Suporte à Degradação Controlada em Sistemas de Tempo Real

Ana Carolina Sokolonski[1] **, George Lima**[1] **, Luciano Porto Barreto**[1]

[1]Programa de Pós-Graduação em Mecatrônica
Departamento de Ciência da Computação
Instituto de Matemática
Universidade Federal da Bahia
Av. Adhemar de Barros s/n - Ondina - Salvador/BA

{anaanton, gmlima, lportoba}@ufba.br

Abstract. *In real-time systems, an error-recovery task is usually modelled/designed as an aperiodic task that must finish before its deadline without comprimising the temporal correctness of the whole system. To achieve this goal, there is an emerging need for proper and efficient task scheduling mechanisms. This paper describes a scheduling model for fault-tolerant systems in which a recovery task is admitted by gracefully degrading the quality of service of periodic tasks. We evaluated the scheduling model by simulation whose results show the effectiveness of the proposed approach.*

Resumo. *Nos sistemas de tempo real, rotinas de recuperação de falhas podem ser modeladas como tarefas aperiódicas que devem cumprir seus prazos de execução sem inviabilizar a correção temporal do sistema como um todo. Para tanto, é necessário o desenvolvimento de mecanismos adequados e eficientes de escalonamento. Este artigo descreve um modelo de escalonamento para tolerância a falhas no qual as rotinas de recuperação são admitidas mediante à degradação controlada da qualidade do serviço de tarefas periódicas. O modelo foi avaliado por simulação e os resultados encontrados indicam a eficácia da abordagem proposta.*

1. Introdução

Sistemas de tempo real são sistemas computacionais que devem reagir a eventos do ambiente respeitando restrições temporais previamente estabelecidas. Tais sistemas são geralmente modelados como um conjunto de tarefas, ativadas como conseqüência da ocorrência dos eventos aos quais elas estão associadas. As tarefas ativadas possuem prazos máximos de execução ou *deadlines*. Para garantir a correção de tais sistemas deve-se assegurar que todas as suas tarefas produzam resultados corretos sem violar suas especificações temporais.

Sistemas de controle são exemplos clássicos de sistemas de tempo real. As tarefas deste tipo de sistema são ativadas contínua e periodicamente para obter o valor da variável controlada (ex. temperatura), e atuar de forma condizente para manter seu valor num patamar desejável. Nesse modelo, o tempo máximo de execução e o período entre ativações de cada tarefa são conhecidos *a priori*. Outro tipo comum de tarefa é a esporádica, para

*Ana Carolina é bolsista CAPES. George Lima recebe apoio do CNPq (475851/2006-4) e FAPESB (7630/2006 e 3381/2005).

a qual conhece-se o intervalo mínimo de tempo entre duas ativações consecutivas. Pode-se dizer que tanto tarefas periódicas quanto esporádicas são bem comportadas, o que facilita o projeto do escalonador, principal mecanismo de suporte a execução de tais sistemas. Uma terceira classe de tarefas é denominada aperiódica, para a qual não se têm informações sobre a regularidade de ativação. Exemplos típicos de sistemas compostos predominantemente por tarefas aperiódicas são os de telefonia, multimídia etc. Para tais sistemas, o mais comum é garantir apenas níveis de qualidade de serviço, pois estes sofrem, eventualmente, períodos de sobrecarga (ex: num aplicativo de exibição de vídeo, podemos garantir um número mínimo de quadros/segundo em períodos de sobrecarga).

Sistemas de tempo real modernos são compostos tanto de tarefas periódicas ou esporádicas quanto de tarefas aperiódicas. Por exemplo, pode-se integrar num sistema de controle, tarefas de processamento de áudio e vídeo, melhorando a interface com o operador. Nesta situação, o escalonador é projetado para garantir os *deadlines* das tarefas periódicas e esporádicas, ao passo que, a qualidade de serviço das tarefas aperiódicas é maximizada.

Além do atendimento às restrições temporais, é desejável que um sistema de tempo real forneça mecanismos para tolerância a falhas. Entretanto, capacitar um sistema de tempo real para esse fim introduz custos computacionais e complexidade no projeto das aplicações. Por exemplo, o instante da detecção de um erro é inerentemente imprevisível e pode disparar rotinas de recuperação cujo objetivo é levar o sistema a um estado seguro ou consistente. A ausência de regularidade dessas rotinas onera excessivamente o custo das políticas de escalonamento e seu tratamento inadequado pode incorrer em perdas de *deadlines* de outras tarefas do sistema. Uma solução para esse problema utiliza técnicas de mascaramento de falhas através da replicação espacial das tarefas (ex.: programação com N-versões e replicação de tarefas em sistemas distribuídos). Contudo, tais soluções exibem alto custo de implementação associado ao sincronismo exigido entre as tarefas [Kopetz 1997, Poledna 1996]. Por esta razão, tais soluções são recomendáveis apenas para sistemas críticos, onde vidas humanas podem estar em risco, por exemplo. Uma alternativa mais simples consiste em modelar as rotinas de recuperação como tarefas aperiódicas e adaptar a política de escalonamento com o intuito de permitir a execução das tarefas de recuperação com baixo impacto frente às tarefas periódicas. Este trabalho segue esta segunda abordagem.

Incorporar tarefas aperiódicas, doravante consideradas como rotinas de recuperação, ao modelo de escalonamento, no entanto, requer mecanismos de escalonamento mais apropriados aos cenários de erros. Para tanto, consideramos que cada tarefa periódica disponibiliza um elenco de implementações com custos computacionais distintos e decrescentes, os quais representam níveis de degradação da qualidade do serviço prestado. Este trabalho propõe meios através dos quais é possível ajustar dinamicamente os tempos de execução das tarefas periódicas quando da ativação de tarefas de recuperação. Mais especificamente, o método proposto consiste em efetuar uma *degradação controlada* (*graceful degradation*) do conjunto de tarefas periódicas com o objetivo de permitir a execução de tarefas aperiódicas de recuperação. A idéia principal do trabalho pode ser resumida da seguinte forma. Através de um teste de admissão, determina-se a possibilidade de perda de *deadlines* causadas por ativações de tarefas aperiódicas. Em caso positivo, o escalonador degrada a qualidade das tarefas periódicas,

escolhendo as versões das mesmas com custos reduzidos. Tal mecanismo utiliza como base o escalonador EDF (*Earliest Deadline First*), pois este oferece alta capacidade de adaptação do sistema em tempo de execução.

Em suma, as principais contribuicões deste trabalho são:

1. Derivamos um teste de escalonamento, executado no instante de ativação da tarefa aperiódica, que permite saber, em tempo de execução, se há risco de perda de *deadlines* de tarefas periódicas;
2. Propomos um algoritmo que efetua degradação controlada em sistemas de tempo real de modo que tarefas aperiódicas sejam executadas sem afetar a correção temporal das tarefas periódicas;
3. Adaptamos uma técnica bastante usada em escalonamento de tarefas aperiódicas não críticas, o servidor TBS [Spuri and Buttazzo 1996], e mostramos suas limitações no contexto de tolerância a falhas;
4. Através de simulação, avaliamos várias configurações de escalonamento em cenários nos quais as tarefas aperiódicas são executadas concorrentemente às tarefas periódicas. Apesar da proposta estar ainda em estágios preliminares, os resultados das simulações indicam a superioridade da técnica descrita.

O restante deste artigo é estruturado da seguinte forma. A seção 2 traz um resumo dos principais resultados sobre esquemas de escalonamento para sistemas de tempo real tolerantes a falhas. Na seção 3, descreveremos o modelo do sistema e a notação utilizada. A seção 4 descreve a abordagem de escalonamento proposta. Resultados de avaliação são apresentados na seção 5. Por fim, a seção 6 conclui o trabalho e aborda perspectivas futuras.

2. Trabalhos Relacionados e Motivação

Considerar os efeitos de rotinas de recuperação no escalonamento de tarefas em sistemas de tempo real críticos não é uma preocupação nova. Tanto escalonadores estáticos quanto dinâmicos têm sido adaptados para tal fim. A maioria dos trabalhos, contudo, são destinados a equacionar os efeitos que rotinas de recuperação exercem no escalonamento. Para tanto, ou impõe-se um modelo restritivo de falhas ou de tarefas.

Esquemas de escalonamento estático consideram que as prioridades das tarefas ou os tempos de processador a elas alocados são determinados em tempo de projeto. Apesar deste modelo dificultar a incorporação de rotinas de recuperação, por estas serem ativadas aperiodicamente, escalonamento estático é considerado mais previsível [Liu 2000]. Algumas abordagens considerando tolerância a falhas são muito restritivas e, portanto, pouco práticas. Por exemplo, em [Liestman and Campbell 1986], os autores consideraram apenas tarefas periódicas, cujos períodos são múltiplos entre si. Há alguns trabalhos que restringem as rotinas de recuperação a simples re-execuções de tarefas [Ghosh et al. 1998, Ghosh et al. 1995]. Em [Kandasamy et al. 1999], os autores descrevem um esquema que requer que todo o escalonamento de tarefas e a reserva de tempo para recuperação seja realizada em tempo de projeto. Um modelo mais flexível foi proposto por [Ramos-Thuel and Strosnider 1991], onde um servidor, cujo tempo de execução é previamente alocado, é responsável por executar rotinas de recuperação. Modelos mais genéricos, mas que consideram modelos de falhas específicos também podem ser encontrados [Burns et al. 1996, Lima and Burns 2003, Lima and Burns 2005].

No contexto de escalonamento dinâmico, baseado no EDF, outros autores [Liberato et al. 2000] analisaram, através de simulação do escalonamento, a possibilidade de recuperar tarefas falhas usando a capacidade ociosa do sistema. Este esquema foi em seguida melhorado [Aydin 2004]. Em [Caccamo and Buttazzo 1998], os autores propuseram um elaborado modelo de tarefas com o objetivo de fornecer diferentes níveis de qualidade de serviço em cenários de falhas para tarefas não críticas. Em [Buttazzo and Stankovic 1993], os autores propuseram um modelo robusto de escalonamento, baseado no EDF, para lidar com falhas temporais. No entanto, tal modelo considera que degradação controlada é feita descartando tarefas que podem provocar perdas de *deadlines*. A necessidade de prover tolerância a falhas adaptativa foi preocupação de alguns outros trabalhos. Por exemplo, um modelo adaptativo, que escolhe dinamicamente técnicas de tolerância a falhas, foi proposto no contexto de sistemas distribuídos [González et al. 1997].

O principal objetivo das abordagens citadas acima é modelar e equacionar os efeitos causados pela execução de rotinas de recuperação durante a execução do sistema. Em outras palavras, elas visam a inclusão destes efeitos na *análise de escalonamento*. O efeito degradativo que a ocorrência de tarefas aperiódicas provoca no escalonamento foi estudado em outros trabalhos [de Jesus and Lima 2005, Marucheck and Strosnider 1995]. Mas, tal estudo não considera degradação de tarefas. Este trabalho tem semelhante objetivo, mas considera que tarefas periódicas podem ser degradadas para que rotinas de recuperação possam ser executadas. Assim, buscamos tornar o sistema mais adaptável, considerando que em situações emergenciais, é preferível reduzir a qualidade dos serviços prestados pelo sistema em função de sua correção.

3. Modelo e Notação

Consideraremos um sistema de tempo real uniprocessado com preempção, composto por dois tipos de tarefas, periódicas, $\Gamma_p = \{\tau_1^p, \tau_2^p, ..., \tau_m^p\}$, e aperiódicas, por $\Gamma_a = \{\tau_1^a, \tau_2^a, ..., \tau_n^a\}$. Usaremos indiscriminadamente a notação τ_i para significar uma tarefa, periódica ou não, sempre que não houver relevância quanto ao tipo de tarefa.

A ativação de tarefas aperiódicas é causada por eventos aperiódicos. Estes podem estar associados à detecção de erros, à recepção de sinais do ambiente ou da aplicação indicando alguma situação anômala. Por exemplo, suponha que uma tarefa espera uma mensagem/sinal/leitura de um determinado sensor. A ausência desta recepção pode ser interpretada como um erro e a ativação da tarefa aperiódica é necessária para levar o sistema a um estado consistente. Erros de software, tais como divisão por zero, tratadores de exceções, etc, podem também causar a ativação de tarefas aperiódicas. Tarefas aperiódicas modelam, desta forma, rotinas de recuperação, ou versões alternativas de serviços, que devem ser executadas em situações anômalas. Várias linguagens de programação dão suporte a este modelo de tarefas, a exemplo das que possuem tratadores de exceção.

Cada ativação $k > 0$ da tarefa τ_i representa uma de suas instâncias. Se τ_i é uma tarefa periódica, o instante de ativação da k-ésima instância é dado por $(k-1)T_i^p$, onde $T_i^p > 0$ é o período de τ_i^p. O *deadline* relativo de τ_i é o intervalo máximo de tempo em que qualquer instância de τ_i deve executar, representado por D_i. Em particular, o *deadline* absoluto da k-ésima instância de τ_i^p é dado por $(k-1)T_i^p + D_i^p$. Assumimos que $D_i^p \leq T_i^p$.

Uma tarefa é considerada ativa no instante t se uma de suas instâncias foi ativada

num instante $r \leq t$ e ainda não finalizou antes de t. As tarefas ativas num instante t são representadas por *ativas(t)*. O conjunto das tarefas ativas num determinado intervalo $[t, t')$ é dado por $A(t, t') = \bigcup_{\forall s \in [t,t')} ativas(s)$.

Assumimos que tarefas periódicas possuem uma ou mais versões de execução. Para cada uma destas versões, assumimos que o tempo de execução mais longo é conhecido. Esta é uma hipótese comum em sistemas de tempo real e é necessária para que seja possível demonstrar sua correção temporal [Burns and Wellings 2001, Liu 2000]. As diferentes versões das tarefas periódicas são representadas pelos seus diferentes tempos máximos de execução. Desta forma, a execução de τ_i^p pode requerer $C_{i,0}^p, C_{i,1}^p, \ldots,$ ou $C_{i,m}^p$. Dizemos que $C_{i,0}^p$ representa o custo de execução τ_i^p com a mais alta qualidade de serviço. Em situações anômalas, quando é necessária a execução de alguma tarefa aperiódica, o sistema pode escolher escalonar uma versão de τ_i^p com qualidade reduzida, $C_{i,j}^p$, $j > 0$. Dizemos então que τ_i^p foi degradada, com nível de degradação j. Sem perda de generalidade, assumimos que $C_{i,j+1}^p < C_{i,j}^p$ ($j = 0, 1, \ldots, m-1$). Por simplicidade, assumimos que existem m níveis distintos de degradação para cada tarefa τ_i^p de Γ_p. Atentemos à diferença desta proposta e a computação imprecisa [de Oliveira 1997], que utiliza tarefas cujo tempo máximo de execução é dividido em duas partes, uma obrigatória e outra opcional. Aqui, versões de uma mesma tarefa podem ser completamente diferentes.

Dado que cada instância de tarefa periódica pode executar numa de suas m versões, existe um número exponencial de configurações. Obviamente, não é possível, em tempo de execução, escolher uma configuração apropriada. Assim, para tratar o problema em tempo de execução, assumimos a seguinte simplificação. Inicialmente, o sistema executa suas tarefas periódicas com qualidade máxima. Se for necessário reduzir a qualidade de serviço destas tarefas para admitir uma tarefa aperiódica, todas as instâncias de todas as tarefas que sofrerão degradação executarão no mesmo nível de degradação no intervalo considerado. Ao término do intervalo, o sistema retorna para o nível zero de degradação.

Nem todas as instâncias das tarefas periódicas podem sofrer degradação. Assumimos que instâncias que já iniciaram suas execuções não são passíveis de serem degradadas. Esta restrição evita tratar problemas de possíveis inconsistências causadas por interrupções da execução de tais instâncias. Assim, representamos o conjunto de tarefas ativas que já iniciaram suas execuções no instante t como sendo $Ex(t)$.

Por definição, o tempo de chegada de cada tarefa aperiódica τ_i^a é desconhecido. Seus custos de execução no pior caso, C_i^a, são conhecidos apenas no instante de sua ativação. Como tarefas aperiódicas são aquelas que representam rotinas de recuperação, por simplicidade, não assumimos que as mesmas possuem versões degradadas. Na prática, tais tarefas contém trechos de código essenciais para manter a consistência do sistema.

Assumimos que o sistema é escalonado de acordo com a política EDF [Liu and Layland 1973], que escolhe, dentre as instâncias das tarefas prontas para executar, aquela que possui o menor *deadline* absoluto. Além disso, assumimos que o conjunto Γ_p é escalonável no nível de degradação $j = 0$, dado que não há ativação de tarefas aperiódicas. Como estaremos preocupados em verificar o efeito da execução de rotinas de recuperação no escalonamento de tarefas periódicas, assumiremos que pode haver sobre-

carga de tarefas aperiódicas em algumas situações. Neste caso, o sistema poderá rejeitar a execução de tais tarefas para conservar a correção temporal das tarefas periódicas. Com isto, nossa intenção é fornecer subsídios de comparação de modelos de escalonamento no que se refere à execução de tarefas aperiódicas.

4. Definição dos Modelos de Escalonamento

4.1. Idéia Básica

Escalonadores dinâmicos conseguem se adaptar muito bem às variações de carga do sistema. No caso do EDF, isto se deve à atribuição de prioridades de acordo com os *deadlines* no instante de ativação das tarefas. Por exemplo, caso o sistema esteja executando alguma instância com *deadline* d e haja a ativação de alguma outra tarefa com *deadline* $d' < d$, a tarefa d' terá maior prioridade. No entanto, justamente por causa desta capacidade de adaptação, o EDF sofre o *efeito dominó* quando o sistema está sobrecarregado [Liu 2000], como ilustra o gráfico contido na Figura 3(b). Por exemplo, se a instância de tarefa com *deadline* d' não consegue cumprir seu *deadline*, pode ser que o tempo restante para executar a outra instância não seja suficiente. Neste caso, ambas perderão os seus *deadlines*. Em geral, para proteger o sistema deste efeito, usa-se um teste de admissão de tal forma que tarefas aperiódicas são rejeitadas caso haja risco de perda de *deadlines*.

Como estamos modelando tarefas aperiódicas como rotinas de recuperação, deveremos também prover o sistema de um teste de admissão. No entanto, ao invés de rejeitar tarefas, estamos interessados em determinar o menor nível de degradação das tarefas periódicas de tal forma que todas as tarefas do sistema cumpram seus *deadlines*. Nesta seção, discutiremos duas maneiras de conseguir este objetivo. Ambas descrevem a estratégia descrita pelo Algoritmo 1, que deve ser executado no momento t em que o escalonador percebe a ativação da tarefa aperiódica.

Algoritmo 1: Busca do nível de degradação apropriado, j.

```
1 j = 0;
2 enquanto j ≤ m ∧ S(t, j) faça  j ← j + 1;
3 se j > m então  j = −1;
```

O algoritmo busca um nível de degradação j que seja apropriado à admissão da tarefa aperiódica. O teste de admissão $S(t, j)$ é usado verificar se o sistema permanece escalonável com o nível de degradação j. Caso $j \leq m$ não seja encontrado, rejeita-se a tarefa aperiódica, o que é indicado fazendo-se $j < 0$. É importante enfatizar que rejeição de tarefas aperiódicas não é desejável. Por isso, usamos a rejeição como uma das métricas de comparação de diferentes estratégias de escalonamento (seção 5). Obviamente, em passos futuros da pesquisa, pretendemos derivar expressões que forneçam um limite a partir do qual tarefas aperiódicas seriam rejeitadas. Tal limite representaria uma medida de resiliência do sistema.

As seções 4.2 e 4.3 descrevem o teste de admissão $S(t, j)$ para dois modelos de escalonamento.

4.2. Modelo de Escalonamento Baseado no TBS

O modelo de escalonamento considerado nesta seção baseia-se no conceito de servidor de tarefas aperiódicas, ou simplesmente servidor. Um servidor é uma tarefa virtual que

visa executar as tarefas aperiódicas (não críticas) controlando a interferência sobre as tarefas periódicas (críticas). Nesta seção, adaptaremos o conceito de servidores para o escalonamento de tarefas aperiódicas críticas.

O servidor possui uma *fila de prontos*, onde as tarefas aperiódicas que estão prontas para executar encontram-se em ordem de prioridade. A prioridade do servidor dependerá do seu *deadline*, calculado dinamicamente em função da tarefa aperiódica no topo da sua fila. Um dos servidores bastantes utilizados, e considerado neste trabalho, é o TBS (*Total Bandwidth Server*) [Spuri and Buttazzo 1996]. O TBS é definido pelo percentual de utilização do processador destinado à execução de tarefas aperiódicas. Durante o projeto do sistema, destina-se um percentual máximo para execução de tarefas aperiódicas. Em geral, se as tarefas periódicas demandam u_p ($0 \leq u_p < 1$) de processador, pode-se reservar $u_s = 1 - u_p$ para executar as tarefas aperiódicas e, desta forma, pode-se obter o uso máximo de processador sem risco de perdas de *deadlines*. Considerando o nível de degradação $j = 0$, u_s é dado por

$$u_s = 1 - \sum_{\tau_i^p \in \Gamma_p} \frac{C_{i,0}^p}{T_i^p} \,. \tag{1}$$

O cálculo da prioridade do TBS, ou seja, do seu *deadline*, é dado da seguinte forma. Suponha que uma tarefa aperiódica τ_i^a é ativada no instante t e seu custo de execução é C_i^a unidades de tempo. Como a execução de τ_i^a não pode ocupar mais que u_s do processador, atribui-se o menor *deadline* d_s ao servidor tal que $C_i^a/(d_s - t) \leq u_s$. Em outras palavras, $d_s = t + C_i^a/u_s$. Como a derivação de d_s é feita para cada tarefa aperiódica, pode-se dizer que há várias instâncias do servidor. Assim, $d_{s,k}$ representa o *deadline* calculado para cada instância k do servidor. Cada instância do servidor é escalonada juntamente com as demais tarefas do sistema pela política EDF.

Considere duas instâncias consecutivas do servidor, com *deadlines* $d_{s,k-1}$ e $d_{s,k}$, respectivamente. Como a execução da instância k pode iniciar antes do instante $d_{s,k-1}$, corre-se o risco de exceder o percentual u_s de processador. Desta forma, pode-se generalizar o cálculo de d_s pela seguinte equação:

$$d_{s,k} = \max(t, d_{s,k-1}) + \frac{C_i^a}{u_s} \,. \tag{2}$$

A equação (2) é usada para determinar $d_{s,k}$ e não considera o *deadline* da tarefa aperiódica d_i^a, o que é necessário no contexto deste trabalho. Para efeito de degradação de tarefas periódicas, é necessário ainda diferenciar aquelas passíveis de serem degradadas das que não podem ser consideradas, pois já iniciaram suas execuções. Para tanto, precisa-se inicialmente determinar o percentual $u_s(t, j)$ disponível para executar a tarefa aperiódica ativada no instante t. Este valor é dado por u_s adicionado do valor conseguido pela degradação das tarefas periódicas. Em outras palavras,

$$u_s(t, j) = u_s + \sum_{\tau_i^p \in A(t, d_i^a) - Ex(t)} \frac{C_{i,0}^p - C_{i,j}^p}{T_i^p} \,. \tag{3}$$

Usando a equação (2) e (3), pode-se determinar o teste de admissão

$$S_{TBS}(t, j) \equiv d_i^a \geq \max(t, d_{s,k-1}) + \frac{C_k^a}{u_s(t, j)} \,, \tag{4}$$

onde d_i^a é o *deadline* da tarefa aperiódica no topo da fila do servidor TBS no instante t.

Como ilustração, considere a Figura 1 que representa a execução de um sistema com duas tarefas periódicas τ_1^p e τ_2^p, com tempos de execução $C_{1,0}^p = 2$, $C_{1,1}^p = 1$, $C_{2,0}^p = 5$ e $C_{2,1}^p = 3$. Os períodos destas tarefas são iguais aos seus *deadlines* relativos, que valem $D_1 = 7$ e $D_2 = 9$, respectivamente. Considere que $r_{1,1}^p = 0$ e $r_{2,1}^p = 0$. No instante 1, uma tarefa aperiódica τ_i^a é ativada no sistema com $C_i^a = 1,8$ e $d_i^a = 6,8$. A Tarefa τ_i^a não pode ser admitida com nível de degradação $j = 0$, pois $u_s = 1 - 0,84 = 0,16$ e $d_{s,1} = 1 + 1,8/0,16 = 11,25$. Assim, $S_{TBS}(1,0)$ é falso. Como no instante $t = 1$ a tarefa τ_1^p está em execução, a única tarefa ativa no intervalo passível de degradação é τ_2^p. Portanto, o valor de $u_s(1,1) = 0,16 + (5-3)/9 \approx 0,38$. Com esta configuração, $d_{s,1} = 1 + 1,8/0,38 \approx 5,74 < d_i^a$ e, portanto, τ_i^a pode ser executada sem risco de perdas de *deadlines*, como ilustra a figura.

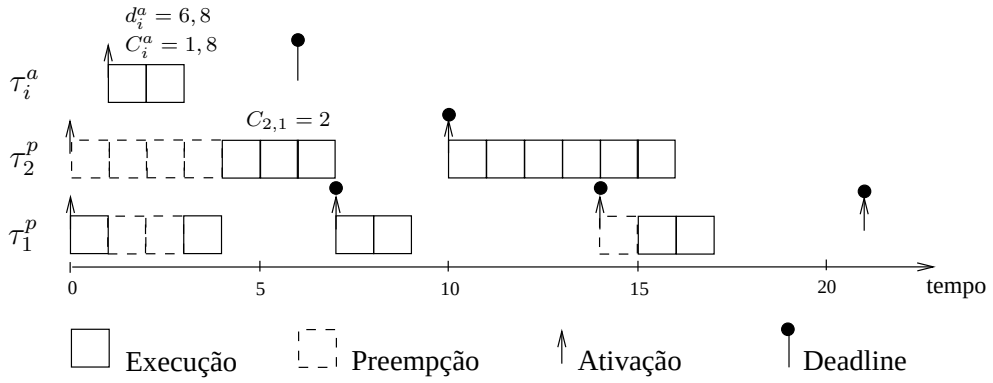


Figura 1. Admissão de τ_i^a assumindo degradação controlada.

Como tarefas aperiódicas não são admitidas no sistema caso haja risco de violação de *deadlines*, fica claro que se o sistema é escalonável antes da ativação da tarefa aperiódica, ele continuará sendo após a admissão da mesma. É importante observar que o escalonamento de tarefas aperiódicas baseado no TBS considera tais tarefas uma por vez. Considerando o exemplo da Figura 1, nenhuma tarefa aperiódica pode ser considerada antes do instante $t = 2,8$. Desta forma, uma tarefa aperiódica urgente pode ter que esperar demasiadamente a execução da tarefa aperiódica admitida anteriormente. A abordagem desenvolvida na próxima sub-seção trata deste problema.

4.3. Modelo Baseado no EDF

O teste de admissão derivado nesta seção baseia-se no cálculo da demanda por processador no intervalo entre a ativação de uma tarefa aperiódica e o seu *deadline*. Se, de acordo com tal demanda, não há risco de violar *deadlines*, a tarefa aperiódica é admitida. O modelo de escalonamento assumido é simplesmente o EDF.

Para compreender melhor a derivação do teste de admissão, suponha uma tarefa aperiódica, τ_i^a, ativada no instante t, com tempo máximo de execução C_i^a e *deadline* d_i^a. O objetivo é determinar se τ_i^a pode ser admitida no instante t sem causar perda de algum *deadline*, seja de tarefas periódicas ativas entre $[t, d_k^a)$, ou de outras tarefas aperiódicas já admitidas anteriormente. Considere, por enquanto, apenas as tarefas periódicas (ver

Figura 2). Inicialmente, é preciso saber quantas instâncias de tarefas periódicas executarão no intervalo $[t, d_k^a)$. Posteriormente, determina-se qual a demanda de processador para executá-las.

Na Figura 2 podem ser observadas as ativações de instâncias de uma tarefa periódica τ_i^p ao longo da linha do tempo. Para determinar quantas destas instâncias executam no intervalo $[t, d_k^a)$, é preciso considerar a última ativação de τ_i^p antes de t se esta ainda não terminou sua execução até o instante t. Definimos então $r_i(t)$ como sendo o instante desta ativação, ou seja, $r_i(t) = (k-1)T_i^p \leq t < kT_i^p$, onde $k > 0$ e $t \geq 0$ é um instante de tempo qualquer.

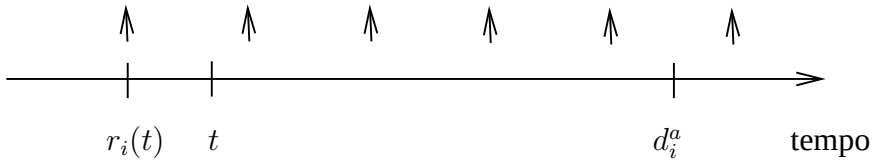


Figura 2. Número de instâncias de τ_i^p consideradas em $[t, d_k^a)$

Assim, o número máximo de instâncias de τ_i^p ativas no intervalo $[t, d_k^a]$ é dado por

$$\left\lceil \frac{d_k^a - r_i(t)}{T_i^p} \right\rceil \tag{5}$$

Determinar a demanda por processador relativa às instâncias de τ_i^p requer a soma do custo computacional de cada uma delas. Todas as instâncias que já iniciaram suas execuções, sejam elas periódicas ou não, contribuirão para esta demanda com o tempo necessário para que elas terminem. Assim, definimos $c_i(t) \geq 0$ o tempo restante para completar a execução da instância ativa de τ_i em $Ex(t)$. Todas as instâncias de tarefas periódicas ativas no intervalo $[t, d_i^a)$ que ainda não iniciaram suas execuções demandarão $C_{i,j}^p$, onde j é o nível de degradação desejado. Em resumo, a demanda por processador das tarefas periódicas no intervalo $[t, d_i^a)$ é dada por

$$\sum_{\tau_i \in Ex(t)} c_i(t) + \sum_{\tau_i^p \in A(t, d_i^a) - Ex(t)} \left\lceil \frac{d_i^a - r_i(t)}{T_i^p} \right\rceil C_{i,j}^p \,. \tag{6}$$

Considerando que o sistema é escalonável antes de admitir a tarefa τ_i^a, o sistema continuará sendo escalonável se todas as tarefas consideradas na equação (6), incluindo τ_i^a, finalizarem suas execuções no intervalo $[t, d_i^a)$. De fato, de acordo com a política EDF, as tarefas que sofrerão influência da execução de τ_i^a são aquelas que têm *deadlines* maiores ou iguais a d_i^a. Assim, admitir o intervalo $[t, d_k^a)$ para executar todas as instâncias de tarefas ativas, assegura que nenhuma tarefa perderá o *deadline* após a admissão de τ_i^a.

Resumindo, para cada ativação de uma tarefa aperiódica τ_i^a, a relação (7) assegura a escalonabilidade do sistema após sua admissão:

$$S_{EDF}(t, j) \equiv \sum_{\tau_i \in Ex(t)} c_i(t) + \sum_{\tau_i^p \in A(t, d_i^a) - Ex(t)} \left\lceil \frac{d_i^a - r_i(t)}{T_i^p} \right\rceil C_{i,j}^p \leq d_i^a - t \,. \tag{7}$$

A relação (7) pode ser usada no algoritmo 1. Com relação à implementação deste algoritmo, é importante observar que, conhecendo-se $S_{EDF}(0)$), é desnecessário efetuar o cálculo para todos os níveis de degradação $j = 1, 2, \ldots, m$. A demanda por processador pode ser obtida subtraindo o valor $C^p_{i,j-1} - C^p_{i,j}$ da demanda por processador encontrado para o nível $j - 1$, para cada tarefa a ser degradada τ^p_i. Desta forma, o custo de implementação é consideravelmente reduzido.

Como ilustração, considere o mesmo exemplo da Figura 1. No instante $t = 1$ a tarefa τ^a_i não pode ser admitida com nível de degradação $j = 0$, pois a demanda total por processador no intervalo $[t, d^a_1)$ é de 8 unidades de tempo, o que significa que $S_{EDF}(1, 0)$ é falso. Fazendo $j = 1$, reduz-se a demanda total por processador para 6 unidades de tempo tornando verdadeiro o teste $S_{EDF}(1, 1)$. O escalonamento produzido é o mesmo apresentado na figura.

5. Simulação e Resultados

Para avaliar as abordagens descritas, desenvolvemos um simulador de escalonamento de tempo real, em Java, e simulamos extensivamente a execução de conjuntos de tarefas. O modelo de simulação foi construído da seguinte forma. Utilizamos nove conjuntos de tarefas periódicas, cada qual composto por oito tarefas. Os valores de $C_{i,0}$ foram gerados de acordo com uma distribuição exponencial com média $u_p/10$, onde $u_p = 10\%, 20\%, \ldots, 90\%$ é o percentual de utilização do conjunto de tarefas periódicas considerado. Para efeitos de simulação, consideramos que $C_{i,j+1} = 90\%C_{i,j}$ ($j = 0, 1, \ldots, 9$). Os períodos foram gerados de acordo com uma distribuição uniforme no intervalo de 80 a 500 unidades de tempo e os *deadlines* foram considerados iguais aos períodos.

O tempo de simulação considerado foi de 100.000 unidades de tempo, o que permitiu observar o comportamento das tarefas periódicas no longo prazo. Nos cenários de teste considerados, por exemplo, a tarefa com período mais longo é ativada 200 vezes durante a simulação.

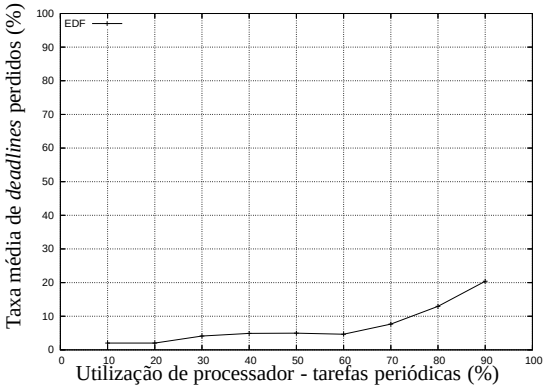
A ocorrência de falhas seguiu o seguinte procedimento. Durante a simulação, foram gerados eventos aleatórios de acordo com uma distribuição de *Poisson* com parâmetro $\lambda = 1000$. Tais eventos representam uma janela de instabilidade no sistema, cujo tamanho foi definido como sendo de 100 unidades de tempo. Dentro desta janela, a ocorrência de erros foi simulada. A ocorrência de cada erro seguiu uma distribuição de *Bernoulli* com parâmetro $p = 0,05$. Assim, procuramos modelar a ocorrência de erros em rajadas (*burst*). Para cada erro gerado, uma tarefa aperiódica, representando uma rotina de recuperação, foi ativada. Os custos e *deadlines* de tais tarefas foram gerados de tal forma que uma tarefa aperiódica τ^a_i, gerada no instante t, requer, em média, $u_a = 40\%$ de processador no intervalo de $[t, d^a_i)$. Os valores de C^a_i foram gerados de acordo com uma distribuição exponencial com parâmetro $1/40$. Seus *deadlines* foram gerados de acordo com uma distribuição normal com média $2,5\, C^a_i$ e variância $0,01\, C^a_i$. Desta forma, estamos considerando um sistema onde os erros ocorrem em janelas de 100 unidades de tempo, em forma de rajadas, gerando rotinas de recuperação com distribuição normal e utilização em torno de 40%.

A simulação da execução de todas as tarefas considerou quatro alternativas de escalonamento: (a) EDF; (b) EDF-SD; (c) EDF-CD; (d) TBS-CD. A primeira alternativa corresponde simplesmente ao escalonador EDF, sem utilizar rejeição de tarefas

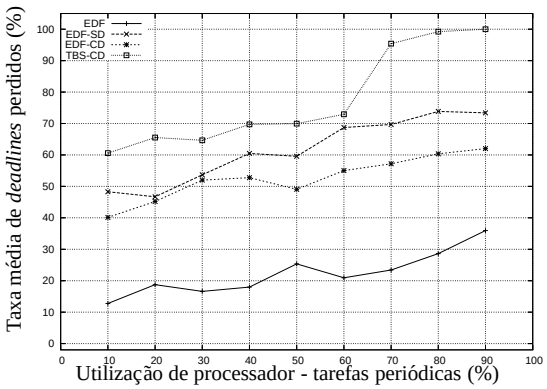
aperiódicas nem tentar degradar tarefas periódicas. A abordagem EDF-SD usa o teste de admissão proposto pela equação (7) para rejeitar tarefas aperiódicas que possam causar violação de *deadlines*. Já a abordagem EDF-CD usa o mesmo teste e considera degradação de tarefas aperiódicas. Finalmente, TBS-CD corresponde ao modelo que usa o TBS, com teste de admissão representado pela equação (4) considerando degradação.

5.1. Nível de Interferência das Tarefas Aperiódicas

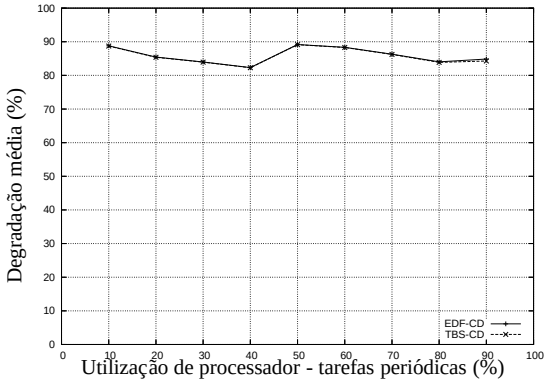
Observamos o nível de interferência que a execução de tarefas aperiódicas exerce na execução de tarefas periódicas. Obviamente, esta interferência é nula para as abordagens de escalonamento que realizam testes de admissão para tarefas aperiódicas (EDF-SD, EDF-CD e TBS-CD). Assim, esta seção apresenta os resultados desta interferência considerando apenas o escalonador EDF, ilustrado no gráfico da Figura 3(a), medida através da taxa de *deadlines* perdidos das tarefas periódicas. Como pode ser observado, o EDF possui um desempenho ruim, como esperado. A partir de 60% de carga das tarefas periódicas a taxa de *deadlines* perdidos cresce muito rapidamente, o que ilustra o efeito dominó mencionado na seção 4.1.



(a) Percentual de *deadlines* perdidos de tarefas periódicas



(b) Percentual de rejeição de tarefas aperiódicas



(c) Degradação média do sistema

Figura 3. Comportamento do sistema para $u_a = 40\%$

5.2. Eficiência das Técnicas de Degradação

A taxa de rejeição das tarefas aperiódicas para as três abordagens que usam testes de admissão foi medida durante a simulação. Este é um parâmetro importante de comparação, visto que o mesmo indica a eficiência do modelo de escalonamento no que refere à admissão de tarefas aperiódicas e o respectivo efeito do algoritmo que provê degradação controlada.

O gráfico da Figura 3(b) apresenta os resultados encontrados. Para efeito de comparação, é mostrado no gráfico o desempenho do EDF com e sem o teste de admissão. Para o EDF sem teste de admissão (EDF), os resultados devem ser interpretados como sendo taxa de *deadlines* perdidos de tarefas aperiódicas, visto que todas as tarefas aperiódicas são admitidas no sistema. Apesar de o EDF apresentar resultados médios melhores com relação a taxa de rejeição de tarefas aperiódicas, deve-se considerar que tarefas periódicas estão também perdendo *deadlines* (Figura 3(a)), o que faz o uso do EDF uma abordagem inapropriada. Se equipamos o EDF com teste de admissão, pode-se notar que a taxa de rejeição de tarefas aperiódicas continua alta. Neste contexto, a implementação de métodos para prover degradação controlada é extremamente útil, como pode ser observado com a curva relativa a EDF-CD. É interessante notar que o TBS-CD mostra-se muito inferior às demais abordagens, mesmo fazendo uso de degradação controlada. Isto se deve ao fato de que tarefas aperiódicas são admitidas seqüencialmente no sistema.

5.3. Nível de Degradação Média

Aferimos o nível de degradação média do sistema, ilustrado na Figura 3(c). O nível de degradação foi medido como sendo o percentual médio de redução de C_i^p, para cada tarefa periódica τ_i^p, necessário para se admitir as tarefas aperiódicas. Como pode ser observado, a degradação do sistema se mantém entre 80% e 90%. Obviamente, este resultado não pode ser analisado separadamente dos resultados da rejeição de tarefas aperiódicas. De fato, as duas abordagens que contemplam degradação controlada são equivalentes quanto ao nível de degradação do sistema. No entanto, o TBS-CD rejeita um número maior de tarefas aperiódicas, como mencionado na seção 5.2.

Outro aspecto importante que deve ser mencionado é a simplicidade do algoritmo de degradação. De fato, na realidade, o espaço de busca por um nível adequado de degradação é exponencial. Tal espaço foi substancialmente reduzido pelas políticas propostas na seção 4 para que as mesmas possam ser viáveis em tempo de execução. Melhorando o algoritmo, certamente, o nível de degradação das tarefas será reduzido, pois, em alguns casos, basta degradar uma tarefa para atingirmos a configuração necessária para a execução da tarefa aperiódica, não sendo necessário degradar todas as tarefas de uma só vez. De qualquer forma, os resultados indicam que o benefício em prover mecanismos de degradação controlada pode compensar os seus custos.

6. Conclusões e Trabalhos Futuros

Este trabalho descreveu e avaliou propostas de escalonamento no contexto de tolerância a falhas. Assumimos um modelo de tarefas que contempla tarefas periódicas, que devem ser executadas regularmente pelo sistema, e tarefas aperiódicas, que são escalonadas em resposta à detecção de erros. Caso tais tarefas sejam ativadas no sistema, o escalonador adapta o sistema, degradando controladamente as tarefas periódicas, a fim de cumprir os

requisitos temporais das rotinas de recuperação. Uma das abordagens propostas baseou-se no servidor TBS, freqüentemente usado para escalonar tarefas aperiódicas não críticas. Mostramos, através de simulação, que esta proposta não é adequada para tratar rotinas de recuperação. Ilustramos ainda que, equipando o EDF com um teste de admissão, é possível melhorar bastante a eficiência do escalonador, protegendo o sistema de perdas de *deadlines* de tarefas periódicas.

Como o principal objetivo aqui foi verificar os efeitos que rotinas de recuperação podem exercer no desempenho do escalonamento, o modelo de degradação de tarefas apresentado foi bastante simplificado. Passos futuros deste trabalho deverão incluir técnicas de otimização para determinar qual das configurações de degradação é mais apropriada para ser escolhida. Melhorias no teste de admissão devem também ser consideradas. Outro desdobramento possível é o desenvolvimento de uma análise de escalonamento para determinar a capacidade do sistema em tolerar falhas de tal forma que nenhum *deadline* de tarefas, periódicas ou aperiódicas, seja violado. Em outras palavras, tal análise deve ser capaz de determinar a taxa máxima/média de erros suportada pelo sistema, por exemplo. Dada a relevância do tema abordado neste trabalho, os resultados apresentados aqui subsidiarão tais desdobramentos em futuros passos de pesquisa.

Referências

Aydin, H. (2004). "On Fault-Sensitive Feasibility Analysis of Real-Time Task Sets". In *Proc. of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, pages 426–434. IEEE Computer Society.

Burns, A., Davis, R. I., and Punnekkat, S. (1996). "Feasibility Analysis of Fault-Tolerant Real-Time Task Sets". In *Proc. of the Euromicro Real-Time Systems Workshop*, pages 29–33. IEEE Computer Society Press.

Burns, A. and Wellings, A. J. (2001). *"Real-Time Systems and Programming Languages"*. Addison-Wesley, 3rd edition.

Buttazzo, G. C. and Stankovic, J. (1993). "Red: A Robust Earliest Deadline Scheduling Algorithm". In *Proc. of 3rd International Workshop on Responsive Computing Systems*.

Caccamo, M. and Buttazzo, G. (1998). "Optimal Scheduling for Fault-Tolerant and Firm Real-Time Systems". In *Proc. of the 5th Conference on Real-Time Computing and Applications (RTCSA)*, pages 223–231.

de Jesus, E. O. and Lima, G. (2005). "Escalonamento para Sistemas de Tempo-Real Tolerantes a Falhas: Um Estudo Empírico". In *Proc. of the Brazilian Real-Time Workshop (WTR 05)*, pages 69–72. Brazilian Computer Society.

de Oliveira, R. S. (1997). "Computação Imprecisa". *Revista de Informática Teórica e Aplicada - RITA*, 4(1).

Ghosh, S., Melhem, R. G., and Mossé, D. (1995). "Enhancing Real-Time Schedules to Tolerate Transient Faults". In *Proc. of the 16th Real-Time Systems Symposium (RTSS)*, pages 120–129. IEEE Computer Society Press.

Ghosh, S., Melhem, R. G., Mossé, D., and Sarma, J. S. (1998). "Fault-Tolerant Rate-Monotonic Scheduling". *Real-Time Systems*, 15(2):149–181.

González, O., Shrikumar, H., Stankovic, J. A., and Ramamritham, K. (1997). "Adaptive Fault Tolerance and Graceful Degradation under Dynamic Hard Real-Time Scheduling". In *Proc. of the 18th Real-Time Systems Symposium (RTSS)*. IEEE Computer Society Press.

Kandasamy, N., Hayes, J. P., and Murray, B. T. (1999). "Tolerating Transient Faults in Statically Scheduled Safety-Critical Embedded Systems". In *Proc. of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 212–221.

Kopetz, H. (1997). *"Real-Time Systems Design for Distributed Embedded Applications"*. Kluwer Academic Publishers.

Liberato, F., Melhem, R. G., and Mossé, D. (2000). "Tolerance to Multiple Transient Faults for Aperiodic Tasks in Hard Real-Time Systems". *IEEE Transactions on Computers*, 49(9):906–914.

Liestman, L. and Campbell, R. H. (1986). "A Fault-Tolerant Scheduling Problem". *IEEE Transaction on Software Engineering*, 12(11):1089–1095.

Lima, G. and Burns, A. (2005). "Scheduling Fixed-Priority Hard Real-Time Tasks in the Presence of Faults". In *Dependable Computing: Second Latin-American Symposium, LADC*, volume LNCS 3747, pages 154–173. Springer-Verlag.

Lima, G. M. A. and Burns, A. (2003). "An Optimal Fixed-Priority Assignment Algorithm for Supporting Fault Tolerant Hard Real-Time Systems". *IEEE Transaction on Computers*, 52(10):1332–1346.

Liu, C. L. and Layland, J. W. (1973). "Scheduling Algorithms for Multiprogram in a Hard Real-Time Environment". *Journal of ACM*, 20(1):40–61.

Liu, J. W. S. (2000). *"Real-Time Systems"*. Prentice-Hall.

Marucheck, M. and Strosnider, J. (1995). "An Evaluation of the Graceful Degradation Properties of real-time Schedulers". In *Proc. of the 25th Annual International Symposium on Fault-Tolerant Computing*.

Poledna, S. (1996). *"Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism"*. Kluwer Academic Publishers.

Ramos-Thuel, S. and Strosnider, J. K. (1991). "The Transient Server Approach to Scheduling Time-Critical Recovery Operations". In *Proc. of the 12th Real-Time Systems Symposium (RTSS)*, pages 286–295. IEEE Computer Society Press.

Spuri, M. and Buttazzo, G. (1996). "Scheduling Aperiodic Tasks in Dynamic Priority Systems". *Real Time Systems*, 10(2):179–210.