Avaliando o Uso de Servidores para Tarefas Aperiódicas para fins de Tolerância a Falhas em Sistemas com Escalonamento Dinâmico

Ana Carolina Sokolonski Anton[1], **George Marconi de Araújo Lima**[1]

[1]Laboratório de Sistemas Distribuídos – Universidade Federal da Bahia (UFBA)
Av. Ademar de Barros s/n - Ondina - Salvador/BA

anaanton@ufba.br, gmlima@ufba.br

Abstract. *A real-time system is designed to react to stimuli from the environment within a priori defined time interval. Ensuring the logical and temporal correctness of such systems is a function of both scheduling and fault tolerance mechanisms. Integrating these mechanisms in a suitable way is not simple and is currently a research issue. In this work we evaluate by simulation the behavior of some well known scheduling mechanisms in the presence of faults. The experiment results indicate how well these mechanisms can deal with fault scenarios and can be used to subside further research in the field.*

Resumo. *Um Sistema de Tempo Real é projetado para reagir, dentro de prazos pré-estabelecidos, a estímulos provocados pelo ambiente. Garantir a correção lógica e temporal de tais sistemas é função de elaborados mecanismos de escalonamento e tolerância a falhas. Integrá-los de forma apropriada não é simples e é tema corrente de pesquisas. Neste trabalho avaliamos, por simulação, o comportamento de alguns dos mais conhecidos mecanismos de escalonamento na presença de falhas. Os resultados dos experimentos indicam como tais mecanismos lidam com cenários de falhas, podendo subsidiar futuras pesquisas na área.*

Palavras-chave: *escalonamento dinâmico, sistema de tempo real, tolerância a falhas.*

1. Introdução

Um Sistema de Tempo Real (STR) é um sistema computacional projetado para reagir a estímulos provocados pelo ambiente. Tais estímulos estão associados a uma ou mais tarefas. Tarefas são unidades de processamento seqüenciais que usam de forma concorrente um ou mais recursos computacionais do sistema. Cada tarefa possui um prazo máximo de execução chamado *deadline*. Num STR, as tarefas devem apresentar correção lógica e temporal. Quando a violação do *deadline* de todas as tarefas do sistema não causa graves danos, chamamos este sistema de Sistema de Tempo Real Não-Crítico (STRNC). Quando a violação do *deadline* da tarefa não é tolerada, esta é chamada de crítica. Um sistema que possui ao menos uma tarefa crítica é chamado de Sistema de Tempo Real Crítico (STRC) [Burns and Wellings 2001].

Um importante mecanismo, responsável por garantir a correção temporal num STR, é o

escalonador. O escalonador deve escolher qual tarefa deve ser executada em função do tempo de acordo com algum crítério. Os critérios de escalonamento comumente usados são baseados em prioridades. O escalonador se limita a executar a tarefa com mais alta prioridade. Prioridade representa, portanto, urgência de execução.

Geralmente, o escalonador é projetado sem considerar a possibilidade da ocorrência de falhas. Em sistemas críticos, no entanto, é preciso especificar as ações que devem ser tomadas pelo sistema quando alguma tarefa falha. Uma das técnicas que pode ser usada é baseada na detecção e recuperação de erros, de acordo com a qual, após a detecção de um erro, uma *tarefa de recuperação* é submetida para execução. Como o momento da ocorrência de erros não pode ser previsto, a tarefa de recuperação pode causar interferências em tarefas menos prioritárias. O mecanismo de escalonamento, deve, portanto, prover o isolamento temporal de tal forma que tarefas de recuperação não causem violações de deadlines de outras tarefas no sistema.

Este fenômeno é parecido com o que ocorre quando executa-se tarefas críticas e não-críticas num mesmo sistema, pois não se sabe o instante de chegada das tarefas não-críticas (i.e. tarefas aperiódicas). Neste caso, usa-se servidores de tarefas aperiódicas, ou simplesmente servidores, para prover o isolamento temporal desejado.

Neste trabalho investigamos, por simulação, o comportamento de diversos tipos de servidores para executar tarefas de recuperação. O uso de servidores para tolerância a falhas ainda não havia sido realizado de maneira satisfatória em sistemas baseados em escalonamento dinâmico. Em particular, usamos uma das políticas de escalonamento mais usadas em sistemas de tempo real, EDF (*Earliest Deadline First*). De acordo com tal política, as prioridades das tarefas variam em função do estado do sistema durante sua execução, provendo portanto melhor adaptação do sistema em cenários de falhas. Além disso, sabe-se que EDF é uma política ótima [Liu 2000] e com baixo custo de implementação. Ser ótima significa que se há um escalonamento possível, onde nenhuma tarefa perde o *deadline*, então o EDF também gera uma escala de execução onde nenhum *deadline* é violado. Os resultados da simulação ilustram as diferenças entre os servidores estudados e podem ser utilizados para subsidiar futuras pesquisas na área de escalonamento para sistemas de tempo real tolerante a falhas.

Na próxima seção, apresentaremos o modelo de escalonamento utilizado e a motivação do trabalho. Na seção 3, explicaremos, com detalhes, os servidores utilizados no experimento. Na seção seguinte, explicaremos como a simulação foi realizada apresentando seus resultados. Finalmente, finalizamos o artigo com uma seção expondo as conclusões extraídas do experimento.

2. Modelo e Motivação

Num STR podem existir três tipos de tarefas: periódicas, aperiódicas e esporádicas. *Periódicas* são aquelas que são reativadas em intervalos regulares de tempo chamados *períodos*. *Aperiódicas* são tarefas que não têm período definido, assim podem ser reativadas a qualquer momento. *Esporádicas* são tarefas que não têm período definido, assim como as aperiódicas, porém um intervalo mínimo entre suas ativações é conhecido. No nosso trabalho, ao ocorrer uma falha no momento da execução de uma tarefa periódica

será gerada uma tarefa de recuperação com os mesmos atributos da tarefa que falhou. Ao compararmos tarefas aperiódicas e tarefas de recuperação, podemos perceber que ambas podem ocorrer a qualquer momento. Abordamos então, as tarefas de recuperação, no nosso trabalho, como tarefas aperiódicas. Assim, temos dois tipos de tarefas executando em nosso sistema, periódicas e de recuperação.

Usaremos a seguinte notação: Tarefas serão representadas pela letra j. Assim, j_i é a tarefa de identificação i. A *k-ésima* instância de j_i é denotada por $j_{i,k}$. O período de j_i é denotado por p_i. O tempo máximo de execução de j_i é representado por c_i. O *deadline* relativo de j_i é o tempo total em que j_i deve iniciar e concluir sua execução, este é simbolizado por D_i. O *deadline* absoluto de j_i em sua ativação k, que corresponde ao valor absoluto do *deadline* (instante do início da ativação k somado ao *deadline* relativo) é representado por $d_{i,k}$. O percentual de utilização do processador solicitado por j_i é representado por $u_i = \frac{c_i}{p_i}$.

Consideramos um STR uniprocessado com preempção, que pode ter em sua fila de execução dois tipos de tarefas, periódicas e recuperações. O modelo de escalonamento que utilizamos faz uso do algoritmo EDF para escalonar suas tarefas. Assumiremos que as tarefas não compartilham recursos (são independentes) e $D_i = p_i$. Nestas condições, sabe-se que se $\sum u_i \leq 1$, o sistema é escalonável [Liu 2000]. Em outras palavras, nenhuma tarefa perde o *deadline*.

Para ilustrar melhor a motivação deste trabalho, considerem a figura 1, onde um exemplo de escalonamento EDF é apresentado. As tarefas periódicas são indicadas por j_1, j_2 e j_3, a linha horizontal é a linha temporal, j_1 tem $c_1 = 3$ e $D_1 = 10$, j_2 tem $c_2 = 4$ e $D_2 = 12$ e j_3 tem $c_3 = 2$ e $D_3 = 7$. As três tarefas são iniciadas no instante zero. Os quadrados cheios representam tarefas em execução. Os quadrados tracejados indicam que a tarefa está em preempção.

No instante 0, a tarefa j_3 é a mais prioritária por apresentar menor *deadline* absoluto, $d_{3,1} = 7$. No instante 2 a tarefa j_3 termina de executar e a tarefa j_1 passa a ser a tarefa mais prioritária. No instante 5, quando a tarefa j_1 terminou sua execução, j_2 passa a executar. No instante 7 a tarefa j_3 é reiniciada, porém seu *deadline* agora é 14, assim ela espera a tarefa j_2, que naquele momento é a mais prioritária, com *deadline* 12, terminar de executar.

A ilustração apresentada na figura 1 contempla apenas tarefas periódicas. Se falhas forem consideradas, ou seja, se introduzirmos tarefas de recuperação, usar apenas EDF pode não ser suficiente. Por exemplo, na figura 2, após a ocorrência de falhas que afetaram as tarefas j_1 e j_3, as mesmas recuperam e a tarefa j_2, que não falhou, perde seu *deadline*. Podemos ver, pelo exemplo 2, que durante a execução de um sistema podem ocorrer erros, levando-o a um estado inconsistente. Tolerar falhas significa manter o sistema em funcionamento correto mesmo na presença de erros. Existem diversas técnicas de tolerância a falhas que se baseiam na detecção e recuperação de erros [Laprie 1993, Laprie 1992]. Tais técnicas são usadas em sistemas computacionais em geral, mas assumem um papel de extrema importância em STRC, pois estes têm

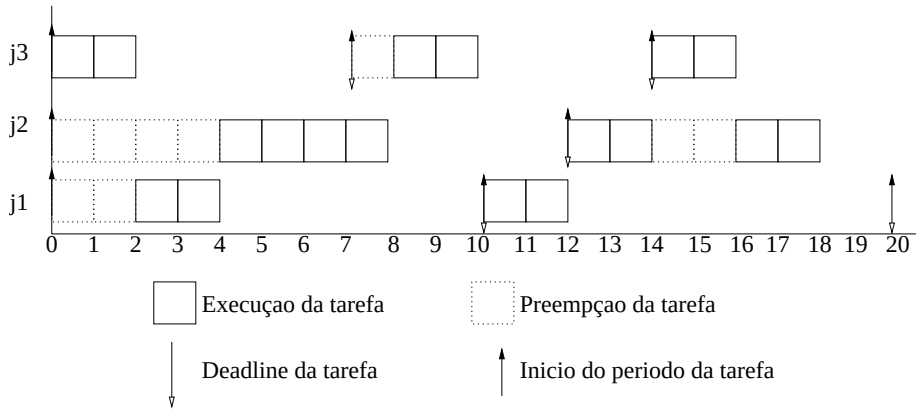

Figura 1. Escalonamento das tarefas j_1, j_2 e j_3 segundo o EDF

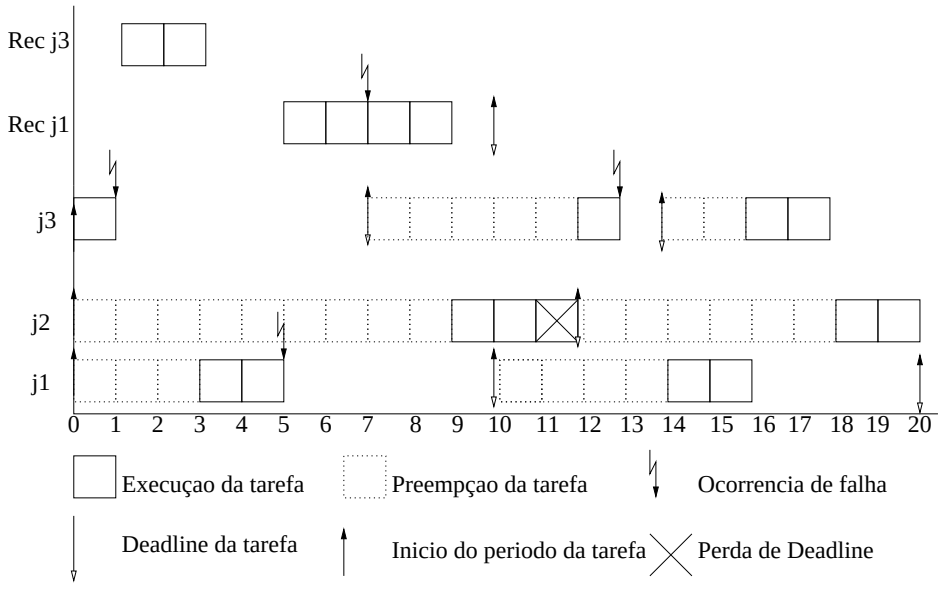

Figura 2. Escalonamento das tarefas j_1, j_2 e j_3 segundo o EDF com ocorrência de falhas

um agravante adicional, pois erros são eventos imprevisíveis e existem *deadlines* para serem cumpridos. Ao tentar recuperar o sistema de um dado erro, por exemplo, corre-se o risco de causar interferência em tarefas menos prioritárias, fazendo com que as mesmas percam seus *deadlines*. Obviamente, isto é indesejável. De maneira simplificada, assumimos que a re-execução de tarefas falhas é suficiente para recuperar o sistema.

Servidores de tarefas aperiódicas, que são utilizados comumente em STRNC, provêm isolamento entre a execução de tarefas aperiódicas sobre tarefas periódicas com *deadlines* críticos. Comparando-se tarefas aperiódicas e rotinas de recuperação de erros, percebe-se que há alguma semelhança: tais rotinas podem ocorrer a qualquer momento (devido à ocorrência de erros) e sua execução não pode interferir no cumprimento dos *deadlines* das demais tarefas do sistema. Motivados por esta semelhança, investigamos o uso de servidores para escalonar rotinas de recuperação.

3. Servidores de Tarefas Aperiódicas

Um servidor de tarefa aperiódica é um mecanismo que visa executar as tarefas aperiódicas, no nosso caso, tarefas de recuperação, controlando a interferência sobre as tarefas periódicas. Pode-se definir um servidor como uma tarefa virtual de período p_s e tempo máximo de execução c_s. Sua prioridade dependerá da política de escalonamento adotada e/ou das prioridades das tarefas de recuperação que estão em sua fila. O servidor diminui o tempo de espera das tarefas de recuperação ao mesmo tempo em que não afeta a escalonabilidade do sistema.

Os servidores podem assumir alguns estados: ***backlogged***, quando existem tarefas de recuperação para executar, ou seja, a fila associada ao servidor não está vazia, e o servidor tem *carga* para executar; **ocioso**, quando a fila de tarefas de recuperação encontra-se vazia, ou não encontra-se vazia, mas o servidor não tem *carga* para executar; **ativo**, quando uma tarefa de recuperação está executando.

Pode-se dividir os diferentes tipos de servidores em duas classes: servidores *polling* e servidores com preservação de tamanho de banda. Como o primeiro apresenta desempenho inferior ao segundo, focaremos nosso estudo nos servidores com preservação de tamanho de banda. Os servidores com preservação de tamanho de banda, como o nome indica, são servidores que preservam suas cargas para num tempo futuro executar tarefas de recuperação. Tais servidores possuem regras para estabelecer quando estes consumirão (decrementarão) sua *carga*, que fora guardada, e como atribuirão (incrementarão) um novo valor à mesma. As principais diferenças entre os servidores com preservação de tamanho de banda estão relacionadas a essas regras que são comumente chamadas de regras de consumo e regras recarga, respectivamente.

3.1. Servidor *Deferrable* Dinâmico

O servidor *deferrable* dinâmico (DDS - *Dynamic Deferrable Server*) é o tipo mais simples de servidor com preservação de tamanho de banda. Sua *carga* é realizada periodicamente a cada intervalo p_s. Ao detectar que a fila de tarefas de recuperação está vazia, o servidor não executa e mantém sua *carga* intacta. Ao chegar uma tarefa de recuperação na fila, o DDS é acionado e executa a tarefa até a *carga* esgotar. Seu

deadline, d_s, está sempre definido [Liu 2000] e é atualizado a cada período p_s.

Regras de Consumo: A *carga* é consumida a uma taxa de uma execução por unidade de tempo. A mesma só poderá ser consumida com o DDS em estado ativo.

Regra de Recarga: A *carga* é realizada para c_s no início de cada período.

Para melhor ilustrarmos o funcionamento do DDS, considere a figura 3, onde j_1, j_2 e j_3 são tarefas periódicas. Os atributos destas tarefas são $c_1 = 2$ e $D_1 = 10$, $c_2 = 4$ e $D_2 = 12$, $c_3 = 2$ e $D_3 = 7$. O DDS tem $c_s = 2$ e $D_s = 10$. O sistema funciona idêntico ao EDF mostrado na figura 1 até o momento 7, quando uma falha ocorre e uma tarefa de recuperação j_a é gerada, com tempo máximo de execução, c_a, igual a 4. A partir deste momento o DDS é escalonado junto com as tarefas periódicas do sistema, pois passa de estado ocioso para *backlogged*, com *deadline* absoluto $d_{s,1} = 10$ e com 2 unidades de tempo para executar, que é seu tempo máximo de execução.

Quando sua prioridade é máxima, o mesmo executa durante 2 unidades de tempo. Apesar da tarefa de recuperação ainda não ter concluído sua execução, pois precisa executar 4 unidades de tempo, o DDS passa ao estado ocioso. Somente será re-escalonado no período seguinte, com $d_{s,2} = 20$.

No pior caso, podem existir duas execuções consecutivas do DDS. Basta, para isso, que ele comece a executar uma tarefa de recuperação a c_s unidades de tempo da sua próxima ativação, com a *carga* igual a c_s e que a fila de prontos do servidor não esteja vazia, estando o mesmo com máxima prioridade. Isto significa que o servidor pode atrasar a execução de tarefas menos prioritárias, no pior caso, em $2c_s$ unidades de tempo. Num sistema com n tarefas periódicas, preemptivas e independentes, que contém um DDS e segue a política de escalonamento EDF, o isolamento das tarefas é garantido se [Liu 2000]:

$$\sum_{k=1}^{n} \frac{c_k}{\min(d_k, p_k)} + u_s(1 + (\frac{p_s - c_s}{d_i})) \leq 1. \quad (1)$$

3.2. Servidor de Constante Utilização

O servidor de constante utilização (CUS - *Constant Utilization Server*) [Deng et al. 1997] é definido pelo seu tamanho u_s, que representa a fração do processador reservada ao servidor. Assim, como o DDS, o *deadline*, d_s, do CUS está sempre definido. O CUS se torna elegível para execução, quando sua *carga* é maior do que zero e se torna inelegível quando a *carga* termina.

Regras de Consumo: A *carga* do CUS somente é consumida quando ele está em execução. Para tanto, o servidor deve estar executando com permissão do escalonador EDF, segundo sua prioridade (*deadline*). Alguma tarefa deve estar na fila de tarefas de recuperação, para que, conseqüentemente, o CUS esteja no estado *backlogged*.

Regras de Recarga: Seja c_s o tempo máximo de execução de um CUS e d_s o seu respectivo *deadline*. Seja j_a uma tarefa de recuperação, c_a seu tempo máximo de

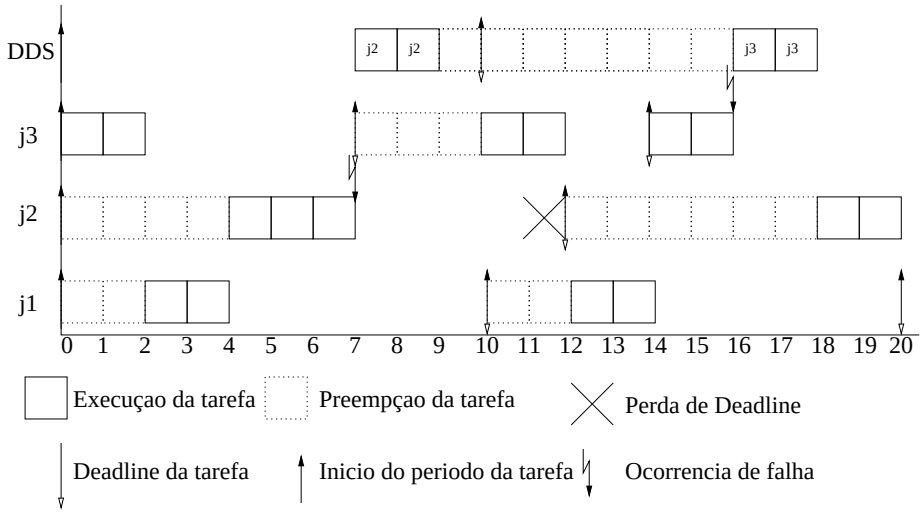

Figura 3. Escalonamento das tarefas j_1, j_2 e j_3 segundo o EDF, com DDS para executar as tarefas de recuperação

execução e D_a o seu *deadline* relativo. Inicialmente, *carga* $= 0$ e $d_s = \infty$. Quando j_a chega no tempo t estando a fila de tarefas aperiódicas vazia, temos que se $t \leq d_s$, o servidor não precisa fazer nada. Se $t > d_s$, o servidor atribui um novo valor a d_s, $d_s = t + \frac{c_a}{u_s}$, e iguala a *carga* a c_a. Ao atingir o *deadline* atual, $d_{s,k}$, o servidor terá seu *deadline* modificado segundo as seguintes regras: Seja a tarefa de recuperação j_a, que se encontra no topo da fila de prontos do servidor, com tempo máximo de execução c_a e *deadline* relativo d_a, (a) se o servidor está em estado *backlogged*, o *deadline* do servidor é modificado para $d_s = d_s + \frac{c_a}{u_s}$ e sua *carga* é modificada para c_k. (b) se o servidor estiver em estado ocioso, nada deverá ser feito [Deng et al. 1997]. Quando o servidor está *backlogged*, ele é escalonado junto com as tarefas periódicas, segundo o algoritmo EDF.

Considere a figura 4, sendo j_1, j_2 e j_3 as mesmas tarefas utilizadas no exemplo anterior. O CUS tem $u_s = 0,2$. O escalonamento é idêntico ao mostrado na figura 1 até o instante 7, quando uma falha ocorre, gerando assim uma tarefa de recuperação j_a com tempo máximo de execução, c_a, igual a 4.

A partir deste momento, o CUS é escalonado junto com as tarefas periódicas do sistema, pois passa de estado ocioso para *backlogged*. Quando a tarefa de recuperação j_a chega, o *deadline* do servidor é calculado para 27, pois $d_{s,1} = 7 + (\frac{4}{0,2}) = 27$. No instante 9, o CUS é a tarefa mais prioritária e executa j_a durante uma unidade de tempo. No instante 11, quando a tarefa de recuperação j_b chega, o *deadline* do CUS não é alterado e j_b é colocada em sua fila de tarefas de recuperação.

Num sistema com uma série de tarefas periódicas com taxa de utilização do processador u_p e um CUS com taxa de utilização do processador u_s. Nenhuma tarefa crítica perderá seu *deadline*, segundo o algoritmo EDF, se e somente se, a equação

$$(u_s + u_p) \leq 1 \tag{2}$$

for válida [Spuri and Buttazzo 1996].

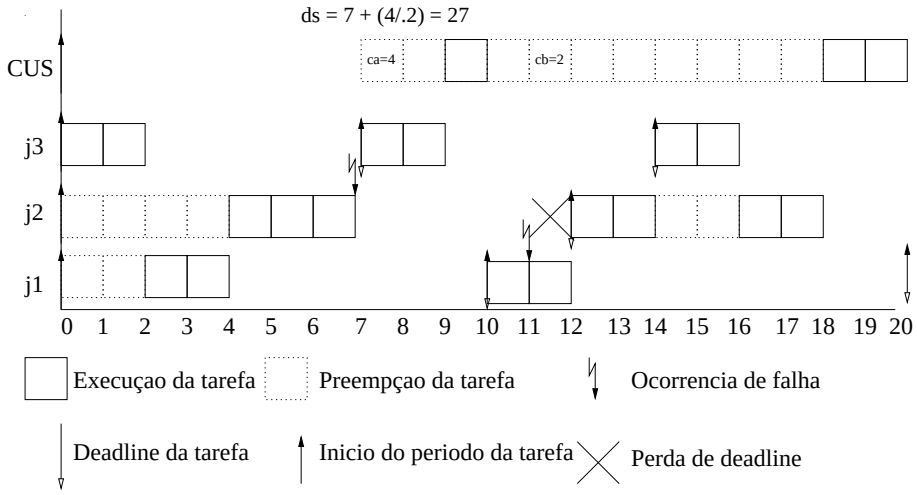

Figura 4. Escalonamento das tarefas j_1, j_2 e j_3 segundo o EDF, com CUS para executar as tarefas de recuperação

3.3. Servidor de Banda Total

O servidor com preservação total de tamanho de banda (TBS - *Total Bandwidth Server*) [Spuri and Buttazzo 1996] melhora o aproveitamento do CUS, através da utilização do tempo de *background* do processador não utilizado pelas tarefas periódicas. No momento em que a *carga* é esgotada, caso o servidor esteja *backlogged*, ele é imediatamente recarregado. Caso contrário, a *carga* é realizada assim que o servidor atingir o estado *backlogged*.

A determinação dos *deadlines* tem que ser feita para cada tarefa de recuperação, visando que a total utilização do processador nunca exceda o valor máximo da utilização do processador do servidor, u_s. Quando a *k-ésima* tarefa chegar, no tempo t, o servidor recebe a seguinte definição do seu *k-ésimo deadline*:

$$d_{s,k} = \max(t, d_{s,k-1}) + (\frac{c_a}{u_s}), \tag{3}$$

onde c_a é o tempo máximo de execução da tarefa de recuperação que se encontra no topo da fila e u_s é a utilização do processador reservada ao TBS. Por definição, $d_{s,0} = \infty$.

O algoritmo do TBS é, essencialmente, o mesmo do CUS. A única diferença entre eles é que no algoritmo TBS, quando uma tarefa que estava no topo da fila de prontos do servidor termina, o TBS realiza sua recarga imediata, se a fila de prontos não estiver vazia. Desta forma, pode-se usar o tempo de *background*, quando não existe tarefas periódicas para executar, para diminuir o tempo de resposta das tarefas de recuperação.

Intuitivamente, a definição dos *deadlines* é tal que, em cada intervalo de tempo, a razão alocada pelo EDF para as tarefas de recuperação não excede a utilização do processador do TBS, u_s.

Regras de Consumo: Idênticas às do CUS.

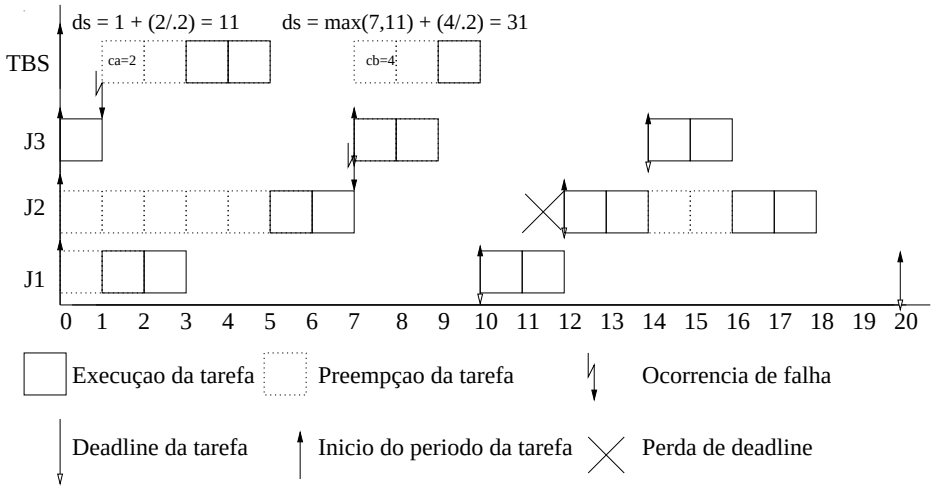

Figura 5. Escalonamento das tarefas j_1, j_2 e j_3 segundo o EDF, com TBS para executar as tarefas de recuperação

Regras de Recarga: Inicialmente, *carga* $= 0$ e $d_{s,0} = \infty$. Quando uma tarefa de recuperação com tempo de execução c_a chega no tempo t, com a fila de tarefas de recuperação do TBS vazia, aplica-se a equação 3 e *carga* é igualada a c_a.

Quando o TBS completa sua tarefa de recuperação corrente, esta é removida da fila de prontos. Se o servidor está em estado *backlogged*, seja a tarefa j_a a tarefa de recuperação que se encontra no topo da fila de prontos do servidor com tempo máximo de execução c_a, o *deadline* $d_{s,k}$ do servidor será igualado a $d_{s,k-1} + \frac{c_a}{u_s}$ e sua *carga* será c_a. Se o TBS estiver ocioso, nada será feito.

Para entendermos melhor o funcionamento do TBS, podemos observar a figura 5. Consideremos as tarefas j_1, j_2 e j_3 usadas nos exemplos anteriores, quando chega uma tarefa de recuperação j_a, no instante 1, com tempo máximo de execução, c_a, igual a 2, o *deadline* do TBS é calculado, $d_{s,1} = t + (\frac{c_a}{u_s}) = 1 + (\frac{2}{0,2}) = 11$. O TBS passa de estado ocioso para *backlogged* e é escalonado junto com as tarefas periódicas do sistema.

No instante 3, o TBS é a tarefa de mais alta prioridade e este executa duas unidades de tempo, concluindo a execução da tarefa j_a. A tarefa de recuperação j_b chega no instante 7, com $c_b = 4$. j_b é colocada em sua fila de tarefas de recuperação. Como a fila encontra-se vazia, o *deadline* do servidor é recalculado, $d_{s,2} = \max(t, d_{s,1}) + (\frac{c_b}{u_s}) = \max(7, 11) + (\frac{4}{0,2}) = 31$. O TBS passa a ter baixa prioridade e as tarefas periódicas executam. No instante 12, a tarefa periódica correspondente a j_b, j_2, atinge o *deadline* e j_b é retirada da fila de tarefas do TBS.

Em cada intervalo de tempo $[t_1, t_2]$, se T_a é o tempo total de execução demandado pelas tarefas de recuperação que chegaram em t_1, ou depois de t_1, e têm *deadline* menor ou igual a t_2, então a equação $T_a \leq \frac{(t_2 - t_1)c_s}{p_s}$ é válida [Spuri and Buttazzo 1996].

3.4. Servidor de Banda Constante

O servidor com preservação de tamanho de banda constante (CBS - *Constant Bandwidth Server*) [Abeni and Buttazzo 2004] garante que se u_s é uma fração do tempo do processador designada para o servidor, esta contribuição para a total utilização não é maior do que u_s, mesmo que as tarefas de recuperação solicitem uma parcela maior. Esta propriedade não é válida para o TBS, nem para o CUS, nos quais contribuições são limitadas a u_s se, e somente se, assume que todas as tarefas servidas executam não mais que o declarado.

A idéia básica do CBS é que quando uma nova tarefa entra no sistema é atribuído um escalonamento adequado ao seu *deadline*, sendo inserida na fila de prontos do EDF. Se a tarefa tentar executar mais tempo do que o permitido, seu *deadline* é adiado, eliminando a interferência desta tarefa nas demais. Notemos que a tarefa permanecerá elegível a execução. Desta forma, o CBS se comporta como um algoritmo de conservação, explorando a folga disponível no sistema.

Um CBS se caracteriza pelo par ordenado (c_s, p_s), onde c_s é o máximo valor de *carga* disponível ao servidor e p_s é o período de recarga do servidor. A razão $u_s = \frac{c_s}{p_s}$ é denominada como a utilização total do processador do servidor. A cada ativação k do CBS, um *deadline*, $d_{s,k}$, é definido. Inicialmente, $d_{s,0} = \infty$.

Regras de Consumo: Para cada tarefa atendida, $j_{i,l}$, é determinado um *deadline* dinâmico, $d_{i,l}$, igual ao *deadline* corrente do CBS. Sempre que uma tarefa da fila de prontos do servidor executa, a *carga* é decrementada na mesma quantidade de unidades de tempo.

Regras de Recarga: Quando $carga = 0$, esta é igualada a c_s, e um novo *deadline* é determinado através da equação $d_{s,k+1} = d_{s,k} + p_s$.

Quando uma tarefa de recuperação $j_{i,l}$ chega e o servidor está ativo, ela é colocada na fila de prontos do servidor de acordo com a política de escalonamento da mesma. Quando uma tarefa de recuperação $j_{i,l}$ chega e o servidor está ocioso, se a *carga* for $\geq (d_{s,k} - t)u_s$, então o servidor gerará um novo *deadline* $d_{s,k+1} = t + p_s$ e a *carga* será realizada, caso contrário, a tarefa será atendida com o último *deadline* do servidor $d_{s,k}$, usando a *carga* corrente. Quando uma tarefa termina de executar, a próxima tarefa que estiver no topo da fila de prontos, caso exista, é atendida usando a *carga* e o *deadline* correntes. Se não existirem tarefas na fila, o servidor passa ao estado ocioso.

Um exemplo de CBS pode ser visto na figura 6, onde utilizamos as mesmas tarefas periódicas dos demais exemplos. O CBS tem $c_s = 2$ e $p_s = 10$, ou seja, é formado pelo par ordenado $(2, 10)$. O escalonamento é idêntico ao mostrado na figura 1 até o instante 7, quando uma tarefa de recuperação j_a com tempo máximo de execução, c_a, igual a 4 chega ao sistema. Neste instante, o CBS passa de estado ocioso para *backlogged*.

Quando a falha ocorre no momento 7, é gerada a tarefa de recuperação j_a, a *carga* do CBS é comparada com $(d_{s,1} - t)u_s = (10 - 7)0,2 = 0,8$, como a *carga* é maior

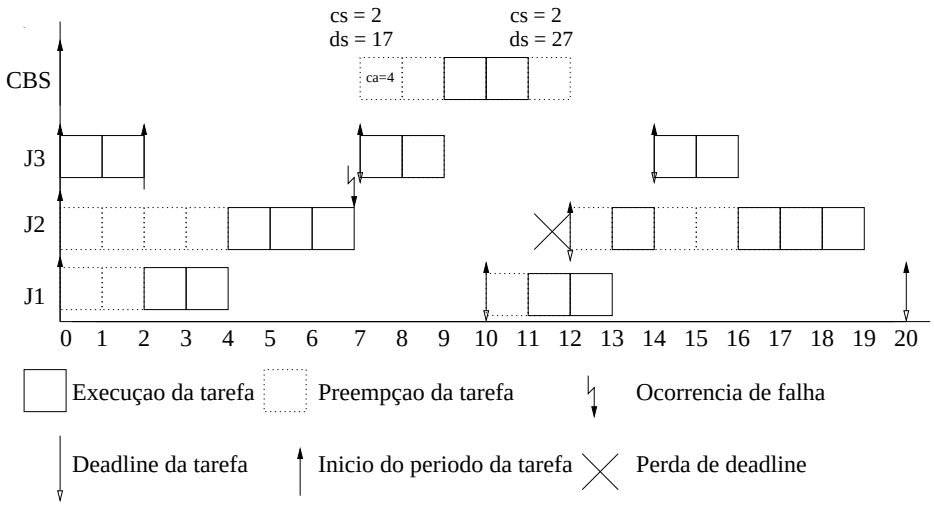

Figura 6. Escalonamento das tarefas j_1, j_2 e j_3 segundo o EDF, com CBS para executar as tarefas de recuperação

que o inteiro mais próximo de $0,8$, que é 1, um novo *deadline* deve ser calculado. $d_{s,2} = t + p_s = 7 + 10 = 17$. No instante 9, o CBS passa a ser a tarefa mais prioritária e começa a executar j_a, porém no instante 11 sua *carga* atinge zero e seu *deadline* é recalculado para $d_{s,3} = d_{s,2} + p_s = 17 + 10 = 27$. Logo, o CBS deixa de ser o mais prioritário e é preemptado para a execução das tarefas periódicas. No instante 12 a tarefa j_a atinge seu *deadline* e é retirada da fila do CBS.

A utilização do processador do CBS com parâmetros (c_s, p_s) é $u_s = \frac{c_s}{p_s}$, independente dos tempos computacionais e do padrão chegada das tarefas atendidas. Dada uma série de n tarefas periódicas com utilização do processador u_p e um CBS com utilização do processador u_s. O sistema não causa perda de *deadlines* se, e somente se, a equação 2 for válida [Spuri and Buttazzo 1996].

4. Simulação

Assumimos um modelo de sistema para simulação ilustrado na figura 7. Um escalonador EDF escolhe, entre duas filas, qual tarefa irá executar. Há uma fila de tarefas periódicas e uma fila de tarefas de recuperação. A segunda é controlada por um servidor de tarefas aperiódicas. As tarefas de recuperação submetidas ao sistema representam a re-execução das tarefas que sofreram erros durante sua execução. Neste experimento, um servidor de tarefas aperiódicas foi adaptado como um mecanismo que visa executar as tarefas de recuperação críticas controlando a interferência sobre as tarefas periódicas críticas.

Baseado neste modelo, variamos o tipo de servidor a fim de avaliar suas características com relação à capacidade do sistema em executar rotinas de recuperação. Cada servidor escalona sua fila de tarefas aperiódicas segundo o algoritmo EDF.

O conjunto de tarefas simuladas foi composto de nove tarefas periódicas. O tempo de simulação considerado foi de *10.000* unidades de tempo. Desta forma, acreditamos que a simulação foi longa o suficiente para observarmos o comportamento dos servidores, pois o período das tarefas foram limitados a 500 unidades de tempo. Durante a simulação,

erros foram gerados aleatoriamente, a taxa de erros foi de 10% das unidades de tempo disponíveis.

A tarefa em execução no instante do erro sofre interrupção e uma rotina de recuperação (re-execução) é inserida na fila do servidor. Não foram considerados na simulação tempo de troca de contexto ou tempo de detecção de erro. Estes foram considerados instantâneos.

Para efeitos de comparação, consideramos também o escalonador EDF como critério de escalonamento, ou seja, sem o uso de servidores. Neste caso, as tarefas de recuperação são re-enfileiradas na fila de tarefas periódicas. Esta tarefa de recuperação deve cumprir a meta da tarefa periódica que falhou, assim, elas têm os mesmos atributos, como *deadline* e tempo máximo de execução.

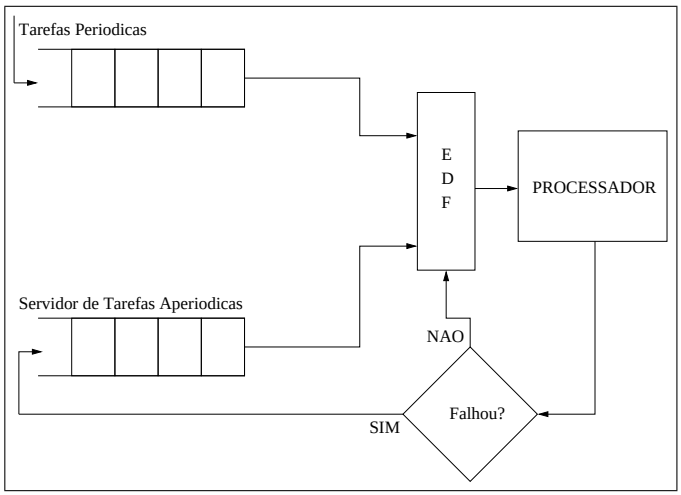

Figura 7. Modelo do sistema

A avaliação do sistema simulado se baseou em quatro parâmetros de comparação:

1. *Taxa de recuperação* - representa o percentual de tarefas recuperadas no sistema e é calculada através da divisão da quantidade de tarefas que recuperaram com sucesso pela quantidade total de tarefas que tentaram recuperar.
2. *Taxa de interferência* - taxa de *deadlines* perdidos de tarefas periódicas, é calculada como a divisão da quantidade de tarefas que perderam o *deadline*, sem serem afetadas por falhas ao longo de sua execução, pela quantidade total de tarefas do sistema.
3. *Taxa de deadlines perdidos* - significa o percentual de tarefas que perderam seus *deadlines*. Esta é obtida através da divisão da quantidade total de tarefas que perderam *deadline*, por qualquer motivo, pela quantidade total de tarefas executadas no sistema.
4. *Utilização real* - representa o percentual do processador utilizado pelo sistema. Esse parâmetro serve para mostrar que, se todas as unidades de tempo foram usadas para executar tarefas, a utilização foi total.

4.1. Avaliação dos Resultados

Como pode ser observado na tabela 1, o escalonador EDF apresentou um alto índice de recuperação, porém provoca interferência nas tarefas que não foram afetadas por erros.

Este é o efeito indesejado que queremos eliminar com o uso de servidores. O DDS

Política de escalonamento	EDF	DDS	CUS	TBS	CBS
Qtde de tarefas executadas	1242	424	1135	1149	1148
Taxa de recuperação	0,312	0,0287	0,220	0,220	0,266
Taxa de interferência	0,02	0	0	0	0
Taxa de *deadlines* perdidos	0,148	0,559	0,172	0,169	0,161
Utilização real	1	0,3407	0,9845	0,9995	1

Tabela 1. Tabela comparativa de desempenho de diversos servidores, feita a partir de simulação realizada neste trabalho

apresenta um baixo índice de recuperação, por isso não é indicado para ser usado em sistemas que precisam recuperar suas tarefas falhas. Além disso, desperdiça uma grande parte da banda, na medida em que só utiliza uma quantidade fixa de tempo por período. Quando sobra unidades de tempo devido a ociosidade do processador num período, essas não são aproveitadas pelo DDS, subutilizando o potencial de processamento do sistema.

O CUS apresenta um melhor índice de recuperação do que o DDS, porém ainda não faz uso de toda a banda disponível no sistema. Esse servidor tem a característica de tentar utilizar o máximo de banda possível. Por ser conservador, utiliza apenas o tamanho de banda seguramente disponível, evitando qualquer possibilidade de interferência nas demais tarefas.

O TBS, que é um melhoramento do CUS, tem um índice de recuperação maior, pois ele usa uma parte maior da banda de utilização disponível do sistema. Realiza sua *carga* com mais frequência, porém ainda não faz utilização de toda a banda. Melhor do que o CUS, devido a uma pequena, mas importante, modificação nas regras de recarga, este servidor, que realiza sua *carga* assim que possível executa, em alguns casos, algumas unidades de tempo a mais que o CUS.

O CBS tenta usar todo o tempo disponível do sistema em prol das tarefas de recuperação. Isto fez com que sua utilização fosse máxima e sua taxa de recuperação crescesse, porém ainda encontra-se inferior a taxa de recuperação do EDF, com a vantagem de não ocasionar interferência no sistema. Este servidor foi o que apresentou melhor taxa de recuperação, com menor taxa de deadlines perdidos.

Como pôde ser observado, o CBS apresentou uma resposta melhor do que os demais, porém seu nível de recuperação apresentado é um pouco inferior (menos que 5%) quando comparado com o escalonador EDF. Porém, ele não causa interferências nas tarefas periódicas.

5. Conclusão

A teoria de escalonamento em Sistemas de Tempo Real é bastante complexa e apresenta muitas abordagem de escalonadores para diferentes propósitos. Neste estudo, apresentamos uma verificação do uso de várias técnicas de uma das abordagens: servidores de tarefas aperiódicas.

Servidores são comumente usados para prover isolamento temporal entre as tarefas críticas e não-críticas. Seu uso para tolerância a falhas ainda não havia sido realizado de maneira satisfatória em sistemas baseados em escalonamento dinâmico.

A simulação realizada focou em alguns dos mais conhecidos servidores. Os resultados encontrados mostram as diferenças práticas entre os servidores no uso para fins de tolerância a falhas, possibilitando importantes observações. os resultados apontam o CBS como a melhor das técnicas aqui apresentadas para o propósito de tolerância a falhas. Acreditamos contudo, que possa-se adaptar algumas das técnicas estudadas especificamente para tolerãncia a falhas. Neste contexto, o estudo realizado deve servir de base para subsidiar futuras pesquisas, onde os aspectos específicos de tolerância a falhas possam ser levados em consideração na definição das regras dos servidores.

Referências

Abeni, L. and Buttazzo, G. (2004). Resource reservation in dynamic real-time systems. *Real-Time Syst.*, 27(2):123–167.

Burns, A. and Wellings, A. (2001). *Real-Time Systems ans Programming Languages*. Addison Wesley, third edition.

Deng, Z., Liu, J. W.-S., and Sun, J. (1997). A scheme for scheduling hard real-time applications in open system environment. In *Proceedings of Ninth Euromicro Workshop on Real-Time Systems*, Toledo, Spain. IEEE Computer Society.

Laprie, J.-C., editor (1992). *Dependability: Basic Concepts and Terminology*, volume 5 of *Dependable Computing and Fault Tolerance*.

Laprie, J. C. (1993). Dependability: from concepts to limits. *12th IFAC International Conference on Computer Safety, Reliability and Security (SAFECOMP'93)*, 1(1):157–168. Communication invitée.

Liu, J. (2000). *Real-Time Systems*. Prentice Hall, first edition.

Spuri, M. and Buttazzo, G. (1996). Scheduling aperiodic tasks in dynamic priority systems. volume 10, pages 179–210. Real-Time Systems.