MECANISMOS PARA PREVISÃO DE PERDA DE DEADLINE PARA THREADS DISTRIBUÍDAS TIPO PIPELINE

PATRICIA DELLA MEA PLENTZ, CARLOS MONTEZ, RÔMULO SILVA DE OLIVEIRA

Programa de Pós-Graduação em Eng. Elétrica – Universidade Federal de Santa Catarina (UFSC)

88.040-900 – Florianópolis – SC – Brasil -- {plentz, montez, romulo}@das.ufsc.br

Resumo. Threads distribuídas tempo real são entidades escalonáveis com um deadline fim a fim, as quais transpõem nodos, carregando seu contexto de escalonamento. Em cada nodo, a thread será localmente escalonada segundo um deadline local gerado por um método de particionamento de deadline. A previsão de perda de deadlines fim a fim pode ser realizada através da definição de deadlines locais estimados. Este artigo propõe e analisa alguns mecanismos de previsão de perda de deadlines fim a fim. Previsões corretas sobre a perda de deadlines possibilitam a realização de ações para melhorar o desempenho do sistema. Simulações mostram que o mecanismo proposto apresenta bons resultados na previsão de perda de deadlines em sistemas sobrecarregados.

Palavras-chave: mecanismo de previsão de perda de deadlines fim a fim, threads distribuídas, sistema tempo real.

1. Introdução

Nos últimos anos a comunidade acadêmica tem pesquisado vários aspectos do escalonamento fim a fim [10][9]. O correto funcionamento de sistemas distribuídos de tempo real que possuem tarefas com restrições temporais depende de uma política de escalonamento adequada [7][1].

Uma Thread Distribuída (TD) é uma entidade escalonável que pode transpor nodos, levando seu contexto de escalonamento (restrições temporais) entre as entidades responsáveis pelo escalonamento naqueles nodos [1]. Este conceito pode ser usado como uma abstração central para sistemas tempo real distribuídos em aplicações de controle e supervisão, por exemplo. Tarefas de controle são executadas localmente e possuem deadlines *hard.* Já tarefas de supervisão têm deadlines *firm*[1] e são distribuídas, isto é, visitam vários nodos para coletar informações para a realização de análises e diagnósticos. Este último tipo de tarefa pode ser criada em instantes pré-programados, ou quando algum alarme de supervisão é ativado, e podem ser modeladas como threads distribuídas. Em [3] e [4] são descritos outros exemplos desta abstração.

De maneira geral, o deadline fim a fim de uma tarefa distribuída é definido como parte dos requerimentos do sistema. Torna-se necessário, portanto, definir uma forma para particionar a restrição temporal entre as tarefas locais que representam esta tarefa distribuída.

Neste trabalho definimos uma arquitetura de sistema que suporta a abstração TD com restrições temporais. Este tipo de thread representa tarefas distribuídas aperiódicas de tempo real com deadlines *firm*. Pretendemos, com esta arquitetura, cumprir o maior número possível de deadlines *firm* das tarefas distribuídas sem comprometer o escalonamento das tarefas locais com deadlines *hard*.

O objetivo deste artigo é discutir diferentes estratégias para estimar a probabilidade de deadlines fim a fim serem alcançados. Assumimos que threads distribuídas são aperiódicas e possuem deadlines fim a fim do tipo *firm*. A previsão de perda de deadlines pode ser feita pela geração de deadlines locais estimados, os quais são definidos a partir de métodos de particionamento de deadlines fim a fim. Bons mecanismos de previsão de perda de deadlines permitem que ações corretivas sejam realizadas a tempo de melhorar o desempenho do sistema.

Este artigo está organizado da seguinte forma: as seções 2 e 3 descrevem trabalhos relacionados e os principais conceitos relacionados com TDs. Na seção 4 são apresentados métodos de particionamento de deadlines fim a fim encontrados na literatura. O modelo proposto e os mecanismos para previsão de perdas de deadlines são descritos nas seções 5 e 6, respectivamente. Resultados da simulação mostram, na seção 7, a aplicabilidade e o desempenho dos mecanismos de previsão de perdas de deadlines propostos. A seção 8 traz as conclusões e perspectivas futuras.

2. Trabalhos Relacionados

Implementações de threads tempo real distribuídas são descritas por vários autores [4, 10, 9, 11, 8]. Em [4] os autores propõem um framework de escalonamento fim a fim no nível de aplicação. Este framework é baseado no middleware CORBA 2.0 e suporta a noção de TD como abstração de

[1] Em sistemas de tempo real, o deadline é o instante máximo desejado para a conclusão de uma tarefa. A perda de um deadline *hard* pode ter conseqüências catastróficas. Já a perda de um deadline *firm* não tem conseqüências catastróficas, porém não existe valor em concluir a execução da tarefa com atraso.

programação para sistemas tempo real distribuídos. O objetivo do framework *Distributed Scheduling Service* (DSS) [10] é obter escalonabilidade fim a fim coerente e gerência de sobrecarga usando a potencialidade dos sistemas locais. Este framework também considera threads distribuídas CORBA 2.0 e utiliza o método de particionamento de deadlines *Effective Deadline* [3]. Nestes trabalhos supõem-se que todas as subtarefas que compõem uma TD são conhecidas, com seus respectivos tempos de computação e nodos onde serão executadas. Em [9] o deadline fim a fim é o principal requerimento do sistema e o autor sugere um framework de escalonamento fim a fim integrado. Em [11] é apresentada uma comparação de desempenho entre TDs e canais de eventos. O protocolo de sincronização *Release Guard* [9] força cada subtarefa a ser periódica, permitindo uma análise online com baixo *overhead*. O trabalho também propõe a integração deste protocolo com TDs para melhorar a escalonabilidade delas. Em [8], propõe-se uma arquitetura de sistema que suporta escalonamento fim a fim de TDs tempo real. A *Real-Time Specification for Java* (RTSJ) é usada na implementação de TDs tempo real, tal que a arquitetura proposta é flexível o suficiente para acomodar diferentes algoritmos de escalonamento.

Muitos trabalhos têm o foco voltado para o problema de particionamento de deadlines fim a fim [3, 7, 2, 6]. Em [3] os autores propõem algoritmos para derivar deadlines de subtarefas a partir do deadline fim a fim de uma tarefa distribuída. Algumas estratégias de particionamento de deadlines propostas naquele trabalho são : *Ultimate* (UD), *Effective Deadline* (ED), *Equal Slack* (EQS) e *Equal Flexibility* (EQF). A taxa de perda de deadlines é comparada entre as estratégias propostas. Os autores observaram que a estratégia EQF melhora significativamente o desempenho das tarefas distribuídas, enquanto tarefas locais têm boas chances de cumprir seus deadlines. A técnica *Slicing* é proposta em [7], onde janelas de tempo são atribuídas para tarefas usando o conceito de caminho crítico. O caminho crítico, conforme definido pelos autores, é aquele que minimiza a folga total do grafo de tarefas. O trabalho propõe o uso de duas métricas: *Fair Assignment* e *Assignment Proportional to the Workload*. A primeira é baseada no número de tarefas que fazem parte do caminho crítico; a segunda atribui deadlines baseada no tempo de execução das tarefas. A técnica *Adaptive Slicing Technique* (AST), proposta em [2], é um aprimoramento da técnica *Slicing*. Ela incorpora novas métricas que são capazes de se adaptarem às mudanças de carga e de tamanho do sistema. Estas métricas consideram o nível de paralelismo do grafo de tarefas. Um conjunto de experimentos utilizando grafos de tarefas demonstrou que a técnica AST apresenta desempenho superior à técnica *Slicing*. Em [6], os algoritmos propostos testam a aceitabilidade de um novo fluxo. Se ele for aceito, os algoritmos devem distribuir o deadline fim a fim deste fluxo nos nodos a serem visitados por ele. Os autores sugerem dois novos métodos de distribuição da folga, *Fair Laxity Distribution* (FLD) e *Unfair Laxity Distribution* (ULD). O desempenho destes métodos propostos é comparado com dois métodos clássicos propostos em [7]. Os resultados obtidos são comparados em função do número de fluxos aceitos no sistema. O melhor desempenho foi obtido pelo algoritmo ULD quando escalonamento *Earliest Deadline First* (EDF) [5] não-preemptivo é utilizado.

3. Threads Distribuídas

Uma Thread Distribuída (TD) é uma entidade abstrata que pode atravessar nodos físicos carregando seus parâmetros de escalonamento. Usualmente, uma TD é implementada a partir de um seqüênciamento de execuções de threads locais (Figura 1) [1]. Desta forma, cada TD atua como uma thread local em um nodo particular do sistema distribuído no qual ela executa.

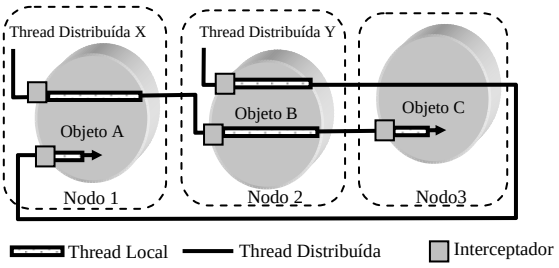

Figure 1. Threads Distribuídas e segmentos locais.

Assim, é possível visualizar uma TD como sendo composta por segmentos locais de execução. Em cada nodo em que uma TD executa, um segmento local desta TD é criado e é implementado por uma thread local. Embora ela execute em vários nodos, cada instância de uma TD é uma entidade única, com um identificador único válido em todo o sistema.

O nodo no qual uma TD é criada é conhecido como *nodo origem*. *Nodos segmentos* são todos os nodos que hospedam parte da execução de uma TD. Uma thread distribuída, em qualquer instante no tempo, deverá estar elegível para execução (ou suspensa) em um único nodo no sistema distribuído. Esse nodo segmento recebe o nome especial de *nodo cabeça*. Sempre que uma TD chega em um nodo significa que ela irá executar um método neste nodo. Quando uma TD faz uma invocação remota, significa que ela irá partir deste nodo para outro.

A TD carrega parâmetros e outros atributos da tarefa de computação quando ela atravessa um nodo. Uma TD executa operações dentro de objetos distribuídos cujos atributos podem ser modificados e acumulados, em uma forma aninhada [1]. Políticas de escalonamento local são usadas para escalonar uma TD quando ela inicia sua execução em um nodo do sistema. Com isto, a coerência deve ser mantida pelo modelo de escalonamento fim a fim entre todas as políticas de escalonamento em cada nodo

segmento da TD. TDs tempo real podem compartilhar recursos físicos (processador, disco, E/S, por exemplo) e recursos lógicos (locks, por exemplo). Estes recursos podem estar sujeitos a restrições de exclusão mútua [4]. Um programa consiste de múltiplas TDs executando concorrentemente e assincronamente.

A execução de uma TD pode terminar no mesmo nodo onde ela iniciou, ou em outro nodo do sistema. Uma TD em execução pode criar uma nova thread tempo real distribuída (através de invocações *one-way*, por exemplo). Esta nova thread tempo real distribuída pode iniciar sua execução no mesmo nodo onde foi criada ou em outro nodo do sistema. Este segundo caso de invocação, porém, não é considerado neste trabalho.

4. Particionamento do Deadline Fim a Fim

Usualmente, o deadline fim a fim de uma TD é definido como parte dos requerimentos do sistema. Esta restrição temporal e outras informações, como a seqüência de nodos que a TD percorreu em uma dada ativação, podem ser armazenados em um histórico. Este histórico é carregado com a TD, conforme ela transpõe os nodos do sistema, e é atualizado a cada nova ativação.

Diferentes métodos de particionamento de deadlines fim a fim são apresentados na literatura [3, 9] e podem ser utilizados para particionar deadlines de TDs. Neste trabalho, consideramos apenas os métodos de particionamento de deadlines que não utilizam informação sobre a carga do sistema. Embora o uso de informação sobre a carga do sistema melhore o particionamento do deadline, ela também exige uma infra-estrutura mais sofisticada e *overhead* adicional. *Ultimate Deadline* (UD) é um método bastante simples o qual atribui o deadline fim a fim para cada segmento local da TD. Considerando deadline local (*dl*), deadline fim a fim (*d*) e segmento local (*si*), UD é definido como:

$$dl(si) = d(TD)$$

O método *Effective Deadline* (ED) [3] é um melhoramento em relação ao método anterior. Os tempos de computação estimados dos segmentos locais são usados na definição dos deadlines locais. Considerando tempo de execução previsto (*pex*) e número de segmentos locais da TD (*m*), ED é definido como:

$$\mathrm{dl(si)} = d(TD) - \sum_{j=i+1}^{m} pex(sj)$$

Equal Flexibility (EQF) [3] é um outro método que propõe que a folga restante total da TD seja dividida entre os segmentos locais que a compõem na proporção de seus tempos de execução estimados. Considerando tempo de chegada (*ar*), EQF é definido como:

$$\mathrm{dl(si)} = ar(si) + pex(si) + \left(d(TD) - ar(si) - \sum_{j=i}^{m} pex(sj) \right) * \left(pex(si) \Big/ \sum_{j=i}^{m} pex(sj) \right)$$

Com o método EQF, cada segmento local recebe folga suficiente para executar porque a definição de folga considera o tempo de execução estimado proporcionalmente deste segmento em relação aos outros. De acordo com o trabalho descrito em [3], TDs com deadlines fim a fim particionados pelo algoritmo EQF têm maiores chances de cumprir seus deadlines quando comparados com outros métodos. Isto ocorre porque EQF divide a folga da TD proporcionalmente ao tempo de execução de cada segmento local. Os métodos acima referenciados consideram escalonamento não-preemptivo de tarefas distribuídas do tipo pipeline.

5. Modelo Proposto

Neste trabalho consideramos um sistema tempo real distribuído como uma unidade automatizada de uma fábrica. Em um dado momento, somente uma aplicação executa em todos os nodos do sistema. TDs tempo real são usadas na implementação desta aplicação. As conseqüências de uma falha temporal de uma TD são da mesma ordem de magnitude que os benefícios do sistema quando em operação normal.

Em cada nodo do sistema, existem tarefas periódicas locais com deadlines *hard* e tarefas distribuídas aperiódicas com deadlines *firm*. As tarefas periódicas locais são consideradas críticas para a aplicação. Com isso, o escalonamento destas tarefas não deve ser prejudicado pelo escalonamento das tarefas aperiódicas distribuídas. Estas tarefas aperiódicas são recorrentes, isto é, elas podem executar várias vezes. Pelo fato delas poderem ser criadas em tempo de execução, este tipo de sistema tem carga dinâmica e podem existir situações de sobrecarga.

Supõe-se que sempre que uma TD tempo real é criada, um deadline fim a fim e um tempo de execução são definidos. Estas restrições temporais são carregadas por cada TD tempo real conforme elas transpõem os nodos do sistema. Existe um escalonador em cada nodo do sistema. Eles não colaboram uns com os outros de forma explícita e são considerados independentes. O escalonamento é apenas influenciado pelos atributos temporais associados com cada TD.

Servidores de tarefas aperiódicas são usados em cada nodo do sistema distribuído de tempo real. A figura 2 mostra a arquitetura presente em cada nodo do sistema. Os interceptadores são responsáveis pelo atendimento das TDs tempo real aperiódicas. Eles são threads locais com tempos de execução previamente definidos, isto é, eles são considerados como mais uma tarefa periódica local do sistema. A tarefa de coordenar a execução dos segmentos locais das TDs fica a cargo dos servidores de aperiódicas.

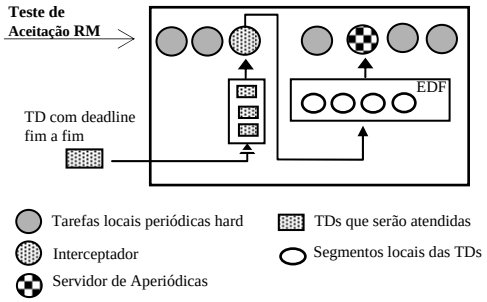

Figura 2 – Arquitetura de sistema para escalonamento de TDs.

Em cada nodo deste sistema será utilizado o algoritmo *Rate Monotonic* (RM) [5] que irá escalonar de forma preemptiva as tarefas locais periódicas com deadlines *hard*, assim como o interceptador e o servidor de aperiódicas. Sempre que uma TD tempo real chega em um nodo do sistema (nodo cabeça), ela é atendida pelo interceptador, o qual mantém uma lista de segmentos locais de TDs tempo real que executam (ou executaram) em um nodo. Para cada thread tempo real distribuída que chega em um determinado nodo, o interceptador verifica se um segmento local desta TD já existe. Em caso a*firm*ativo, este segmento local é ativado para executar em nome da TD aperiódica que chegou. Senão, o interceptador cria um segmento local com a função de executar em nome desta nova TD.

Nestas duas situações, todas as propriedades da thread tempo real distribuída aperiódica (tais como identificador e restrições temporais) são herdados pelos seus segmentos locais. Estas propriedades são armazenadas em um histórico mantido por cada segmento local da DT. As últimas ativações da TD são também armazenadas neste histórico.

O interceptador coloca este segmento local na fila do servidor de aperiódicas daquele nodo (figura 2). O algoritmo *Earliest Deadline First* (EDF) [5] é usado para escalonar, de forma preemptiva, a fila do servidor de aperiódicas.

Existem vários métodos de escalonamento para tarefas distribuídas presentes na literatura [1][4]. O método de escalonamento usado neste trabalho é composto por dois estágios: particionamento do deadline fim a fim e escalonamento local. Este não é um procedimento incomum na literatura de tempo real [9]. O objetivo de escalonamento da arquitetura proposta é garantir os deadlines das tarefas locais *hard*. Ao mesmo tempo, ela deve reduzir o tempo de resposta das tarefas aperiódicas para atender os deadlines dos segmentos locais das TDs e, consequentemente, cumprir os deadlines fim a fim das TDs.

6. Mecanismos Propostos para Previsão de Perda de Deadlines Fim a Fim

Para cada chegada de uma TD no sistema, um mecanismo de previsão de perda de deadlines pode ser usado para determinar a probabilidade do deadline fim a fim da TD ser cumprido. Este tipo de previsão é importante em sistemas distribuídos de tempo real porque permite a realização de ações corretivas para recuperar tarefas distribuídas, tal que elas possam cumprir seus deadlines. Efetuar ajustes nos deadlines das tarefas é um exemplo de ação corretiva que pode ser realizada.

Neste trabalho, a previsão de perda de deadlines é realizada a partir da definição de deadlines locais estimados, denominados *milestones*. Ao final da execução da TD, avalia-se a acurácia da previsão. Alguns métodos de particionamento de deadlines conhecidos na literatura foram utilizados como mecanismos de previsão de perda de deadlines. Os métodos utilizados foram: *Ultimate Deadline* (UD), *Effective Deadline* (ED) e *Equal Flexibility* (EQF) [3]. Através da observação do comportamento destes mecanismos de previsão, pretendemos verificar se algum deles apresenta resultados confiáveis para que ações corretivas possam ser realizadas no sentido de melhorar o desempenho do sistema.

Os métodos de particionamento de deadlines são utilizados neste trabalho para fins de escalonamento local e para fins de previsão de perda de deadlines. O método EQF é utilizado para fins de escalonamento local. Assim, o deadline fim a fim de uma TD é particionado segundo este método, gerando deadlines locais. Em seguida, os métodos UD, ED e EQF são utilizados para fins de previsão de perda de deadlines, através da geração de *milestones*.

Para comparar a qualidade das previsões realizadas pelos mecanismos estudados, torna-se necessário a definição de uma métrica. Seja Ek(z) o erro associado com a previsão de perda de deadlines realizada pelo mecanismo z para a TD k, e é definido como:

$Ek(z) = 1 - Pk(z)$ *se* $Rk \leq Dk$;
$Ek(z) = Pk(z)$ *se* $Rk > Dk$, onde z

representa o mecanismo de previsão usado.

O valor de Ek é necessariamente entre 0 e 1. A variável P é definida da seguinte forma:

Slack = (Mk – Rk) / Mk;
Pk(z) = slack + 0.5;
If Pk(z) > 1 then Pk(z) = 1;
If Pk(z) < 0 then Pk(z) = 0;

A variável *Slack* representa o quão distante foi a previsão local (*Mk*) em relação ao tempo de resposta local (*Rk*). Os exemplos a seguir ilustram o comportamento da métrica usada neste trabalho:

- O algoritmo z estima que a TD irá perder seu deadline, o que de fato ocorre; então Ek(z) = Pk(z) = 0;

- O algoritmo z estima que existe uma chance de 80% da TD cumprir seu deadline, Pk(z) = 0.8, o que de fato ocorre; então Ek(z) = 1 – 0.8 = 0.2;

- O algoritmo z estima que existe uma chance de 50% da TD cumprir seu deadline, Pk(z) = 0.5; neste caso Ek(z) = 0.5 independente de a TD cumprir (1 – 0.5) ou não (0.5) seu deadline.

A taxa de erro de um dado mecanismo é definida como: $E(z) = \sum all\ k\ \ Ek(z) / nk$. É importante

observar que esta métrica considera a convicção de um algoritmo sobre cumprir ou não um deadline. Por exemplo, supondo que para a tarefa Tk temos a previsão de dois algoritmos y e z, sendo: Pk(y) = 0.6 e Pk(z) = 0.8. Se a TD cumpre seu deadline, teremos Ek(y) = 0.4 e Ek(z) = 0.2. Conforme o número nk aumenta, teremos uma medida da capacidade de cada mecanismo em fazer previsões corretas. Um bom mecanismo de previsão de perda de deadlines deve gerar um erro igual a zero ao longo de suas execuções.

7. Simulações

O objetivo da simulação é avaliar o uso dos métodos de particionamento de deadlines fim a fim *Ultimate* (UD), *Effective Deadline* (ED) e *Equal Flexibility* (EQF) como mecanismos de previsão de perda de deadlines. Inicialmente, o método EQF é utilizado para fins de escalonamento, com o particionamento do deadline fim a fim em deadlines locais. Na seqüência, para fins de previsão de perda de deadlines, os algoritmos UD, ED e EQF são utilizados para a definição dos *milestones*. Neste trabalho, propomos o uso do servidor de aperiódicas *Sporadic Server* [5] para o escalonamento dos segmentos locais da TD aperiódica. A fila do servidor de aperiódicas será escalonada de maneira preemptiva pelo algoritmo EDF [5]. Três diferentes cenários de carga serão simulados e analisados. As condições da simulação são descritos a seguir.

7.1 Condições da Simulação

O sistema é composto por quatro nodos interconectados. Para cada nodo do sistema, o uso do processador é de 50% para as tarefas locais periódicas com deadlines *hard* e 50% para o servidor de aperiódicas. Existem quatro tarefas locais periódicas em cada nodo, cujos períodos são uniformemente distribuídos entre 10 e 100. Suas utilizações são de 20% para as tarefas com períodos menores e 10% para cada uma das outras três tarefas. As tarefas locais têm deadlines relativos iguais aos seus períodos.

Existem quatro TDs tempo real aperiódicas do tipo pipeline que percorrem os nodos do sistema (Figura 3).

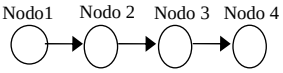

Figura 3 – Threads distribuídas do sistema.

As TDs aperiódicas sempre executam um método no nodo onde elas foram criadas. Estas TDs transpõem os nodos do sistema para coletar informações. O tempo entre chegadas destas TDs segue uma distribuição exponencial de 1000 unidades de tempo – u.t. e o tempo de execução de cada segmento local varia de 5 u.t. até 60 u.t. Assumimos que não existe particionamento na rede e o atraso na comunicação da rede segue uma distribuição uniforme entre 1 u.t. e 2 u.t. Um *Sporadic Server* é usado, com capacidade de 5 u.t. e período igual a 10 u.t. Conforme descrito anteriormente, o algoritmo RM será utilizado neste sistema para escalonar as tarefas locais periódicas, o interceptador e o servidor de aperiódicas.

Para fins de escalonamento local, os deadlines dos segmentos locais de cada TD foram definidos através do método de particionamento EQF. Todas as TDs possuem 7 segmentos locais, onde os segmentos de 1 a 4 representam o caminho de execução dos nodos 1 a 4. Os segmentos locais de 5 a 7 representam o caminho sobre os nodos 3 a 1, isto é, o retorno da TD para o nodo origem (Figura 4).

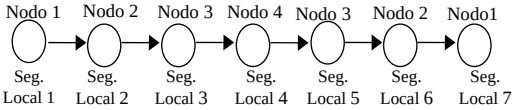

Figura 4 – Segmentos locais das TDs do sistema.

Nas simulações, focamos nossa atenção no segmento local 4 porque ele representa a metade do pipeline. Se neste ponto uma boa previsão for realizada, ações corretivas podem ser tomadas com o objetivo de melhorar o desempenho do sistema.

7.2 Resultados da Simulação

As tabelas 1, 2 e 3 apresentam os resultados da simulação. Três tipos de distribuição de carga foram utilizados. No primeiro caso, a maioria dos deadlines é alcançada. No segundo caso, existe uma distribuição balanceada da carga, onde o número de deadlines fim a fim perdidos e alcançados é equilibrado. No último caso, a maioria dos deadlines é perdida. Para cada tipo de distribuição de carga, existem três medidas: *Right*, *Wrong* e *Error*. Conforme a quantidade de TDs que chegam no sistema, *Right* expressa a quantidade de acertos de cada previsor. A segunda medida, *Wrong*, é complementar à primeira e expressa a quantidade de erros de cada previsor. *Error* expressa a taxa de erro de cada previsor e está definida na seção 6. Para cada tipo de carga, os três mecanismos de previsão foram simulados. Foram simuladas as chegadas de aproximadamente 40 TDs. O padrão de chegadas das TDs é o mesmo para todos os mecanismos testados.

Tabela 1 – Carga Leve, deadline fim a fim = 560

	MET	MISS	RIGHT	WRONG	ERROR
UD	28	7	28	7	0,19
ED	28	7	29	6	0,16
EQF	28	7	31	4	0,36

Tabela 2 – Carga Balanceada, deadline fim a fim = 520

	MET	MISS	RIGHT	WRONG	ERROR
UD	10	13	10	13	0,52
ED	10	13	12	11	0,38
EQF	10	13	23	0	0,22

Tabela 3 – Carga Pesada, deadline fim a fim = 480

	MET	MISS	RIGHT	WRONG	ERROR
UD	4	18	6	16	0,67
ED	4	18	9	13	0,45
EQF	4	18	19	3	0,19

Quando o sistema possui uma carga leve (Tab. 1) onde a maioria dos deadlines fim a fim das TDs são cumpridos, os mecanismos de previsão de perda de deadlines apresentam resultados similares. Nenhum deles demonstrou ser consistentemente superior aos outros. A taxa de erro dos mecanismos UD e ED apresentou melhores resultados que o mecanismo EQF. Isto acontece porque UD e ED podem ser considerados algoritmos otimistas, isto é, eles acreditam que a maioria das TDs irá cumprir seus deadlines e isto de fato acontece neste tipo de carga.

Quando o sistema possui uma carga balanceada (Tabela 2) onde o número de deadlines alcançados e perdidos é equilibrado, observamos que o mecanismo EQF apresentou melhores resultados. Enquanto UD fez 10 previsões corretas (*Right*) e 13 previsões incorretas (*Wrong*), EQF fez 23 previsões corretas e nenhuma previsão incorreta. Além disso, a taxa de erro do EQF é menor em relação aos demais mecanismos.

Um bom resultado a favor do mecanismo de previsão EQF também foi observado quando o sistema possui carga pesada (Tabela 3). Este caso é especialmente interessante porque EQF demonstra ser um mecanismo confiável o suficiente para ser usado como base para realização de ações corretivas em sistemas sobrecarregados. Ao contrário do mecanismo UD que pode ser considerado um mecanismo otimista, EQF pode ser visto como um mecanismo pessimista, isto é, ele prevê que a maioria das TDs perderá seus deadlines e isto de fato acontece neste cenário de carga. Usando EQF como mecanismo de previsão de perda de deadlines em sistemas sobrecarregados, é possível realizar ações corretivas para melhorar o desempenho do sistema. Um exemplo de ação corretiva poderia ser aumentar o deadline fim a fim da TD para que ela consiga cumprir suas restrições temporais. Descartar a TD seria outra possibilidade de ação corretiva para controle de sobrecarga, aumentando a taxa de deadlines fim a fim alcançados.

8. Conclusões

Neste artigo consideramos uma arquitetura de sistema que oferece suporte para threads tempo real distribuídas. Os objetivos da arquitetura foram garantir o cumprimento das restrições temporais das tarefas locais periódicas e, ao mesmo tempo, tentar cumprir o máximo de deadlines fim a fim das TDs. Três mecanismos de previsão de perdas de deadlines foram estudados. Estes mecanismos fazem a previsão a partir da geração de *milestones* (deadlines locais estimados).

Simulações foram realizadas para analisar o comportamento dos mecanismos de previsão propostos. Verificamos que o mecanismo EQF apresenta bons resultados, principalmente em sistemas sobrecarregados. O fato de o mecanismo EQF considerar o tempo de computação de cada segmento local da TD na geração dos *milestones* faz com que estes tenham valores menores em relação aos demais mecanismos (UD e ED), tornando a previsão sobre a perda de deadlines mais realista neste tipo de sistema.

Neste artigo analisamos somente TDs do tipo pipeline. Em trabalhos futuros, pretendemos estender esta pesquisa para TDs com fluxos de controle de execução probabilisticamente conhecidos.

Referências Bibliográficas

1. Clark, R.K., Jensen, E.D., Reynolds, F.D.: *An architectural overview of the Alpha Real-Time Distributed Kernel*. In: Proc. USENIX Workshop on Microkernels and Other Kernel Architectures, Seatlle, 1992, p.p. 127–146.
2. Jonsson, J., Shin, K. G.: *A Robust Adaptive Metric for Deadline Assignment in Heterogeneous Distributed Real-Time Systems*. In: Proceedings of the IEEE International Parallel Processing Symposium, pp. 678-687, 1999.
3. Kao, B., Molina, H.G.: *Deadline Assignment in a Distributed Soft Real-Time System*. In: IEEE Trans. Parallel and Distributed Systems, vol. 8, nb. 12, pp.1268-1274, Dec 1997.
4. Li, P., et al.: *Scheduling Distributable Real-Time Threads in Tempus Middleware*. In Proc. ICPADS, Newport Beach, California, pp. 187, Jul. 2004.
5. Liu, J.W.S.: *Real-Time Systems*. Prentice Hall, 2000.
6. Marinca, D., Minet, P., George, L.: *Analysis of Deadline Assignment Methods in Distributed Real-Time Systems*. In: Computer Communications, Vol.27, issue 15, Elsevier, June 2004.
7. Natale, M. D., Stankovic, J. A.: *Dynamic End-to-End Guarantees in Distributed Real-Time Systems*. In: Proceedings of Real-Time Symposium, San Juan, Puerto Rico, 7-9 December, 1994.
8. Plentz, P., Oliveira, R. S., Montez, C.: *Scheduling of the Distributed Thread Abstraction with Timing Constraints using RTSJ*. In: ETFA'2005 - 10th IEEE International Conference on Emerging Technologies and Factory Automation, Catania, Italy, 19-22 September 2005, pp. 23-30.
9. Sun, J.: *Fixed-Priority End-To-End Scheduling in Distributed Real-Time Systems*. PHd Thesis, Graduate College, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1997.
10. Zhang, J., et al: *A Real-Time Distributed Scheduling Service For Middleware Systems*. In: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, pp. 59-65, 2005.
11. Zhang, Y., et al: *A Real-Time Performance Comparison of Distributable Threads and Event Channels*. In: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium, pp 497-506, 2005.