processamento para executar interrupções é $f(L)$.

Teorema 1 *O conjunto τ de n tarefas periódicas ou esporádicas e o conjunto Δ de m gerenciadores de interrupção é escalonável pelo EDF se e somente se*

$$\forall L \geq 0 \qquad g(0, L) \leq L - f(L), f(L) \text{ calcula-se}$$

$$f(0) = 0$$

$$f(L) = \begin{cases} f(L-1) + 1, \\ \quad if \sum_{i=1}^{m} \lceil \frac{L}{TH_i} \rceil CH_i > f(L-1) \\ f(L-1), \\ \quad \text{caso contrário} \end{cases} \quad (4)$$

A prova deste teorema é similar a prova do método de demanda de processador em Baruah. A diferença está no fato de que a cada intervalo de tamanho L, a quantidade de tempo que o processador pode dedicar para as tarefas de aplicação é igual a $L - f(L)$.

Utilizando este método, a subtarefa B_i é modelada como um gerenciador de interrupções, subtarefas A_i e C_i são implementadas como subtarefas executando no *EDF* e a escalonabilidade verificada utilizando o teorema 1. O teorema 1 como descrito por Jeffay e Stone assume que o sistema é sincrono com um sistemas de tarefas no qual os deadlines são iguais aos períodos.

Este teorema é extendido utilizando-se a demanda de processador para representar um sistema assíncrono com deadlines menores ou iguais aos períodos. Neste caso, subtarefas A_i chegam no tempo zero ($\Phi_{A_i} = 0$) e C_i chega no tempo Φ_{C_i}. Neste momento, assumimos que num tempo específico uma interrupção inicia B_i (utilizando a mesma designação de Jeffay e Stone). Para assegurar que a subtarefa C_i somente será executada após a subtarefa B_i utiliza-se um offset $\Phi_{C_i} = D_{B_i}$. O novo teste de escalonabilidade no qual todas as subtarefas C_i possuem offsets e subtarefas B_i são modeladas como interrupções é:

$$\forall L \geq 0 \qquad g(t_1, t_2) \leq (t_2 - t_1) - F(t_1, t_2)$$

Diferentemente de um sistema sincrono, onde a escalonabilidade pode ser verificada testando-se todos os períodos ocupados L até H(hiperperíodo). Os testes de escalonabilidade para sistemas assincronos de tarefas possuem alta complexidade algoritmica e torna-se necessário testar todos os períodos ocupados de 0 até $2{\cdot}H{+}\Phi$.

No problema do intervalo de tempo, subtarefas B possuem uma janela de tempo durante a qual podem estar ativas. Aplicar diretamente o teorema 1 seria pessimista por contabilizar a influência de interrupções onde elas não poderiam ocorrer. Uma melhoria seria inserir um offset ΦH_i em $\lceil \frac{L-\Phi H_i}{TH_i} \rceil$ para representar o fato que uma interrupção não pode ocorrer antes de $Bmin$. No algoritmo 1 assume-se que $F(t_1, t_2)$ representa a demanda de processador resultante das interrupções em $[t_1, t_2]$. No pior caso, o algoritmo possui complexidade $O(H^2)$. Infelizmente, no pior caso o hiperperíodo é o produto de todos os períodos $\prod_{i=1}^{n} T_i$. Assim, o algoritmo pode ser aplicado apenas quando os períodos do sistema de tarefas resultam num hiperperíodo pequeno.

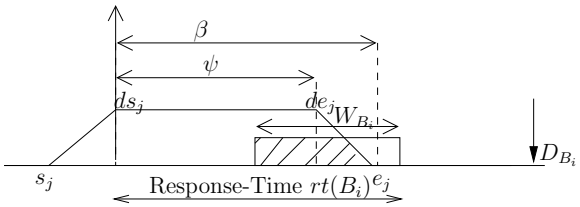
Algoritmo 1 Escalonabilidade - primeira parte

```
for all t1 such that 0 ≤ t1 ≤ 2·H + Φ do
  for all t2 such that t1 ≤ t2 ≤ 2·H + Φ do
    g(t1,t2) = Σ_{i=1}^{n} max{0, ⌊(t2+Ti−Di−Φi)/Ti⌋ −
    ⌈(t1−Φi)/Ti⌉} · Wi
    F(t1,t2) = f(t2) − f(t1)
    if g(t1,t2) > (t2 − t1) − F(t1,t2) then
      return nonfeasible
    end if
  end for
end for
{Escalonável, aplica a segunda parte}
return feasible
```

4.2.3 Teste de escalonabilidade para as subtarefas não preemptivas

Diferentemente das subtarefas A_i e C_i que são escalonadas pelo *EDF*, subtarefas B_i são escalonadas com base uma prioridade fixa com valores associados através de uma regra heurística. A regra possui duas métricas, **criticalidade** e **fator de deslocamento**. Na primeira métrica, subtarefas podem ser rígidas ou cumulativas. Subtarefas com benefício rígido correspondem a um grupo de subtarefas com prioridade mais alta do que subtarefas com benefício cumulativo. Dentro de cada grupo, prioridades são associadas inversamente ao fator de deslocamento, computado como $sf_i = \frac{\psi}{W_{B_i}}$. O fator de deslocamento é relacionado a capacidade da subtarefa de ser postergada e ainda executar dentro do intervalo ideal, obtendo o maior QoS.

A escalonabilidade de B_i é verificada calculando o seu tempo de resposta (rt), assumindo que todas as subtarefas B_i são sempre liberadas em ds_j como mostrado na figura 6. Na mesma figura, utiliza-se β para descrever o intervalo de tempo entre a liberação em ds_j até o ponto e_j. Em subtarefas com métrica cumulativa (figura 2) é possível que B_i termine após o intervalo ideal, resultando num baixo valor de QoS. Em contraste, subtarefas com métrica rígida (figura 3) demandam sua execução completa dentro do intervalo ideal. Assim, é necessário verificar se no prior cenário possível $rt(B_i) \leq \psi$. Note que numa subtarefa com métrica rígida B_i, $s_j = ds_j$, $de_j = e_j$.



$rt(B_i)$	QoS
$rt \leq \psi$	100%
$rt \geq \beta + W_{B_i}$	0%
$\psi < rt < \beta + W_{B_i}$	$QoS(B_i, rt(B_i) - W_{B_i}, rt(B_i))$%

Figura 6: QoS em relação ao rt.

O tempo de reposta pode ser dividido em pior tempo de resposta ($wcrt$) e melhor tempo de resposta ($bcrt$). O $wcrt$ representa o pior cenário possível para a execução de B_i e neste sentido o QoS obtido é o mínimo possível. O $bcrt$ representa o melhor cenário possível para B_i resultando no maior QoS.

Através do cálculo do $wcrt$ e do $bcrt$ de uma subtarefa B_i é possível obter um QoS como mostrado na figura 6. Desta forma, aplicando o $wcrt$ de uma subtarefa B_i como seu tempo de resposta na figura 6 resulta no mínimo QoS possível. Aplicando o $bcrt$ como seu tempo de resposta, resulta no máximo QoS possível. A primeira linha da tabela na figura 6 cobre o caso em que todo B_i executa dentro de seu intervalo de tempo ideal. A segunda linha cobre o caso em que a execução acontece fora do intervalo de tempo e a terceira linha cobre o caso em que parte de B_i executa dentro do intervalo de tempo.

4.2.4 Tempo de resposta

O tempo de resposta de pior caso de subtarefas esporádicas não preemptivas pode ser determinado pela soma de três fatores $wcrt(B_i) = W(B_i) + max(W(B_j), j \in lp(i)) + \sum_{j \in hp(i)} \cdot W(B_j)$. O primeiro termo na equação é tempo de execução de pior caso da subtarefa B_i. O segundo termo é o máximo tempo que B_i pode ficar bloqueada por uma subtarefa que se encontra em execução no momento que B_i é liberada. Contabiliza-se este valor como o maior $wcet$ entre as subtarefas B_j com prioridade menor (lp) do que B_i, deixando a interferência das subtarefas de prioridade mais alta (hp) para o próximo termo. O último termo é

o máximo tempo de bloqueio causado por subtarefas B_j com prioridades mais altas. Contabiliza-se este valor adicionando todas as subtarefas B_j com prioridade mais alta do que B_i. O melhor tempo de resposta para B_i ocorre quando B_i não sofre qualquer interferência de subtarefas B_j, sendo assim, $bcrt(B_i) = W(B_i)$.

5 Avaliação experimental

Esta seção ilustra-se o teste de escalonabilidade comprando-se com uma simulação. O experimento é composto por três tarefas (τ_1,τ_2 e τ_3), como mostrado na tabela 1. Tarefa τ_1 e τ_3 possuem métrica de benefício cumulativo e a tarefa τ_2 possui métrica de benefício rígida. As prioridades são $Prio_{B_1} = 3$, $Prio_{B_2} = 1$, $Prio_{B_3} = 2$ e os demais parâmetros são $\rho_{B1} = 12$, $\rho_{B2} = 8$, $\rho_{B3} = 14$, $\psi_{B1} = 10$, $\psi_{B2} = 8$, $\psi_{B3} = 8$. Os resultados do teste offline podem ser vistos na tabela 2. Em virtude do uso de um teste pessimista os valores reais para os benefícios podem ser maiores do que os valores resultantes do teste offline.

τ	subtask	W_i	D_i	T_i	Φ_i	$Bmin_i$	$Bmax_i$
	A_1	4	10	40	0		
τ_1	B_1	6	31	40	11	10	20
	C_1	2	40	40	31		
	A_2	3	20	40	0		
τ_2	B_2	2	34	40	20	20	26
	C_2	2	40	40	34		
	A_3	2	15	60	0		
τ_3	B_3	6	31	60	18	15	20
	C_3	1	60	60	31		

Tabela 1: Exemplo com três tarefas.

subtask	wcrt	bcrt	min benefit	max benefit
B_1	14	6	41.6%	100.0%
B_2	8	2	100.0%	100.0%
B_3	14	6	25.00%	100.0%

Tabela 2: Resultados através do teste offline

O conjunto de tarefas foi simulado por 10.000 unidades de tempo, com o tempo de chegada escolhido uniformemente entre $Bmin$ e $Bmax$ (tabela 3). Subtarefas B_i e C_i são requisitados em 90% das ativações de τ_i. A simulação mostra um resultado consistente com o teste offline onde os valores para o benefício são iguais ou maiores do que os resultantes do teste offline. Desta maneira, o teste pode ser utilizado para garantir que durante sua execução nenhuma tarefa obterá um benefício inferior ao calculado pelo teste offline.

6 Conclusões e trabalhos futuros

Este artigo apresentou o modelo de tarefas do intervalo de tempo para descrever requisitos temporais que não admitem expressão em termos de deadlines e períodos. Abordagens da literatura de

subtask	wcrt	bcrt	min benefit	max benefit
B_1	14	6	41.6%	100.0%
B_2	7	2	100.0%	100.0%
B_3	13	6	41.6%	100.0%

Tabela 3: Resultados através de simulação

tempo real foram adaptadas para este problema de escalonamento objetivando criar um teste offline. Além de uma resposta aceite/rejeição para tarefas com métrica de benefício rígida o teste offline fornece valores mínimos e máximos para o benefício esperado nos casos de tarefas com métricas de benefício cumulativo. Como trabalho futuro, pretende-se investigar como as subtarefas não preemptivas podem ser re-ordenadas em tempo de execução para incrementar o benefício obtido.

Referências

Blazewicz, J. (1976). Scheduling Dependent Tasks with Different Arrival Times to Meet Deadlines, *Proceedings of the International Workshop on Modelling and Performance Evaluation of Computer Systems*, North-Holland, pp. 57–65.

Burns, A., Prasad, D., Bondavalli, A., Giandomenico, F. D., Ramamritham, K., Stankovic, J. and Strigini, L. (2000). The Meaning and Role of Value in Scheduling Flexible Real-Time Systems, *Journal of Systems Architecture* **46**: 305–325.

Buttazzo, G. C., Spuri, M. and Sensini, F. (1995). Value vs. Deadline Scheduling in Overload Conditions, *IEEE RTSS*, pp. 90–99.

Jeffay, K. and Stone, D. L. (1993). Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems, *Proceedings of the* 14^{th} *IEEE Symposium on Real-Time Systems*, pp. 212–221.

k. Baruah, S., Howell, R. R. and Rosier, L. E. (1990). Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor, *Real-Time Systems* **2**: 301–324.

Li, P. (2004). *Utility Accrual Real-Time Scheduling: Models and Algorithms*, PhD thesis, Virginia Polytechnic Institute and State University.

Lipton, R. J. and Tomkins, A. (1994). Online Interval Scheduling, *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, PA, USA, pp. 302–311.

Liu, J., Shih, W.-K., Lin, K.-J., Bettati, R. and Chung, J.-Y. (1994). Imprecise computations, *Proceeding of the IEEE*, Vol. 82, pp. 83–94.

Mazzini, R. and Armentano, V. A. (2001). A Heuristic For Single Machine Scheduling With Early And Tardy Costs, *European Journal of Operational Research* **1**: 129–146.

Pellizzoni, R. and Lipari, G. (2005). Improved Schedulability Analysis Of Real-Time Transactions With Earliest Deadline Scheduling, *Proceedings of the 11th IEEE RTAS*, pp. 66–75.

Ravindran, B., Jensen, E. D. and Li, P. (2005). On Recent Advances In Time/Utility Function Real-Time Scheduling And Resource Management, 8^{th} *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 55–60.

Tindell, K. (1992). Using offset information to analyse static priority pre-emptively scheduled task sets, *Technical report*, University of York, YCS-92.