Universidade Federal de Santa Catarina
Programa de Pós-Graduação em Engenharia Elétrica

Mecanismos de Previsão de Perda de Deadline para Sistemas Baseados em Threads Distribuídas Tempo Real

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de **Doutora em Engenharia Elétrica**

Patricia Della Méa Plentz

Florianópolis, Abril de 2008

Lista de Abreviaturas

ASQ: *Aperiodic Server Queue Length*

CORBA: *Commom Object Request Broker Architecture*

ED: *Effective Deadline*

EDF: *Earliest Deadline First*

EQF: *Equal Flexibility*

EQS: *Equal Slack*

FIFO: *First In First Out*

GUS: *Generic Utility Scheduling*

HUA: *Handler-assured Utility Accrual*

JCP: *Sun's Java Community Process*

JSR: *Java Specification Request*

JVM: *Java Virtual Machine*

LLF: *Least Laxity First*

OMG: *Object Management Group*

RM: *Rate Monotonic*

RMI: *Remote Method Invocation*

RPC: *Remote Procedure Call*

RTSJ: *Real-Time Specification for Java*

TAO: *The ACE ORB*

TPR: *Thread Polling with bounded Recovery*

TUF: *Time/Utility Function*

Lista de Símbolos

C_i = tempo de computação da tarefa i

P_i = período da tarefa i

D_i = deadline da tarefa i

J_i = release jitter da tarefa i

ar_i = tempo de chegada da tarefa i

r_i = tempo de liberação da tarefa i

st_i = tempo de início da tarefa i

ct_i = tempo de término da tarefa i

s_i = subtarefa i

$rl(s_i)$ = tempo de resposta local da subtarefa i

$dl(s_i)$ = deadline local da subtarefa i

min = intervalo mínimo entre duas ativações consecutivas de uma tarefa

TD = thread distribuída

I_i = itinerário i de uma thread distribuída

$DPond_i$ = deadline ponderado do nodo i

$PC(z)$ = taxa de previsões corretas realizada pelo mecanismo z

$E(z)$ = taxa de erro associado com a previsão realizada pelo mecanismo z

$Prob_k(z)$ = probabilidade definida pelo mecanismo z da thread distribuída k cumprir seu deadline fim a fim

Rff_k = tempo de resposta fim a fim da thread distribuída k

Dff_k = deadline fim a fim da thread distribuída k

Y_i = tempo de resposta estimado da thread distribuída em um nodo

X_i = tamanho da fila do servidor de aperiódicas do nodo em questão.

Δ = tempo máximo para envio de mensagens entre dois nodos quaisquer

σ = ajuste de probabilidade dos mecanismos de previsão

Resumo

Este trabalho trata do problema de previsão de perda de deadline em sistemas distribuídos de tempo real. O contexto do trabalho refere-se aos sistemas com restrições temporais não críticas. O objetivo é fornecer mecanismos de previsão de perda de deadline para melhorar o desempenho desses sistemas. São propostos diferentes mecanismos para sistemas construídos a partir de threads distribuídas. Para tanto, define-se um modelo de tarefas e uma arquitetura de sistema. O modelo de tarefas é constituído de tarefas locais periódicas com deadlines críticos e tarefas distribuídas aperiódicas com deadlines firmes. A arquitetura de sistema, presente em cada nodo, é composta por uma tarefa interceptadora e uma tarefa servidora, além de tarefas periódicas locais. A seguir, os mecanismos de previsão de perda de deadlines propostos são descritos. O mecanismo baseado em *Milestones* utiliza restrições temporais da thread distribuída para definir um tempo de resposta estimado (um *milestone*). O mecanismo de previsão baseado na folga restante (FR) considera as execuções condicionais da thread distribuída para fins de previsão. O mecanismo *Aperiodic Server Queue Length* (ASQ) utiliza informações a respeito da carga computacional do sistema, além de informações a respeito da thread distribuída. Os mecanismos propostos foram avaliados através de simulações, considerando diferentes contextos de execução. Os resultados mostram que os mecanismos FR e ASQ geram melhores previsões que o mecanismo baseado em *Milestones*. De uma maneira geral, os mecanismos de previsão fornecem uma estratégia adequada para melhorar o comportamento do sistema, porque permitem a antecipação de decisões acerca das medidas necessárias para aumentar seu desempenho.

Abstract

This work considers the deadline miss prediction problem in distributed real-time systems. The interest area of this work refers to systems with non-critical timing constraints. The objetive is to provide deadline miss prediction mechanisms for systems implemented by distributed threads. To achieve this goal, it is defined a task model and a system architecture. The task model is composed by periodic local tasks with hard deadlines and aperiodic distributed tasks with firm deadlines. The system architecture, present in each system node, is composed by an interceptor task and a server task. The prediction mechanisms proposed in this work use information about temporal constraints of a distributed task and information about workload of the system. The mechanism based on Milestones uses timing constraints of the current distributed thread to define an estimated response time (milestone). The mechanism based on available slack (*Folga Restante - FR*) considers the conditional execution of the distributed thread to carry out the prediction. The Aperiodic Server Queue Length (ASQ) mechanism uses information about system workload and information about the current distributed thread. The proposed mechanisms were evaluated through simulations, using different execution scenarios. The results show that FR and ASQ mechanisms generate better predictions than the mechanism based on Milestones. In the general, the prediction mechanisms provide an adequate strategy to improve the system behavior because they allow the anticipation of decisions about necessary measures to improve system performance.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Atualmente, o uso de sistemas computacionais de tempo real é amplo e abrange diversas áreas de aplicação, como sistemas de controle de tráfego aéreo, sistemas embutidos em equipamentos industriais e sistemas multimídia. Nestes tipos de sistemas, a correção temporal é tão importante quanto a correção lógica. A correção temporal está associada com o cumprimento de um tempo máximo de execução - um *deadline* [1].

Sistemas de tempo real podem ser diferenciados por sua criticalidade. Sistemas críticos são aqueles em que a perda de um deadline pode ter conseqüências irreversíveis. Já os sistemas não-críticos toleram a perda de alguns deadlines, desde que esta não ocorra repetidamente e indiscriminadamente.

Um sistema tempo real é formado por tarefas, que podem ser classificadas quanto a sua periodicidade de ativações. Uma tarefa é dita *periódica* se a cada período P de tempo sempre ocorrer uma ativação. Entretanto, quando a ativação da tarefa responde a eventos internos ou externos, definindo uma característica aleatória nas ativações, a tarefa é dita *aperiódica*. Uma tarefa aperiódica é também *esporádica* se existir um intervalo mínimo de tempo (maior que zero) entre duas ativações sucessivas [1].

Na medida em que as técnicas e metodologias aplicadas no desenvolvimento de sistemas

[1] Os termos em inglês serão traduzidos para o português quando isto não acarretar prejuízo ao entendimento do trabalho. Na literatura de tempo real, o termo deadline é bastante comum e usado sem tradução.

tempo real consolidam-se, aumenta o interesse do emprego destas em sistemas distribuídos. Políticas usadas em sistemas tempo real locais são estendidas para incorporar os aspectos relacionados a um ambiente distribuído.

Em sistemas distribuídos de tempo real, o emprego de um modelo de execução fim a fim é fundamental para garantir a previsibilidade do sistema. Usualmente, um sistema distribuído é implementado a partir de um conjunto de tarefas locais dispostas em diferentes nodos que colaboram entre si para alcançar a solução de escalonamento empregada. A implementação destas tarefas locais colaboradoras envolve muitas questões que são inerentes ao sistema distribuído. O ideal seria a existência de uma tarefa distribuída, conhecida em todo sistema através de um identificador, que carrega suas restrições temporais e parâmetros da computação que ela representa conforme visita os nodos do sistema, sem onerar o programador da aplicação com questões de implementação desta tarefa.

O conceito *threads distribuídas* surgiu para fornecer suporte a esta idéia. Thread distribuída é a abstração de um fluxo de controle fim a fim que se estende e se retrai através do sistema distribuído via invocações de métodos remotos e possui um identificador conhecido em todo o sistema distribuído. Um sistema implementado a partir desta abstração facilita a visão do programador na medida em que ele não precisa tratar explicitamente os aspectos subjacentes relacionados com a implementação da tarefa distribuída.

O contexto deste trabalho refere-se aos sistemas distribuídos tempo real não-críticos. As aplicações para este sistema são implementadas a partir do conceito thread distribuída, as quais são aperiódicas e possuem deadlines fim a fim firmes. O método de escalonamento proposto neste trabalho é composto por dois estágios, que são: particionamento do deadline fim a fim e escalonamento local. No primeiro estágio, o retorno da thread distribuída ao nodo onde a invocação remota foi disparada, bem como seus fluxos condicionais de execução (representados por estruturas de programação do estilo *If-Then*) são considerados na definição de deadlines locais. No segundo estágio, define-se uma arquitetura de sistema, presente em cada nodo, composta por uma tarefa interceptadora e uma tarefa servidora de aperiódicas. Estas duas tarefas são responsáveis pelo atendimento e escalonamento da thread distribuída em cada nodo do sistema que ela visita durante sua execução.

Cada nodo do sistema possui, portanto, threads distribuídas aperiódicas com deadlines firmes e tarefas locais periódicas com deadlines críticos, caracterizando um conjunto híbrido

de tarefas. O objetivo de escalonamento da arquitetura de sistema proposta é garantir os deadlines das tarefas locais periódicas e ao mesmo reduzir o tempo de resposta das threads distribuídas aperiódicas.

Sistemas distribuídos tempo real não-críticos podem fazer uso de mecanismos de previsão de perda de deadlines como forma de melhorar seu desempenho. A detecção antecipada a respeito de uma possível perda de deadline permite que ações corretivas sejam executadas a tempo de melhorar o desempenho do sistema distribuído.

Sob a perspectiva de detecção antecipada de perda de deadlines, surgem questões importantes que devem ser discutidas, como por exemplo, qual o momento (ou nodo) adequado para realizar a previsão e como incorporar, em um mecanismo de previsão, execuções condicionais de uma tarefa distribuída.

Poucos trabalhos na literatura propõem mecanismos de previsão de perda de deadlines em ambientes distribuídos de tempo real não-críticos. Não foi encontrado na literatura nenhum estudo amplo sobre como implementar mecanismos de previsão de perda de deadlines para sistemas baseados em threads distribuídas, considerando suas execuções condicionais.

1.2 Objetivos da Tese

O objetivo geral desta tese é a elaboração e avaliação de mecanismos de previsão de perda de deadlines para sistemas construídos a partir do conceito threads distribuídas. Para atender o objetivo geral desta tese, os seguintes objetivos específicos foram definidos:

- Desenvolver uma arquitetura de sistema e um modelo de tarefas para aplicações não críticas implementadas a partir de threads distribuídas;
- Elaborar mecanismos de previsão de perda de deadlines considerando as execuções condicionais das threads distribuídas e a carga computacional do sistema;
- Avaliar a qualidade das previsões realizadas pelos mecanismos considerando diferentes contextos de execução.

1.3 Adequação às Linhas de Pesquisa do Curso

O trabalho descrito nesta tese está inserido no contexto da Área de Concentração em Automação e Sistemas do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina. Este trabalho está perfeitamente integrado com os demais trabalhos de pesquisa sobre tempo real e com as atividades do Curso de Pós-Graduação em Engenharia Elétrica desta Universidade.

1.4 Organização do Texto

A organização do texto desta tese revela as diferentes etapas de pesquisa acerca dos temas estudados. Em uma forma ampla, ele pode ser dividido em duas partes, onde a primeira delas apresenta uma revisão bibliográfica sobre assuntos que constituem os pilares desta tese - sistemas de tempo real e threads distribuídas. Da união destes assuntos juntamente com a definição de algumas premissas evoluímos para a segunda parte deste texto, com a proposição de uma arquitetura de sistema. Nesta arquitetura, focamos a atenção sobre mecanismos de previsão de perda de deadlines, assunto alvo desta tese. A elaboração e a análise de mecanismos de previsão para sistemas baseados em threads distribuídas norteou a pesquisa deste trabalho.

Esta tese é composta por 9 capítulos, sendo que este primeiro apresentou os aspectos que motivaram esta pesquisa, os objetivos desta tese e a adequação às linhas de pesquisa do curso.

No capítulo 2 são descritos conceitos sobre sistemas de tempo real relacionados com o tema de pesquisa desta tese. Especificamente, revisitamos a bibliografia sobre escalonamento de tarefas, servidores de aperiódicas e sistemas distribuídos, em especial a questão sobre particionamento do deadline fim a fim.

No capítulo 3 são exploradas algumas definições acerca de threads distribuídas. Iniciamos apresentando sua conceituação no sistema operacional *Alpha* e, na seqüencia, os trabalhos que tratam desta abstração computacional no *Real-Time CORBA 1.2*. Este capítulo também mostra implementações de threads distribuídas na linguagem de programação Java. Trabalhos publicados nesta área consideram a abstração com e sem requisitos temporais. O

esforço de revisão bibliográfica deste capítulo foi sobre threads distribuídas com restrições temporais.

A partir dos capítulos 2 e 3, definimos algumas premissas e elaboramos uma arquitetura de sistema que acomoda threads distribuídas tempo real. O capítulo 4 apresenta esta arquitetura e o modelo de tarefas adotado nesta tese. Definimos um método de escalonamento para threads distribuídas, considerando o particionamento do seu deadline fim a fim. Nesse capítulo são levantadas algumas questões sobre previsão de perda de deadlines e introduzimos a métrica utilizada para medir a qualidade das previsões realizadas pelos mecanismos propostos neste trabalho. O capítulo é finalizado com a descrição do modelo geral de funcionamento dos mecanismos de previsão propostos.

No capítulo 5 o mecanismo de previsão baseado em *Milestones* é descrito. Este é o primeiro de uma série de três mecanismos propostos nesta tese. São apresentadas diferentes formas para definir os *Milestones* de uma thread distribuída. Para cada *Milestone* um exemplo é utilizado, mostrando como ocorre a previsão de perda de um deadline fim a fim. A qualidade das previsões realizadas por este mecanismo é avaliada através de simulações.

Outro mecanismo de previsão é apresentado no capítulo 6. O mecanismo baseado na *Folga Restante* (FR) é introduzido e exemplificado através da descrição de um cenário de execução. Os resultados gerados a partir de simulações são avaliados e servem como apoio para a definição de um terceiro mecanismo.

O mecanismo *Aperiodic Server Queue Length* (ASQ) é descrito no capítulo 7. Para realizar a previsão de perda de deadline de uma thread distribuída, este mecanismo utiliza informações fornecidas pelas demais threads distribuídas ativas no sistema, além de informações a respeito das restrições temporais da thread distribuída em questão. Os resultados das simulações são apresentados e discutidos.

No capítulo 8 são descritos trabalhos encontrados na literatura, em algumas áreas da computação, que propõem algoritmos que fazem a previsão de tempo de resposta de tarefas e de sistemas. São discutidas as limitações destes trabalhos no que se refere a implementação destes considerando a arquitetura de sistema, o modelo de tarefas e as premissas adotadas nesta tese.

Esta tese é finalizada com o capítulo de conclusões, o qual apresenta a revisão dos

objetivos, o resumo do trabalho realizado, as contribuições ao estado da arte bem como perspectivas para futuras pesquisas.

Capítulo 2

Sistemas Tempo Real

2.1 Introdução

Sistemas tempo real são sistemas computacionais que demandam que seus resultados estejam lógica e temporalmente corretos, isto é, devem produzir resultados corretos no tempo especificado. Em geral, um prazo máximo para a execução das tarefas é definido e várias políticas de escalonamento propõem formas de garantir que todas as tarefas (ou o maior número possível) cumpram suas restrições temporais.

A consolidação das técnicas aplicadas no desenvolvimento de sistemas tempo real motivou seu emprego em aplicações distribuídas. Técnicas e políticas usadas em sistemas tempo real locais são estendidas para incorporar os aspectos relacionados a um ambiente distribuído.

O objetivo deste capítulo é apresentar conceitos sobre sistemas tempo real distribuídos. Para tanto, na seção 2.2 são apresentados os tipos de tarefas que compõem uma aplicação tempo real e suas características temporais. O escalonamento deste tipo de tarefa é detalhado através da explanação das principais técnicas existentes na literatura. A seguir, na seção 2.3 são descritos os principais algoritmos que atuam como servidores de aperiódicas, suas vantagens e desvantagens. Os aspectos relacionados com sistemas distribuídos tempo real estão na seção 2.4, onde métodos de particionamento de deadlines fim a fim são discutidos. Este capítulo é finalizado com a seção 2.5, que expõe observações sobre os conceitos e técnicas vistos no decorrer deste texto.

2.2 Conceitos Básicos

Sistemas tempo real são aqueles que devem reagir a eventos do ambiente, respeitando restrições temporais impostas pela aplicação. O correto comportamento destes sistemas depende, portanto, não somente do resultado da computação mas também do tempo em que ela é realizada.

Usualmente, a unidade de concorrência de um sistema é representada por uma tarefa, a qual pode receber dados, executar um algoritmo específico e gerar algum tipo de saída. Na maioria dos sistemas de software, uma tarefa é dita correta no aspecto lógico se gerar uma saída correta em função dos dados de entrada. Em sistemas tempo real, as tarefas devem estar corretas também sob o aspecto temporal, isto é, devem produzir um resultado correto dentro de um prazo satisfatório. Um resultado que ocorra além do prazo especificado pode ser sem utilidade ou até representar uma ameaça [2]. Este tempo máximo no qual uma tarefa deve concluir sua execução é chamado de *deadline*.

Dependendo das conseqüências que podem ocorrer em função da perda de um deadline, as tarefas tempo real são usualmente classificadas em *Críticas (Hard)*, *Não-Críticas (Soft)* ou *Firmes*:

- *Tarefa Crítica*: a perda de um deadline implica em falhas catastróficas no sistema de tempo real e/ou no ambiente controlado pelo sistema. Essas falhas podem representar danos irreversíveis em equipamentos ou ainda, em perda de vidas humanas.

- *Tarefa Não-Crítica*: a perda de um deadline implica em diminuição de desempenho do sistema. O desvio do comportamento normal do sistema não representa um custo muito significativo.

- *Tarefa Firme*: a perda de um deadline não implica em falhas catastróficas no sistema, porém não existe benefício em executar a tarefa até a sua conclusão. Se a tarefa perde o deadline, ela deixa de ter valor para o sistema.

Um sistema composto por tarefas críticas é dito sistema tempo real crítico. As aplicações incluem, tipicamente, tarefas críticas e não críticas e portanto, um sistema tempo real crítico deve ser projetado para executar ambos os tipos de tarefas através de estratégias específicas. Quando uma aplicação possui um conjunto híbrido de tarefas, o objetivo do sistema deve ser garantir as restrições temporais das tarefas críticas e minimizar o tempo de resposta médio

das tarefas não-críticas.

Além do deadline, outras restrições temporais podem ser definidas para uma tarefa tempo real. Os seguintes parâmetros são geralmente utilizados na caracterização de uma tarefa [3]:

- Tempo de Chegada: é o tempo no qual uma tarefa se torna pronta para execução.
- Tempo de Liberação: é o instante no qual uma tarefa é inserida na fila de pronto (*ready queue* - fila de tarefas prontas para serem executadas);
- Tempo de Computação: tempo necessário para o processador executar a tarefa;
- Tempo de Início: tempo no qual uma tarefa inicia sua execução;
- Tempo de Resposta: tempo no qual uma tarefa conclui sua execução;
- Importância: representa a importância relativa da tarefa com relação às outras do sistema;
- *Lateness*: representa o atraso no tempo de resposta de uma tarefa. Se a tarefa conclui sua execução antes do seu deadline, o *lateness* é negativo.
- Folga: é o tempo máximo que uma tarefa pode ser atrasada na sua ativação para que o tempo de resposta seja menor ou igual ao deadline da tarefa.

O tempo de liberação de uma tarefa pode ou não coincidir com o tempo de chegada da tarefa. Geralmente, assume-se que uma tarefa é imediatamente encaminhada para a fila de pronto, tão logo sua instância chegue no sistema. Ocorre que este encaminhamento pode ser atrasado por um escalonador ativado por tempo ou pelo bloqueio na recepção de uma mensagem (onde tarefas são ativadas por mensagens). Essa não-coincidência entre o tempo de chegada e a liberação da tarefa é conhecido como *Release Jitter*, que representa a máxima variação dos tempos de liberação das instâncias das tarefas [2].

O tempo de resposta de uma tarefa compreende o intervalo de tempo entre a chegada da tarefa no sistema e o término de sua execução. O tempo máximo de resposta de uma tarefa é o maior tempo de resposta que esta poderá apresentar dentro da aplicação em questão. Como as demais tarefas da aplicação podem interferir na execução da tarefa corrente, o tempo de

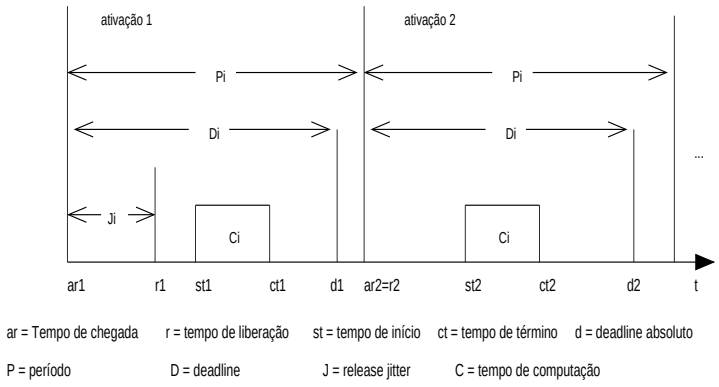

Figura 2.1: Ativações de uma Tarefa Periódica.

resposta de uma tarefa não é definido apenas pelo comportamento da tarefa em questão mas sim por todas as tarefas da aplicação.

Com respeito à periodicidade das ativações, uma tarefa pode ser classificada como *periódica* ou *aperiódica*. No primeiro caso, se a cada período P de tempo sempre ocorrer uma ativação, ela é dita periódica. No segundo caso, quando a ativação da tarefa responde a eventos internos ou externos, definindo uma característica aleatória nas ativações, a tarefa é dita aperiódica. Uma tarefa aperiódica é também *esporádica* se existir um intervalo mínimo de tempo (maior que zero) entre duas ativações sucessivas [1].

A partir das restrições temporais descritas acima e da periodicidade da ativação de uma tarefa, o comportamento temporal de uma tarefa periódica pode ser descrito pela quádrupla (J, C, P, D) onde C representa o tempo de computação da tarefa, P é o período da tarefa, D é o deadline e J é o *Release Jitter* da tarefa (figura 2.1).

Na figura 2.1 cada ativação da tarefa é definida a partir dos tempos absolutos: tempos de chegada (*arrival time* - ar_i), tempos de liberação (*release time* - r_i), tempos de início (*start time* - st_i), tempos de término (*conclusion time* - ct_i) e os deadlines absolutos (d_i).

Uma tarefa esporádica é definida por (C, D, min) onde *min* representa o intervalo mínimo entre duas ativações consecutivas. Já uma tarefa aperiódica é definida pelas restrições C e D ou apenas C no caso desta não possuir um deadline [4] (figura 2.2).

Algumas vezes, em função de dependências semânticas, as tarefas de uma aplicação não podem executar em uma ordem arbitrária. É necessário seguir uma ordem de precedência, onde uma tarefa poderá executar somente após o término de outra. Esta implicação semântica

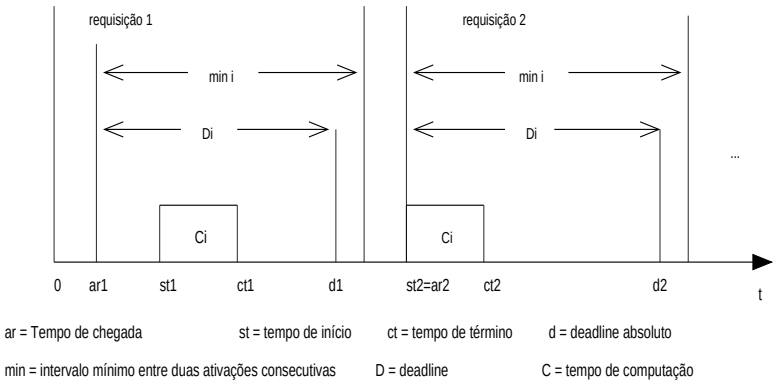

Figura 2.2: Ativações de uma Tarefa Aperiódica.

é chamada de *Relação de Precedência* entre tarefas de uma aplicação.

As relações de precedência de um conjunto de tarefas podem ser representadas por grafos acíclicos orientados, onde os nós representam as tarefas e os arcos representam as relações de precedência existentes entre as tarefas. A figura 2.3 mostra as relações de precedência de cinco tarefas. A tarefa J1 é a única que pode iniciar sua execução porque ela não possui tarefas predecessoras. Assim que J1 conclui sua execução, a tarefa J2 ou a tarefa J3 pode iniciar sua execução. A tarefa J4 pode iniciar sua execução somente depois que a tarefa J2 é concluída. A tarefa J5 deve esperar pela conclusão das tarefas J2 e J3.

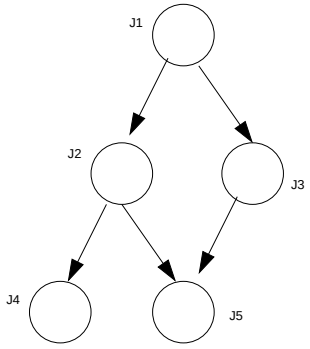

Figura 2.3: Relações de Precedência entre Cinco Tarefas.

2.2.1 Escalonamento de Tarefas Tempo Real

Escalonar tarefas significa ordená-las na fila de prontos. Logo, o componente responsável, em tempo de execução, pela gestão do processador é o escalonador. Ele executa uma política de escalonamento que ordena um conjunto de tarefas para execução em um processador. As escalas produzida por um escalonador, se forem viáveis, garantem o

cumprimento das restrições temporais das tarefas tempo real [1].

Uma política de escalonamento pode ser classificada em relação à preempção [3]:

- **Preemptiva**: uma política de escalonamento preemptiva pode interromper a execução de uma tarefa, a qualquer tempo, atribuindo o processador para outra tarefa de maior prioridade (ou de maior importância).
- **Não-Preemptiva**: em uma política de escalonamento não-preemptiva, uma tarefa conclui sua execução sem ser interrompida por qualquer outra.

Uma política de escalonamento pode ainda ser classificada como estática ou dinâmica, conforme a seguir [3]:

- **Estática**: uma política de escalonamento é estática quando as decisões de escalonamento são baseadas em parâmetros fixos e são atribuídas para as tarefas antes de sua execução.
- **Dinâmica**: políticas de escalonamento dinâmicas utilizam parâmetros dinâmicos que podem mudar durante a evolução do sistema.

Uma política de escalonamento pode também ser classificada como *offline* ou *online* [3]:

- ***Offline***: uma política de escalonamento offline é executada antes da ativação das tarefas no sistema. A escala é gerada e armazenada em uma tabela. O escalonador, em tempo de execução, consulta a tabela e cumpre a escala nela definida.
- ***Online***: uma política de escalonamento é online quando as decisões de escalonamento ocorrem em tempo de execução, sempre que uma tarefa chega no sistema ou quando uma tarefa conclui sua execução.

Uma aplicação tempo real pode apresentar carga estática ou dinâmica, de acordo com os tipos de tarefas que ela possui. Se todas as tarefas que compõem a aplicação são periódicas ou esporádicas, a carga do sistema é estática porque é possível conhecer os tempos em que

elas estarão no sistema e quais são suas restrições temporais. Neste caso, o conjunto de tarefas é conhecido em tempo de projeto e as situações de pior caso podem ser determinadas [2].

De outra forma, quando alguma tarefa da aplicação é do tipo aperiódica, a carga do sistema assume um caráter dinâmico já que não é possível conhecer os tempos de chegada desta tarefa. Neste caso, onde os tempos de chegada de uma tarefa não podem ser antecipados, a situação de pior caso não pode ser conhecida em tempo de projeto.

Na classe de políticas de escalonamento *online* encontram-se as que são dirigidas por prioridades, onde a tarefa com maior prioridade é escalonada e executada. Os algoritmos dirigidos por prioridades diferem uns dos outros na maneira como uma prioridade é atribuída a uma tarefa. Desta forma, estes algoritmos podem ser classificados em fixos ou dinâmicos. Um algoritmo de *Prioridade Fixa* atribui a mesma prioridade para todas as ativações de uma mesma tarefa. Nesta classe de algoritmos, a prioridade de uma tarefa é fixa em relação às demais tarefas [4].

Em contraste, os algoritmos de *Prioridade Dinâmica* atribuem prioridades diferentes para cada ativação de uma mesma tarefa. Nesta classe de algoritmos, a prioridade de uma tarefa em relação as demais muda conforme as ativações desta tarefa.

Muitos algoritmos de Prioridade Fixa são citados na literatura [4, 2]. Um deles, bastante conhecido e que é de interesse neste trabalho é o Taxa Monotônica (*RM - Rate Monotonic*) [4]. Este algoritmo atribui prioridades para as tarefas usando como base seus períodos. Quanto menor o período da tarefa, maior sua prioridade. A taxa de liberações entre ativações de uma tarefa é inversamente proporcional ao seu período e, assim, quanto maior sua taxa, maior sua prioridade.

Na classe de algoritmos de Prioridade Dinâmica, destacamos o *Earliest Deadline First* (EDF) [3] que atribui prioridades dinâmicas para as ativações de uma tarefa de acordo com seus deadlines absolutos. A cada chegada de uma ativação da tarefa no sistema, a fila de prontos é reordenada, considerando o seu deadline absoluto.

2.3 Servidores de Aperiódicas

Usualmente, as aplicações tempo real são compostas por tarefas periódicas e aperiódicas, definindo um modelo híbrido de tarefas. Para lidar com este tipo de conjunto de tarefas, al-

gumas técnicas são propostas na literatura. Uma das mais conhecidas é chamada *Servidor de Aperiódicas* [3], onde uma tarefa periódica utiliza seu tempo de computação para executar tarefas aperiódicas, caso exista alguma no sistema. O servidor de aperiódicas é escalonado com a mesma política usada para escalonar tarefas periódicas e, quando ativo, atende às requisições aperiódicas usando sua capacidade (tempo de computação). O ordenamento das requisições aperiódicas não depende do algoritmo de escalonamento usado para tarefas periódicas. Ele pode ser realizado usando o tempo de chegada, o tempo de computação, o deadline ou outro parâmetro.

Os servidores de aperiódicas podem ser usados segundo políticas de prioridade fixa ou dinâmica. Neste trabalho focamos sobre tipos de servidores de aperiódicas que possam ser utilizados com políticas de prioridade fixa, como Taxa Monotônica, por exemplo. Esta escolha se deve ao fato da arquitetura de sistema proposta neste trabalho utilizar esta política de escalonamento. Assumimos as tarefas periódicas como críticas, necessitando, portanto, de garantias em tempo de projeto para as situações de pior caso. As sobras na escala da carga periódica são determinadas estaticamente, em tempo de projeto. Posteriormente, em tempo de execução, estas folgas são atribuídas ao processamento aperiódico usando o conceito de servidor [1]. A seguir são descritos alguns tipos de servidores de prioridade fixa.

- *Servidor de Background*: Quando não existem tarefas periódicas para serem executadas, este servidor atende às requisições aperiódicas. Se a carga periódica for muito alta, o tempo de resposta das tarefas aperiódicas será muito elevado. Por isso, apesar de sua simplicidade, o servidor *Background* pode ser adotado apenas em situações em que as tarefas aperiódicas não sejam críticas e que a carga periódica não seja muito alta.

- *Polling Server*: a tarefa servidora Polling possui um período P e um tempo de computação C que será usado para o atendimento das requisições aperiódicas. Em cada período de ativação, a tarefa servidora executa as requisições aperiódicas pendentes dentro do limite da sua capacidade C. Quando não houver requisições aperiódicas pendentes, a tarefa servidora se suspende até seu próximo período. A capacidade não utilizada é entregue para a execução de tarefas periódicas pendentes. Se uma requisição aperiódica chega logo após a tarefa servidora ter examinado a fila, seu atendimento ocorrerá somente no próximo período da tarefa servidora.

A abordagem *Polling Server* melhora o tempo de resposta médio das tarefas aperiódicas,

se comparado com a abordagem *Background*. Entretanto, *Polling Server* não oferece serviço de resposta imediato para as requisições aperiódicas. É necessário esperar o próximo período da tarefa servidora para que ela atenda uma requisição aperiódica.

- *Deferrable Server*: da mesma forma que a abordagem *Polling Server*, uma tarefa servidora periódica também é criada para o atendimento de requisições aperiódicas, usualmente com prioridade mais alta que as demais tarefas periódicas da aplicação. *Deferrable Server* conserva sua capacidade até o final do período, mesmo não havendo requisições aperiódicas para serem atendidas. No início de cada período, a capacidade do servidor é restaurada. Esta abordagem melhora o tempo de resposta em relação ao *Polling Server*.

- *Sporadic Server*: Esta abordagem é um refinamento da abordagem *Deferrable Server* em relação a forma de restauração da capacidade da tarefa servidora. Ao contrário da abordagem anterior, que restaura a capacidade sempre no início de cada período, a abordagem *Sporadic Server* faz a restauração somente quando sua capacidade tenha sido utilizada.

2.4 Sistemas Tempo Real Distribuídos

Existem diversas definições para sistemas distribuídos. Em [5], um sistema distribuído é definido como aquele em que os componentes, localizados em uma rede de computadores, comunicam e coordenam suas atividades através da troca de mensagens. O estudo sobre sistemas distribuídos é bastante amplo e abrange questões sobre heterogeneidade, segurança, tratamento de faltas, escalabilidade, concorrência, entre outras.

Aplicações em um sistema tempo real distribuído fazem uso de modelos e técnicas para que suas restrições temporais sejam atendidas. Neste sentido, muitos aspectos são pesquisados na literatura, como por exemplo, atribuição de tarefas aos nodos, protocolos de sincronização para acesso a recursos e modelos de escalonamento fim a fim [4, 3].

Para acomodar uma aplicação com restrições temporais em um ambiente distribuído, surge a necessidade de decompor tais restrições para que estas sejam usadas no escalonamento local de cada nodo do sistema distribuído. Usualmente, o deadline de uma tarefa distribuída é definido como parte da especificação da aplicação, surgindo a partir de um tempo de resposta máximo permitido para esta tarefa. Contudo, essa tarefa será escalonada localmente, em cada nodo que ela atravessa, tornando-se necessário, portanto, definir formas de particionar o

deadline fim a fim em deadlines locais, os quais serão usados nos escalonamentos dos trechos de código locais em cada nodo.

A literatura apresenta diferentes abordagens para particionar o deadline de uma tarefa distribuída [6, 7, 8, 9]. A seguir são descritos os principais trabalhos sobre este assunto.

2.4.1 Propostas para Particionamento do Deadline Fim a Fim

Na classe de sistemas distribuídos tempo real críticos, a literatura apresenta trabalhos que propõem soluções para o particionamento de deadlines das tarefas. Em [7] é descrito um algoritmo heurístico chamado HOPA (*Heuristic Optimized Priority Assignment*) para otimizar a atribuição de prioridades para tarefas e mensagens. HOPA realiza o particionamento de deadlines e análise de escalonabilidade recursivamente até encontrar uma escala viável.

O algoritmo faz a distribuição do deadline fim a fim de cada tarefa do sistema entre suas subtarefas. Uma vez que cada subtarefa recebe um deadline local, prioridades são atribuídas em cada recurso e a análise de todo o sistema é realizada. Como resultado da análise, novos deadlines locais intermediários são calculados. A iteração procede até que uma solução viável seja encontrada ou alguma condição de parada seja alcançada.

A redistribuição de deadlines locais leva em consideração o quanto uma subtarefa está distante de sua escalonabilidade. Os novos deadlines intermediários são obtidos como função de dois fatores: a distância de cada subtarefa relativo às outras que compõem outras tarefas, e a distância de cada subtarefa relativa às outras que utilizam o mesmo recurso.

O estudo sobre particionamento de deadline também aparece em sistemas distribuídos não-críticos, como sistemas de vídeo sob demanda. Em [10], dois algoritmos são propostos, *Fair Laxity Distribution* (FLD) e *Unfair Laxity Distribution* (ULD). Estes algoritmos definem deadlines locais para uma tarefa distribuída incorporando a carga do nodo em questão através da inspeção dos tempos de computação das demais tarefas presentes no nodo. Os algoritmos fazem parte de um teste de aceitação que somente aceita a tarefa para execução no nodo caso sua inclusão mantenha o conjunto de tarefas escalonável. No caso da tarefa ser aceita no nodo, ambos os algoritmos calculam a folga desta tarefa. A folga é definida como diferença entre o deadline fim a fim da tarefa e o somatório dos deadlines locais nos nodos que pertencem ao

caminho da tarefa. A diferença entre os algoritmos reside na forma em que a folga é distribuída para a tarefa. O algoritmo FLD divide a folga pelo número de nodos que fazem parte do caminho da tarefa. O outro algoritmo proposto, ULD, divide a folga proporcionalmente aos deadlines locais.

Os algoritmos propostos em [10] não foram comparados com outros em relação ao *overhead* gerado pela inspeção dos tempos de computação de todas as tarefas presentes em cada nodo que compõe o caminho da tarefa, para a definição dos deadlines locais. Uma limitação do uso destes algoritmos em outras aplicações surge do fato que uma tarefa precisa inicialmente percorrer todos os nodos que farão parte do seu caminho de execução para definir os deadlines locais e somente depois iniciar sua execução. Em outras palavras, a seqüência de nodos precisa ser previamente estabelecida. Este tipo de abordagem não é adequado para aplicações que não podem estabelecer o caminho, em tempo de execução, para então, em seguida, iniciar sua execução.

Kao e Garcia-Molina propõem em [6] dois conjuntos de algoritmos de particionamento de deadlines fim a fim para as subtarefas que compõem uma tarefa distribuída. No primeiro conjunto, as subtarefas executam em série apenas (como um *pipeline*), isto é, não são concorrentes entre si. No segundo conjunto são propostos algoritmos para subtarefas paralelas (aquelas que executam concorrentemente). Este tipo de subtarefa está fora do escopo deste trabalho e por isso os algoritmos para particionamento de deadlines relacionados à subtarefas paralelas não serão descritos.

No conjunto de algoritmos propostos em [6] para subtarefas seqüenciais, a técnica *Ultimate Deadline* (UD) é bastante simples porque o próprio deadline fim a fim da tarefa distribuída é atribuído para cada uma de suas subtarefas. Nesta técnica, deadline local (dl) de uma subtarefa s_i é definido como [6]:

$$dl(s_i) = d(TD).$$

onde d é o deadline fim a fim da thread distribuída TD.

Uma desvantagem desta técnica é que ela fornece, aos escalonadores locais, informação incorreta sobre a folga que uma subtarefa possui para executar. As subtarefas iniciais de uma tarefa distribuída receberão uma estimativa de folga excessiva, e poderão receber baixas

prioridades com relação a outras tarefas que estão executando no mesmo processador. Boa parte da folga estimada será gasta com as primeiras subtarefas, fazendo com que as últimas possam perder seus deadlines e, conseqüentemente, o deadline fim a fim da própria tarefa distribuída.

Na técnica *Effective Deadline* (ED) [6] os tempos de computação estimados das subtarefas são levados em consideração para efeitos de cálculo. Assim, o deadline de uma subtarefa é igual a diferença entre o deadline fim a fim da tarefa distribuída e os tempos de computação estimados das subtarefas subseqüentes a esta. Este método é definido como:

$$dl(s_i) = d(TD) - \sum_{j=i+1}^{n} C(s_j),$$

onde C é o tempo de computação estimado da subtarefa e n é o número de subtarefas da tarefa distribuída. Uma desvantagem desta técnica é que a primeira subtarefa da tarefa distribuída recebe toda a folga disponível, fazendo com que as demais subtarefas não recebam folga suficiente para executar, ocasionando uma possível perda de seus deadlines.

Equal Slack (EQS) [6] é uma técnica que faz a divisão da folga da tarefa distribuída entre todas as subtarefas que a compõem, isto é, a folga é igualmente dividida entre todas as subtarefas da tarefa distribuída. Formalmente o EQS é definido como:

$$dl(s_i) = ar(s_i) + C(s_i) + (d(TD) - ar(s_i) - \sum_{j=1}^{n} C(s_j)) \div (n - i + 1),$$

onde ar é o tempo de chegada da subtarefa. A desvantagem desta técnica é que ela faz a divisão da folga da tarefa distribuída entre suas subtarefas sem considerar os tempos de computação destas. Assim, uma subtarefa com tempo de computação maior recebe o mesmo valor de folga que outra subtarefa com tempo de computação menor, o que é inadequado porque as subtarefas deveriam receber uma folga proporcional ao seu tempo de computação.

Com o intuito de melhorar a técnica EQS, a técnica *Equal Flexibility* (EQF) [6] propõe que a folga da tarefa distribuída seja dividida entre suas subtarefas na proporção de seus tempos de execução estimados. Neste método é definido o conceito de *flexibilidade*, que é o quociente da folga da tarefa distribuída pelo seu tempo de computação. Com isso, quanto mais flexível uma tarefa distribuída é, menos estritas são suas restrições temporais. Com EQF, apesar das subtarefas de uma mesma tarefa distribuída possuírem diferentes valores de

folga, elas possuem a mesma flexibilidade. A técnica EQF é definida como:

$$dl(s_i) = ar(s_i) + C(s_i) + (d(TD) - ar(s_i) - \sum_{j=1}^{m} C(s_j)) \times (\frac{C(s_i)}{\sum_{j=1}^{m} C(s_j)}),$$

Nesta técnica, cada subtarefa recebe folga suficiente para executar, já que a definição da folga de uma subtarefa considera, de forma proporcional, o tempo de computação desta em relação às demais.

Nas simulações realizadas em [6] para comparar o desempenho deste conjunto de algoritmos, a métrica utilizada foi o número de deadlines perdidos. O sistema é composto por tarefas locais (que executam apenas em um nodo do sistema) e globais (que executam em vários nodos do sistema), ambas com deadlines não-críticos. Estes dois tipos de tarefas foram comparados, onde tarefas globais sempre perdem mais deadlines que as locais, independente do algoritmo usado. Observou-se que nos casos onde a carga do sistema é moderada (nem muito alta nem muito baixa) o desempenho do algoritmo ED fica entre UD e EQF, e que EQS é muito parecido com EQF (nos casos em que eles diferem, EQF é superior).

Sun propõe em [8] o algoritmo de particionamento de deadlines chamado *Normalized Proportional Deadline Monotonic* (NPDM). O algoritmo divide o deadline fim a fim da tarefa entre suas subtarefas proporcionalmente aos seus tempos de execução e considera a utilização do processador para efeitos de cálculo. NPDM é proposto como parte de um framework de escalonamento distribuído para tarefas periódicas. O desempenho daquele algoritmo foi comparado com o EQF (proposto em [6]) e os resultados mostraram que ambos apresentam resultados muito próximos.

Os métodos de particionamento de deadlines podem ser classificados em dois grupos, onde o primeiro utiliza os parâmetros e restrições temporais apenas da tarefa que terá seu deadline particionado (algoritmos locais) e o segundo grupo utiliza dados a respeito da carga do sistema, isto é, parâmetros e restrições temporais das tarefas presentes em cada nodo do sistema (algoritmos globais). Usualmente, os métodos locais são executados estaticamente (em tempo de projeto), e os métodos globais são executados dinamicamente (em tempo de execução) já que exigem informações acerca das demais tarefas presentes no sistema. Os algoritmos propostos por Kao e Garcia-Molina em [6] são exemplos de algoritmos locais. Já os algoritmos propostos por Sun [8] e Marinca [10] são do tipo global.

A vantagem dos algoritmos locais é a simplicidade dos cálculos no particionamento do deadline, que consideram apenas parâmetros e restrições temporais da tarefa em questão. A vantagem dos algoritmos globais é que eles podem apresentar melhor desempenho já que consideram, para a definição dos deadlines locais das tarefas, informações sobre a carga do sistema. Entretanto, conforme os resultados apresentados em [8], o desempenho de ambos os algoritmos é muito próximo, o que justifica a utilização de algoritmos locais para o particionamento do deadline fim a fim de tarefas distribuídas.

2.5 Conclusão

Este capítulo apresentou os principais conceitos sobre sistemas de tempo real. Em especial, descrevemos sobre políticas de escalonamento dirigidas por prioridades, com ênfase para a política de prioridade fixa Taxa Monotônica e para a política de prioridade dinâmica EDF. A escolha destas políticas de escalonamento deve-se à sua importância na literatura e porque elas são utilizadas na arquitetura de sistema proposta nesta tese.

Servidor de Aperiódicas é uma técnica bastante conhecida para escalonamento de um conjunto híbrido de tarefas. Descrevemos as principais técnicas para políticas de escalonamento de prioridade fixa, como *Sporadic Server*. Esta técnica apresenta melhores resultados que as demais porque fornece atendimento imediato de requisições aperiódicas e suas regras de restauração da capacidade são mais eficientes.

No âmbito dos sistemas tempo real distribuídos, apresentamos trabalhos que tratam sobre o particionamento de deadlines de tarefas fim a fim. Entre os algoritmos locais, destacamos o *Equal Flexibility* (EQF) pela sua facilidade de implementação e bons resultados apresentados. Estes aspectos motivaram a escolha do EQF como algoritmo de particionamento de deadline na arquitetura de sistema proposta nesta tese.

A literatura sobre algoritmos de particionamento de deadlines é extensa e abrangente. Ela inclui propostas para sistemas de banco de dados distribuídos, para sistemas que utilizam o conceito de *workflows* e propostas no contexto de sistemas tempo real críticos. O anexo A desta tese contém uma descrição detalhada desses trabalhos.

Capítulo 3

Threads Distribuídas

3.1 Introdução

Para alcançar previsibilidade em um sistema distribuído de tempo real é necessário um modelo de execução fim a fim. Neste modelo, as restrições temporais devem ser usadas no gerenciamento de recursos (escalonamento) de forma consistente em cada nodo envolvido no processamento distribuído.

Neste contexto, o conceito de threads distribuídas pode ser utilizado como uma abstração central. Basicamente, uma thread distribuída é uma entidade escalonável que pode transpor nodos, conduzindo seu contexto de escalonamento (restrições temporais) entre as instâncias de escalonamento naqueles nodos [11].

Este capítulo apresenta os principais conceitos e trabalhos relacionados a threads distribuídas. Noções sobre esta abstração são descritos na seção 3.2. Na seqüência, a visão inicial de threads distribuídas, proposta no sistema operacional tempo real *Alpha* [11], é descrita na seção 3.3. As threads distribuídas adotadas no *Real-Time CORBA 1.2* [12], bem como seus mecanismos e interfaces, estão na seção 3.4. A penúltima seção deste capítulo (3.5) trata sobre implementações de threads distribuídas na plataforma de programação Java, onde elas são implementadas no topo das máquinas virtuais cooperantes de um sistema distribuído. As conclusões (seção 3.6) finalizam este capítulo.

3.2 Principais Conceitos

Thread Distribuída é a abstração de um fluxo de controle fim a fim que se estende e se retrai (retorna) através das instâncias dos objetos distribuídos via invocações de métodos remotos. Cada thread distribuída possui um identificador único conhecido em todo sistema [11]. Thread distribuída é uma forma de implementar tarefas distribuídas.

Threads distribuídas se assemelham às threads convencionais no sentido que ambas são uma abstração de execução seqüencial. A diferença entre elas reside no fato que, ao contrário das threads convencionais, as quais estão confinadas a um único espaço de endereçamento, as threads distribuídas realizam execuções seqüenciais em métodos de objetos que podem residir em diferentes nodos físicos, transpondo-os de maneira transparente (figura 3.1).

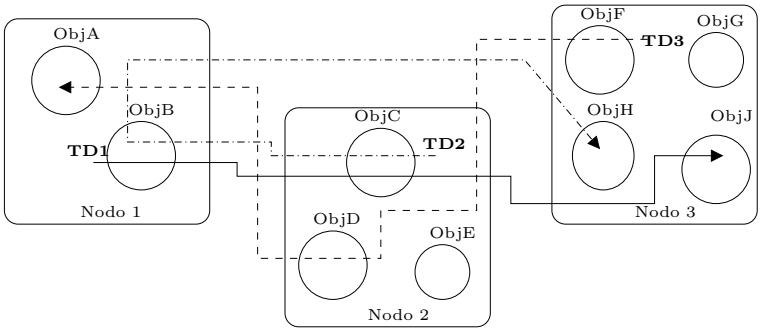

Figura 3.1: Modelo de Threads Distribuídas.

A abstração thread distribuída pode ser implementada como parte do sistema operacional (ex. sistema Alpha), ou como parte do middleware (ex. *Real-Time CORBA 1.2*) ou ainda como parte da linguagem de programação (ex. propostas existentes em Java). Usualmente uma thread distribuída é implementada através da concatenação de threads locais (figura 3.2) [13].

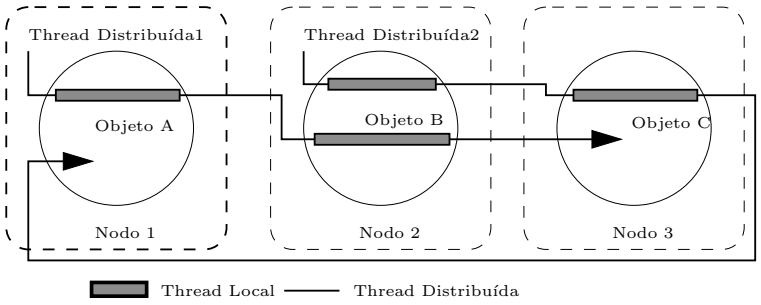

Figura 3.2: Implementação de uma Thread Distribuída.

Todos os nodos que hospedam parte da execução de uma thread distribuída são denominados *nodos segmentos*. Uma thread distribuída, em qualquer ponto 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*. O nodo no qual a thread distribuída é criada é denominado *nodo origem*.

O fluxo de controle de uma thread distribuída pode ser bifurcado, criando ou acordando outras threads distribuídas. Um programa pode consistir de múltiplas threads distribuídas executando concorrentemente e assincronamente.

Sempre que uma thread distribuída transpõe um nodo, ela carrega os parâmetros e outros atributos da computação que ela representa. Esses atributos podem ser modificados e acumulados, de maneira aninhada, conforme ela executa operações dentro de objetos distribuídos [11]. Quando uma thread distribuída inicia sua execução em um nodo do sistema, este é escalonado segundo a política de escalonamento local. Um modelo de escalonamento fim a fim coerente deve manter a mesma política de escalonamento em cada nodo segmento da thread distribuída. Um exemplo seria o de threads distribuídas conduzindo valores de deadline como restrição temporal, e cada nodo segmento implementando uma política EDF. Dessa forma, um modelo flexível de threads distribuídas deve permitir que o programador da aplicação selecione e/ou instale sua política de escalonamento em cada nodo que fará parte de sua aplicação.

Qualquer operação remota na thread distribuída afetará um ou mais nodos que atualmente hospedam a execução da thread distribuída. Um exemplo é a ocorrência de uma exceção síncrona no nodo cabeça da thread, ou uma exceção assíncrona em algum nodo segmento dela. Nesse caso, o fluxo de execução normal da thread distribuída é interrompido, e a exceção tratada pela thread distribuída (threads distribuídas sempre manipulam suas próprias exceções, preservando a correspondência entre ela e a computação que elas representam). Se não existir um tratador no nodo cabeça, essa exceção é propagada para trás, até um nodo que possua um tratador adequado ao tipo de exceção ocorrida.

3.3 Threads Distribuídas no Sistema Operacional Distribuído Tempo Real *Alpha*

O conceito de threads distribuídas foi introduzido no contexto do sistema operacional tempo real distribuído Alpha [11], formando a base do modelo de programação do kernel desse sistema.

Alguns dos conceitos apresentados na seção 3.2, embora não façam parte do sistema Alpha, foram fortemente influenciados pelas idéias propostas nesse sistema. Threads distribuídas fazem parte do modelo de programação do kernel do Alpha. Estas percorrem os nodos físicos do sistema distribuído, carregando atributos tempo real e outros atributos da computação que elas representam, com o objetivo de facilitar o gerenciamento de recursos do sistema [11].

Threads distribuídas do sistema Alpha são a unidade de escalonamento daquele sistema e são preemptáveis. A estratégia de escalonamento de recursos do Alpha, de acordo com as restrições temporais, é baseada no modelo *Utility Accrual*[11], o qual avalia uma tarefa de acordo com a utilidade que sua conclusão traz ao sistema. Quando o escalonador do sistema detecta que existe uma thread distribuída pronta para executar e que sua execução provavelmente aumentará o benefício obtido pelo sistema em relação àquela que está atualmente executando, o sistema pode preemptar a thread distribuída que está executando em favor daquela que está pronta. Os custos de preempção e o tempo de conclusão esperado da thread distribuída que assumirá o processador são levados em consideração na tomada dessa decisão.

3.3.1 Tratamento de Exceções no Sistema Alpha

Threads distribuídas estão sujeitas a exceções, que podem ser síncronas (ex. instruções de teste de hardware) ou assíncronas (ex. expiração de uma restrição de tempo real). O kernel do sistema Alpha fornece mecanismos para criar tratadores de exceções através de blocos de exceção. Esses blocos são delimitados pelas operações *begin* e *end* e podem ser aninhados. A operação *begin* abre um escopo de execução onde são definidos os tratadores de exceção que serão usados para os tipos de exceções especificados naquele bloco enquanto a thread distribuída está executando dentro dele. A operação *end* fecha o bloco de exceção

mais interno.

Quando ocorre uma exceção de um tipo particular, o controle da thread distribuída é movido para o tratador definido pelo bloco de exceção mais interno. Mesmo que a thread distribuída esteja executando dentro do kernel do sistema (ou esteja em uma operação bloqueante), uma exceção pode forçá-la a sair do kernel para executar o tratador apropriado.

Para manter a correspondência entre uma thread distribuída e a computação que ela representa, cada thread distribuída manipula suas próprias exceções. Quando ocorre uma exceção, os atributos da thread distribuída são ajustados pelo kernel para que cada tratador de exceções seja executado com atributos apropriados pelo bloco de exceção naquele ponto. Isso garante, entre outras coisas, que os parâmetros de escalonamento apropriados sejam associados com o tratamento de exceções [11].

3.4 Threads Distribuídas no Real-Time CORBA 1.2

A OMG (*Object Management Group*) é um consórcio internacional de empresas, sem fins lucrativos, criado na década de 80, que tem por objetivo definir padrões na área da computação com objetos distribuídos [12].

O CORBA (*Commom Object Request Broker Architecture*) foi definido pela OMG e permite o desenvolvimento de aplicações em várias linguagens, e também a integração destas com aplicações legadas. Ele é baseado em componentes (objetos) que podem descobrir um ao outro e se comunicar através de um barramento de objetos (um *middleware*), além de oferecer vários outros serviços.

Em 1999, a OMG estendeu o CORBA com interfaces para desenvolvimento de aplicações de tempo real. Nessas interfaces, que ficaram conhecidas como *Real-time CORBA 1.0* [12], já existia o conceito de restrição temporal que se propagava em cada nodo do caminho da invocação, e que era usado de forma coerente no escalonamento em cada nodo. Contudo, essas especificações previam apenas valores de prioridades, conhecidos como prioridades CORBA, usados como mecanismos para suportar restrições temporais fim a fim. A especificação *Real-time CORBA 1.2* [12] veio com objetivo de generalizar as interfaces Real-time CORBA 1.0, permitindo, dentre outras coisas, que os programadores das aplicações especifiquem suas restrições temporais fim a fim, além de permitir que escalonadores (políticas de escalonamento)

sejam instalados em nodos da aplicação distribuída.

A especificação Real-time CORBA 1.2 define Thread Distribuída como a abstração fundamental para a execução de aplicações. Estas threads são as entidades escalonáveis do sistema distribuído. Assim como no sistema Alpha, uma thread distribuída tem um identificador único conhecido em todo sistema distribuído e somente um ponto de execução em um dado instante de tempo. Além disso, uma thread distribuída pode ser preemptada por outras de maior prioridade.

Dentro de cada nodo do sistema distribuído, o fluxo de controle da thread distribuída é mapeado para a execução de uma thread local, fornecida pelo sistema operacional. De forma similar a uma thread local que executa invocações de operações locais aninhadas, uma thread distribuída executa uma seqüência de código consistindo de invocações de operações locais e/ou distribuídas aninhadas. Assim, ela pode estender e retrair seu ponto de execução (através de invocações e retornos CORBA) entre operações em instâncias de objetos, que podem residir em diferentes nodos do sistema distribuído.

Nas interfaces da especificação Real-time CORBA 1.2, a operação *spawn()* permite criar uma nova thread distribuída. Esta thread é composta por *segmentos de escalonamento*, os quais representam uma sequência de fluxo de controle onde um conjunto de parâmetros de escalonamento pode ser associado. Estes parâmetros de escalonamento podem ser, por exemplo, prioridade, importância e restrições temporais (deadline). As operações *begin_scheduling_segment*(BSS) e *end_scheduling_segment* (ESS) delimitam um segmento de escalonamento no código da aplicação (figura 3.3). É possível também atualizar dinamicamente um parâmetro de escalonamento associado à thread distribuída através da operação *update_scheduling_segment* (USS). Essas operações trazem grande flexibilidade para as aplicações, permitindo mudanças dinâmicas nas restrições temporais (escalonamento dinâmico).

Os *Interceptadores Portáveis* que aparecem na figura 3.3 são pontos de monitoramento e controle de uma seqüência de requisição/resposta entre objetos distribuídos [14]. No contexto do Real-Time CORBA 1.2, uma seqüência de requisição/resposta se refere a uma thread distribuída. A principal função de um interceptador portável CORBA é habilitar serviços da camada ORB (*Object Request Broker*) a transferir informações de contexto entre clientes e servidores.

Uma thread distribuída que executa fora do contexto de um segmento de escalonamento

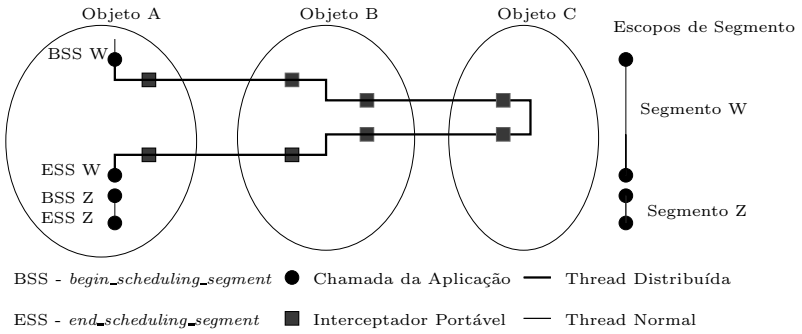

Figura 3.3: Segmentos de escalonamento do Real-time CORBA 1.2.

não tem parâmetros de escalonamento associados a ela, sendo escalonada pela política nativa do sistema operacional (normalmente baseada em prioridades).

Dentro de uma thread distribuída, segmentos de escalonamento podem ser seqüenciais e/ou aninhados. Um aninhamento cria escopos de escalonamento. A Figura 3.4 ilustra um segmento aninhado. Nesse caso, o segmento X está aninhado dentro do segmento W. No ponto onde o segmento X inicia, o contexto de escalonamento do segmento W é empilhado, e os parâmetros de escalonamento do segmento X são usados para a thread distribuída. Quando o segmento X termina, a thread distribuída volta a considerar os parâmetros de escalonamento do segmento W.

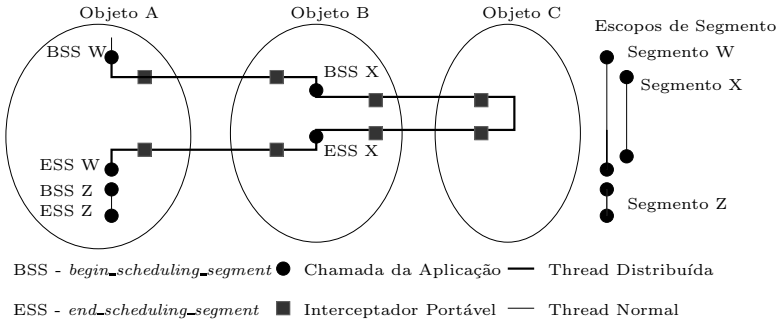

Figura 3.4: Thread distribuída com segmentos aninhados.

O parâmetro de escalonamento criado em uma instância de objeto deve ser considerado em outras instâncias de objetos conforme a thread distribuída os transpõe. Apesar disso, uma thread distribuída que está executando em uma única instância de objeto pode, em diferentes tempos, ter restrições temporais distintas.

Para implementar políticas de escalonamento dinâmico, toda instância do escalonador deve monitorar as restrições temporais de cada thread distribuída que está atualmente execu-

tando em seu nodo. Apesar das especificações do Real-Time CORBA 1.2 determinarem que isso será feito através de Interceptadores Portáveis do CORBA, essas especificidades são deixadas para as implementações de cada escalonador. Alguns pontos de escalonamento, onde o escalonador precisaria atuar, são apontados na especificação [12]: na criação e no término de uma thread distribuída; no início, término ou atualização de um segmento de escalonamento; em cada ponto de invocação ou retorno de invocação; e no bloqueio e liberação de recursos (ex. operações com mutexes).

Para o disparo de exceções, a especificação Real-Time CORBA 1.2 define as seguintes interfaces:

- *CORBA::SCHEDULE_FAILURE* - esta exceção é disparada quando a thread distribuída viola algum de seus parâmetros de escalonamento. A perda de deadline é um exemplo de violação dos parâmetros de escalonamento.

- *CORBA::THREAD_CANCELLED* - esta exceção indica que a thread distribuída foi cancelada. Uma thread distribuída pode cancelar a execução de outra. Esta exceção é lançada no nodo cabeça subseqüente da thread distribuída cancelada.

Esta especificação não traz interfaces relacionadas à propagação de exceção no caminho da invocação. Apenas cita que exceções devem ser propagadas na thread distribuída, entre os nodos segmentos, até alcançar o nodo cabeça, notificando o seu tratador de exceção. A propagação e a notificação de exceções devem ser realizadas assim que possível, se a thread distribuída está executando ou assim que ela se torne a mais elegível para executar.

3.4.1 Trabalhos sobre Threads Distribuídas do *Real-time CORBA 1.2*

A literatura apresenta trabalhos sobre diferentes aspectos relacionados a threads distribuídas no contexto do Real-Time CORBA 1.2 [15] [16]. A seguir são descritos alguns destes trabalhos.

Em [15], os autores propõem a utilização do algoritmo de escalonamento *GUS* (*Generic Utility Scheduling*) para escalonar threads distribuídas periódicas no contexto do middleware *Tempus*. Este algoritmo de escalonamento utiliza critérios relacionados com a utilidade acumulada (*Utility Accrual*) [11] de uma tarefa para escaloná-la. O Real-Time CORBA 1.2 oferece interfaces para disciplinas de escalonamento que utilizam este tipo de critério, além

de outras como RM (*Rate Monotonic*), EDF (*Earliest Deadline First*) e LLF (*Least Laxity First*).

Para o escalonamento de tarefas, o algoritmo *GUS* utiliza valores gerados por uma *Time/Utility Function (TUF)* [11]. Conforme descrito na seção 3.3, uma TUF especifica para o sistema a utilidade de uma tarefa como função do seu tempo de conclusão. Os parâmetros de escalonamento, incluindo os valores gerados pela TUF, são propagados conforme a thread distribuída transpõe nodos do sistema. Instâncias locais do algoritmo de escalonamento *GUS* usam estes parâmetros propagados para construir escalas locais de forma a maximizar a utilidade acumulada localmente da thread distribuída e com isso obter uma utilidade acumulada globalmente ótima.

As avaliações experimentais apresentadas em [15] mostram que os principais componentes que contribuem para o *overhead* do Tempus são: o serviço de nomes, o software do lado cliente, a rede de comunicação e o software do lado servidor. Destes componentes, observou-se que a rede de comunicação apresenta o maior *overhead*. Entretanto, os valores médios do *overhead* são pequenos (na ordem de poucos milisegundos) quando comparados com restrições temporais de aplicações, que são na ordem de centenas de milisegundos ou mesmo segundos.

Comparações de desempenho também foram realizadas entre o algoritmo GUS e políticas de escalonamento bem conhecidas como RM e EDF. Três tipos de TUFs foram utilizadas com diferentes conjuntos de threads distribuídas. As avaliações utilizaram dois cenários: com e sem compartilhamento de recursos. Em ambos os cenários o algoritmo *GUS* apresentou melhores resultados que os demais. Apesar do algoritmo GUS cumprir um menor número de deadlines que os algoritmos RM e EDF, ele alcança uma utilidade acumulada maior para o sistema que os demais algoritmos.

Em [16], são realizadas comparações de desempenho entre threads distribuídas e Canais de Eventos (*Event Channel*) do CORBA. O protocolo de sincronização inter-processador *Release Guard* [8] é integrado nestes dois modelos de comunicação fim a fim, para prover neles maior previsibilidade. Este protocolo de sincronização garante que o tempo entre duas liberações consecutivas da mesma subtarefa não será menor que o período. Isto permite realizar análise de escalonabilidade do sistema assumindo as subtarefas como periódicas.

A implementação do protocolo *Release Guard* foi realizada sobre o TAO (*The ACE*

ORB)[17], um ORB C++ que segue a maioria das especificações sobre características e serviços do CORBA 3.x e projetado para aplicações de tempo real. Para o escalonamento das subtarefas, foi utilizado um algoritmo de escalonamento não preemptivo. Três tipos de topologias de comunicação foram utilizadas: um canal seqüencial de tarefas, um grafo de tarefas estático e outro dinâmico, onde algumas subtarefas são executadas em mais de um processador, garantindo a execução destas caso ocorra algum problema que impossibilite executá-las em um dos processadores do sistema.

Utilizando as topologias de comunicação especificadas, foram realizados três testes distintos, e cada um deles foi implementado com threads distribuídas e canais de eventos. No terceiro teste, o sistema operou alternando entre dois grafos de tarefas, estático e dinâmico.

Nos experimentos realizados em [16] foi observado que o uso do protocolo *Release Guard* aumenta a previsibilidade do sistema distribuído. Da perspectiva de desempenho, os canais de eventos são mais eficientes para aplicações com topologias de comunicação estáticas (canal seqüencial de tarefas e grafo estático de tarefas) e as threads distribuídas são mais adequadas para aplicações que utilizam topologias de comunicação dinâmicas, como o grafo dinâmico de tarefas.

3.5 Threads Distribuídas no Java

Para prover as funcionalidades de uma plataforma de objetos distribuídos, o Java disponibiliza o serviço RMI (*Remote Method Invocation*) [18] que oferece suporte para a implementação de invocações de métodos remotos. Com o RMI uma aplicação pode exportar seus objetos, os quais podem ser remotamente referenciados e seus métodos podem ser invocados por uma aplicação que executa em diferentes máquinas virtuais Java (*JVM - Java Virtual Machine*).

O RMI incorpora, naturalmente, o conceito de vários fluxos de controle locais que colaboram em um ambiente distribuído na medida em que a aplicação não fica restrita ao espaço de endereçamento de uma única JVM. Entretanto, o RMI não fornece suporte transparente para o reconhecimento desses fluxos locais cooperantes como um único fluxo que representa a mesma computação.

Alguns trabalhos na literatura tratam de questões que relacionam threads distribuídas

e RMI [19, 13, 20]. Estes trabalhos descrevem as limitações do RMI quando utilizado em um ambiente multithread, em especial, a inabilidade de lidar com métodos e blocos de código sincronizados em Java. Esses trabalhos usam a abstração threads distribuídas como base nas soluções propostas para tais limitações e são apresentados na próxima seção. Estes três trabalhos, entretanto, não consideram threads distribuídas com restrições temporais.

Os trabalhos [21, 22] levam em conta restrições temporais de threads distribuídas. O primeiro apresenta uma arquitetura de sistema que oferece suporte para threads distribuídas aperiódicas. O segundo tem o foco sobre um protocolo que permite o lançamento e tratamento de exceções de threads distribuídas. Estes trabalhos são descritos na seção 3.5.3.

3.5.1 Implementação de Threads Distribuídas pela Transformação dos Bytecodes de Aplicações Java

Para implementar um suporte transparente para o reconhecimento de fluxos locais cooperantes como um único fluxo que representa a mesma computação, isto é, uma thread distribuída, uma ferramenta chamada Transformador DTI foi proposta em [19]. Esta ferramenta transforma os bytecodes de programas Java inserindo um identificador para a thread distribuída que será reconhecido em cada nodo que ela executar.

Este identificador é definido no momento da criação da thread distribuída. Este comportamento é encapsulado dentro da classe que implementa threads distribuídas. A propagação do identificador de uma thread distribuída é de responsabilidade do Transformador DTI que estende a assinatura de cada método com um argumento adicional (o identificador da thread distribuída). Todos os métodos invocados dentro do corpo de um método também têm sua assinatura estendida com este argumento adicional. O Transformador DTI encapsula uma operação que permite que a aplicação verifique qual thread distribuída está executando em um dado momento.

A estratégia de implementação usada em [19] foi escolhida por três razões. Primeiro, ela estende a linguagem Java de forma transparente ao usuário (ex. através de um carregador de classes personalizado que executa a transformação dos bytecodes em tempo de carga). Segundo, a noção de identidade de threads distribuídas pode ser adicionada à JVM sem modificá-la, tornando esse mecanismo portável para qualquer sistema que possua uma JVM instalada. Terceiro, essa estratégia permite integração dinâmica com o RMI.

O trabalho apresentado em [20] segue esta mesma linha de pesquisa, isto é, propõe instrumentação nos bytecodes de aplicações Java. Entretanto, em vez de reescrever todos os métodos da aplicação cliente (como proposto em [19]) inserindo um argumento que representa o identificador da thread distribuída, a proposta descrita em [20] realiza a transformação apenas dos bytecodes dos *stubs RMI* gerados.

Esta técnica surgiu da observação de que quase todos os mecanismos de middleware estilo RPC (*Remote Procedure Call*) precisam gerar *stubs* para os métodos remotamente invocáveis. Com a transformação destes *stubs* apenas quando são gerados, o identificador da thread distribuída é propagado para todas as invocações remotas, sem sobrecarregar invocações locais. Segundo os autores, esta técnica tem um *overhead* menor que o método descrito em [19].

3.5.2 Implementação de Threads Distribuídas com Alterações no RMI

Em [13], os autores descrevem três limitações relacionadas ao RMI. A primeira delas se refere à reentrância de sincronização, quando um objeto possui mais de um método sincronizado. Um método sincronizado garante acesso exclusivo a uma thread. Cada objeto possui um monitor associado a ele, o qual controla o acesso de threads aos métodos sincronizados pertencentes ao objeto. A seguinte situação demonstra o problema (figura 3.5).

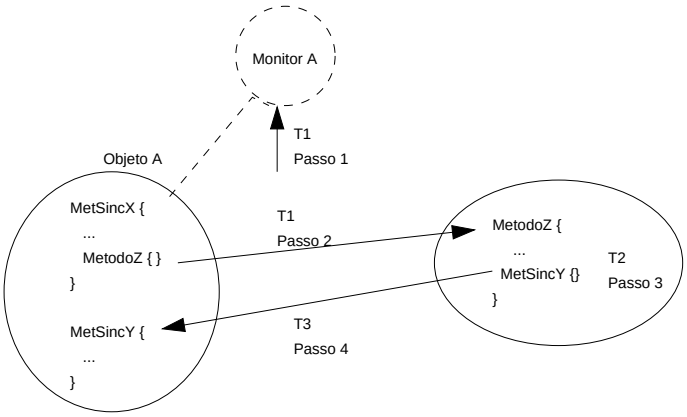

Figura 3.5: Reentrância de Sincronização.

Os métodos *MetSincX* e *MetSincY* são métodos sincronizados de um mesmo objeto. A thread local T1, que representa uma thread distribuída, inicia a execução de *MetSincX*, e obtém o lock sobre o objeto correspondente (*Passo 1* na figura3.5). Em seguida, essa

thread distribuída invoca um método remoto, o qual é atendido pela thread local T2 que representa a thread distribuída naquele nodo (*Passo 2*). Dentro desse método remoto, a thread distribuída invoca o método sincronizado *MetSincY*, o qual é sincronizado sobre o mesmo objeto de *MetSincX*. Uma nova thread local (T3) atende essa invocação, porém fica bloqueada porque o objeto monitor correspondente já está alocado para a thread T1 (que executa *MetSincX*)(*Passo 3*). A situação de deadlock ocorre porque T3 está bloqueada esperando obter o lock sobre o objeto correspondente, porém este está alocado para T1 que está bloqueada a espera do retorno de T3.

A solução proposta em [13] sugere o uso da mesma thread local, em cada nodo do sistema distribuído, para representar uma thread distribuída. Com isso, todas as invocações remotas que ocorrerem em um dado nodo e que sejam da mesma thread distribuída serão atendidas sempre pela mesma thread local.

O segundo problema descrito em [13] está relacionado com a chamada de métodos remotos que estejam dentro de um bloco de código sincronizado. No RMI, os métodos remotos são acessados por classes especiais, chamadas stubs, que servem como um proxy para acessar a implementação correspondente. O proxy encaminha todas as chamadas para um objeto servidor que contém a implementação desses métodos. O RMI não permite que um método remoto seja declarado dentro de um bloco sincronizado porque quando uma thread executa um bloco sincronizado em um objeto proxy, o lock desse objeto é alocado. Isso permite que diferentes threads, que usam diferentes objetos proxy para o mesmo objeto remoto, possam executar blocos sincronizados sobre essas referências concorrentemente.

Para transpor essa limitação, os autores propõem que o monitor remoto do objeto servidor seja adquirido para que o bloco sincronizado local seja executado, em vez de adquirir apenas o lock do objeto proxy. Para atender essa solução foi definida uma API com as operações *rmiAcquire* e *rmiRelease* guardadas por um bloco *try/finally*. A aquisição do monitor remoto é possível porque as operações *rmiAcquire* e *rmiRelease* são implementadas para disparar uma invocação de método remoto especial com capacidade de envio de um "retorno antecipado".

No lado cliente, *rmiAcquire* é disparado dando início à invocação remota. Então, um bloco sincronizado no objeto remoto é iniciado e o monitor remoto correspondente é adquirido. Depois disso, um "retorno antecipado"é emitido de volta para o cliente e recebido por *rmi-*

Acquire. Com isso, o cliente adquiriu efetivamente o monitor remoto. No objeto remoto, a thread que está atendendo essa invocação remota permanece dentro do bloco sincronizado esperando pela mensagem *rmiRelease*. No lado cliente, o bloco sincronizado é processado até a sua conclusão quando então a operação *rmiRelease* é enviada para o objeto remoto. Esta operação instrui a thread a deixar o bloco sincronizado, liberar o monitor remoto e finalmente retornar. O retorno dessa operação informa ao cliente que o monitor remoto está liberado.

A aquisição do monitor remoto é possível somente em objetos que estejam localizados sobre máquinas virtuais diferentes. O Java não oferece suporte para "retornos antecipados"de métodos e por isso não existe simulação local para aquisição do monitor remoto. A solução para essa falta de transparência de localização é alcançada através da transformação do código fonte. O código resultante primeiro verifica se o objeto em questão é um objeto remoto e se ele realmente está localizado remotamente. Se este for o caso, a aquisição do monitor remoto é executada da forma descrita anteriormente. Senão, uma sincronização Java local é executada. A sincronização sobre um objeto remoto que reside sobre uma máquina virtual local se dá pela aquisição da referência do objeto que contém a implementação ao invés de adquirir o proxy usado para acesso pela aplicação. A terceira limitação descrita naquele trabalho se refere ao mecanismo de interrupção de threads distribuídas. Esse mecanismo funciona bem para threads normais do Java, mas não para threads distribuídas porque um sinal de interrupção enviado para um segmento que representa uma thread distribuída, mas que está atualmente inativo, não alcança o segmento cabeça e assim não será tratado na thread local antes do retorno da invocação remota.

Na solução proposta em [13], uma interrupção enviada para um segmento local inativo da thread distribuída deve ser encaminhado até o segmento cabeça, onde a interrupção pode ser tratada. Como existe uma única thread local por nodo que representa uma thread distribuída, um segmento inativo pode receber uma interrupção e encaminhá-la em uma invocação fora de ordem junto com uma chamada de método remoto pendente. Nesta solução pode ocorrer uma condição de corrida quando acontece o retorno do método (assim o segmento local antes inativo torna-se o segmento cabeça) e a interrupção encaminhada se sobrepõem. Nesse caso, a interrupção remota é descartada.

Em [13] as soluções para as limitações encontradas no RMI foram implementadas e avaliadas através de *benchmarks*. Os *benchmarks* foram executados em um cluster com 16

nodos conectados com *Fast Ethernet* e *Myrinet* (utilizando o software *ParaStation*). O RMI permite a utilização de TCP/IP apenas. Por isso, foi utilizado o KaRMI, que permite a troca da camada de transporte. O KaRMI permite usar Fast Ethernet sobre TCP/IP ou *Myrinet* sobre o *ParaStation*. Três fontes de *overhead* foram encontradas, além da latência normal de rede. Duas dessas fontes são decorrentes das decisões de implementação. Sobre a camada TCP/IP, foi verificado que os benefícios de um ambiente de threads distribuídas completamente transparentes podem ser alcançados incorrendo em um *overhead* de cerca de 30%. Este resultado é ainda cerca de 40% mais rápido que a implementação RMI normal, que não gerencia o conceito de threads distribuídas. Sobre a camada *ParaStation*, a latência pura da rede é reduzida para 36 microssegundos, enquanto o *overhead* introduzido pela extensões propostas em [13] permanecem quase o mesmo.

3.5.3 Implementações de Threads Distribuídas com Restrições Temporais

Os trabalhos descritos até o momento consideram threads distribuídas sem requisitos temporais. No contexto de sistemas distribuídos de tempo real, o escalonamento de tarefas considerando restrições temporais é necessário para a correta operação deste tipo de sistema. A comunidade acadêmica tem pesquisado vários aspectos do escalonamento fim a fim, inclusive a utilização de threads distribuídas como entidade escalonável de sistemas distribuídos de tempo real ([12, 23]).

Em [21] é proposta a implementação de threads distribuídas com restrições temporais sobre a Especificação Tempo Real Java (*RTSJ - Real-Time Specification for Java*) [24]. Esta especificação estende a plataforma Java com novos conceitos e mecanismos, criando um ambiente que permite a criação de aplicações tempo real. Como a RTSJ não foi projetada para sistemas distribuídos, o trabalho [21] apresenta uma arquitetura capaz de acomodar a abstração thread distribuída usando a RTSJ como sistema subjacente para suporte de execução local. Esta arquitetura, como mostra a figura 3.6, está presente em cada nodo do sistema distribuído e utiliza interceptadores e servidores de aperiódicas para o escalonamento das threads distribuídas. Os interceptadores são responsáveis por atender threads distribuídas quando estas chegam no nodo e encaminhá-las ao servidor de aperiódicas, que por sua vez realiza o escalonamento dos segmentos locais de threads distribuídas presentes no nodo em questão.

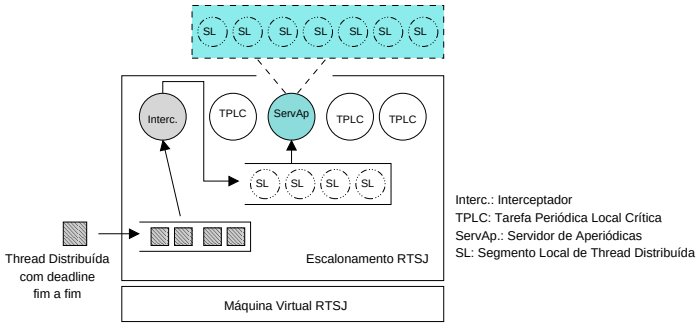

Figura 3.6: Arquitetura de Sistema para Threads Distribuídas Aperiódicas.

A RTSJ não oferece nenhum algoritmo que implemente um servidor de aperiódicas. Entretanto, ela oferece um mecanismo chamado *Processing Group Parameters* que permite associar um objeto a um grupo de objetos escalonáveis, criando um servidor de aperiódicas lógico. Este mecanismo foi utilizado na implementação do servidor de aperiódicas da arquitetura de sistema proposta em [21].

O escalonador padrão da RTSJ utiliza prioridade fixa com preempção. Por esta razão, na implementação da arquitetura proposta foi utilizado o algoritmo RM (*Rate Monotonic*) [4]. Simulações foram realizadas com o objetivo de observar a sensibilidade da arquitetura proposta com relação aos tipos de servidores usados, *Background Server* e *Polling Server*, e com relação às políticas de escalonamento utilizadas na fila do servidor de aperiódicas, FIFO e EDF. A métrica utilizada foi a taxa de deadlines alcançados.

Os resultados obtidos em [21] mostraram que, para aquele contexto simulado, considerando a política de escalonamento EDF, os servidores de aperiódicas *Background* e *Polling Server* apresentam melhores resultados, quando comparados com a política FIFO. As experiências mostraram uma grande sensibilidade da arquitetura proposta com relação a política de escalonamento empregada.

O trabalho descrito em [22] apresenta a implementação de threads distribuídas sobre a DRTSJ (*Distributed Real-time Specification for Java*) [25]. Esta especificação está em desenvolvimento na JSR-50 (*Java Specification Request*) do JCP (*Sun's Java Community Process*). Seu objetivo é definir extensões na RTSJ para ambientes distribuídos, incorporando o conceito de threads distribuídas. Esta especificação está em fase inicial e vários aspectos deste contexto ainda precisam ser estudados [23].

Os autores daquele trabalho apresentam o algoritmo de escalonamento HUA (*Handler-*

assured Utility Accrual) e o protocolo TPR (*Thread Polling with bounded Recovery*) para garantir a integridade da thread distribuída. Através desse algoritmo de escalonamento e desse protocolo, o artigo trata questões relacionadas com falhas na execução de uma thread distribuída. As falhas consideradas são:

- quando a thread distribuída perde seu deadline;
- quando ocorre algum problema em um nodo que o impossibilita de continuar funcionando normalmente.

Se uma thread distribuída perde seu deadline, uma exceção é lançada e tratadores de exceção são disparados em todos os nodos que a hospedam. No caso em que a thread chega em um nodo que apresenta falha, o protocolo TPR envia notificações sobre esta falha para todos os outro nodos que hospedam parte da execução da thread distribuída.

O algoritmo HUA utiliza TUFs [11] para definir as restrições temporais das threads distribuídas e faz a análise de escalonabilidade considerando o tempo de computação da thread distribuída e do seu tratador de exceção. O objetivo é maximizar a utilidade acumulada total das tarefas. Além disso, as operações relacionadas com a recuperação de uma thread distribuída devem ser limitadas. As operações relacionadas com a manutenção dos segmentos locais da thread distribuída ficam a cargo do protocolo TPR, que executa no nodo origem da thread distribuída e dispara mensagens de notificações e reconhecimentos (*ACKs*) aos demais nodos que hospedam parte da execução da thread distribuída. O protocolo especifica *timeouts* para o recebimento de respostas e caso este *timeout* expire, operações de remoção de segmentos locais órfãos são realizadas para manter a consistência dos dados.

3.6 Conclusão

O modelo de programação baseado em threads distribuídas facilita a construção de aplicações, na medida em que simplifica a visão do programador durante o desenvolvimento de programas no contexto distribuído. De certa forma, desde os trabalhos originais sobre RPC (*Remote Procedure Call*) nos anos 80, tem-se buscado esconder do programador as peculiaridades do ambiente distribuído. Threads distribuídas dão um passo nesta direção, ao esconder não somente as trocas de mensagens (algo que RPC já fazia), como também a

existência de diversas threads locais que juntas implementam o conceito de thread distribuída [26].

É possível construir uma aplicação distribuída sem a abstração threads distribuídas. Porém, diversos aspectos teriam que ser tratados explicitamente pelo programador (ex. propagação das restrições temporais nos nodos do sistema distribuído conforme a aplicação transpõe esses nodos).

Com relação ao Real-Time CORBA 1.2, pode-se observar que ele é bastante flexível no que diz respeito ao escalonamento, definindo o conceito de segmentos de escalonamento. Este conceito permite a alteração dinâmica de restrições temporais. Além disso, o Real-Time CORBA 1.2 define facilidades como os *escalonadores conectáveis* que permite a implementação de novos algoritmos.

A implementação da abstração threads distribuídas no Java sem qualquer alteração na JVM e utilizando o RMI como mecanismo de comunicação apresenta um ambiente interessante para a programação distribuída. Entretanto, nesse contexto, o RMI apresenta algumas deficiências. No decorrer deste capítulo, descrevemos os trabalhos [19], [13] e [20] que propõem soluções para tais deficiências, as quais são baseadas no identificador da thread distribuída e em alterações no RMI.

Os trabalhos [21] e [22] tratam de threads distribuídas que possuem restrições temporais. No primeiro, o qual é parte dos primeiros trabalhos desta tese, uma arquitetura de sistema foi proposta para acomodar threads distribuídas do tipo aperiódicas. Esta arquitetura, presente em cada nodo do sistema, utiliza a RTSJ como sistema subjacente local. Foram exploradas políticas de escalonamento para o servidores de aperiódicas e também para as filas desses servidores. No segundo trabalho [22], os autores propõem a implementação de threads distribuídas sobre a DRTSJ. A propagação e o tratamento de exceções é o foco daquele trabalho. É proposto o uso conjunto de um algoritmo de escalonamento e um protocolo que fica hospedado no nodo origem da thread distribuída e é responsável pelo envio e recebimento de mensagens de controle para os segmentos locais de uma thread distribuída.

O estudo da arquitetura proposta em [21] mostrou sua sensibilidade com relação às políticas de escalonamento empregadas e norteou pesquisas mais aprofundadas sobre outros aspectos, como por exemplo, o emprego de algoritmos mais sofisticados no servidor de aperiódicas para o escalonamento dos segmentos locais de threads distribuídas; a busca por

métodos de particionamento de deadlines mais eficientes para serem usados na partição do deadline fim a fim de uma thread distribuída e também a definição de caminhos de execução que uma thread distribuída pode percorrer em um sistema distribuído.

Sobre este último aspecto, notamos que uma thread distribuída, em um ambiente dinâmico, possui caminhos de execução (itinerários) bastante variados e que podem ser probabilisticamente conhecidos antes do início de sua execução. Isto é, os métodos remotos que serão invocados pela thread dependem do estado do sistema e serão conhecidos apenas durante sua execução. Esta observação nos conduziu para a definição dos principais itinerários que uma thread distribuída pode seguir. O detalhamento deste aspecto será apresentado no capítulo seguinte.

Capítulo 4

Arquitetura de Sistema

4.1 Introdução

Nesta tese, a arquitetura de sistema refere-se a definição do suporte necessário para implementação e escalonamento de threads tempo real distribuídas. Neste capítulo serão descritas a abordagem de escalonamento adotada neste trabalho bem como as premissas usadas para fins de escalonamento local e de previsão de perda de deadline.

Inicialmente descrevemos o modelo de tarefas adotado nesta tese (seção 4.2). A seguir, apresentamos a arquitetura de sistema presente em cada nodo, a qual oferece o suporte necessário para a implementação e escalonamento de threads distribuídas. Para tanto, na seção 4.3 definimos o método de escalonamento utilizado, composto pelo particionamento do deadline fim a fim e escalonamento local. A seção 4.4 estabelece as premissas usadas nos mecanismos de previsão de perda de deadline propostos nesta tese. Nas seções 4.5 e 4.6, são descritas, respectivamente, as métricas utilizadas para avaliar a qualidade das previsões e o modelo geral de funcionamento dos mecanismos propostos. A última seção apresenta as conclusões deste capítulo.

4.2 Modelo de Tarefas Adotado

Esta seção apresenta o modelo de tarefas adotado nesta tese e que será utilizado nos capítulos seguintes. São descritos os parâmetros que caracterizam as tarefas tempo real bem

como a arquitetura de sistema que está presente em cada nodo e que atende às tarefas da aplicação distribuída de tempo real.

Neste trabalho consideramos um sistema tempo real distribuído aquele onde todos os nodos do sistema executam uma única aplicação durante um período de tempo. Esta aplicação é composta por tarefas locais e tarefas distribuídas. A abstração threads distribuídas é utilizada para a implementação das tarefas distribuídas.

O sistema distribuído alvo é formado por um conjunto de nodos (ou nós), conectados através de um meio de comunicação. O envio de mensagens entre dois nodos quaisquer é delimitado por um tempo máximo Δ. Para efeito de escalonamento, o tempo para envio de mensagens entre tarefas situadas no mesmo nodo é considerado zero. O protocolo de comunicação (camadas de enlace, rede, transporte e RMI - *Remote Method Invocation*) será executado por um processador auxiliar, não competindo, portanto, com tarefas da aplicação.

Em cada nodo do sistema, existem tarefas periódicas locais com deadlines críticos e threads distribuídas aperiódicas com deadlines firmes. As tarefas periódicas locais são consideradas críticas para a aplicação. Com isso, o escalonamento destas tarefas não deve ser prejudicado pelo escalonamento das threads distribuídas aperiódicas. As threads distribuídas são recorrentes, isto é, podem executar várias vezes no sistema. Como os tempos de chegada das threads distribuídas não são previamente conhecidos, o sistema possui carga dinâmica e podem existir situações de sobrecarga.

Supõe-se que na criação de uma thread distribuída tempo real, um deadline fim a fim e um tempo de execução médio são definidos. Os métodos remotos que a thread distribuída poderá executar na aplicação são conhecidos em tempo de projeto e com base nesta informação, calcula-se uma estimativa do seu tempo de execução. O deadline fim a fim é especificado pela aplicação. Estas restrições temporais são carregadas pela thread distribuída tempo real conforme ela transpõe os nodos do sistema.

Para fins de particionamento do deadline fim a fim e escalonamento local, consideramos neste modelo de tarefas o retorno da thread distribuída ao objeto no qual a chamada remota foi disparada. Após executar um método remoto, a thread distribuída pode retornar ao nodo origem e finalizar sua execução (figura 4.1); ou pode retornar ao nodo origem e realizar novas invocações remotas (figura 4.2) ou pode, ainda, a partir do objeto onde ela se encontra realizar invocações remotas (figura 4.3).

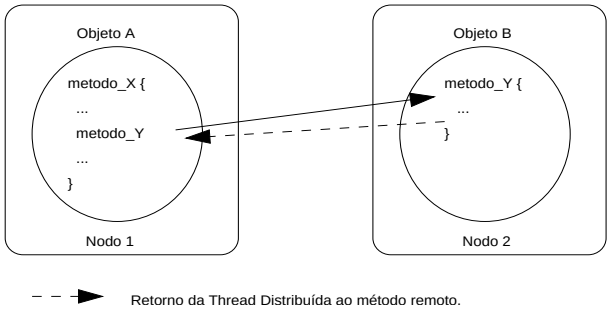

Figura 4.1: Retorno da thread distribuída e finalização da execução.

Threads distribuídas podem incorporar uma natureza autônoma, no sentido que a seqüência de métodos remotos que serão executados por elas depende das variações do ambiente distribuído no qual as threads executam. Se, por exemplo, threads distribuídas são utilizadas na implementação de tarefas de supervisão de uma unidade automatizada de uma fábrica que controla robôs e tornos mecânicos, a thread distribuída será programada para coletar dados a respeito da produção destes equipamentos. Ela visita os nodos que são responsáveis pelo controle destes equipamentos e executa métodos que verificam o estado e capturam informações sobre eles. Como na maior parte do tempo os equipamentos estarão operando normalmente, a thread distribuída irá executar uma determinada seqüência de métodos remotos. Por outro lado, se a thread distribuída detecta problemas em um dos nodos (ou ele se encontra inativo), outra seqüencia de métodos remotos será executada. Em ambas as situações, esta seqüência de métodos remotos executados pode ser vista como um caminho ou itinerário que a thread distribuída irá seguir.

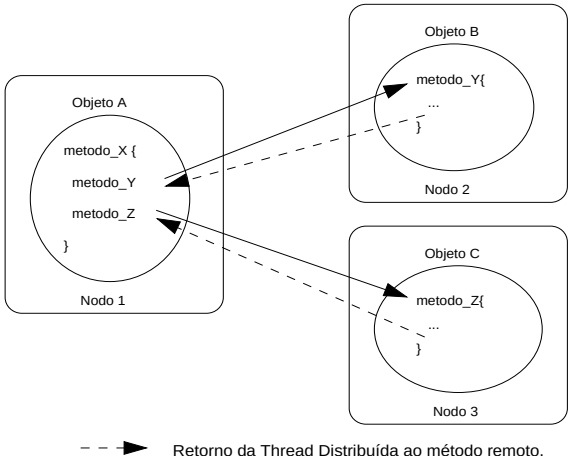

Figura 4.2: Retorno da thread distribuída e nova chamada remota.

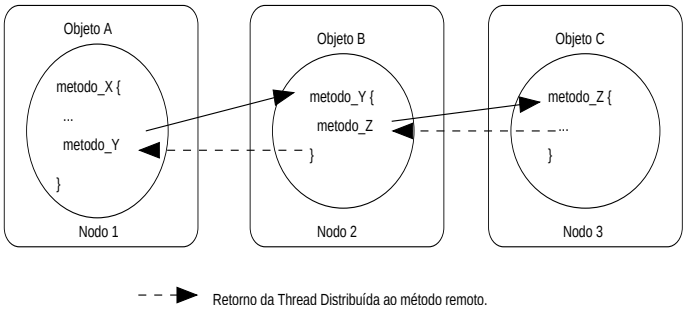

Figura 4.3: Novas invocações antes do retorno da thread distribuída.

4.3 Método de Escalonamento para Threads Distribuídas

Existem vários métodos de escalonamento para tarefas distribuídas presentes na literatura [4, 3]. O método de escalonamento usado nesta tese 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 [8]. O objetivo de escalonamento da arquitetura de sistema proposta é garantir os deadlines das tarefas locais críticas. Ao mesmo tempo, ela deve reduzir o tempo de resposta das threads distribuídas não-críticas para atender os deadlines dos seus segmentos locais e, conseqüentemente, cumprir seu deadline fim a fim.

Neste trabalho realizamos o particionamento do deadline fim a fim de uma thread distribuída definindo, inicialmente, o algoritmo que será usado para este fim. Entre os algoritmos de particionamento estudados [6, 10, 9], utilizamos o *Equal Flexibility* (EQF) [6] pela sua simplicidade de implementação e baixo overhead de execução. Entretanto, o particionamento do deadline fim a fim deve considerar a natureza autônoma de execução de uma thread distribuída, que se altera conforme as variações do sistema distribuído em que ela executa. A seguir tratamos sobre particionamento do deadline fim a fim de threads distribuídas considerando este aspecto e sobre escalonamento local, onde será descrita a arquitetura de sistema adotada neste trabalho.

4.3.1 Particionamento do Deadline Fim a Fim de Threads Distribuídas

A escolha por diferentes caminhos de execução (itinerários) de uma thread distribuída reflete a idéia de execuções condicionais, representadas por estruturas de programação *If-Then*, muito comuns em sistemas de controle e automação. A literatura sobre threads distribuídas e particionamento de deadlines, até o presente momento, aborda este aspecto de

forma superficial. A maioria dos trabalhos considera o caminho de uma tarefa como sendo apenas um pipeline [27, 9, 10]. Quando muito, reconhecem a existência de estruturas de controle condicionais [28, 29, 30], mas justificam que a definição de um algoritmo de particionamento de deadlines para tais estruturas é de difícil modelagem.

Nesta tese consideramos que o particionamento do deadline fim a fim de uma thread distribuída pode ser realizado reconhecendo estruturas condicionais de programação. A definição dos principais itinerários que uma thread distribuída pode percorrer é o primeiro passo em direção ao particionamento do seu deadline fim a fim. Os possíveis itinerários a serem executados por uma thread distribuída são conhecidos em tempo de projeto mas, em cada ativação, o itinerário pode variar em função do estado do sistema distribuído.

Ao final de cada ativação, a thread distribuída atualiza seu histórico armazenando o itinerário percorrido. Como a thread distribuída inicia sua execução sempre no mesmo nodo (nodo origem), é possível manter o histórico nesse nodo. Antes da thread distribuída iniciar sua execução, o histórico é verificado e as informações contidas nele são usadas no particionamento do deadline fim a fim. É possível observar, por exemplo, que alguns itinerários são executados um maior número de vezes que outros. Estes itinerários podem ser usados para particionar o deadline fim a fim da thread distribuída fazendo com que ela alcance um maior número de deadlines. Nesta tese, os itinerários serão representados por grafos direcionados acíclicos, com arcos indicando a seqüência de métodos que podem ser executados pela thread distribuída. Entre todos os itinerários que uma thread distribuída pode executar, é possível identificar os principais, que são:

- Itinerário Maior Número de Saltos: O itinerário *Maior Número de Saltos* verifica, entre todas as possíveis seqüências de execução de métodos remotos de uma thread distribuída, aquela que apresenta o maior número de nodos visitados (saltos). A figura 4.4 mostra nodos contendo os possíveis métodos remotos a serem executados pela thread distribuída e o tempo médio de execução de cada um deles. Neste exemplo, o itinerário com o maior número de saltos é composto pelos nodos 1, 3, 4, 8, 4, 3, 1. O retorno da thread distribuída para o nodo no qual a invocação remota foi realizada será representado por uma seta pontilhada.

Quando a thread distribuída retorna ao objeto onde uma chamada remota foi disparada, ela pode executar alguma computação neste objeto antes de finalizar sua execução, como

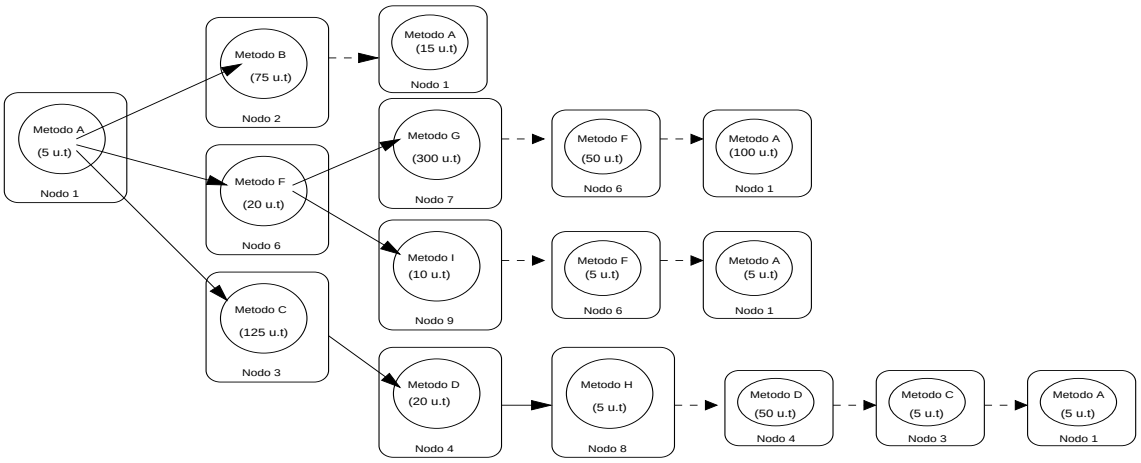

Figura 4.4: Itinerário Maior Número de Saltos com retorno.

mostra a figura 4.4. Por isso, é necessário considerar também o tempo médio de computação gasto no retorno da thread distribuída ao objeto no qual a chamada remota foi disparada.

Conforme aumenta o tamanho do itinerário que a thread distribuída executa (número de métodos remotos), o espaço necessário para ilustrar o seu retorno torna-se muito grande. Por este motivo, neste trabalho, sempre que for possível, o tempo médio de computação do retorno da thread distribuída até o nodo origem será ilustrado em um retângulo pontilhado contendo o somatório dos tempos médios de computação dos métodos que fazem parte deste retorno. Com isso, a figura 4.4 é modificada conforme mostra a figura 4.5

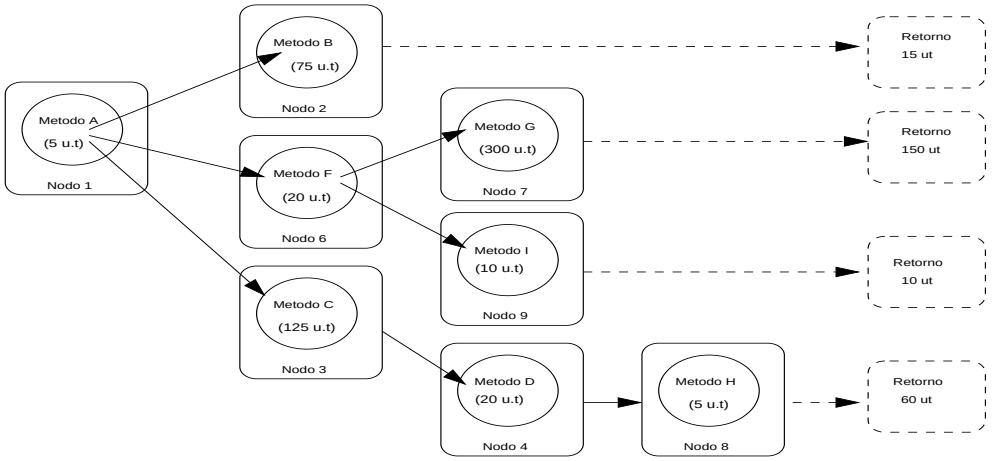

Figura 4.5: Itinerário Maior Número de Saltos com retorno resumido.

Se não for necessário ilustrar os tempos médios de computação dos métodos remotos executados pela thread distribuída, eles serão ocultados das figuras, assim como os métodos que fazem parte do retorno da thread distribuída ao nodo origem. Desta forma, a figura

anterior toma a forma apresentada na figura 4.6.

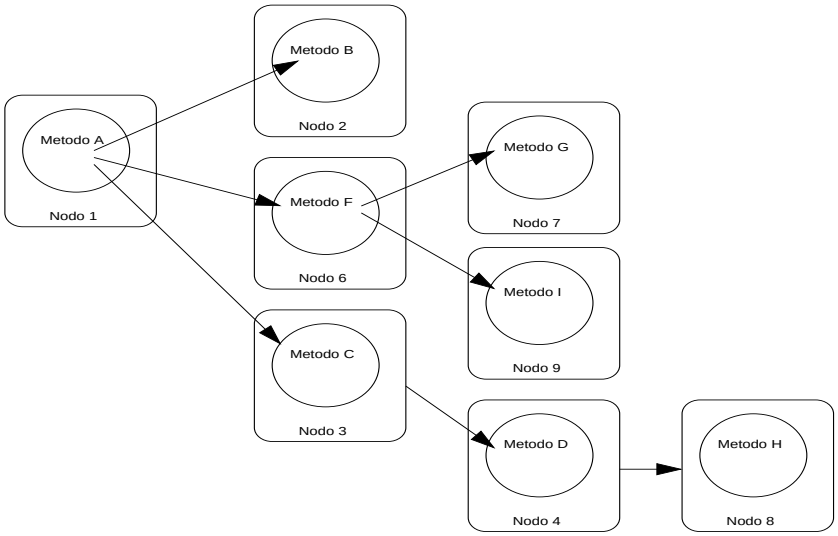

Figura 4.6: Itinerário Maior Número de Saltos.

Em caso de empate, onde mais de um itinerário apresenta o mesmo número de nodos que fazem parte dele, o critério de desempate é arbitrário, isto é, escolhe-se um entre os que estão em empate. Isto é válido para os demais itinerários definidos a seguir.

- Itinerário Menor Número de Saltos: O itinerário *Menor Número de Saltos* verifica, entre todas as possíveis seqüências de execução de métodos remotos de uma thread distribuída, aquela que apresenta o menor número de nodos visitados (saltos). Este itinerário é representado na figura 4.6 pelos nodos 1 e 2.

- Itinerário Mais Provável: O itinerário *Mais Provável* representa a execução mais freqüente de uma seqüência de métodos remotos. Ele será obtido a partir da observação do histórico da thread distribuída que armazena a seqüência de métodos remotos executados nas ativações passadas.

O itinerário *Mais Provável* pode ser representado por um grafo direcionado acíclico com arcos contendo a probabilidade da thread distribuída executar um determinado método remoto. Esta probabilidade é definida a partir das anotações contidas no histórico da thread distribuída. A figura 4.7 ilustra nodos contendo os possíveis métodos remotos a serem executados pela thread distribuída e o respectivo itinerário que é executado o maior número de vezes por ela.

Devido sua natureza autônoma, a thread distribuída pode desviar a execução para um

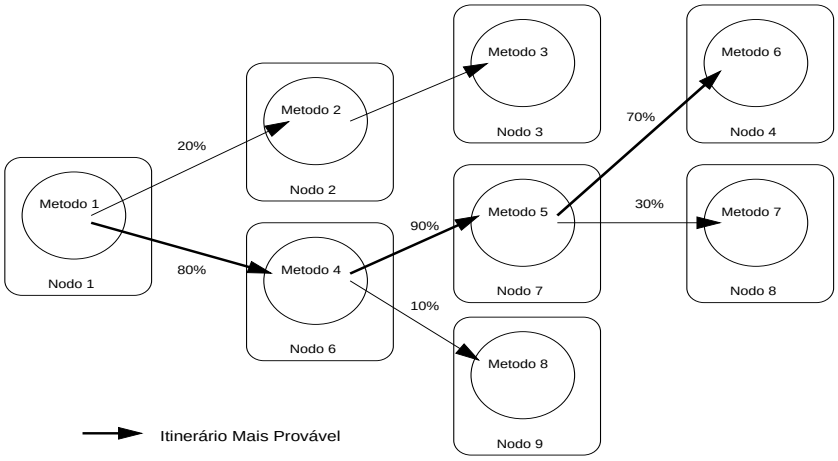

Figura 4.7: Itinerário Mais Provável.

outro itinerário, diferente daquele definido no início da sua execução, em função do estado do sistema distribuído. Todos os itinerários descritos acima apresentam a limitação de não realizar o particionamento do deadline fim a fim considerando itinerários alternativos. Isto é, se o particionamento do deadline é realizado utilizando, por exemplo, o itinerário *Maior Número de Saltos* e, durante sua execução a thread distribuída executa um método remoto que está fora deste itinerário, o deadline local deste e dos demais métodos subseqüentes devem estar definidos.

Em função desta possível situação, propomos neste trabalho que o particionamento do deadline fim a fim de uma thread distribuída seja realizado da seguinte forma: define-se um itinerário base para o particionamento do deadline (por exemplo, *Maior Número de Saltos*) e, para todos os demais itinerários, calcula-se o particionamento a partir do primeiro método que não pertence ao itinerário base, considerando o deadline local imediatamente anterior a este método.

Como um exemplo, na figura 4.7 definimos o itinerário *Maior Número de Saltos* como base para o particionamento. Este itinerário é composto pelos métodos *M1, M4, M5* e *M6*, que recebem deadlines locais a partir do método de particionamento EQF. Para definir os deadlines locais dos métodos *M2* e *M3*, que não pertencem ao itinerário base, o EQF é aplicado para estes métodos considerando o deadline local do método *M1* como instante de início do método *M2* e o deadline fim a fim da thread distribuída.

- Itinerário Ponderado: O itinerário *Ponderado* foi criado para transpor a limitação descrita anteriormente. Todos os possíveis itinerários, com a respectiva probabilidade de

serem executados, são levados em consideração para efeitos de cálculo do particionamento do deadline fim a fim.

Inicialmente define-se a probabilidade de cada itinerário ser executado pela thread distribuída. Isto é feito através do cálculo do produto das probabilidades de cada nodo que pertence a um itinerário. A seguir, o particionamento do deadline fim a fim é realizado para cada itinerário, gerando deadlines locais para cada nodo pertencente a um itinerário. Como um nodo pode pertencer a mais de um itinerário, ele poderá ter mais de um deadline local. Neste caso, calcula-se o deadline ponderado deste nodo através da multiplicação de cada deadline local pela probabilidade do itinerário a que ele pertence. Estes valores são somados e divididos pelo somatório das probabilidades dos itinerários que contém o nodo em questão. O valor resultante é um deadline local ponderado, que representa uma média dos deadlines locais de um mesmo nodo que pertence a mais de um itinerário.

A figura 4.8 ilustra um exemplo. Considerando que os nodos de 1 a 6 contém métodos remotos que são executados com as seguintes probabilidades: 80% de probabilidade de executar o método remoto contido no nodo 2, 20% de probabilidade de executar o método remoto contido no nodo 5, 60% de probabilidade de executar o método remoto contido no nodo 3 e 40% de probabilidade de executar o método remoto contido no nodo 6.

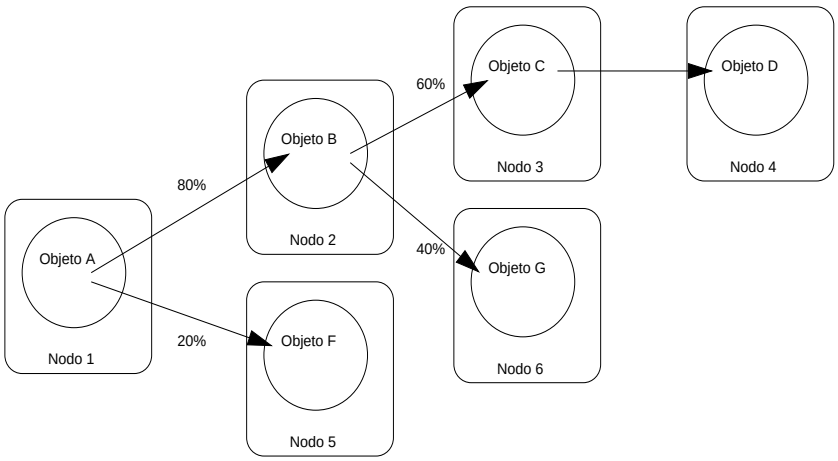

Figura 4.8: Itinerário Ponderado.

Os possíveis itinerários que a thread distribuída pode executar são:

Itinerário 1 (I1): composto pelos nodos 1, 2, 3 e 4;

Itinerário 2 (I2): composto pelos nodos 1, 2, e 6;

Itinerário 3 (I3): composto pelos nodos 1 e 5.

A probabilidade de cada itinerário é definida como:

$I_1 = 0.8(80\%) \times 0.6(60\%) = 0.48(48\%)$;

$I_2 = 0.8(80\%) \times 0.4(40\%) = 0.32(32\%)$;

$I_3 = 0.2(20\%) = 0.20(20\%)$.

Para cada um destes itinerários, é realizado o particionamento do deadline fim a fim gerando deadlines locais conforme o número de nodos que compõem o itinerário. Observa-se, entretanto, que o nodo 1, por exemplo, está presente nos itinerários I_1, I_2 e I_3 e terá três deadlines locais diferentes, um para cada itinerário. Nos casos em que um nodo possui mais de um deadline, calcula-se uma média ponderada dos deadlines com as probabilidades do respectivo itinerário. O deadline ponderado do nodo 1 (DP_1) é definido como:

$$DP_1 = ((dl_1I_1 \times 0.48) + (dl_1I_2 \times 0.32) + (dl_1I_3 \times 0.20))/(0.48 + 0.32 + 0.20),$$

onde dl_1I_1 é o deadline local do nodo 1 no itinerário *I1*, dl_1I_2 é o deadline local do nodo 1 no itinerário *I2* e dl_1I_3 é o deadline local do nodo 1 no itinerário *I3*.

O mesmo ocorre com o nodo 2, que está presente nos itinerários *I1*, *I2*. O deadline ponderado do nodo 2 (DP_2) é definido como:

$$DP_2 = ((dl_2I_1 \times 0.48) + (dl_2I_2 \times 0.32)) \div (0.48 + 0.32);$$

Como os nodos 4, 5 e 6 fazem parte, cada um, de apenas um itinerário, não existe necessidade de calcular o deadline ponderado destes nodos.

Experimentos de simulação realizados com todos os itinerários descritos neste texto mostraram o itinerário *Maior Número de Saltos* com melhores resultados [31], isto é, threads distribuídas com deadlines particionados segundo este itinerário alcançaram um maior número de deadlines fim a fim. Em função destes resultados, adotamos neste trabalho o itinerário *Maior Número de Saltos* como padrão para threads distribuídas. Assim, cada segmento local de uma thread distribuída recebe um deadline local proveniente do particionamento realizado utilizando o EQF no itinerário *Maior Número de Saltos*.

4.3.2 Parâmetros para Previsão

Cada thread distribuída possui, em seu nodo origem, um histórico que armazena informações a respeito de sua execução, como os tempos médios de computação de cada método remoto, o número de vezes que ele foi executado, restrições temporais e informações necessárias para os cálculos de previsão de perda de deadline. Este histórico se mantém fixo no nodo origem e é atualizado ao final de cada ativação da thread distribuída.

Para percorrer o sistema distribuído executando seus métodos remotos, a thread distribuída precisa carregar consigo suas restrições temporais que são usadas para fins de escalonamento local. Para realizar a previsão de perda de deadlines, a thread distribuída precisa carregar consigo outras informações durante sua passagem pelo sistema distribuído.

Para isso, em cada ativação, antes da thread distribuída partir do nodo origem, ela cria uma estrutura auxiliar onde armazena estas informações adicionais, necessárias nos cálculos de previsão. Esta estrutura, chamada *Parâmetros para Previsão* será carregada pela thread distribuída conforme ela transpõe os nodos do sistema.

No início da sua execução, no nodo origem, a thread distribuída define qual método remoto irá executar. Antes de partir para o nodo que contém o método remoto em questão, a thread distribuída faz uma cópia dos possíveis itinerários que ela poderá seguir a partir deste método remoto, além das informações necessárias para realizar a previsão. É importante ressaltar que não é realizada uma cópia de todos os itinerários da thread distribuída e sim apenas daqueles itinerários que podem ser executados considerando o método remoto que ela irá executar. Também não é realizada uma cópia de todo o histórico, apenas das informações usadas nos cálculos de previsão de perda de deadline. As restrições temporais fazem parte da thread distribuída, e não precisam ser copiadas para esta estrutura auxiliar.

Quando a thread distribuída finaliza sua execução em um nodo, o tempo de resposta local é copiado para a estrutura *Parâmetros para Previsão*. Esta estrutura tende a manter um tamanho equilibrado porque a medida que a thread distribuída transpõe o sistema distribuído realizando sua execução, alguns dados são descartados e outros são armazenados. Assim que a thread define executar um determinado método remoto, é possível definir quais os itinerários que ela irá seguir. Com isso, os demais itinerários armazenados nesta estrutura podem ser descartados. Ao final da execução da thread distribuída, não haverá itinerários armazenados

na estrutura *Parâmetros para Previsão*. Por outro lado, esta estrutura armazena os tempos de resposta locais e, ao final da execução da thread distribuída, no seu retorno ao nodo origem, a estrutura *Parâmetros para Previsão* terá os tempos de resposta de cada nodo em que a thread distribuída executou.

A estrutura *Parâmetros para Previsão* é utilizada pelos mecanismos de previsão de perda de deadline propostos nesta tese e será revista nos capítulos que descrevem estes mecanismos.

4.3.3 Escalonamento Local

A arquitetura de sistema proposta nesta tese está presente em cada nodo do sistema distribuído e possui um escalonador local responsável pelo escalonamento de um conjunto híbrido de tarefas (formado por tarefas periódicas e aperiódicas). Os escalonadores locais não colaboram uns com os outros de forma explícita e são considerados independentes. O escalonamento é apenas influenciado pelos atributos temporais associados com cada tarefa. O sistema utiliza o algoritmo *Rate Monotonic* [4] para escalonar as tarefas locais periódicas e as threads distribuídas aperiódicas. O escalonamento é realizado de forma preemptiva, garantindo que aquelas com maior prioridade tenham preferência de execução.

Utiliza-se o conceito de *Servidor de Aperiódicas* [3] para o escalonamento de threads distribuídas. Um servidor de aperiódicas atua no sistema como uma tarefa periódica, com um período e tempo de computação previamente definidos. Além disso, a tarefa servidora possui uma fila associada a ela que armazena tarefas aperiódicas que chegam no nodo. No seu período de execução, a tarefa servidora verifica sua fila e utiliza seu tempo de computação para executar as tarefas aperiódicas, caso exista alguma na fila. O servidor de aperiódicas utilizado nesta tese é o *Sporadic Server* porque apresenta melhor desempenho em relação aos servidores de prioridade fixa *Background* e *Polling Server*, define regras de reabastecimento (*Replenishment Rules*) mais elaboradas que o *Deferrable Server* e porque possui uma complexidade computacional e de implementação menor que o *Slack Stealer* [3].

Uma thread distribuída é formada por segmentos locais de execução. Cada segmento local atua em um nodo do sistema distribuído. Em um dado momento a thread distribuída estará ativa em apenas um nodo do sistema. Um servidor de aperiódicas tem a função de executar os segmentos locais das diversas threads distribuídas da aplicação. A fila do servidor

de aperiódicas contém, portanto, um segmento local de cada thread distribuída que aguarda para ser executada no nodo em questão.

A figura 4.9 mostra a arquitetura presente em cada nodo do sistema distribuído. O sistema utiliza uma tarefa interceptadora responsável por fazer a recepção da thread distribuída no nodo, o mapeamento dela para o seu segmento local e o envio deste segmento para a fila do servidor de aperiódicas.

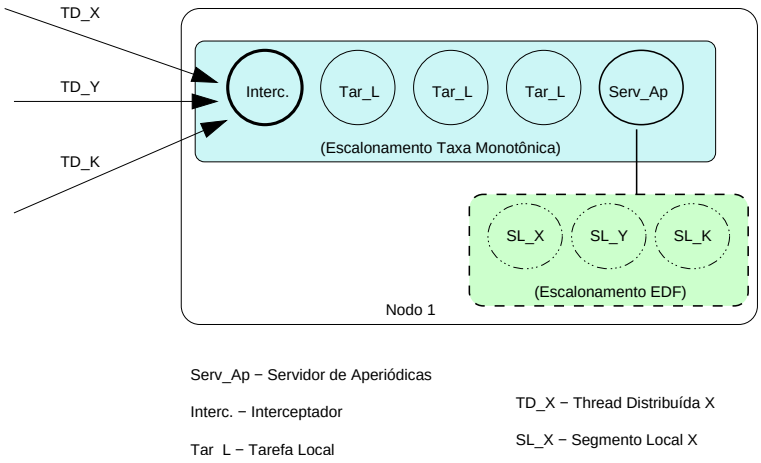

Figura 4.9: Atendimento de threads distribuídas pela tarefa interceptadora.

Sempre que uma thread distribuída tempo real chega em um nodo do sistema (nodo cabeça), ela é atendida pela tarefa interceptadora, a qual mantém uma lista de segmentos locais de threads distribuídas tempo real que executam (ou executaram) em um nodo. Para cada thread distribuída que chega em um determinado nodo, a tarefa interceptadora verifica se um segmento local dessa thread distribuída já existe. Em caso afirmativo, esse segmento local é ativado para executar em nome da thread distribuída aperiódica que chegou. Senão, a tarefa interceptadora cria um segmento local com a função de executar em nome dessa nova thread distribuída.

Nestas duas situações, todas as propriedades da thread distribuída tempo real aperiódica (tais como identificador e restrições temporais) são herdadas pelos seus segmentos locais. A tarefa interceptadora envia o segmento local para a fila do servidor de aperiódicas daquele nodo. O algoritmo *Earliest Deadline First* (EDF) [4] é usado para escalonar, de forma preemptiva, a fila do servidor de aperiódicas. O escalonador utiliza o deadline local do segmento, definido pelo algoritmo de particionamento *Equal Flexibility* (EQF) [6], para fazer o escalonamento dos segmentos locais das threads distribuídas presentes no nodo. Esse algoritmo de

particionamento foi escolhido em função dos bons resultados apresentados por ele em [6].

Conforme mostra a figura 4.9, as threads distribuídas *TD_X*, *TD_Y* e *TD_K* são atendidas pela tarefa interceptadora *Interc.* quando chegam no *Nodo 1.* A tarefa interceptadora verifica se já existem segmentos locais destas threads distribuídas. Em caso afirmativo, estes segmentos são ativados, as propriedades da thread distribuída são copiadas para o segmento e este é enviado para a fila do servidor de aperiódicas (a fila contém os segmentos locais *SL_X*, *SL_Y* e *SL_K*, das respectivas threads distribuídas). Em caso negativo, a tarefa interceptadora cria um segmento local com a função de executar em nome desta nova thread distribuída.

4.4 Previsão de Perda de Deadline em Sistemas Baseados em Threads Distribuídas

O desempenho de sistemas distribuídos de tempo real não críticos pode ser melhorado pela implementação de mecanismos que fazem a previsão de perda de deadline. Tais mecanismos podem ser usados para determinar a probabilidade de uma thread distribuída perder seu deadline fim a fim e a partir disso, ações corretivas podem ser realizadas a tempo com o objetivo de aumentar o desempenho do sistema.

A escolha do nodo adequado para executar o mecanismo de previsão é uma importante decisão. Se o mecanismo for executado quando a thread distribuída estiver nos nodos iniciais, ele pode gerar resultados insatisfatórios. Por outro lado, se o mecanismo for executado quando a thread distribuída estiver nos nodos finais será muito tarde para realizar qualquer ação corretiva.

Entretanto, a escolha entre nodos iniciais ou finais é relativa porque a thread distribuída pode estar nos nodos iniciais e ter gasto quase todo seu deadline fim a fim se os métodos remotos hospedados nestes nodos tiverem tempos de computação grandes. Da mesma forma, uma thread distribuída pode estar em um nodo final e ter gasto poucas unidades de tempo do seu deadline fim a fim se os métodos remotos hospedados nos nodos iniciais tiverem tempos de computação pequenos.

Uma solução seria usar as informações do histórico da thread distribuída para definir

o melhor momento para a previsão. Outra solução seria acionar o mecanismo de previsão considerando o tempo que a thread distribuída ainda dispõe para executar. Nesta tese, consideramos que a previsão de perda deve ocorrer assim que a thread distribuída tenha gasto metade do seu deadline. Desta forma, no caso da previsão mostrar uma possível perda de deadline, ainda haverá tempo suficiente para executar ações que possam ajudar a thread a alcançar o seu deadline fim a fim.

Se o deadline fim a fim de uma thread distribuída é igual a 200 unidades de tempo (ut), por exemplo, o mecanismo de previsão deve ser executado no primeiro nodo onde o tempo de resposta parcial da thread distribuída for igual ou maior que 100ut. Enquanto o tempo de resposta parcial da thread distribuída for menor que 100ut, ela segue sua execução no sistema distribuído.

Nesta tese propomos três mecanismos de previsão de perda de deadline, um baseado na geração de deadlines locais estimados (*Milestones*), outro baseado no cálculo da folga restante da thread distribuída e um terceiro baseado no tamanho da fila do servidor de aperiódicas. Todos os mecanismos executam a previsão assim que o tempo de resposta parcial da thread distribuída é maior ou igual a metade de seu deadline fim a fim. Nas duas próximas seções serão descritas as métricas utilizadas para avaliar os resultados das previsões realizadas pelos mecanismos propostos bem como o modelo geral de funcionamento destes.

4.5 Métricas Utilizadas para Comparação de Desempenho dos Mecanismos Propostos

A seguir são descritas duas métricas usadas para avaliar a qualidade das previsões realizadas pelos mecanismos propostos.

4.5.1 Métrica Taxa Relativa de Erro - $E(z)$

A métrica *Taxa Relativa de Erro* calcula o erro associado com a previsão realizada pelo mecanismo em função do tempo de resposta fim a fim da thread distribuída. Seja $E_k(z)$ o erro associado com a previsão realizada pelo mecanismo z para uma thread distribuída k.

$E_k(z)$ é definido como:

$$E_k(z) = 1 - Prob_k(z) \qquad se \quad Rff \leq Dff,$$
$$E_k(z) = Prob_k(z) \qquad se \quad Rff > Dff,$$

onde $Prob_k(z)$ representa a probabilidade definida pelo mecanismo z da thread distribuída k cumprir seu deadline fim a fim. Cada mecanismo define uma forma diferente para gerar o valor de $Prob_k$. *Rff* e *Dff* representam, respectivamente, o tempo de resposta fim a fim e o deadline fim a fim da thread distribuída.

Os exemplos a seguir ilustram o comportamento da métrica taxa relativa de erro:

- O mecanismo de previsão z estima que a thread distribuída irá perder seu deadline, o que de fato ocorre; então $E_k(z) = Prob_k(z) = 0$;

- O mecanismo de previsão z estima que existe uma chance de 80% da thread distribuída cumprir seu deadline, $Prob_k(z) = 0.8$, o que de fato ocorre; então $E_k(z) = 1 - 0.8 = 0.2$;

- O mecanismo de previsão z estima que existe uma chance de 50% da thread distribuída cumprir seu deadline, $Prob_k(z) = 0.5$; neste caso $E_k(z) = 0.5$ independente da thread distribuída cumprir (1 - 0.5) ou não (0.5) seu deadline.

A taxa relativa de erro de um dado mecanismo é definida como:

$$E(z) = \sum_{k=1}^{n_k} E_k(z) \div n_k,$$

onde n_k é o número de threads distribuídas do sistema. É importante observar que esta métrica representa a convicção de um algoritmo de previsão sobre a capacidade da thread distribuída cumprir ou não um deadline. Por exemplo, supondo que para a tarefa T_k temos a previsão de dois mecanismos y e z, sendo: $Prob_k(y) = 0.6$ e $Prob_k(z) = 0.8$. Se a thread distribuída cumpre seu deadline, teremos $E_k(y) = 0.4$ e $E_k(z) = 0.2$. Conforme o número n_k aumenta, teremos uma medida da capacidade de cada mecanismo em fazer previsões corretas. Um mecanismo perfeito de previsão de perda de deadlines deve gerar um erro igual a zero ao longo de suas execuções.

4.5.2 Métrica Taxa de Previsões Corretas - $PC(z)$

A métrica *Taxa de Previsões Corretas* calcula o número de previsões corretas sobre o número total de previsões realizadas por um determinado mecanismo. Esta métrica considera a probabilidade de uma thread distribuída cumprir seu deadline fim a fim, da seguinte forma:

$$Se \quad (Prob_k(z) < 50\%) \quad e \quad (Rff > Dff) \quad entao \quad PrevisoesCorretas(z) += 1;$$
$$Se \quad (Prob_k(z) \geq 50\%) \quad e \quad (Rff \leq Dff) \quad entao \quad PrevisoesCorretas(z) += 1;$$
$$PC(z) = PrevisoesCorretas(z)/NumTotalPrevisoes(z);$$

onde $Prob_k(z)$ é a probabilidade da thread distribuída k cumprir seu deadline fim a fim, definida a partir de um mecanismo z de previsão; Rff e Dff representam o tempo de resposta fim a fim e o deadline fim a fim da thread distribuída k, respectivamente; $PrevisoesCorretas(z)$ armazena o número de previsões corretas do mecanismo z e $NumTotalPrevisoes(z)$ representa o número de total de previsões realizadas pelo mecanismo z, sendo uma previsão para cada thread distribuída do sistema.

Os exemplos a seguir ilustram o comportamento da métrica taxa de previsões corretas. Considerando que para cada exemplo o mecanismo de previsão z realize duas previsões:

- **Exemplo 1:**

- O mecanismo estima que a thread distribuída k irá perder seu deadline ($Prob_k(z) < 50\%$)), o que de fato ocorre ($Rff > Dff$); então $PrevisoesCorretas(z) += 1$;
- O mecanismo estima que a thread distribuída j irá cumprir seu deadline ($Prob_j(z) \geq 50\%$), o que de fato ocorre ($Rff \leq Dff$); então $PrevisoesCorretas(z) += 2$;
- A taxa de previsões corretas do mecanismo z para estas duas threads distribuídas (k e j) será: $PC(z) = 2/2 = 1$;

- **Exemplo 2:**

- O mecanismo estima que a thread distribuída k irá perder seu deadline ($Prob_k(z) < 50\%$)), o que de fato não ocorre ($Rff \leq Dff$); então $PrevisoesCorretas(z) += 0$;

- O mecanismo estima que a thread distribuída j irá cumprir seu deadline ($Prob_j(z) \geq 50\%$), o que de fato não ocorre ($Rff > Dff$); então $PrevisoesCorretas(z) += 0$;
- A taxa de previsões corretas do mecanismo z para estas duas threads distribuídas (k e j) será: $PC(z) = 0/2 = 0$;

- Exemplo 3:

- O mecanismo estima que a thread distribuída k irá perder seu deadline ($Prob_k(z) < 50\%$)), o que de fato ocorre ($Rff > Dff$); então $PrevisoesCorretas(z) += 1$;
- O mecanismo estima que a thread distribuída j irá cumprir seu deadline ($Prob_j(z) \geq 50\%$), o que de fato não ocorre ($Rff > Dff$); então $PrevisoesCorretas(z) += 1$;
- A taxa de previsões corretas do mecanismo z para estas duas threads distribuídas (k e j) será: $PC(z) = 1/2 = 0,5$;

Quanto mais próximo de 1 for o resultado desta métrica, melhor será a previsão realizada por um dado mecanismo.

4.6 Acionamento dos Mecanismos de Previsão

Os mecanismos de previsão de perda de deadline propostos nessa tese possuem a mesma dinâmica de funcionamento. A diferença entre eles surge no cálculo da probabilidade do cumprimento do deadline fim a fim.

Inicialmente, define-se o itinerário *Maior Número de Saltos* que é aquele em que o número de nodos é maior em relação aos outros itinerários. Através da inspeção dos supostos itinerários que a thread distribuída pode executar, define-se aquele com maior número de nodos.

Em seguida, para fins de previsão, é realizado o particionamento do deadline fim a fim da thread distribuída utilizando o método *Equal Flexibility* (EQF)[6] considerando este suposto itinerário. A equação que define o método EQF é a seguinte:

$$dl(si) = ar(si) + C(si) + (Dff_k - ar(si) - \sum_{j=i}^{n} C(sj)) \times (\frac{C(si)}{\sum_{j=i}^{n} C(sj)}),$$

onde ar é o tempo de chegada do segmento da thread distribuída k e C é o tempo de computação deste segmento; Dff_k é o deadline fim a fim da thread distribuída k e n é o número de segmentos locais da thread distribuída.

Durante a passagem da thread distribuída pelo sistema, ao final da execução em um nodo, a thread distribuída verifica seu tempo de resposta local. Se o tempo de resposta local for igual ou maior que metade do deadline fim a fim, o mecanismo de previsão é acionado. Neste momento, os mecanismos de previsão calculam a probabilidade da thread distribuída cumprir seu deadline fim a fim. Assim que a thread distribuída conclui sua execução no sistema, calcula-se o erro associado com a previsão realizada (seção 4.5).

4.7 Conclusão

Este capítulo apresentou o modelo de tarefas e a arquitetura de sistema adotados nesta tese. O modelo de tarefas inclui tarefas locais periódicas críticas e tarefas distribuídas aperiódicas não-críticas, sendo que estas últimas são implementadas pela abstração threads distribuídas.

A arquitetura de sistema proposta tem como objetivo de escalonamento garantir os deadlines das tarefas locais e reduzir o tempo de resposta das threads distribuídas. Cada nodo do sistema possui uma tarefa interceptadora que recebe threads distribuídas e ativa (ou cria) o segmento local correspondente. Estes segmentos locais são escalonados por um servidor de aperiódicas que utiliza EDF preemptivo para escalonar sua fila.

O método de escalonamento adotado nesta tese inova ao considerar execuções condicionais no particionamento do deadline fim a fim. Foram definidos os principais itinerários que uma thread distribuída pode seguir e o particionamento do deadline fim a fim é realizado considerando um destes itinerários. O reparticionamento do deadline é necessário nos casos em que a thread distribuída executa um método remoto que está fora do itinerário previamente definido. Esta situação ocorre porque a execução de uma thread distribuída varia conforme a dinâmica do sistema distribuído em que ela atua. Desta forma, mesmo definindo um itinerário que supostamente a thread distribuída irá seguir, podem ocorrer situações em que ela desvia desse suposto itinerário e executa métodos remotos criando, em tempo de execução, outro itinerário.

Neste capítulo foram definidas as premissas dos mecanismos de previsão propostos nesta tese. Na seção 4.4 discutimos sobre a escolha do nodo onde os mecanismos de previsão de perda de deadline devem ser executados. Concluímos que, ao contrário de definir um nodo específico, cada nodo do sistema deve monitorar os tempos de resposta parciais da thread distribuída. O momento adequado para executar o mecanismo de previsão de perda de deadline será quando a thread distribuída alcançar metade de seu deadline fim a fim.

Na seção 4.5 foram apresentadas as métricas Taxa de Erro Relativa ($E_k(z)$) e Taxa de Previsões Corretas($PC(z)$), que irão avaliar a qualidade dos mecanismos de previsão propostos nesta tese. Na seção 4.6 o modelo geral de funcionamento dos mecanismos propostos foi descrito, enfatizando o momento em que são acionados.

Capítulo 5

Mecanismos de Previsão Baseados em Milestones

5.1 Introdução

A previsão sobre a perda de um deadline fim a fim de uma thread distribuída pode ser realizada considerando apenas informações da thread distribuída em questão, como os tempos de computação dos segmentos locais da thread. Estas informações são conhecidas antes da ativação da thread distribuída, por conta do histórico de ativações passadas (informações previamente conhecidas). Outra forma de executar a previsão é considerar outras informações sobre o sistema (informações globais), como o tamanho da fila dos servidores dos nodos que pertencem ao itinerário que a thread distribuída está executando, além de informações da thread distribuída em questão. As informações globais serão conhecidas durante a execução da thread distribuída no sistema.

Os mecanismos de previsão de perda de deadlines descritos neste capítulo consideram apenas informações previamente conhecidas para a definição de um tempo de resposta estimado, chamado *Milestone*. São propostas três formas diferentes de gerar os *Milestones*, usando como base os métodos de particionamento de deadlines propostos em [6].

Na seção 5.2 deste capítulo serão descritas diferentes formas de definir os *Milestones*. Na seção 5.3 são apresentadas as condições das simulações e os resultados são descritos na seção 5.4. A seção 5.5 apresenta as conclusões deste capítulo.

5.2 Descrição dos Mecanismos Baseados em Milestones

Os mecanismos de previsão baseados em *Milestones* geram tempos de resposta estimados a partir das equações dos métodos de particionamento propostos em [6]. Nesta tese são propostos três tipos de *Milestones*:

- *MilestoneED* (MED): Baseado no método de particionamento *Effective Deadline* [6];

- *MilestoneEQS* (MEQS): Baseado no método de particionamento *Equal Slack* [6];

- *MilestoneEQF* (MEQF): Baseado no método de particionamento *Equal Flexibility* [6];

Conforme discutido anteriormente, os mecanismos de previsão são executados no momento em que a thread distribuída cumpre metade do seu deadline fim a fim. Esta situação pode ocorrer no primeiro nodo em que a thread distribuída executa, no último ou em qualquer outro, de acordo com a carga do sistema. Em função desta observação, os mecanismos de previsão baseados em *Milestones* definem tempos de resposta estimados para todos os nodos que fazem parte do suposto itinerário que a thread distribuída poderá executar.

No capítulo 4 (seção 4.3.1) foram definidos os principais itinerários que uma thread distribuída pode seguir. Um itinerário representa a composição de uma seqüencia de métodos remotos que a thread distribuída pode executar. Devido à natureza dinâmica da thread distribuída, que varia sua execução conforme as condições do sistema em que ela atua, podem acontecer situações em que a thread distribuída não executa todos os métodos definidos em um itinerário. Nestes casos, um itinerário será percorrido apenas parcialmente. Os itinerários podem ser usados para fins de particionamento do deadline fim a fim (como visto na seção 4.3.1) e para fins de previsão, onde será gerado um *Milestone* para cada método que compõe um itinerário.

Dos itinerários descritos no capítulo anterior, foram escolhidos quatro para serem usados com cada previsor *Milestone*. Os itinerários escolhidos são: *Maior Número de Saltos, Menor Número de Saltos, Ponderado* e *Mais Provável*. Da união de cada previsor *Milestones* com estes quatro itinerários surgem doze previsores:

- MED - Maior Número de Saltos;

- MED - Menor Número de Saltos;

- MED - Ponderado;
- MED - Mais Provável;

- MEQS - Maior Número de Saltos;
- MEQS - Menor Número de Saltos;
- MEQS - Ponderado;
- MEQS - Mais Provável;

- MEQF - Maior Número de Saltos;
- MEQF - Menor Número de Saltos;
- MEQF - Ponderado;
- MEQF - Mais Provável;

A geração dos *Milestones* é realizada no nodo origem, antes da thread distribuída iniciar sua execução. Para cada par *previsor-itinerário*, a previsão é realizada considerando os tempos de computação dos métodos que compõem o itinerário em questão. Os *Milestones* gerados são carregados com a thread distribuída, na estrutura *Registro por Ativação*, conforme ela transpõe os nodos do sistema. Conforme a thread distribuída visita os nodos do sistema, ela descarta os *Milestones* dos nodos já visitados, visando manter o Registro por Ativação com o menor tamanho possível. Quando o mecanismo de previsão é acionado, ele utiliza um *Milestone* para a definição da probabilidade da thread distribuída cumprir seu deadline fim a fim. Depois de realizada a previsão, todos os *Milestones* podem ser descartados do Registro por Ativação, porque eles não são mais necessários. A única informação que a thread distribuída precisa manter nesta estrutura até o final de sua execução é o resultado da previsão, isto é, a probabilidade dela cumprir seu deadline fim a fim.

As próximas seções desse texto apresentam a forma como são gerados os *MilestoneED*, *MilestoneEQS* e *MilestoneEQF*, respectivamente.

5.2.1 *MilestoneED* (MED)

O mecanismo de previsão *MilestoneED* (MED) define estimativas para tempos de resposta locais (*Milestones*) a partir do método de particionamento de deadlines *Effective Deadline* (ED) [6]. Conforme mostra a equação 5.1 este método utiliza apenas os tempos médios de computação dos segmentos locais da thread distribuída para a definição dos *Milestones*.

$$MED(s_i) = Dff_k - \sum_{j=i+1}^{n} C(s_j), \tag{5.1}$$

onde $MED(s_i)$ representa o *Milestone* do segmento local s_i da thread distribuída k, Dff_k é o deadline fim a fim da thread distribuída k, n é o número de segmentos locais que compõem a thread distribuída k em um determinado itinerário e C é o tempo de computação destes segmentos locais.

No tipo de sistema proposto neste trabalho, os tempos de comunicação são muito menores que os tempos de computação. Em função disto, os previsores utilizam em seus cálculos apenas os tempos de computação.

No momento em que o mecanismo de previsão é acionado, define-se o valor da variável *Slack* e do índice P associado à thread distribuída k como sendo:

$$Slack = (MED(s_i) - rl(s_i)) \div Dff_k$$
$$P_k(MED) = slack + \sigma;$$

onde $rl(s_i)$ representa o tempo de resposta efetivo do segmento local s_i da thread distribuída k e a variável σ é usada para ajustar a probabilidade, sendo 0.5 por default. A variável *Slack* recebe a subtração do valor do *Milestone* do nodo em questão pelo tempo de resposta local da thread distribuída. Ao resultado dessa subtração adiciona-se o valor de σ, indicando que se os valores do *Milestone* e do tempo de resposta local forem iguais, a thread distribuída tem uma probabilidade de 50% de cumprir seu deadline fim a fim.

Define-se a probabilidade da thread distribuída k cumprir seu deadline fim a fim como

sendo:

$$Prob_k(MED) = \begin{cases} 0 & P_k(MED) < 0 \\ P_k(MED) & 0 \leq P_k(MED) \leq 1 \\ 1 & P_k(MED) > 1 \end{cases}$$

onde $Prob_k(MED)$ representa a probabilidade da thread distribuída k cumprir seu deadline fim a fim segundo o mecanismo de previsão MED.

O exemplo a seguir ilustra o funcionamento do mecanismo de previsão *MilestoneED* com o itinerário *Maior Número de Saltos*. A figura 5.1 apresenta os possíveis itinerários que a thread distribuída poderá seguir e a figura 5.2 apresenta o histórico da thread distribuída, armazenado no nodo origem. O deadline fim a fim da thread distribuída é igual a 200 ut (unidades de tempo). Para fins de escalonamento, o método de particionamento de deadline usado é o EQF [6], com o itinerário *Maior Número de Saltos*. O histórico contém os itinerários que a thread distribuída pode executar, os segmentos locais que executam os métodos remotos que compõem os itinerários (coluna *sl*), os respectivos tempos de computação de cada segmento local (indicados pela coluna C), os deadlines dos segmentos locais (indicados pela coluna *dl*) e os MilestonesED para cada segmento local da thread distribuída (coluna *MED*).

Neste exemplo é interessante observar que surge um empate para a definição do itinerário *Maior Número de Saltos*. Os itinerários compostos pelos métodos *M1-M4-M7-M8-M7-M4-M1* e *M1-M4-M7-M9-M7-M4-M1* apresentam o mesmo número de nodos (7). Quando ocorrem situações de empate, o itinerário escolhido é arbitrário, isto é, define-se um entre os que estão em empate. Neste caso, assumimos o itinerário composto pelos métodos *M1-M4-M7-M9-M7-M4-M1* tanto para fins de particionamento do deadline fim a fim quanto para fins de previsão.

Os *MilestonesED* definidos para os segmentos locais que irão executar os métodos remotos *M1-M4-M7-M9-M7-M4-M1* possuem, respectivamente, os valores 89ut, 94ut, 104ut, 129ut, 154ut, 188ut e 200ut. Ao final da execução em um nodo, a thread distribuída aciona o mecanismo de previsão se o seu tempo de resposta for maior ou igual a metade do deadline fim a fim. Dependendo do nodo onde for realizada a previsão, será utilizado um dos *milestones* definidos.

Supondo que o tempo de resposta local da thread distribuída no nodo 6 seja igual a 100

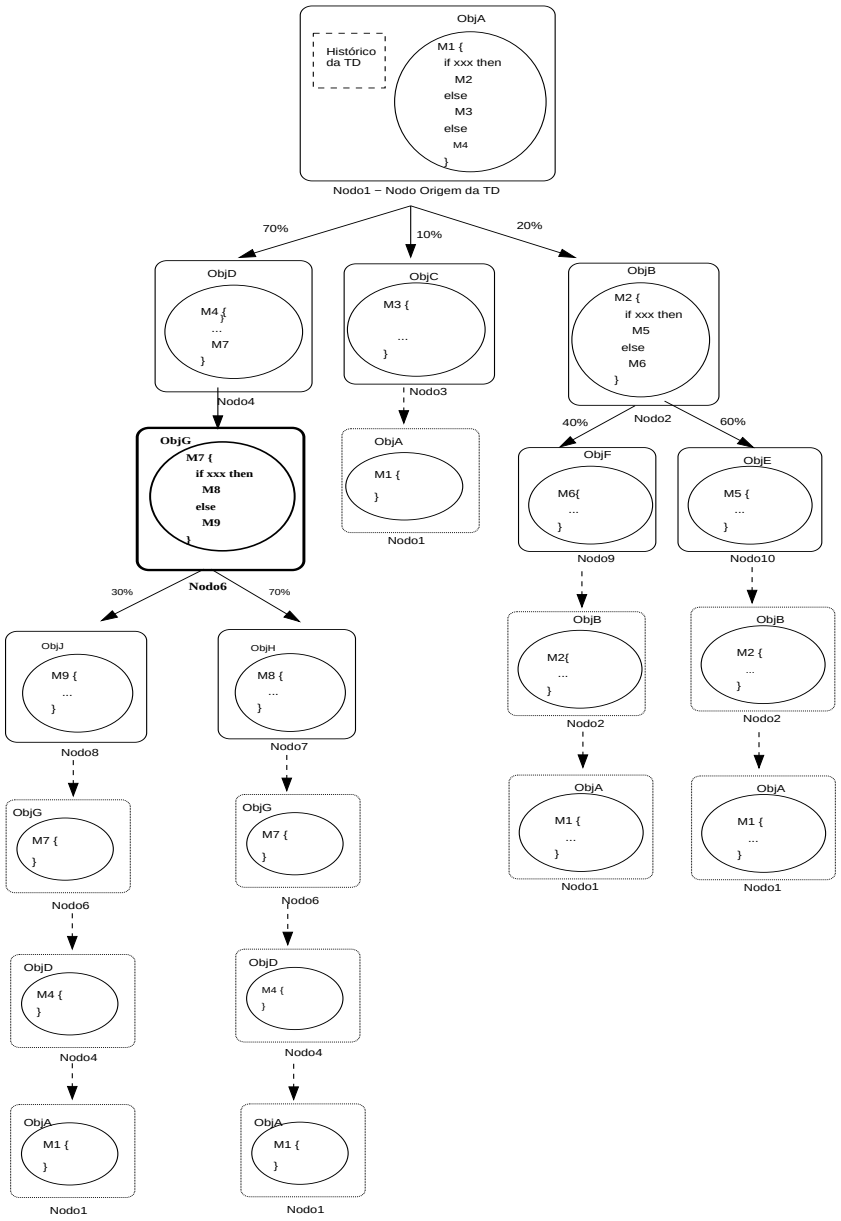

Figura 5.1: *MilestoneED*.

ut, o mecanismo de previsão é acionado neste nodo e a probabilidade da thread distribuída k cumprir seu deadline fim a fim é calculado da seguinte forma:

$$Slack = (104 - 100) \div 200;$$

$$P_k(MED) = slack + 0,5;$$

$$Prob_k(MED) = 0,52;$$

Neste exemplo, a probabilidade da thread distribuída cumprir seu deadline fim a fim, definida

HISTÓRICO DA THREAD DISTRIBUÍDA

Itinerários: Deadline fim a fim = 200ut

I1 = M1−M2−M5−M2−M1 (sl 1−2−5−10−11)

I2 = M1−M2−M6−M2−M1 (sl 1−2−6−12−13)

I3 = M1−M3−M1 (sl 1−3−14)

I4 = M1−M4−M7−M8−M7−M4−M1 (sl 1−4−7−8−15−16−17)

I5 = M1−M4−M7−M9−M7−M4−M1 (sl 1−4−7−9−18−19−20)

sl	C	dl	MED	sl	C	dl	MED
1	40	100	89	11	13	200	200
2	10	113	135	12	7	163	172
3	15	179	196	13	28	200	200
4	5	105	94	14	4	200	200
5	20	137	146	15	13	149	174
6	30	153	165	16	9	167	183
7	10	114	104	17	17	200	200
8	5	123	161	18	25	159	154
9	25	136	129	19	34	189	188
10	41	185	187	20	12	200	200

Figura 5.2: *Histórico do MilestoneED.*

pelo mecanismo *MilestoneED*, é $Prob_k(MED) = 0,52$. Ao final da execução da thread distribuída, é verificado se ela cumpriu ou não o deadline fim a fim e avalia-se a previsão realizada através das métricas definidas na seção 4.5.

5.2.2 *MilestoneEQS* (MEQS)

O mecanismo de previsão *MilestoneEQS* (MEQS) define estimativas para tempos de resposta locais (*Milestones*) a partir do método de particionamento de deadlines *Equal Slack* (EQS) [6] utilizando a seguinte expressão:

$$MEQS(s_i) = ar(s_i) + C(s_i) + (Dff_k - ar(s_i) - \sum_{j=i}^{n} C(s_j)) \div (n - i + 1)$$

$$Slack = (MEQS(s_i) - rl(s_i)) \div Dff_k$$

$$P_k(MEQS) = slack + \sigma$$

onde $ar(s_i)$ é o tempo de chegada do segmento local da thread distribuída k em um nodo.

Define-se a probabilidade da thread distribuída k cumprir seu deadline fim a fim como

sendo:

$$Prob_k(MEQS) = \begin{cases} 0 & P_k(MEQS) < 0 \\ P_k(MEQS) & 0 \leq P_k(MEQS) \leq 1 \\ 1 & P_k(MEQS) > 1 \end{cases}$$

onde $Prob_k(MEQS)$ representa a probabilidade da thread distribuída k cumprir seu deadline fim a fim segundo o mecanismo de previsão MEQS.

Este *Milestone* define tempos de resposta locais estimados para a thread distribuída dividindo igualmente o valor da folga entre todos os segmentos locais da thread distribuída.

Usando o mesmo exemplo da seção anterior (figura 5.1), os *milestones* definidos para os segmentos locais que irão executar o itinerário *Maior Número de Saltos* (métodos remotos *M1-M4-M7-M9-M7-M4-M1*) possuem, respectivamente, os valores 47ut, 59ut, 76ut, 108ut, 140ut, 181ut e 200ut, conforme ilustra a coluna *MEQS* da figura 5.3. Esta figura apresenta apenas o histórico da thread distribuída modificado com os valores destes *Milestones*.

HISTÓRICO DA THREAD DISTRIB.

Deadline fim a fim = 200ut

Itinerários:

I1 = M1−M2−M5−M2−M1 (sl 1−2−5−10−11)
I2 = M1−M2−M6−M2−M1 (sl 1−2−6−12−13)
I3 = M1−M3−M1 (sl 1−3−14)
I4 = M1−M4−M7−M8−M7−M4−M1 (sl 1−4−7−8−15−16−17)
I5 = M1−M4−M7−M9−M7−M4−M1 (sl 1−4−7−9−18−19−20)

sl	C	dl	MEQS	sl	C	dl	MEQS
1	40	100	47	11	13	200	200
2	10	113	77	12	7	163	153
3	15	179	129	13	28	200	200
4	5	105	59	14	4	200	200
5	20	137	113	15	13	149	134
6	30	153	126	16	9	167	163
7	10	114	76	17	17	200	200
8	5	123	101	18	25	159	140
9	25	136	108	19	34	189	181
10	41	185	171	20	12	200	200

Figura 5.3: *MilestoneEQS*.

Como no exemplo anterior, considerando que o mecanismo de previsão é acionado no nodo 6 (tempo de resposta local da thread distribuída é igual a 100ut), a probabilidade da

thread distribuída k cumprir seu deadline fim a fim será:

$$Slack = (76 - 100) \div 200;$$
$$P_k(MEQS) = slack + 0,5;$$
$$Prob_k(MEQS) = 0,38;$$

Quando a thread distribuída conclui sua execução no sistema, a probabilidade definida pelo mecanismo *MilestoneEQS* ($Prob_k(MEQS) = 0,38$) é avaliada pelas métricas definidas na seção 4.5.

5.2.3 *MilestoneEQF* (MEQF)

O mecanismo de previsão *MilestoneEQF* (MEQF) define estimativas para tempos de resposta locais a partir do método de particionamento de deadlines *Equal Flexibility* (EQF) [6] utilizando a seguinte equação:

$$MEQF(s_i) = ar(s_i) + C(s_i) + (Dff_k - ar(s_i) - \sum_{j=i}^{n} C(s_j)) \times (\frac{C(s_i)}{\sum_{j=i}^{n} C(s_j)}).$$
$$Slack = (MEQF(s_i) - rl(s_i)) \div Dff_k$$
$$P_k(MEQF) = slack + \sigma$$

Define-se a probabilidade da thread distribuída k cumprir seu deadline fim a fim como sendo:

$$Prob_k(MEQF) = \begin{cases} 0 & P_k(MEQF) < 0 \\ P_k(MEQF) & 0 \leq P_k(MEQF) \leq 1 \\ 1 & P_k(MEQF) > 1 \end{cases}$$

onde $Prob_k(MEQF)$ representa a probabilidade da thread distribuída k cumprir seu deadline fim a fim segundo o mecanismo de previsão MEQF.

Este *Milestone* define tempos de resposta locais estimados para a thread distribuída dividindo o valor da folga de forma proporcional aos tempos de computação de cada segmento local que a compõem.

Nesta tese, os *milestones* definidos a partir deste mecanismo coincidem com os valores

dos deadlines locais dos segmentos da thread distribuída. Isto ocorre porque o método de particionamento do deadline fim a fim usado neste trabalho foi o *Equal Flexibility* com o itinerário *Maior Número de Saltos*, o mesmo usado para gerar os *MilestonesEQF*.

Usando o mesmo exemplo da seção 5.2.1 (figura 5.1), os *MilestoneEQF* definidos para os segmentos locais que irão executar os métodos remotos do itinerário *Maior Número de Saltos* (*M1-M4-M7-M9-M7-M4-M1*) possuem, respectivamente, os valores 100ut, 105ut, 114ut, 136ut, 159ut, 189ut e 200ut, conforme ilustra a coluna C da figura 5.4. A figura apresenta apenas o histórico da thread distribuída modificado com os valores destes *milestones*.

HISTÓRICO DA THREAD DISTRIB.

Deadline fim a fim = 200ut

Itinerários:

I1 = M1−M2−M5−M2−M1 (sl 1−2−5−10−11)
I2 = M1−M2−M6−M2−M1 (sl 1−2−6−12−13)
I3 = M1−M3−M1 (sl 1−3−14)
I4 = M1−M4−M7−M8−M7−M4−M1 (sl 1−4−7−8−15−16−17)
I5 = M1−M4−M7−M9−M7−M4−M1 (sl 1−4−7−9−18−19−20)

sl	C	dl	MEQF	sl	C	dl	MEQF
1	40	100	100	11	13	200	200
2	10	113	113	12	7	163	163
3	15	179	179	13	28	200	200
4	5	105	105	14	4	200	200
5	20	137	137	15	13	149	149
6	30	153	153	16	9	167	167
7	10	114	114	17	17	200	200
8	5	123	123	18	25	159	159
9	25	136	136	19	34	189	189
10	41	185	185	20	12	200	200

Figura 5.4: *MilestoneEQF*.

Considerando que a thread distribuída k tenha um tempo de resposta igual a 100ut no nodo 6, sua probabilidade de cumprir o deadline fim a fim, calculada segundo o mecanismo MEQF, será:

$$Slack = (114 - 100)/200;$$

$$P_k(MEQF) = slack + 0.5;$$

$$Prob_k(MEQF) = 0,57;$$

Assim que a thread distribuída conclui sua execução, avalia-se a qualidade da previsão realizada ($Prob_k(MEQF) = 0,57$) a partir das métricas definidas na seção 4.5.

5.3 Condições das Simulações

Simulações foram realizadas com o objetivo de avaliar a qualidade das previsões feitas pelos mecanismos baseados em *Milestones*.

A ferramenta utilizada para as simulações chama-se *Frasiteral*, desenvolvida no Departamento de Automação e Sistemas da Universidade Federal de Santa Catarina (DAS-UFSC), sob a coordenação do prof. Rômulo Silva de Oliveira. Trata-se de um *framework* orientado a objetos implementado na linguagem de programação Java e fornece um conjunto de classes e interfaces para a especificação e simulação de tarefas com requisitos temporais e escalonadores de tempo real. Diversos trabalhos do grupo de pesquisa usaram este framework [32], [33].

Foi definida uma lista com 90 threads distribuídas (TDs) diferentes, sendo:

- 30 TDs do tipo Pipeline (Anexo B);
- 30 TDs do tipo Árvore Balanceada (Anexos C);
- 30 TDs do tipo Árvore Não Balanceada (Anexo D)

Considerando estes três tipos de threads distribuídas, foram testados 4 tipos de cargas:

- Carga 1: Conjunto de threads distribuídas do tipo Pipeline apenas;
- Carga 2: Conjunto de threads distribuídas do tipo Árvore Balanceada apenas;
- Carga 3: Conjunto de threads distribuídas do tipo Árvore Não-Balanceada apenas;
- Carga 4: Conjunto misto de threads distribuídas, contendo os três tipos definidos (Pipeline, Árvore Balanceada, Árvore Não-Balanceada).

Para cada tipo de carga foram geradas 100 configurações diferentes. Para cada configuração, foram sorteadas 9 threads distribuídas da lista de 90. No caso da Carga 4 (conjunto misto de threads distribuídas), foram sorteadas 3 threads distribuídas de cada tipo.

Em cada configuração, um valor diferente de deadline fim a fim foi utilizado para todas as threads distribuídas do sistema. Os valores dos deadlines foram escolhidos considerando três condições:

- ***Deadline Folgado***: onde a maioria das threads distribuídas cumpre seu deadline fim a fim;
- ***Deadline Justo***: onde cerca da metade das threads distribuídas cumpre e a outra metade perde seu deadline fim a fim;
- ***Deadline Apertado***: onde a maioria das threads distribuídas perde seu deadline fim a fim;

Para abranger estas três condições de deadlines fim a fim, foi definida uma faixa de deadlines que varia de 100 até 900ut.

Cada thread distribuída possui um número variado de segmentos locais, cada um deles com um tempo de computação. A chegada de uma thread distribuída no sistema segue uma distribuição exponencial, com intervalo médio de 700 ut entre chegadas.

O tempo gasto por uma thread distribuída quando sai de um nodo até chegar em outro (tempo de rede) é igual a 2ut. Cada nodo do sistema distribuído contém 4 tarefas locais periódicas com períodos iguais a 10ut, 20ut, 40ut e 80ut e tempos de computação iguais a 2ut, 2ut, 4ut e 8ut, respectivamente. A tarefa interceptadora está entre as tarefas locais periódicas garantidas. O servidor de aperiódicas também é considerado uma tarefa local periódica com tempo de computação igual a 5ut, período e deadline iguais a 10ut. Como o período das tarefas locais são múltiplos entre si, é possível alcançar 100% de ocupação do processador.

O tempo de simulação foi de 20000ut e foram usadas duas métricas para medir a qualidade das previsões realizadas: métrica Taxa de Erro Relativo ($E(z)$) e a métrica Taxa de Previsões Corretas ($PC(z)$). Na primeira métrica, descrita no capítulo 4, quando mais próximo de zero for o resultado gerado, menor é o erro associado com a previsão realizada. Na segunda métrica, quanto mais próximo de 1 for o resultado gerado, melhor é a capacidade do previsor de realizar previsões corretas.

5.4 Resultados das Simulações

Para fins de comparação, utilizamos juntamente com os previsores *Milestones*, o previsor *Ultimate*. Este previsor não realiza nenhum cálculo para definir *Milestones*. Ele assume cada

Milestone como sendo o valor do próprio deadline fim a fim da thread distribuída.

Para cada previsor *Milestone*, foram usados quatro tipos de itinerários para gerar o valor do tempo de resposta estimado, totalizando doze previsores. Os itinerários usados foram: Maior Número de Saltos (*MaNS*), Menor Número de Saltos (*MeNS*), Ponderado (*Pond*) e Mais Provável (*MProv*). O previsor *Ultimate* não faz uso de itinerários.

As tabelas a seguir contêm resultados com treze previsores:

- Ultimate;
- MED-MaNS, MED-MeNS, MED-Pond, MED-MProv;
- MEQS-MaNS, MEQS-MeNS, MEQS-Pond, MEQS-MProv;
- MEQF-MaNS, MEQF-MeNS, MEQF-Pond, MEQF-MProv

Os valores apresentados nas tabelas a seguir incluem o intervalo de confiança para um grau de confiança de 95%. Este intervalo aparece ao lado de cada valor da tabela e é indicado pela coluna *IC*.

5.4.1 Simulações com a Carga 1 - *Threads Distribuídas Tipo Pipeline*

A tabela 5.1 apresenta os resultados das simulações realizadas com a Carga 1, segundo a métrica Taxa Relativa de Erro ($E(z)$). É possível observar que para cada deadline os resultados de cada previsor não variam em relação aos itinerários supostos no momento da criação dos *Milestones*, isto é, o previsor MED, por exemplo, em cada faixa de deadline, apresenta a mesma taxa de erro nos quatro itinerários definidos. Isto ocorre porque as threads simuladas são do tipo *pipeline* e o uso de diferentes itinerários neste tipo de thread produz sempre a mesma seqüência de nodos que serão visitados por ela.

Em todos os deadlines simulados, o previsor Ultimate gerou as maiores taxas de erro. Já o previsor MEQF gerou melhores resultados, com exceção do deadline 900, onde houve um empate nos resultados do MEQF e MED (embora o IC do previsor MEQF apresente resultados melhores). Nesta situação, o deadline é dito folgado (a maioria das threads distribuídas cumprem o deadline fim a fim) e o previsor MED atribui toda a folga possível ao primeiro segmento local da thread distribuída. Provavelmente o tempo de resposta local efetivo da

Tabela 5.1: Taxa Relativa de Erro para Threads Distribuídas tipo *Pipeline*

	Deadline, Taxa de Erro e Intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,223	±0,012	0,353	±0,014	0,280	±0,023	0,181	±0,027	0,126	±0,024
MED-MaNS	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MED-MeNS	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MED-Pond	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MED-MProv	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MEQS-MaNS	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQS-MeNS	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQS-Pond	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQS-MProv	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQF-MaNS	**0,041**	±0,005	**0,150**	±0,010	**0,160**	±0,009	**0,146**	±0,011	**0,116**	±0,014
MEQF-MeNS	**0,041**	±0,005	**0,150**	±0,010	**0,160**	±0,009	**0,146**	±0,011	**0,116**	±0,014
MEQF-Pond	**0,041**	±0,005	**0,150**	±0,010	**0,160**	±0,009	**0,146**	±0,011	**0,116**	±0,014
MEQF-MProv	**0,041**	±0,005	**0,150**	±0,010	**0,160**	±0,009	**0,146**	±0,011	**0,116**	±0,014

thread distribuída será menor que o tempo de resposta local estimado pelo previsor MED, o que implica em uma previsão de cumprimento do deadline fim a fim. No caso do previsor MEQF, cada segmento local da thread distribuída recebe um valor de folga suficiente para executar e também, provavelmente, o tempo de resposta estimado será menor (no máximo igual) ao tempo de resposta efetivo, produzindo também uma previsão de cumprimento de deadline. No caso do previsor MEQS, como ele divide igualmente a folga entre os segmentos locais da thread distribuída sem levar em consideração o tempo de computação de cada segmento local, a previsão pode indicar o cumprimento do deadline fim a fim, mas o tempo de resposta efetivo nos nodos seguintes pode ultrapassar o deadline fim a fim gerando uma falsa previsão de cumprimento de deadline.

O deadline 500 é considerado um deadline justo, onde cerca da metade das threads distribuídas do sistema cumprem o deadline fim a fim e a outra metade não cumpre. Neste tipo de deadline torna-se mais difícil realizar previsões corretas e os resultados dos mecanismos de previsão ganham mais importância. Neste deadline, o mecanismo MEQF apresentou a menor taxa de erro.

Se o intervalo de confiança de cada previsor para cada deadline for utilizado, pode-se afirmar que houve um empate entre os previsores MED, MEQS e MEQF, porque em todos eles, em algum deadline, a taxa de erro se sobrepõe à taxa de erro de outro mecanismo.

O gráfico da figura 5.5 mostra os valores da tabela 5.1. A taxa de erro dos mecanismos MED, MEQS e MEQF são próximas para os deadlines 100 e 900. Nos demais deadlines (300, 500 e 700), o mecanismo MEQF mantém as taxas de erro menores em relação aos demais mecanismos. O mecanismo Ultimate, na maioria dos deadlines, apresenta a maior taxa de erro. No deadline 900, o *Ultimate* permanece próximo dos demais mecanismos.

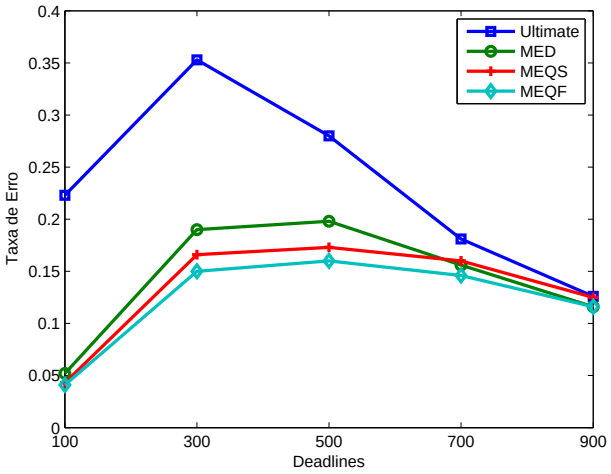

Figura 5.5: *Taxa de Erro dos Previsores Milestones para Threads Distribuídas tipo Pipeline.*

A tabela 5.2 apresenta os resultados das simulações realizadas com a Carga 1, segundo a métrica Taxa de Previsões Corretas ($PC(z)$). O previsor MEQF apresenta a maior taxa de previsões corretas em todos os deadlines, com exceção do deadline com valor 100, onde o resultado é próximo do previsor MEQS.

Considerando o intervalo de confiança de cada mecanismo, observa-se um empate entre eles, onde o resultado de um se sobrepõe ao resultado de outro.

O gráfico da figura 5.6 mostra os valores da tabela 5.2. O previsor *Ultimate* obteve a pior taxa de previsões corretas na maioria dos deadlines, com exceção do deadline 900, onde seu resultado está próximo dos demais mecanismos. O previsor MEQF obteve a melhor taxa de previsões corretas na maioria dos deadlines. No deadline 100, os previsores MED, MEQS e MEQF apresentaram resultados próximos.

Em geral, para um sistema composto somente por threads distribuídas do tipo *pipeline* e que segue as premissas definidas neste trabalho, é possível afirmar que o mecanismo MEQF apresenta melhores resultados, tanto com relação a taxa de erro quanto com relação a taxa de previsões corretas.

Tabela 5.2: Taxa de Previsões Corretas para Threads Distribuídas tipo *Pipeline*

Mecanismos de Previsão	*Deadline, Taxa de Previsões Corretas (PC) e Intervalo de Confiança (IC)*									
	100		300		500		700		900	
	PC	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,705	±0,089	0,6	±0,096	0,714	±0,089	0,831	±0,073	0,893	±0,061
MED-MaNS	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,880	±0,064	0,906	±0,057
MED-MeNS	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,880	±0,064	0,906	±0,057
MED-Pond	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,88	±0,064	0,906	±0,057
MED-MProv	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,880	±0,064	0,906	±0,057
MEQS-MaNS	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQS-MeNS	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQS-Pond	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQS-MProv	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQF-MaNS	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041
MEQF-MeNS	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041
MEQF-Pond	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041
MEQF-MProv	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041

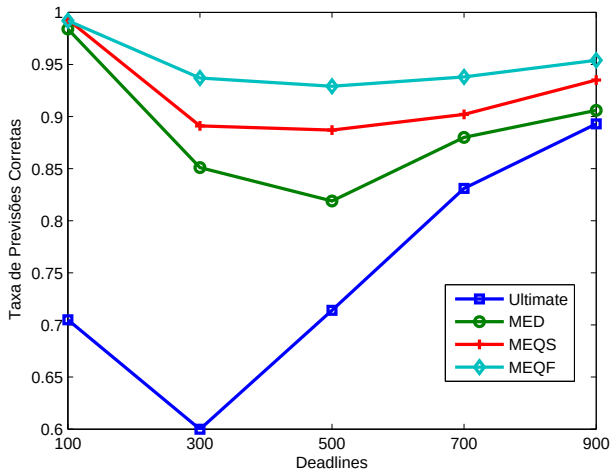

Figura 5.6: *Taxa de Previsões Corretas para Threads Distribuídas tipo Pipeline.*

5.4.2 Simulações com a Carga 2 - *Threads Distribuídas Tipo Árvore Balanceada*

Neste tipo de carga, que considera caminhos de execução alternativos que a thread distribuída pode seguir, os resultados variam conforme o suposto itinerário definido para fins de previsão. A tabela 5.3 apresenta os resultados das simulações realizadas com a Carga 2, segundo a métrica Taxa de Erro Relativo ($E(z)$).

Neste tipo de carga, não houve um previsor que apresentou a menor taxa de erro em

Tabela 5.3: Taxa Relativa de Erro para Threads Distribuídas tipo Árvore Balanceada

	Deadline, Taxa de Erro e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,413	±0,019	0,379	±0,017	0,203	±0,017	0,120	±0,016	0,078	±0,012
MED-MaNS	**0,038**	±0,005	0,263	±0,011	0,178	±0,011	**0,102**	±0,011	**0,064**	±0,009
MED-MeNS	0,102	±0,007	0,301	±0,011	0,186	±0,013	0,110	±0,013	0,070	±0,011
MED-Pond	0,053	±0,005	0,279	±0,011	0,181	±0,012	0,104	±0,012	0,066	±0,010
MED-MProv	0,067	±0,007	0,276	±0,011	0,181	±0,012	0,105	±0,012	0,067	±0,010
MEQS-MaNS	0,051	±0,005	0,223	±0,011	0,190	±0,008	0,120	±0,010	0,072	±0,009
MEQS-MeNS	0,092	±0,005	0,251	±0,012	0,189	±0,009	0,116	±0,010	0,071	±0,009
MEQS-Pond	0,065	±0,005	0,229	±0,011	0,190	±0,008	0,119	±0,010	0,072	±0,009
MEQS-MProv	0,067	±0,005	0,233	±0,011	0,189	±0,009	0,117	±0,010	0,071	±0,009
MEQF-MaNS	0,049	±0,004	**0,213**	±0,010	**0,172**	±0,008	0,110	±0,009	0,068	±0,009
MEQF-MeNS	0,099	±0,006	0,252	±0,012	0,175	±0,009	0,109	±0,010	0,068	±0,009
MEQF-Pond	0,064	±0,005	0,220	±0,010	**0,171**	±0,008	0,109	±0,009	0,068	±0,009
MEQF-MProv	0,069	±0,006	0,227	±0,011	**0,171**	±0,008	0,108	±0,009	0,068	±0,009

todos os deadlines. O previsor MED obteve as menores taxas de erros nos deadlines 100, 700 e 900, com o itinerário *Maior Número de Saltos*. Neste tipo de deadline, as previsões são mais fáceis de serem realizadas porque os deadlines em questão ou são muito apertados (indicando sua provável perda) ou são muito folgados (indicando seu provável cumprimento).

O previsor MEQF obteve as menores taxas de erros nos deadlines 300 e 500, também com o itinerário *Maior Número de Saltos*. Neste tipo de deadline, as previsões são mais difíceis de serem realizadas porque os deadlines em questão são justos, isto é, nem muito apertado nem muito folgado e uma thread distribuída tem a mesma chance de cumprir ou de perder um deadline deste tipo. O fato do previsor MEQF ter apresentando a menor taxa de erro nesta faixa de deadlines indica sua superioridade em relação aos demais previsores.

Considerando os valores próximos em 1% da menor taxa de erro encontrada em um determinado deadline, o previsor *MEQF-MaiorNSaltos*, no deadline 500, apresenta resultado próximo dos previsores *MEQF-Ponderado* e *MEQF-MaisProvável*. O previsor *Ultimate* apresentou a maior taxa de erro em todos os deadlines.

As figuras 5.7, 5.8 e 5.9 mostram gráficos com cada previsor e os respectivos itinerários usados. O comportamento dos itinerários é semelhante nos previsores MED, MEQS e MEQF, isto é, o itinerário *Menor Número de Saltos* (MeNS) apresenta a maior taxa de erro nos deadlines 100 e 300 e o itinerário *Maior Número de Saltos* (MaNS) apresenta a menor taxa

de erro nestes mesmos deadlines. Entretanto, a diferença destes itinerários com relação aos itinerários *Ponderado* (Pond) e *Mais Provável* (MProv) não é significativa. Nos deadlines 500, 700 e 900, a diferença na taxa de erro entre os itinerários destes previsores é pequena.

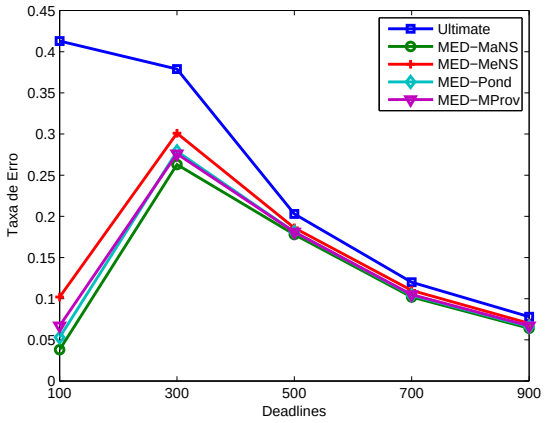

Figura 5.7: Erro do Previsor MED para TDs tipo Árvore Balanceada.

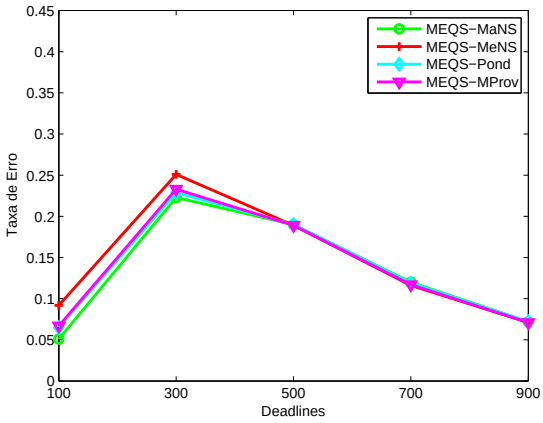

Figura 5.8: Erro do Previsor MEQS para TDs tipo Árvore Balanceada.

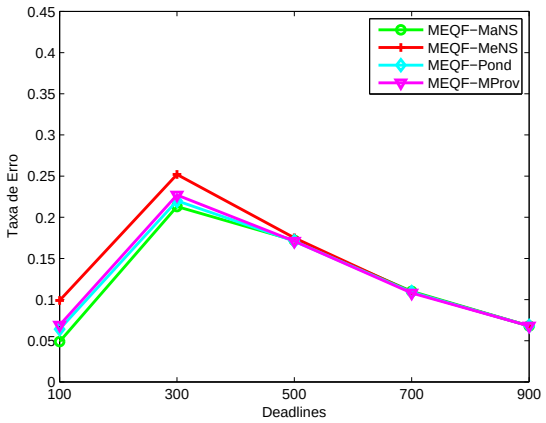

Figura 5.9: Erro do Previsor MEQF para TDs tipo Árvore Balanceada.

A tabela 5.4 apresenta os resultados das simulações realizadas com a Carga 2, segundo a métrica Taxa de Previsões Corretas ($PC(z)$). Para os deadlines justos (onde cerca da metade das threads distribuídas cumpre o deadline e a outra metade não cumpre), que variam de 300 a 700, o previsor MEQF apresenta as maiores taxas de previsões corretas em relação aos demais previsores. No deadline 900, considerando uma variação de 1% sobre a melhor taxa de previsão correta ($0,978$ do previsor *MEQF-MeNS*), os previsores *MED-MaNS*, *MED-Pond*, *MEQS-Pond*, *MEQS-MProv* e todos os demais itinerários do previsor MEQF apresentam

taxas de previsões corretas próximas.

Tabela 5.4: Taxa de Previsões Corretas para Threads Distribuídas tipo *Árvore Balanceada*

	Deadline, Taxa de Previsões Corretas (PC) e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,482	±0,098	0,602	±0,096	0,817	±0,076	0,905	±0,058	0,937	±0,048
MED-MaNS	0,983	±0,025	0,808	±0,077	0,871	±0,066	0,941	±0,046	**0,974**	±0,031
MED-MeNS	0,932	±0,049	0,737	±0,086	0,854	±0,069	0,928	±0,051	0,961	±0,038
MED-Pond	0,978	±0,028	0,778	±0,081	0,862	±0,068	0,940	±0,046	**0,971**	±0,033
MED-MProv	0,949	±0,043	0,782	±0,081	0,865	±0,067	0,937	±0,048	0,967	±0,035
MEQS-MaNS	0,988	±0,021	0,868	±0,066	0,883	±0,063	0,940	±0,047	0,967	±0,035
MEQS-MeNS	0,977	±0,029	0,826	±0,074	0,878	±0,064	0,942	±0,046	0,970	±0,034
MEQS-Pond	0,988	±0,021	0,861	±0,068	0,886	±0,062	0,942	±0,046	**0,969**	±0,034
MEQS-MProv	0,980	±0,027	0,848	±0,070	0,885	±0,063	0,943	±0,046	**0,969**	±0,034
MEQF-MaNS	0,988	±0,021	**0,904**	±0,058	**0,916**	±0,054	**0,951**	±0,042	**0,972**	±0,032
MEQF-MeNS	0,978	±0,029	0,847	±0,070	0,900	±0,059	**0,953**	±0,042	**0,978**	±0,029
MEQF-Pond	**0,990**	±0,019	0,900	±0,059	**0,919**	±0,053	**0,954**	±0,041	**0,974**	±0,031
MEQF-MProv	0,981	±0,027	0,884	±0,063	**0,913**	±0,055	**0,952**	±0,042	**0,974**	±0,031

As figuras 5.10 5.11 5.12 mostram os valores da tabela 5.4. A menor taxa de previsões corretas foi do mecanismo Ultimate, seguido pelos mecanismos MED e MEQS. O previsor MEQF obteve a maior taxa de previsões corretas neste tipo de carga. É possível observar que nos três previsores baseados em *Milestones*, os itinerários MaNS e MeNS obtiveram, respectivamente, a maior e a menor taxa de previsões corretas, nos deadlines 300 e 500. Nos demais deadlines, os resultados entre os itinerários são próximos.

Os resultados da primeira métrica mostram que não houve um previsor que se manteve com a menor taxa de erro em todas as faixas de deadlines. O previsor MED-MaNS obteve os melhores resultados nos deadlines 100, 700 e 900. Já o previsor MEQF com os itinerários MaNS, Pond e MProv obteve as menores taxas de erros nos deadlines 300 e 500.

Na segunda métrica, o previsor MEQF apresentou melhores resultados que os demais previsores em todos os deadlines. Não é possível destacar um itinerário como melhor em todos os deadlines simulados. Nos deadlines 300 e 500 o itinerário MaNS obteve a maior taxa de previsões corretas.

Os resultados de ambas as métricas são especialmente importantes para deadlines justos, onde torna-se mais difícil realizar previsões corretas já que as tarefas não possuem deadlines folgados o suficiente para garantir seu cumprimento e nem deadlines apertados o sufi-

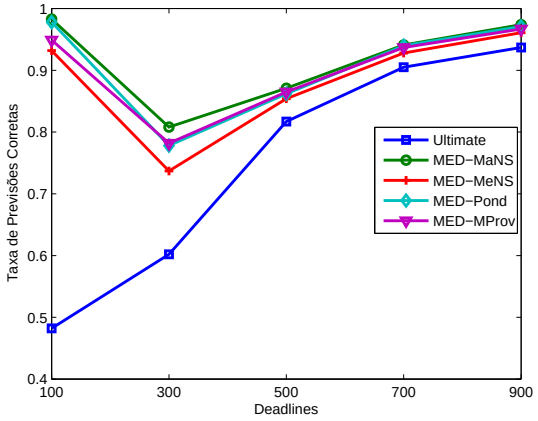

Figura 5.10: PC do Mecanismo MED para TDs tipo Árvore Balanceada.

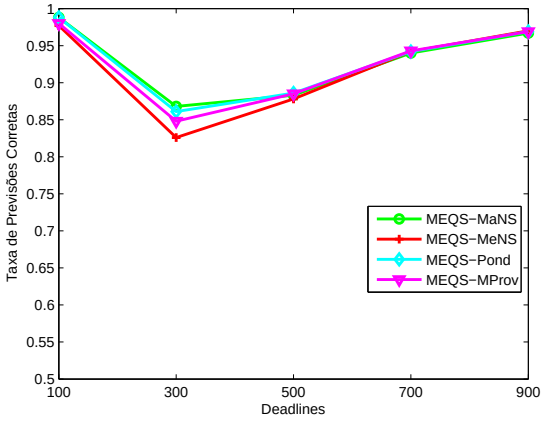

Figura 5.11: PC do Mecanismo MEQS para TDs tipo Árvore Balanceada.

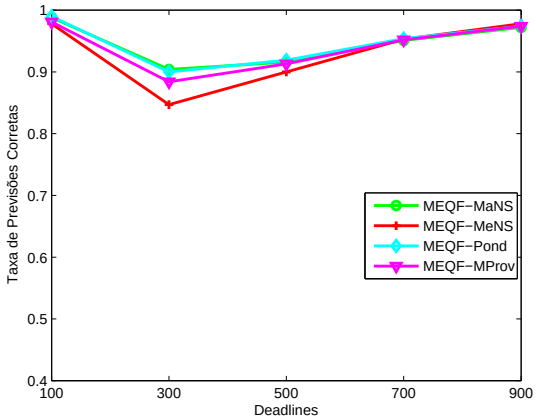

Figura 5.12: PC do Mecanismo MEQF para TDs tipo Árvore Balanceada.

ciente que indiquem a sua provável perda. Nesta faixa de deadlines, o previsor MEQF obteve os melhores resultados.

5.4.3 Simulações com a Carga 3 - *Threads Distribuídas Tipo Árvore Não Balanceada*

A tabela 5.5 apresenta os resultados das simulações realizadas com a Carga 3, segundo a métrica Taxa de Erro Relativo ($E(z)$). Com threads distribuídas do tipo Árvore Não Balanceada, nenhum previsor manteve a menor taxa de erro em todos os deadlines. O previsor MED obteve a menor taxa de erro no deadline 100 com o itinerário *MaNS*, e nos deadlines 500, 700 e 900 com o itinerário *MProv*. O previsor MEQF obteve a menor taxa de erro no deadline 300 com o itinerário *MaNS*.

No deadline 300, considerando uma variação de 1% sobre a menor taxa de erro, o previsor *MEQS-MaNS* apresenta taxa de erro próxima do previsor *MEQF-MaNS*. O previsor *Ultimate* apresentou a maior taxa de erro nos deadlines 100 a 500.

Considerando, em cada deadline, os resultados próximos em 1% da menor taxa de erro, percebe-se que apenas o previsor MEQS-MaNS tem um valor próximo do previsor MEQF-MaNS no deadline 300.

Usando a menor taxa de erro de cada mecanismo e o respectivo intervalo de confiança, observa-se situações de empate nos deadlines 100 e 900 entre os previsores MED e MEQF, e entre os previsores MEQS e MEQF. Nos deadlines 300 e 700, os previsores MEQS e MEQF empatam. Não ocorrem situações de empate entre os previsores no deadline 500.

As figuras 5.13, 5.14 e 5.15 mostram gráficos com cada previsor e os respectivos itinerários usados. Na figura 5.13, o previsor Ultimate apresenta a maior taxa de erro para os deadlines 100, 300 e 500.

Para os deadlines 700 e 900, nenhum previsor apresenta um resultado significativamente melhor que os demais. O itinerário *Menor Número de Saltos* (MeNS) apresenta a maior taxa de erro nos deadlines 100, 300 e 500.

A tabela 5.6 apresenta os resultados das simulações realizadas com a carga 3, segundo a métrica Taxa de Previsões Corretas ($PC(z)$). O previsor MEQF apresentou a maior taxa de acertos em todos os deadlines. No deadline 100, considerando uma variação de 1% na maior

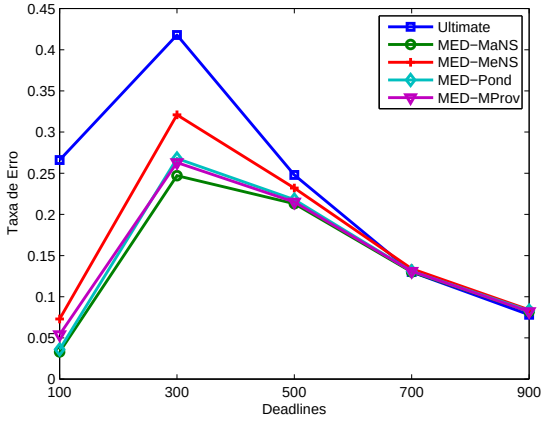

Figura 5.13: Erro do Previsor MED para TDs tipo Árvore Não Balanceada.

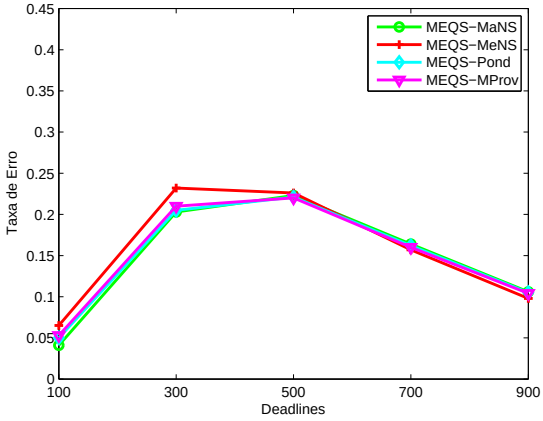

Figura 5.14: Erro do Previsor MEQS para TDs tipo Árvore Não Balanceada.

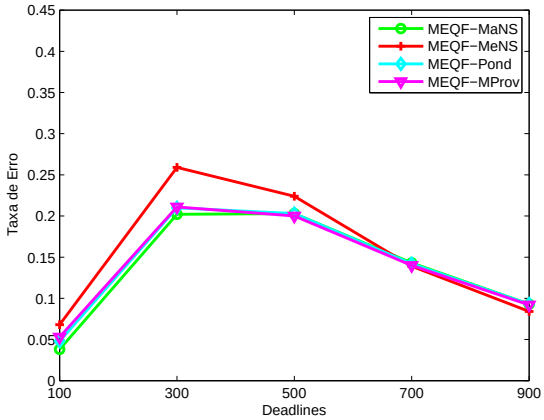

Figura 5.15: Erro do Previsor MEQF para TDs tipo Árvore Não Balanceada.

Tabela 5.5: Taxa Relativa de Erro para Threads Distribuídas tipo Árvore Não Balanceada

	Deadline, Taxa de Erro e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,266	±0,016	0,418	±0,013	0,248	±0,014	0,130	±0,012	0,078	±0,012
MED-MaNS	**0,033**	±0,004	0,247	±0,001	0,213	±0,009	0,130	±0,010	0,082	±0,010
MED-MeNS	0,073	±0,006	0,321	±0,001	0,232	±0,011	0,134	±0,011	0,084	±0,011
MED-Pond	0,036	±0,004	0,268	±0,001	0,218	±0,010	0,131	±0,010	0,083	±0,011
MED-MProv	0,054	±0,006	0,276	±0,001	**0,181**	±0,010	**0,105**	±0,010	**0,067**	±0,011
MEQS-MaNS	0,041	±0,003	**0,203**	±0,009	0,223	±0,008	0,164	±0,010	0,106	±0,009
MEQS-MeNS	0,065	±0,005	0,232	±0,009	0,226	±0,009	0,157	±0,010	0,098	±0,009
MEQS-Pond	0,050	±0,004	0,205	±0,009	0,221	±0,008	0,162	±0,010	0,105	±0,009
MEQS-MProv	0,053	±0,004	0,210	±0,009	0,220	±0,008	0,160	±0,010	0,104	±0,009
MEQF-MaNS	0,038	±0,004	**0,202**	±0,009	0,203	±0,007	0,143	±0,009	0,093	±0,008
MEQF-MeNS	0,068	±0,005	0,259	±0,009	0,224	±0,010	0,139	±0,010	0,084	±0,009
MEQF-Pond	0,048	±0,004	0,210	±0,009	0,203	±0,007	0,142	±0,009	0,093	±0,008
MEQF-MProv	0,053	±0,004	0,211	±0,009	0,200	±0,007	0,140	±0,009	0,092	±0,008

taxa de previsão correta desta faixa de deadline ($0,993$ do previsor *MEQF-Ponderado*), o previsor MEQS e todos os seus itinerários obtiveram resultados próximos da melhor taxa de previsão correta.

Usando a maior taxa de previsão correta de cada mecanismo e o respectivo intervalo de confiança, observa-se situações de empate em todos os deadlines entre todos os previsores. No deadline 500, por exemplo, o previsor MED obteve uma taxa de previsão correta igual a $0,842$, com intervalo de confiança de $\pm 0,072$, significando que o valor desta taxa pode variar até $0,914$. O previsor MEQF obteve uma taxa igual a $0,911$ que pode variar até $0,855$, já que o intervalo de confiança é de $\pm 0,056$. O previsor MEQS obteve uma taxa igual a $0,856$ que pode variar de $0,787$ a $0,925$ porque o intervalo de confiança é igual a $\pm 0,069$.

Analisando individualmente cada previsor, é interessante observar que as maiores taxas de previsões corretas do previsor MED, em todos os deadlines, ocorreram com o itinerário MaNS. Com o previsor MEQS, não houve um itinerários que obteve os melhores resultados em todos os deadlines simulados. Com o previsor MEQF, o itinerário Pond obteve a maior taxa de previsões corretas nos deadlines 100, 300, 500 e 900, e o itinerário MProv obteve a maior taxa de previsões corretas no deadline 700.

As figuras 5.16 5.17 5.18 mostram os valores da tabela 5.6. A menor taxa de previsões corretas foi do mecanismo *Ultimate*, seguido pelos mecanismos MED e MEQS. O previsor

Tabela 5.6: Taxa de Previsões Corretas para Threads Distribuídas tipo *Árvore Não Balanceada*

	Deadline, Taxa de Previsões Corretas (PC) e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,688	±0,091	0,519	±0,098	0,770	±0,082	0,898	±0,059	0,951	±0,043
MED-MaNS	0,982	±0,026	0,809	±0,077	0,842	±0,072	0,913	±0,055	0,957	±0,040
MED-MeNS	0,939	±0,047	0,661	±0,093	0,803	±0,078	0,905	±0,057	0,953	±0,042
MED-Pond	0,980	±0,027	0,771	±0,082	0,830	±0,074	0,911	±0,056	0,955	±0,040
MED-MProv	0,954	±0,041	0,771	±0,082	0,835	±0,073	0,911	±0,056	0,956	±0,040
MEQS-MaNS	**0,990**	±0,019	0,871	±0,066	0,846	±0,071	0,900	±0,059	0,946	±0,044
MEQS-MeNS	0,987	±0,022	0,879	±0,064	0,852	±0,070	0,908	±0,057	0,955	±0,041
MEQS-Pond	**0,990**	±0,019	0,885	±0,063	0,854	±0,069	0,905	±0,057	0,949	±0,043
MEQS-MProv	**0,990**	±0,019	0,883	±0,063	0,856	±0,069	0,910	±0,056	0,951	±0,042
MEQF-MaNS	**0,991**	±0,018	**0,907**	±0,057	**0,903**	±0,058	**0,941**	±0,046	**0,966**	±0,036
MEQF-MeNS	**0,987**	±0,022	0,834	±0,073	0,851	±0,070	0,922	±0,052	**0,963**	±0,037
MEQF-Pond	**0,993**	±0,017	**0,917**	±0,054	**0,911**	±0,056	**0,944**	±0,045	**0,968**	±0,035
MEQF-MProv	**0,991**	±0,018	**0,908**	±0,057	**0,910**	±0,056	**0,945**	±0,045	**0,968**	±0,035

MEQF obteve a maior taxa de previsões corretas neste tipo de carga. É possível observar que o itinerário MeNS dos previsores MED e MEQF obteve a menor taxa de previsões corretas na maioria dos deadlines. Os itinerários do previsor MEQS alcançaram resultados próximos sendo que nenhum deles se destacou com um resultado significativamente melhor que os demais.

Para árvores não balanceadas, os resultados da primeira métrica não mostram um previsor com resultados significativamente melhores que os demais em todas as faixas de deadlines. O previsor MEQF-MProv obteve a menor taxa de erro no deadline 500.

A segunda métrica revela que o previsor MEQF-Pond apresenta a maior taxa de previsões corretas nos deadlines 100, 300 e 500, e o previsor MEQF-MProv nos deadlines 700 e 900. Entretanto, a diferença de resultados entre estes previsores é pequena. Este resultado reafirma a capacidade do previsor MEQF em realizar previsões corretas em sistemas com deadlines justos.

5.4.4 Simulações com a Carga 4 - *Threads Distribuídas Carga Mista*

A tabela 5.7 apresenta os resultados das simulações realizadas com a Carga 4, segundo a métrica Taxa de Erro Relativo ($E(z)$). Nos deadlines 300 e 500 o previsor MEQF apresentou

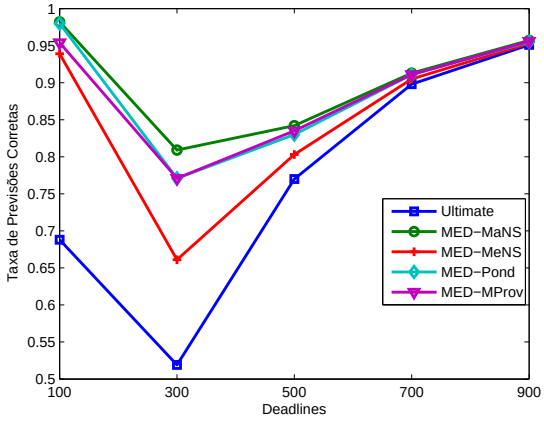

Figura 5.16: PC do Mecanismo MED para TDs tipo Árvore Não Balanceada.

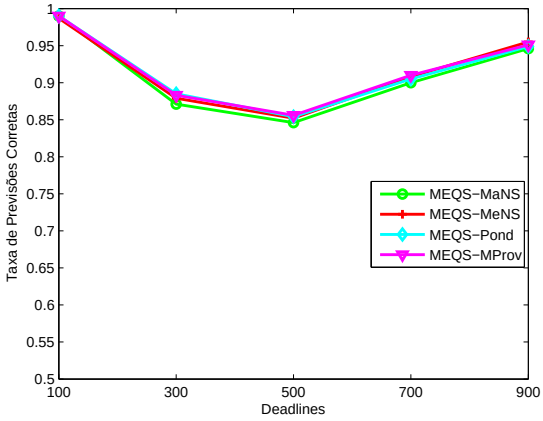

Figura 5.17: PC do Mecanismo MEQS para TDs tipo Árvore Não Balanceada.

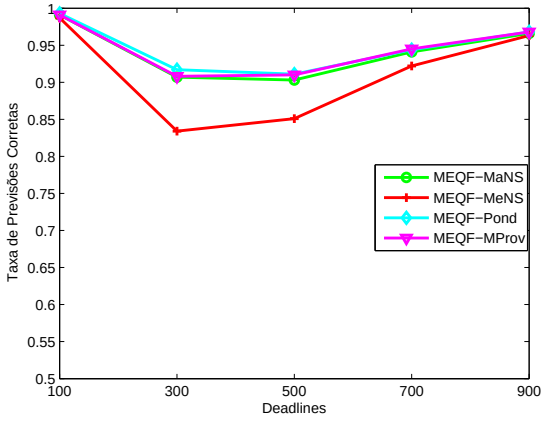

Figura 5.18: PC do Mecanismo MEQF para TDs tipo Árvore Não Balanceada.

a menor taxa de erro, com os itinerários MaNS e MProv, respectivamente. Nos demais deadlines, o previsor MED com o itinerário MaNS apresentou a menor taxa de erro. No deadline 700, considerando uma variação de 1% na menor taxa de erro, os itinerários Pond e MProv do previsor MED apresentaram resultados próximos do itinerário MaNS .

Neste tipo de carga não é possível afirmar que um previsor obteve resultados significativamente melhores que os demais previsores em todos os deadlines. Também não é possível afirmar que um determinado itinerário manteve bons resultados em todas as faixas de deadlines. Entretanto, conforme ocorreu nas outras cargas simuladas, o previsor MEQF (com os itinerários MaNS e MProv) apresentou os melhores resultados em deadlines justos (300 e 500), onde existe maior dificuldade em realizar previsões corretas.

Usando a menor taxa de erro de cada mecanismo e o respectivo intervalo de confiança, observa-se situações de empate entre os três previsores Milestones, nos deadlines 100, 500, 700 e 900. No deadline 300, ocorreu um empate entre os previsores MEQF e MEQS, ambos com o itinerário MaNS.

Analisando individualmente cada previsor, é interessante observar que as menores taxas de erro do previsor MED, em todos os deadlines, ocorreram com o itinerário MaNS. Com o previsor MEQS, não houve um itinerários que obteve os melhores resultados em todos os deadlines simulados. Com o previsor MEQF, o itinerário MaNS obteve a menor taxa de erro nos deadlines 100 e 300; o itinerário MProv nos deadlines 500 e 700, e o itinerário MeNS no deadline 500.

As figuras 5.19, 5.20 e 5.21 mostram gráficos com cada previsor e os respectivos itinerários usados. Os resultados mostram o mesmo padrão de comportamento dos previsores nas outras cargas simuladas. Na figura 5.19, o previsor *Ultimate* apresenta a maior taxa de erro para os deadlines 100, 300 e 500.

Para os deadlines 700 e 900, nenhum previsor apresenta um resultado significativamente melhor que os demais. O itinerário *Menor Número de Saltos* (MeNS) apresenta a maior taxa de erro nos deadlines 100, 300 e 500 e o itinerário *Maior Número de Saltos* (MaNS) apresenta a menor taxa de erro nestes deadlines.

A tabela 5.8 apresenta os resultados das simulações realizadas com a Carga 4, segundo a métrica Taxa de Previsões Corretas ($PC(z)$). O previsor MEQF apresentou a maior taxa

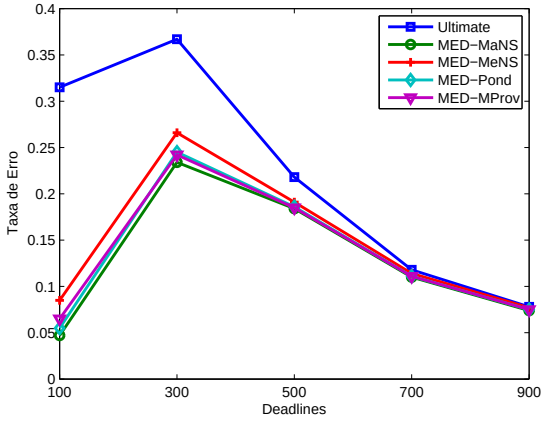

Figura 5.19: Erro do Previsor MED para TDs Carga Mista.

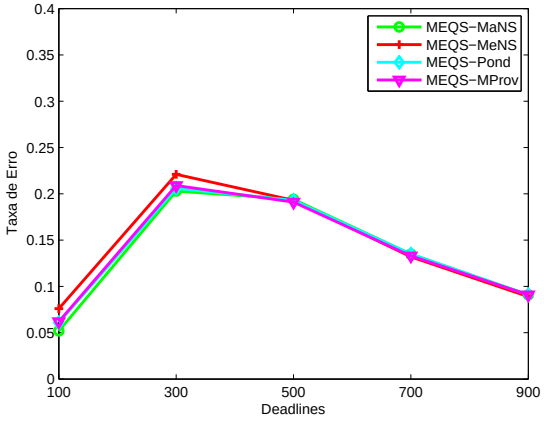

Figura 5.20: Erro do Previsor MEQS para TDs Carga Mista.

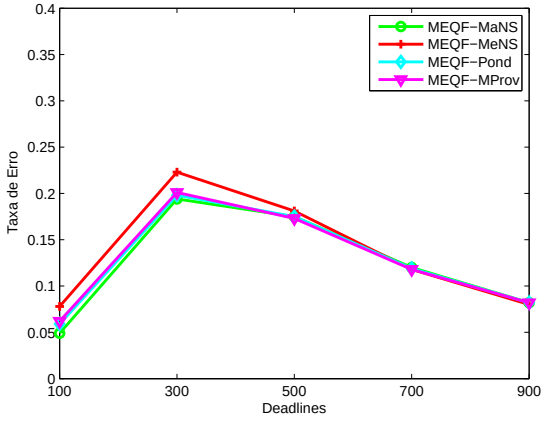

Figura 5.21: Erro do Previsor MEQF para TDs Carga Mista.

Tabela 5.7: Taxa Relativa de Erro para Threads Distribuídas - Carga Mista

	Deadline, Taxa de Erro e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,315	±0,017	0,367	±0,016	0,218	±0,020	0,118	±0,016	0,078	±0,014
MED-MaNS	**0,047**	±0,005	0,234	±0,010	0,184	±0,013	**0,110**	±0,012	**0,074**	±0,012
MED-MeNS	0,085	±0,008	0,266	±0,011	0,191	±0,014	0,114	±0,013	0,077	±0,012
MED-Pond	0,055	±0,006	0,245	±0,010	0,186	±0,013	**0,111**	±0,012	0,075	±0,012
MED-MProv	0,065	±0,007	0,242	±0,010	0,185	±0,013	**0,111**	±0,012	0,075	±0,012
MEQS-MaNS	0,052	±0,005	0,203	±0,011	0,194	±0,010	0,135	±0,011	0,091	±0,011
MEQS-MeNS	0,076	±0,006	0,221	±0,011	0,193	±0,010	0,132	±0,011	0,089	±0,011
MEQS-Pond	0,061	±0,005	0,206	±0,011	0,193	±0,010	0,135	±0,011	0,091	±0,011
MEQS-MProv	0,062	±0,006	0,209	±0,011	0,191	±0,010	0,133	±0,011	0,091	±0,011
MEQF-MaNS	0,049	±0,005	**0,194**	±0,010	0,175	±0,009	0,120	±0,010	0,082	±0,011
MEQF-MeNS	0,078	±0,006	0,223	±0,010	0,181	±0,011	0,118	±0,011	0,080	±0,011
MEQF-Pond	0,059	±0,005	0,198	±0,010	0,175	±0,009	0,119	±0,010	0,082	±0,011
MEQF-MProv	0,062	±0,006	0,201	±0,010	**0,173**	±0,009	0,118	±0,010	0,082	±0,011

de previsão correta em relação aos demais previsores, em todos os deadlines.

No deadline 100, considerando uma variação de 1% sobre a menor taxa de erro (0.989 do previsor *MEQF-MaNS* e *MEQF-Pond*), os previsores MED-MaNS e MEQS (MaNS, MeNS, Pond e MProv) apresentam resultados próximos.

Não houve um itinerário do previsor MEQF que tenha apresentado a maior taxa de previsão correta em todos os deadlines. No deadline 100, por exemplo, os melhores resultados foram dos itinerários Maior Número de Saltos (MaNS) e Ponderado (Pond). No deadline 300, o melhor resultado foi do itinerário Ponderado. No deadline 500, o itinerário Mais Provável obteve o melhor resultado. Já no deadline 700, os itinerários Ponderado e Mais Provável apresentaram os melhores resultados e no deadline 900 os itinerários Menos Número de Saltos, Ponderado e Mais Provável apresentaram as maiores taxa de previsões corretas.

As figuras 5.22 5.23 5.24 mostram os valores da tabela 5.8. O padrão de comportamento dos previsores é o mesmo encontrado na Carga 3, isto é, O previsor Ultimate obteve a menor taxa de previsões corretas, seguido pelos mecanismos MED e MEQS. A maior taxa de previsões corretas foi alcançada pelo previsor MEQF. O itinerário MeNS dos previsores MED e MEQF obteve a menor taxa de previsões corretas na maioria dos deadlines. No previsor MEQS nenhum itinerário se destacou com um resultado significativamente melhor que os demais.

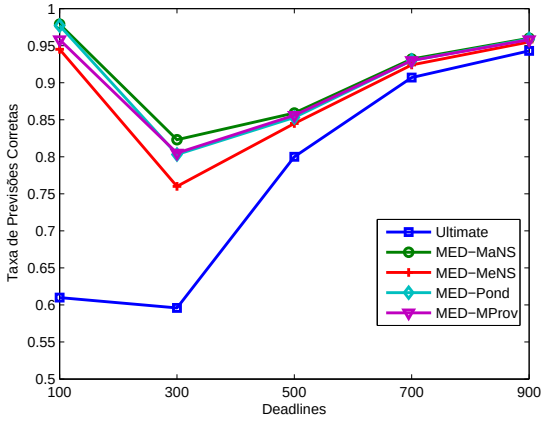

Figura 5.22: PC do Mecanismo MED para TDs Carga Mista.

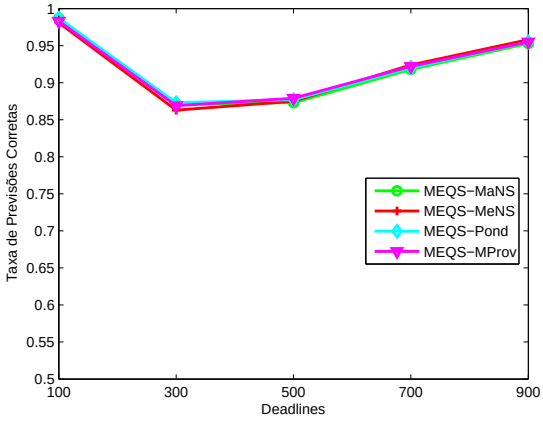

Figura 5.23: PC do Mecanismo MEQS para TDs Carga Mista.

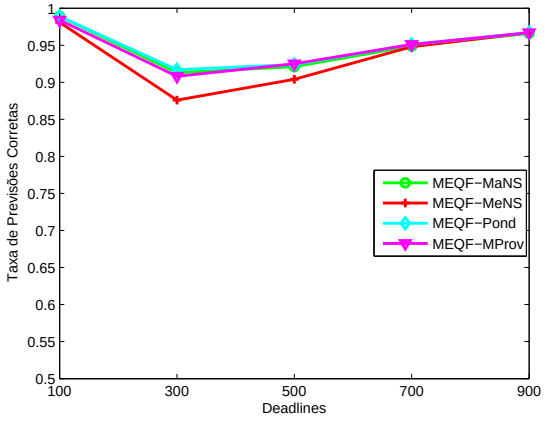

Figura 5.24: PC do Mecanismo MEQF para TDs Carga Mista.

Tabela 5.8: Taxa de Previsões Corretas para Threads Distribuídas tipo *Carga Mista*

	Deadline, Taxa de Previsões Corretas (PC) e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,61	±0,096	0,596	±0,096	0,800	±0,078	0,907	±0,057	0,943	±0,045
MED-MaNS	**0,979**	±0,028	0,823	±0,075	0,859	±0,068	0,932	±0,049	**0,960**	±0,038
MED-MeNS	0,945	±0,045	0,760	±0,084	0,845	±0,071	0,924	±0,052	0,955	±0,041
MED-Pond	0,977	±0,029	0,803	±0,078	0,853	±0,069	0,930	±0,050	**0,959**	±0,039
MED-MProv	0,958	±0,039	0,805	±0,078	0,856	±0,069	0,930	±0,050	**0,958**	±0,039
MEQS-MaNS	**0,987**	±0,022	0,871	±0,066	0,873	±0,065	0,918	±0,054	0,953	±0,041
MEQS-MeNS	**0,981**	±0,027	0,863	±0,067	0,875	±0,065	0,924	±0,052	**0,958**	±0,039
MEQS-Pond	**0,987**	±0,022	0,873	±0,065	0,877	±0,064	0,920	±0,053	0,955	±0,041
MEQS-MProv	**0,983**	±0,025	0,869	±0,066	0,879	±0,064	0,922	±0,053	0,955	±0,040
MEQF-MaNS	**0,989**	±0,021	**0,913**	±0,055	**0,921**	±0,053	**0,949**	±0,043	**0,966**	±0,035
MEQF-MeNS	**0,981**	±0,027	0,876	±0,065	0,904	±0,058	**0,948**	±0,044	**0,967**	±,035
MEQF-Pond	**0,989**	±0,020	**0,917**	±0,054	**0,924**	±0,052	**0,951**	±0,042	**0,967**	±0,035
MEQF-MProv	**0,984**	±0,024	0,908	±0,057	**0,925**	±0,052	**0,951**	±0,042	**0,967**	±0,035

Na carga mista, os resultados da primeira métrica revelam que nenhum previsor apresentou resultados significativamente melhores que os demais em todas as faixas de deadlines. O previsor MEQF-MProv obteve a menor taxa de erro no deadline 500.

A segunda métrica mostra que o previsor MEQF obteve a maior taxa de previsões corretas em todos os deadlines simulados. Entretanto, nenhum itinerário do previsor MEQF alcançou a maior taxa de previsões corretas em todos os deadlines. Em alguns deadlines, o itinerário Pond obteve os melhores resultados e em outros deadlines, os itinerários MProv e Mans obtiveram as maiores taxas de previsões corretas.

5.5 Conclusão

Este capítulo descreveu os mecanismos baseados em *Milestones*, que definem um tempo de resposta estimado para a thread distribuída considerando apenas informações previamente conhecidas de execuções passadas.

Foram propostas três formas de definir os *Milestones*, usando os métodos de particionamento de deadlines propostos em [6] como base para a criação dos tempos de respostas estimados (*Milestones*). Após a execução da thread distribuída em um nodo, é verificado se o tempo de resposta local efetivo é igual ou maior que metade do deadline fim a fim. Se

esta condição for verdadeira, o mecanismo de previsão é acionado. O mecanismo calcula a probabilidade da thread distribuída cumprir seu deadline fim a fim subtraindo o valor do *Milestone* daquele nodo pelo tempo de resposta local efetivo. Ao resultado desta subtração adiciona-se o valor $0,5$, indicando que se os valores do *Milestone* e do tempo de resposta local efetivo forem exatamente iguais, a thread distribuída possui 50% de probabilidade de cumprir seu deadline fim a fim.

Os mecanismos propostos são simples e implicam em baixo overhead porque os cálculos para a definição dos *Milestones* podem ser realizados estaticamente, no nodo origem da thread distribuída.

Para as simulações, duas métricas foram utilizadas, *Taxa Relativa de Erro* e *Taxa de Previsões Corretas*. A primeira mostra o quão distante foi a previsão em relação ao que de fato aconteceu e a outra calcula a taxa de previsões corretas sobre o total de previsões realizadas.

Na métrica *Taxa Relativa de Erro*, o previsor MEQF destacou-se na carga 1 (*pipeline*) com a menor taxa de erro entre os previsores em todos os deadlines simulados. Nas demais cargas, não houve um previsor com resultados significativamente melhores que os outros em todos os deadlines. Entretanto, com uma pequena diferença nos resultados, o previsor MEQF com o itinerário *Maior Número de Saltos* manteve-se melhor que os demais previsores na maioria das cargas e deadlines simulados.

Na métrica Taxa de Previsões Corretas, o previsor MEQF apresentou os melhores resultados nas cargas 1 (*pipeline*) e 3 (árvores não balanceadas). Com relação aos itinerários, nenhum se manteve como melhor em todas as cargas e deadlines simulados. Houve uma alternância principalmente entre o itinerário *Maior Número de Saltos - MaNS*, *Ponderado - Pond* e *Mais Provável - MProv*, com uma pequena diferença em favor do itinerário *MaNS*.

A principal fragilidade dos mecanismos baseados em *Milestones* se refere à forma como eles utilizam os possíveis itinerários de uma thread distribuída para gerar os *Milestones*. O grafo que representa a thread distribuída é desmembrado em *pipelines*, sendo que um determinado itinerário é usado como critério para realizar este desmembramento. Por exemplo, se o itinerário escolhido para gerar os *Milestones* é o *Maior Número de Saltos*, busca-se no grafo o *pipeline* com o maior número de nodos que uma thread distribuída possa executar em uma determinada ativação. Os *Milestones* são gerados a partir deste pipeline, usando os tempos

médios de computação dos métodos remotos que o compõem. A seguir, um outro pipeline é escolhido, considerando o mesmo critério e a definição dos *Milestones* é então realizada para este pipeline. Este processo se repete até que o grafo tenha sido completamente esgotado e todos os nodos tenham recebido um *Milestone*.

Ocorre que após ter sido realizada a previsão, a thread distribuída pode seguir para um nodo que possui um método remoto que pertence a outro itinerário, diferente daquele que foi usado como base para a geração do *Milestones*. Neste caso provavelmente a previsão irá falhar porque foi realizada considerando um determinado itinerário que a thread distribuída não seguiu.

O estudo sobre os mecanismos baseados em *Milestones* e seus resultados conduziu à elaboração de outros mecanismos mais refinados. O capítulo seguinte apresenta um outro mecanismo, que também utiliza o tempo de resposta local efetivo da thread distribuída e o tempo de computação restante para o cálculo da probabilidade, porém realiza este cálculo de forma diferente do mecanismo baseado em *Milestones* e considera, de forma mais abrangente, os possíveis itinerários que thread distribuída pode executar em um dada ativação.

Capítulo 6

Mecanismo de Previsão Baseado na Folga Restante da Thread Distribuída

6.1 Introdução

Todas as possíveis seqüências de métodos remotos que uma thread distribuída pode executar podem ser representadas por um grafo direcionado acíclico, com os nodos contendo os métodos remotos e as arestas indicando a seqüência em que eles serão executados. Cada seqüência de execução de métodos remotos pode ser vista como um possível itinerário a ser seguido pela thread distribuída.

Os mecanismos de previsão descritos no capítulo anterior utilizam um possível itinerário como critério para desmembrar em *pipelines* o grafo de execuções da thread distribuída e gerar os *milestones*. Desta forma, cada método remoto do grafo possui um *milestone* que será usado para fins de previsão.

Entretanto, devido à natureza autônoma da thread distribuída, ela pode executar uma seqüência de métodos remotos diferente daquela estabelecida antes de iniciar sua execução. Considerando esta característica, o mecanismo de previsão baseado na *Folga Restante* (FR) da thread distribuída realiza o cálculo da previsão considerando todos os itinerários que a thread distribuída poderá percorrer, a partir do nodo em que ela se encontra no momento

em que o mecanismo é acionado.

Com o objetivo de melhorar os resultados gerados pelos previsores baseados em *Milestones*, este capítulo descreve o mecanismo *Folga Restante* (FR), que utiliza o tempo de execução de todos os itinerários que a thread distribuída poderá percorrer, a partir do nodo em que ela se encontra, com a respectiva probabilidade de cada itinerário ser executado.

Este capítulo está estruturado da seguinte forma: a seção 6.2 descreve as características do mecanismo FR; a seção 6.3 apresenta as condições de simulação usadas, mostra e avalia os resultados das simulações realizadas. A seção 6.4 apresenta as conclusões deste capítulo.

6.2 Descrição do Mecanismo *Folga Restante* (FR)

A probabilidade de uma thread distribuída cumprir seu deadline fim a fim é definida pelo mecanismo FR através da seguinte expressão:

$$P_k(FR) = (Dff_k - TRespostaLocal_k - TCompRestante_k)/TCompRestante_k$$

$$Prob_k(FR) = \begin{cases} 0 & P_k(FR) < 0 \\ P_k(FR) & 0 \leq P_k(FR) \leq 1 \\ 1 & P_k(FR) > 1 \end{cases}$$

onde Dff_k é o deadline fim a fim da thread distribuída k, $TRespostaLocal$ é o tempo de resposta local efetivo da thread distribuída k até o momento em que o mecanismo de previsão foi acionado e $TCompRestante$ é o somatório dos tempos médios de computação dos segmentos locais da thread distribuída k que ainda serão executados.

A variável $TCompRestante$ considera o tempo de computação de todos os itinerários que a thread distribuída poderá percorrer. Para isso, o mecanismo FR consulta a estrutura *Parâmetros para Previsão* que contém todos os itinerários possíveis que a thread distribuída pode executar a partir do nodo que ela se encontra. Para cada itinerário, o mecanismo calcula a soma dos tempos médios de computação de cada método que pertence ao itinerário e multiplica pela probabilidade do itinerário ser executado.

O seguinte exemplo ilustra como este mecanismo realiza o cálculo de previsão de perda de deadlines para threads distribuídas. A fig. 6.1 apresenta 9 nodos, cada um de-

les hospedando métodos remotos que poderão ser executados pela thread distribuída e a figura 6.2 apresenta o histórico da thread distribuída, armazenado no nodo origem. A coluna *sl* indica os segmentos locais da thread distribuída, a coluna *C* mostra os tempos médios de computação de cada segmento local, a coluna *dl* mostra o valor do deadline local de cada segmento, gerado a partir do método de particionamento EQF [6] no itinerário *Maior Número de Saltos* (usado para fins de escalonamento).

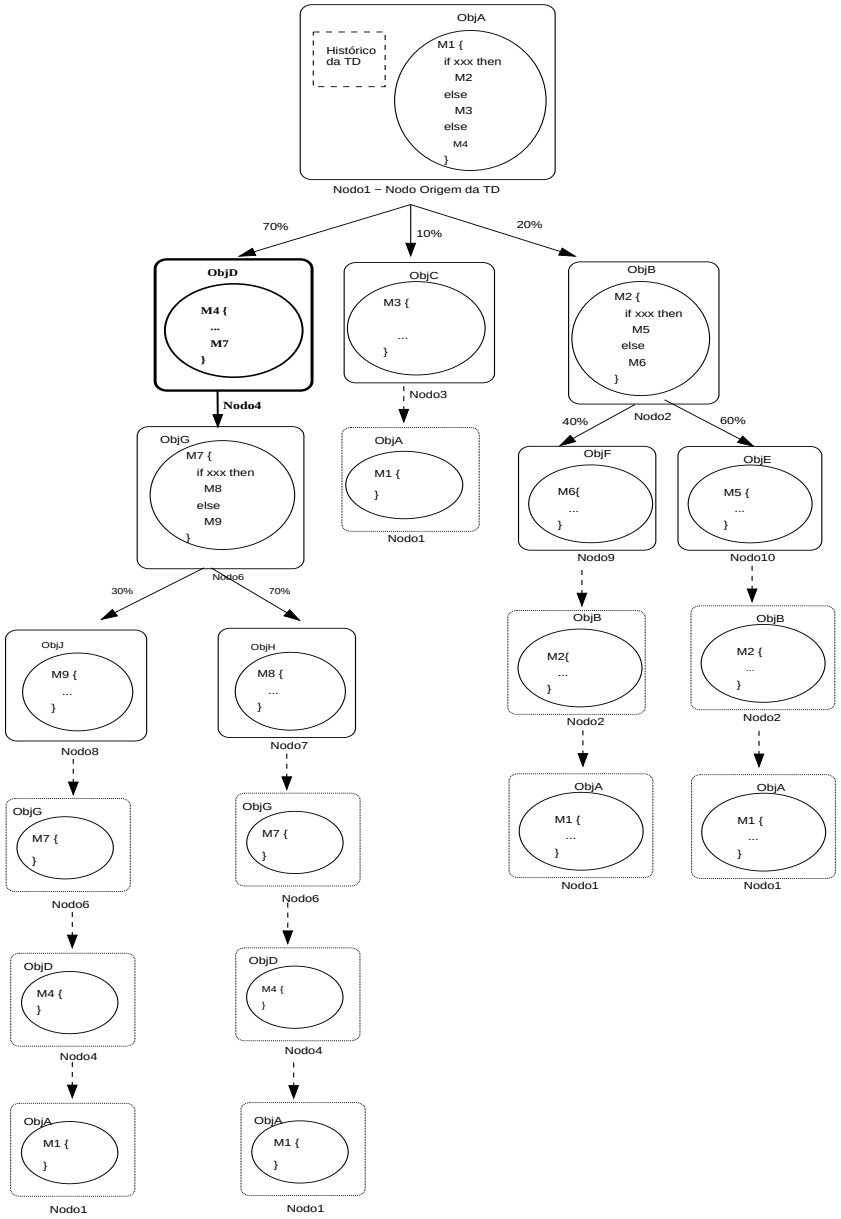

Figura 6.1: Itinerários que a thread distribuída pode executar.

Considerando que o tempo de resposta local da thread distribuída no nodo 4 seja igual a 100 ut, o mecanismo de previsão FR é acionado neste nodo porque este valor representa

HISTÓRICO DA THREAD DISTRIBUÍDA

Itinerários: Deadline fim a fim = 200ut

I1 = M1−M2−M5−M2−M1 (sl 1−2−5−10−11)

I2 = M1−M2−M6−M2−M1 (sl 1−2−6−12−13)

I3 = M1−M3−M1 (sl 1−3−14)

I4 = M1−M4−M7−M8−M7−M4−M1 (sl 1−4−7−8−15−16−17)

I5 = M1−M4−M7−M9−M7−M4−M1 (sl 1−4−7−9−18−19−20)

sl	C	dl	sl	C	dl
1	40	100	11	13	200
2	10	113	12	7	163
3	15	179	13	28	200
4	5	105	14	4	200
5	20	137	15	13	149
6	30	153	16	9	167
7	10	114	17	17	200
8	5	123	18	25	159
9	25	136	19	34	189
10	41	185	20	12	200

Figura 6.2: Histórico da thread distribuída.

metade do deadline fim a fim (200 ut) gasto pela thread distribuída até o momento.

A variável $TCompRestante$ deverá ser composta pelo somatório ponderado dos tempos médios de computação dos itinerários que a thread distribuída ainda pode percorrer. Para cada itinerário, deve ser considerada a probabilidade dele ser executado. Neste exemplo, os itinerários que poderão ser executados são compostos pelos métodos *M7-M8-M7-M4-M1* (itinerário I4 da figura 6.1), com probabilidade 70% (0,7). Este itinerário será executado pelos segmentos locais 7, 8, 15, 16 e 17, respectivamente. O itinerário *M7-M9-M7-M4-M1* (itinerário I5 da figura 6.1) possui probabilidade de 30% (0,3) e será executado pelos segmentos locais 7, 9, 18, 19 e 20. Desta forma calcula-se o tempo médio de execução de cada itinerário

a partir dos tempos médios de execução de cada segmento local, como mostrado a seguir:

$$
\begin{aligned}
I4 &= (10 + 5 + 13 + 9 + 17) \times 0,7 \\
I4 &= 37,0; \\
I5 &= (10 + 25 + 25 + 34 + 12) \times 0,3 \\
I5 &= 31,8; \\
TCompRestante &= I4 + I5 \\
TCompRestante &= 68,8; \\
P_k(FR) &= (200 - 100 - 68,8)/68,8 \\
P_k(FR) &= 0,453; \\
Prob_k(FR) &= 0,453;
\end{aligned}
$$

Neste exemplo, a probabilidade da thread distribuída cumprir seu deadline fim a fim, definida pelo mecanismo FR é igual a $Prob_k(FR) = 0,453$. Ao final da execução da thread distribuída no sistema, avalia-se a qualidade da previsão realizada através da métrica $E_k(FR)$ e $PC_k(FR)$.

6.3 Simulações com o Mecanismo Folga Restante (FR)

Simulações foram realizadas com o objetivo de avaliar a qualidade das previsões realizadas pelo mecanismo *FR*. As condições das simulação foram as mesmas usadas nos previsores *Milestones*, definidas na seção 5.3.

6.3.1 Simulações com a Carga 1 - *Threads Distribuídas Tipo Pipeline*

A tabela 6.1 apresenta os resultados das simulações realizadas com a carga 1, segundo a métrica Taxa de Erro Relativo ($E(z)$). Neste tipo de carga, as threads distribuídas não fazem uso de possíveis itinerários a serem seguidos, para fins de previsão.

Conforme mostra a tabela 6.1, o mecanismo *FR* apresentou a menor taxa relativa de erro, em todas as faixas de deadlines. A diferença em relação aos previsores *Milestones* é significativa, mesmo usando o intervalo de confiança (IC) do previsor *FR* em cada deadline,

Tabela 6.1: Taxa Relativa de Erro para Threads Distribuídas tipo *Pipeline*

	Deadline, Taxa de Erro e Intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,223	±0,012	0,353	±0,014	0,280	±0,023	0,181	±0,027	0,126	±0,024
MED-MaNS	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MED-MeNS	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MED-Pond	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MED-MProv	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MEQS-MaNS	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQS-MeNS	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQS-Pond	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQS-MProv	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQF-MaNS	0,041	±0,005	0,150	±0,010	0,160	±0,009	0,146	±0,011	0,116	±0,014
MEQF-MeNS	0,041	±0,005	0,150	±0,010	0,160	±0,009	0,146	±0,011	0,116	±0,014
MEQF-Pond	0,041	±0,005	0,150	±0,010	0,160	±0,009	0,146	±0,011	0,116	±0,014
MEQF-MProv	0,041	±0,005	0,150	±0,010	0,160	±0,009	0,146	±0,011	0,116	±0,014
FR	**0,028**	±0,006	**0,102**	±0,007	**0,119**	±0,010	**0,102**	±0,016	**0,078**	±0,017

ele mantém as menores taxas de erro. Considerando os resultados próximos em 1% da taxa de erro do mecanismo FR, nenhum previsor *Milestone* alcança resultados próximos do previsor *FR*.

Analisando as taxas de erro do mecanismo FR nos deadlines simulados, percebe-se que a menor taxa de erro foi alcançada no deadline 100. Conforme discutido anteriormente, este deadline pode ser considerado apertado, onde a maioria das threads distribuídas perde seu deadline. Nesta situação, a previsão de perda de deadline torna-se mais fácil de ser realizada. O gráfico da figura 6.3 mostra os valores da tabela 6.1.

Os resultados das simulações com esta carga segundo a métrica *Taxa de Previsões Corretas* ($PC(z)$) são apresentados na tabela 6.2. O previsor FR não obteve a maior taxa de previsões corretas em nenhum deadline, sendo que o previsor *MEQF* manteve os melhores resultados na maioria dos deadlines.

No deadline 100, a taxa de previsão correta do mecanismo FR foi menor que a do previsor MED. No deadline 300, esta taxa ficou entre as taxas dos previsores MEQS e MEQF. Nos deadlines 500, 700 e 900, a taxa de previsões corretas do previsor FR ficou entre as taxas dos previsores MED e MEQS.

Usando o intervalo de confiança (IC) de cada resultado do previsor FR, percebe-se que

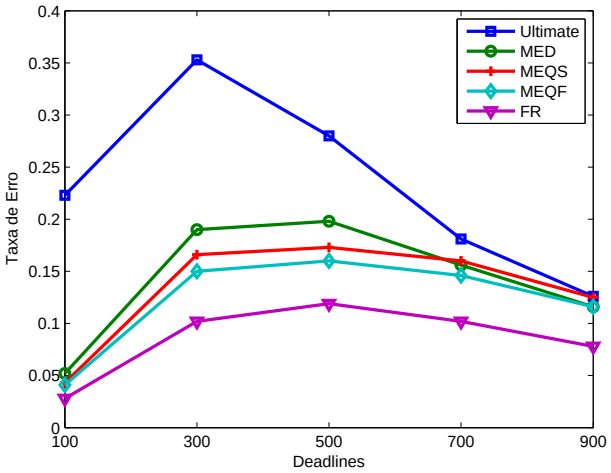

Figura 6.3: Taxa de Erro do Previsor FR para Threads Distribuídas tipo Pipeline.

ele empata com todos os previsores, em todos deadlines. Os mecanismos que possuem taxas de previsões corretas próximas em 1% do previsor FR são o previsor MED no deadline 100 e o previsor MEQS nos deadlines 500 e 700. O melhor resultado do previsor FR, isto é, a maior taxa de previsão correta foi no deadline 100, onde ele alcançou uma taxa de previsão correta de $0,976$

Tabela 6.2: Taxa de Previsões Corretas para Threads Distribuídas tipo *Pipeline*

	Deadline, Taxa de Previsões Corretas (PC) e Intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,705	±0,089	0,600	±0,096	0,714	±0,089	0,831	±0,073	0,893	±0,061
MED-MaNS	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,880	±0,064	0,906	±0,057
MED-MeNS	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,880	±0,064	0,906	±0,057
MED-Pond	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,88	±0,064	0,906	±0,057
MED-MProv	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,880	±0,064	0,906	±0,057
MEQS-MaNS	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQS-MeNS	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQS-Pond	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQS-MProv	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQF-MaNS	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041
MEQF-MeNS	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041
MEQF-Pond	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041
MEQF-MProv	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041
FR	0,976	±0,030	0,902	±0,058	0,886	±0,062	0,898	±0,059	0,923	±0,061

O gráfico da figura 6.4 mostra os valores da tabela 6.2. O previsor *Ultimate* obteve a pior

taxa de previsões corretas na maioria dos deadlines, com exceção do deadline 900, onde seu resultado está próximo dos demais mecanismos. O previsor MEQF obteve a melhor taxa de previsões corretas na maioria dos deadlines. No deadline 100, os previsores FR, MED, MEQS e MEQF apresentaram resultados próximos. Os previsores FR e MEQS ficaram praticamente empatados na maioria dos deadlines (300, 500, 700 e 900).

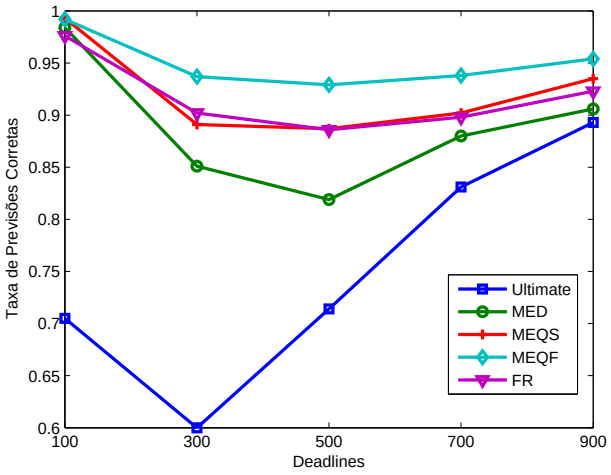

Figura 6.4: Taxa de Previsões Corretas para Threads Distribuídas tipo Pipeline.

As simulações realizadas com esta carga mostram que o previsor FR apresenta melhores resultados que os demais previsores na métrica Taxa Relativa de Erro. A probabilidade gerada pelo FR é melhor que a dos demais mecanismos, porque seu cálculo considera todos os possíveis itinerários que a thread distribuída pode executar. Os casos em que o mecanismo FR erra a previsão não são suficientes para compensar a grande maioria dos casos onde sua estimativa de probabilidade é melhor que a dos mecanismos baseados em Milestones.

Entretanto, na métrica Taxa de Previsões Corretas, o previsor FR apresenta resultados inferiores aos resultados do previsor MEQF. Isto ocorre porque esta métrica considera apenas se a thread distribuída vai ou não cumprir seu deadline, isto é, se a probabilidade calculada é maior ou menor que 50%. A métrica não leva em consideração a certeza a respeito desta previsão. Por exemplo, se o mecanismo FR calcula uma probabilidade igual ou maior que 50%, a métrica PC considera que a thread distribuída irá cumprir seu deadline, sem levar em consideração se essa probabilidade é 50%, 75% ou 99%.

Ocorre que quando a thread distribuída perde seu deadline, a métrica Taxa Relativa de Erro utiliza a probabilidade gerada pelo mecanismo e calcula a distância que separa a

previsão com o que de fato ocorreu. Já a métrica PC utiliza a probabilidade gerada pelo mecanismo para simplesmente contabilizar que o mecanismo errou a previsão, sem considerar de fato qual foi a probabilidade definida pelo mecanismo.

6.3.2 Simulações com a Carga 2 - *Threads Distribuídas Tipo Árvore Balanceada*

Este tipo de carga faz uso de supostos itinerários de execução onde a thread distribuída tem mais de uma sequência de métodos possíveis de serem executados em uma dada ativação. A tabela 6.3 apresenta os resultados das simulações realizadas com esta carga, segundo a métrica Taxa Relativa de Erro ($E(z)$). O mecanismo FR alcança a menor taxa de erro em todos os deadlines. Mesmo usando o intervalo de confiança do mecanismo, o resultado não se sobrepõe aos resultados dos outros mecanismos.

Observando os resultados do previsor FR nos deadlines simulados, percebe-se que a menor taxa de erro foi obtida no deadline 100, seguido pelo deadline 900, 700, 500 e 300. Prever a perda de um deadline quando estes são apertados (deadline 100) ou folgados (900) é mais fácil que em situações onde o deadline é justo, isto é, o número de tarefas que cumpre o deadline é muito próximo do número de tarefas que perde o deadline.

Tabela 6.3: Taxa Relativa de Erro para Threads Distribuídas tipo Árvore Balanceada

	Deadline, Taxa de Erro e Intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,413	±0,019	0,379	±0,017	0,203	±0,017	0,120	±0,016	0,078	±0,012
MED-MaNS	0,038	±0,005	0,263	±0,011	0,178	±0,011	0,102	±0,011	0,064	±0,009
MED-MeNS	0,102	±0,007	0,301	±0,011	0,186	±0,013	0,110	±0,013	0,070	±0,011
MED-Pond	0,053	±0,005	0,279	±0,011	0,181	±0,012	0,104	±0,012	0,066	±0,010
MED-MProv	0,067	±0,007	0,276	±0,011	0,181	±0,012	0,105	±0,012	0,067	±0,010
MEQS-MaNS	0,051	±0,005	0,223	±0,011	0,190	±0,008	0,120	±0,010	0,072	±0,009
MEQS-MeNS	0,092	±0,005	0,251	±0,012	0,189	±0,009	0,116	±0,010	0,071	±0,009
MEQS-Pond	0,065	±0,005	0,229	±0,011	0,190	±0,008	0,119	±0,010	0,072	±0,009
MEQS-MProv	0,067	±0,005	0,233	±0,011	0,189	±0,009	0,117	±0,010	0,071	±0,009
MEQF-MaNS	0,049	±0,004	0,213	±0,010	0,172	±0,008	0,110	±0,009	0,068	±0,009
MEQF-MeNS	0,099	±0,006	0,252	±0,012	0,175	±0,009	0,109	±0,010	0,068	±0,009
MEQF-Pond	0,064	±0,005	0,220	±0,010	0,171	±0,008	0,109	±0,009	0,068	±0,009
MEQF-MProv	0,069	±0,006	0,227	±0,011	0,171	±0,008	0,108	±0,009	0,068	±0,009
FR	**0,012**	±0,003	**0,158**	±0,008	**0,121**	±0,012	**0,066**	±0,012	**0,032**	±0,008

Os bons resultados do previsor FR nesta métrica se deve ao fato dele considerar todos os itinerários possíveis no cálculo da previsão. Este mecanismo faz a soma dos tempos de computação de cada itinerário possível de ser executado a partir do nodo em que a previsão é realizada, considerando a probabilidade de cada itinerário ser percorrido pela thread distribuída. A figura 6.5 mostra um gráfico com os resultados da tabela 6.3. São mostrados apenas o itinerário MaNS de cada previsor *Milestone*, em função dos bons resultados deste itinerário nas simulações realizadas com os previsores *Milestones*.

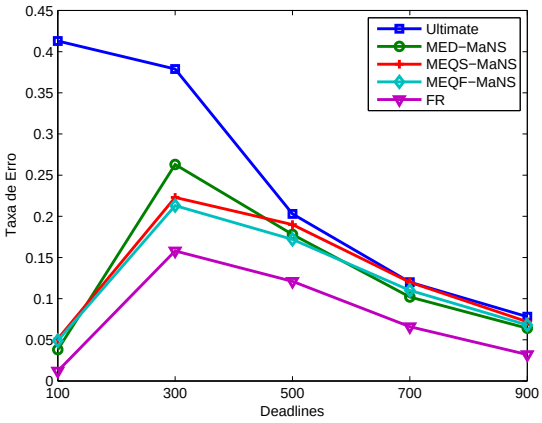

Figura 6.5: Taxa de Erro do Previsor FR para TDs tipo Árvore Balanceada.

A tabela 6.4 apresenta os resultados das simulações realizadas com a carga 2, segundo a métrica *Taxa de Previsões Corretas* ($PC(z)$). Neste tipo de carga, assim como na anterior, as taxas de previsões corretas do previsor FR não apresentaram os melhores resultados. Os previsores MEQF-MaNS e MEQF-Pond apresentaram as maiores taxas de previsões corretas na maioria dos deadlines.

Usando o intervalo de confiança (IC) do previsor FR, percebe-se que seus resultados se sobrepõem aos resultados dos demais previsores *Milestones*. No deadline 500, por exemplo, considerando o grau de confiança de $\pm 0,061$, o IC do previsor FR varia de [0,818 0,940] e as taxas de previsões corretas dos previsores MED, MEQS e MEQF são, respectivamente, $0,871$, $0,886$ e $0,919$.

Considerando os resultados próximos em 1% dos demais previsores, pode-se afirmar que no deadline 100 o previsor FR alcançou um resultado próximo dos previsores MED-MaNS, MED-Pond, MEQS (todos os itinerários) e MEQF (todos os itinerários). No deadline 300 o previsor FR alcançou resultado próximo dos previsores MEQS-MProv e MEQF-Mens. No

deadline 500, o resultado do previsor FR foi próximo dos previsores MED-MaNS e MEQS (todos os itinerários). No deadline 700, o resultado do previsor FR foi próxima dos previsores MED e MEQS, em todos os itinerários e no deadline 900, o resultado do previsor FR foi próximo dos três previsores Milestones, em todos os itinerários.

A maior taxa de previsões corretas do previsor FR foi no deadline 100, seguido pelos resultados nos deadlines 900, 700, 500 e 300.

Tabela 6.4: Taxa de Previsões Corretas para Threads Distribuídas tipo Árvore Balanceada

	Deadline, Taxa de Previsões Corretas (PC) e Intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,482	±0,098	0,602	±0,096	0,817	±0,076	0,905	±0,058	0,937	±0,048
MED-MaNS	0,983	±0,025	0,808	±0,077	0,871	±0,066	0,941	±0,046	**0,974**	±0,031
MED-MeNS	0,932	±0,049	0,737	±0,086	0,854	±0,069	0,928	±0,051	0,961	±0,038
MED-Pond	0,978	±0,028	0,778	±0,081	0,862	±0,068	0,940	±0,046	**0,971**	±0,033
MED-MProv	0,949	±0,043	0,782	±0,081	0,865	±0,067	0,937	±0,048	0,967	±0,035
MEQS-MaNS	0,988	±0,021	0,868	±0,066	0,883	±0,063	0,940	±0,047	0,967	±0,035
MEQS-MeNS	0,977	±0,029	0,826	±0,074	0,878	±0,064	0,942	±0,046	0,970	±0,034
MEQS-Pond	0,988	±0,021	0,861	±0,068	0,886	±0,062	0,942	±0,046	**0,969**	±0,034
MEQS-MProv	0,980	±0,027	0,848	±0,070	0,885	±0,063	0,943	±0,046	**0,969**	±0,034
MEQF-MaNS	0,988	±0,021	**0,904**	±0,058	**0,916**	±0,054	**0,951**	±0,042	**0,972**	±0,032
MEQF-MeNS	0,978	±0,029	0,847	±0,070	0,900	±0,059	**0,953**	±0,042	0,978	±0,029
MEQF-Pond	**0,990**	±0,019	0,900	±0,059	**0,919**	±0,053	**0,954**	±0,041	**0,974**	±0,031
MEQF-MProv	0,981	±0,027	0,884	±0,063	**0,913**	±0,055	**0,952**	±0,042	**0,974**	±0,031
FR	0,987	±0,022	0,850	±0,070	0,879	±0,064	0,934	±0,049	0,968	±0,034

A figura 6.6 apresenta um gráfico com os valores da tabela 6.4. Foi utilizado o itinerário MaNS para os previsores MED, MEQS e MEQF. O previsor Ultimate apresenta a menor taxa de previsões corretas e o previsor MEQF-MaNS apresenta a maior taxa de previsões corretas. O previsor FR obteve resultados próximos do previsor MEQS-MaNS.

Com threads distribuídas do tipo árvore balanceada, o previsor FR apresentou resultados significativamente melhores que os previsores baseados em *Milestones*, considerando a métrica taxa relativa de erro. Já na segunda métrica, taxa de previsões corretas, o previsor FR obteve resultados inferiores aos previsores *Milestones*.

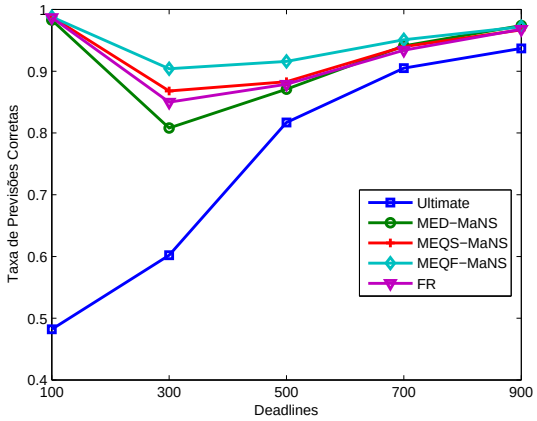

Figura 6.6: Taxa de Previsões Corretas do Mecanismo FR para TDs tipo Árvore Balanceada.

6.3.3 Simulações com a Carga 3 - *Threads Distribuídas Tipo Árvore Não Balanceada*

A tabela 6.5 apresenta os resultados das simulações com a carga 3, segundo a métrica Taxa Relativa de Erro. O mecanismo FR apresenta a menor taxa de erro na maioria dos deadlines, com uma diferença significativa em relação aos demais previsores, com exceção do deadline 100 onde o mecanismo *MED-MaNS* apresentou a menor taxa de erro.

Usando o intervalo de confiança (IC) do previsor FR, percebe-se que seus resultados se sobrepõem aos resultados de alguns dos previsores *Milestones*, nos deadlines 100 e 900. No deadline 100, por exemplo, considerando o grau de confiança dos previsores *Milestones* e FR, o IC do previsor FR varia de [0,029 0,039] e os IC dos previsores MED, MEQS e MEQF se encontram, respectivamente, em [0,029 0,037], [0,038 0,044] e [0,034 0,042].

Considerando os resultados próximos em 1% do resultado do previsor FR, é possível verificar que nenhuma outra taxa de erro dos demais previsores alcança resultados próximos, em todos os deadlines. Observa-se que entre as taxas de erro do previsor FR, a taxa obtida no deadline 100 é a menor, seguida pelas taxas obtidas nos deadlines 900, 700, 500 e 300, exatamente como na carga anteriormente simulada.

A figura 6.7 mostra um gráfico com os valores da tabela 6.5. Foi utilizado o itinerário MaNS dos previsores *Milestones*, além do previsor *Ultimate* e FR. No deadline 100, os previsores baseados em *Milestones* apresentam resultados muito próximos do previsor FR, com exceção do mecanismo *Ultimate* que possui a maior taxa de erro.

Tabela 6.5: Taxa Relativa de Erro para Threads Distribuídas tipo Árvore Não Balanceada

	Deadline, Taxa de Erro e Intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,266	±0,016	0,418	±0,013	0,248	±0,014	0,130	±0,012	0,078	±0,012
MED-MaNS	**0,033**	±0,004	0,247	±0,001	0,213	±0,009	0,130	±0,010	0,082	±0,010
MED-MeNS	0,073	±0,006	0,321	±0,001	0,232	±0,011	0,134	±0,011	0,084	±0,011
MED-Pond	**0,036**	±0,004	0,268	±0,001	0,218	±0,010	0,131	±0,010	0,083	±0,011
MED-MProv	0,054	±0,006	0,276	±0,001	0,181	±0,010	0,105	±0,010	0,067	±0,011
MEQS-MaNS	0,041	±0,003	0,203	±0,009	0,223	±0,008	0,164	±0,010	0,106	±0,009
MEQS-MeNS	0,065	±0,005	0,232	±0,009	0,226	±0,009	0,157	±0,010	0,098	±0,009
MEQS-Pond	0,050	±0,004	0,205	±0,009	0,221	±0,008	0,162	±0,010	0,105	±0,009
MEQS-MProv	0,053	±0,004	0,210	±0,009	0,220	±0,008	0,160	±0,010	0,104	±0,009
MEQF-MaNS	0,038	±0,004	0,202	±0,009	0,203	±0,007	0,143	±0,009	0,093	±0,008
MEQF-MeNS	0,068	±0,005	0,259	±0,009	0,224	±0,010	0,139	±0,010	0,084	±0,009
MEQF-Pond	0,048	±0,004	0,210	±0,009	0,203	±0,007	0,142	±0,009	0,093	±0,008
MEQF-MProv	0,053	±0,004	0,211	±0,009	0,200	±0,007	0,140	±0,009	0,092	±0,008
FR	**0,034**	±0,005	**0,136**	±0,009	**0,135**	±0,009	**0,090**	±0,011	**0,056**	±0,013

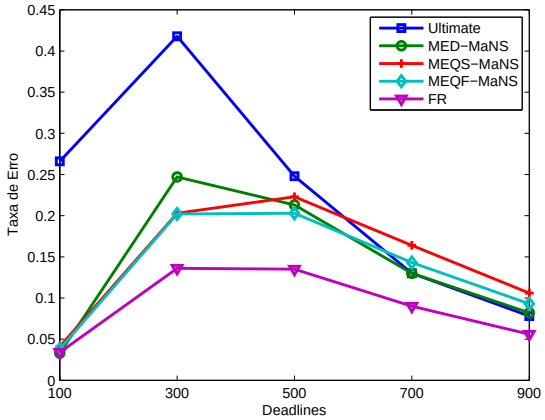

Figura 6.7: Taxa de Erro do Previsor FR para TDs tipo Árvore Não Balanceada.

A tabela 6.6 apresenta os resultados das simulações realizadas com a carga 3, segundo a métrica Taxa de Previsões Corretas ($PC(z)$). Em todos os deadlines simulados, a taxa de previsões corretas do mecanismo FR foi menor que a do previsor MEQF (este previsor obteve os melhores resultados entre os previsores *Milestones*). No deadline 300, o resultado do previsor FR ficou muito próximo do previsor MEQS-MaNS.

No deadline 500, a taxa de previsão correta alcançada pelo mecanismo FR foi melhor que a dos previsores MED e MEQS. Considerando resultados próximos em 1% da taxa de previsão correta do mecanismo FR, percebe-se que os resultados do demais previsores não se aproximam do resultado do FR.

No deadline 700, o mecanismo FR obteve uma taxa de previsão correta próxima dos mecanismos *MEQS-MeNS* e *MEQS-Pond*. Usando o intervalo de confiança do previsor FR neste deadline, observa-se um empate entre ele e todos os previsores *Milestones*. A taxa de previsão correta do mecanismo FR neste deadline fica próxima dos previsores MED e MEQS (em todos os itinerários), se forem considerados os valores próximos em 1%.

No deadline 900, o previsor FR obteve a pior taxa de previsão correta em relação a todos os previsores simulados. O resultado do mecanismo FR neste deadline fica próxima do previsor MEQS (nos itinerários *MaNS*, *Pond* e *MProv*), se forem considerados os valores próximos em 1%.

A figura 6.8 apresenta um gráfico com os valores da tabela 6.6. Foi utilizado o itinerário MaNS para os previsores MED, MEQS e MEQF. O previsor FR apresenta resultados próximos do previsor MEQS, o previsor MEQF obteve as maiores taxas de previsões corretas e o previsor Ultimate as piores taxas de previsões corretas.

É possível observar que os resultados das simulações realizadas com threads distribuídas tipo árvore não balanceada apresentam o mesmo formato da carga anterior, isto é, com a primeira métrica - *Taxa Relativa de Erro*, o mecanismo FR apresenta resultados superiores em relação aos previsores *Milestones* e *Ultimate*. Com a segunda métrica - *Taxa de Previsões Corretas*, o previsor FR alcança resultados próximos dos previsores *Milestones*. O previsor MEQF se mantém com os melhores resultados em todos os deadlines simulados. Nos deadlines 300 e 500, o previsor FR obteve maiores taxas de previsões corretas que os previsores MED e MEQS.

Tabela 6.6: Taxa de Previsões Corretas para Threads Distribuídas tipo Árvore Não Balanceada

Mecanismos de Previsão	*Deadline, Taxa de Previsões Corretas (PC) e intervalo de Confiança (IC)*									
	100		300		500		700		900	
	PC	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,688	±0,091	0,519	±0,098	0,770	±0,082	0,898	±0,059	0,951	±0,043
MED-MaNS	0,982	±0,026	0,809	±0,077	0,842	±0,072	0,913	±0,055	0,957	±0,040
MED-MeNS	0,939	±0,047	0,661	±0,093	0,803	±0,078	0,905	±0,057	0,953	±0,042
MED-Pond	0,980	±0,027	0,771	±0,082	0,830	±0,074	0,911	±0,056	0,955	±0,040
MED-MProv	0,954	±0,041	0,771	±0,082	0,835	±0,073	0,911	±0,056	0,956	±0,040
MEQS-MaNS	**0,990**	±0,019	0,871	±0,066	0,846	±0,071	0,900	±0,059	0,946	±0,044
MEQS-MeNS	0,987	±0,022	0,879	±0,064	0,852	±0,070	0,908	±0,057	0,955	±0,041
MEQS-Pond	**0,990**	±0,019	0,885	±0,063	0,854	±0,069	0,905	±0,057	0,949	±0,043
MEQS-MProv	**0,990**	±0,019	0,883	±0,063	0,856	±0,069	0,910	±0,056	0,951	±0,042
MEQF-MaNS	**0,991**	±0,018	**0,907**	±0,057	**0,903**	±0,058	**0,941**	±0,046	**0,966**	±0,036
MEQF-MeNS	0,987	±0,022	0,834	±0,073	0,851	±0,070	0,922	±0,052	**0,963**	±0,037
MEQF-Pond	**0,993**	±0,017	**0,917**	±0,054	**0,911**	±0,056	**0,944**	±0,045	**0,968**	±0,035
MEQF-MProv	**0,991**	±0,018	**0,908**	±0,057	**0,910**	±0,056	**0,945**	±0,045	**0,968**	±0,035
FR	0,969	±0,034	0,872	±0,065	0,866	±0,067	0,907	±0,057	0,943	±0,046

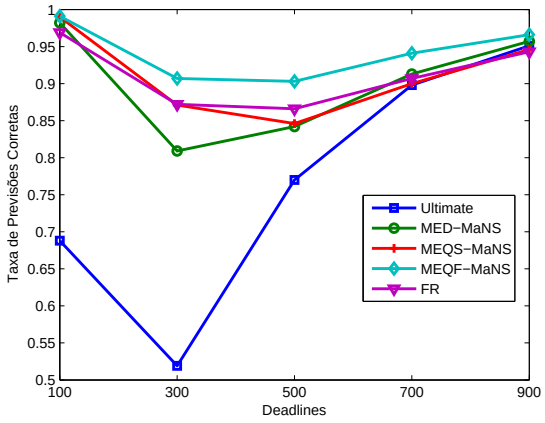

Figura 6.8: Taxa de Previsões Corretas do Mecanismo FR para TDs tipo Árvore Não Balanceada.

6.3.4 Simulações com a Carga 4 - *Threads Distribuídas Carga Mista*

A tabela 6.7 apresenta os resultados das simulações com a carga mista, composta por threads distribuídas do tipo *pipeline*, balanceada e não balanceada, segundo a métrica *Taxa Relativa de Erro*.

A diferença em relação aos mecanismos baseados em *Milestones* é significativa, tanto que mesmo usando o intervalo de confiança de cada resultado do mecanismo FR, os valores gerados por ele e dos mecanismos baseados em *Milestones* não se sobrepõem. Esta diferença se reafirma considerando, em cada deadline, valores próximos em 1% da taxa de erro do mecanismo FR.

Tabela 6.7: Taxa Relativa de Erro para Threads Distribuídas - Carga Mista

	Deadline, Taxa de Erro e Intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,315	±0,017	0,367	±0,016	0,218	±0,020	0,118	±0,016	0,078	±0,014
MED-MaNS	0,047	±0,005	0,234	±0,010	0,184	±0,013	0,110	±0,012	0,074	±0,012
MED-MeNS	0,085	±0,008	0,266	±0,011	0,191	±0,014	0,114	±0,013	0,077	±0,012
MED-Pond	0,055	±0,006	0,245	±0,010	0,186	±0,013	0,111	±0,012	0,075	±0,012
MED-MProv	0,065	±0,007	0,242	±0,010	0,185	±0,013	0,111	±0,012	0,075	±0,012
MEQS-MaNS	0,052	±0,005	0,203	±0,011	0,194	±0,010	0,135	±0,011	0,091	±0,011
MEQS-MeNS	0,076	±0,006	0,221	±0,011	0,193	±0,010	0,132	±0,011	0,089	±0,011
MEQS-Pond	0,061	±0,005	0,206	±0,011	0,193	±0,010	0,135	±0,011	0,091	±0,011
MEQS-MProv	0,062	±0,006	0,209	±0,011	0,191	±0,010	0,133	±0,011	0,091	±0,011
MEQF-MaNS	0,049	±0,005	0,194	±0,010	0,175	±0,009	0,120	±0,010	0,082	±0,011
MEQF-MeNS	0,078	±0,006	0,223	±0,010	0,181	±0,011	0,118	±0,011	0,080	±0,011
MEQF-Pond	0,059	±0,005	0,198	±0,010	0,175	±0,009	0,119	±0,010	0,082	±0,011
MEQF-MProv	0,062	±0,006	0,201	±0,010	0,173	±0,009	0,118	±0,010	0,082	±0,011
FR	**0,026**	±0,006	**0,133**	±0,008	**0,114**	±0,012	**0,068**	±0,011	**0,044**	±0,011

A figura 6.9 mostra um gráfico com cada previsor com o suposto itinerário MaNS. A escolha deste itinerário se deve aos bons resultados que ele apresentou em relação aos outros itinerários. O gráfico apresenta o mesmo padrão de comportamento do previsor FR nas outras cargas simuladas. O previsor Ultimate apresenta a maior taxa de erro para os deadlines 100, 300 e 500.

Para os deadlines 700 e 900, os previsores *Milestones* apresentam uma pequena diferença nos resultados e o previsor *FR* alcança a menor taxa de erro em todos os deadlines, com uma diferença significativa em relação aos *Milestones*.

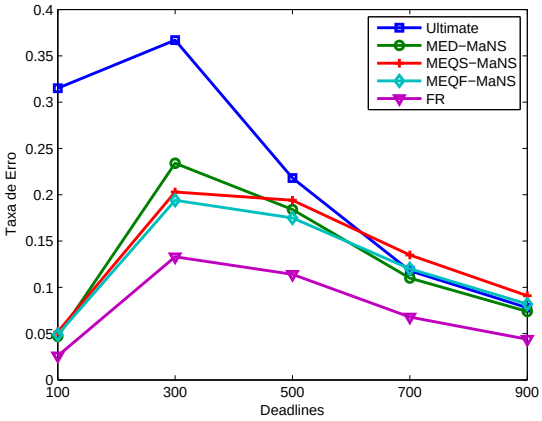

Figura 6.9: Taxa de Erro do Previsor FR para TDs Carga Mista.

A tabela 6.8 apresenta os resultados das simulações realizadas com a carga 4, segundo a métrica Taxa de Previsões Corretas ($PC(z)$). O previsor MEQF apresentou as melhores taxas de previsões corretas em todos os deadlines.

Considerando o intervalo de confiança do previsor FR no deadline 100, pode-se verificar um empate deste previsor com os previsores *Milestones*. Entretanto, se considerar resultados próximos em 1% da maior taxa de previsão correta (previsor *MEQF-Pond*),o mecanismo FR não obteve um resultado próximo dos previsores *Milestones*.

No deadline 300, o previsor FR obteve um resultado próximo do previsor *MEQS-Pond*. Usando o intervalo de confiança do previsor FR neste deadline, percebe-se um empate entre a maioria dos previsores, com exceção dos previsores *Ultimate*, *MED-Pond* e *MED-MProv*, que tiveram as menores taxas de previsão correta. Observa-se que os mecanismos MEQS-MaNS, MEQS-Pond, MEQS-MProv e MEQF-MeNS apresentam resultados próximos em 1% do previsor FR.

No deadline 500, usando o intervalo de confiança do previsor FR neste deadline, percebe-se um empate entre a maioria dos previsores, com exceção do previsor *Ultimate*, que obteve a menor taxa de previsão correta. É possível verificar que a taxa de previsão correta do previsor FR foi melhor que a dos previsores MED e MEQS em todos os itinerários, com exceção do previsor MEQS-MProv, que mantém um resultado próximo do previsor FR considerando valores próximos de 1%.

No deadline 700, o mecanismo FR apresentou resultados próximos em 1% dos previ-

sores MED-MaNS, MED-MeNS, MEQS-MeNS e MEQS-MProv. Considerando o intervalo de confiança do previsor FR neste deadline ($IC \pm 0,050$), é possível afirmar que houve um empate entre ele e os demais previsores.

O previsor FR obteve a mesma taxa de previsão que os mecanismos MED-MeNS, MEQS-Pond, MEQS-MProv no deadline 900. Considerando os resultados próximos em 1% da maior taxa de previsão correta neste deadline (0,967 dos previsores *MEQF-MeNS*, *MEQF-Pond* e *MEQF-MProv*), percebe-se a existência de um empate entre o previsor FR e a maioria dos previsores *Milestones*.

Tabela 6.8: Taxa de Previsões Corretas para Threads Distribuídas - *Carga Mista*

	Deadline, Taxa de Previsões Corretas (PC) e Intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,61	±0,096	0,596	±0,096	0,800	±0,078	0,907	±0,057	0,943	±0,045
MED-MaNS	**0,979**	±0,028	0,823	±0,075	0,859	±0,068	0,932	±0,049	**0,960**	±0,038
MED-MeNS	0,945	±0,045	0,760	±0,084	0,845	±0,071	0,924	±0,052	0,955	±0,041
MED-Pond	0,977	±0,029	0,803	±0,078	0,853	±0,069	0,930	±0,050	**0,959**	±0,039
MED-MProv	0,958	±0,039	0,805	±0,078	0,856	±0,069	0,930	±0,050	**0,958**	±0,039
MEQS-MaNS	**0,987**	±0,022	0,871	±0,066	0,873	±0,065	0,918	±0,054	0,953	±0,041
MEQS-MeNS	**0,981**	±0,027	0,863	±0,067	0,875	±0,065	0,924	±0,052	**0,958**	±0,039
MEQS-Pond	**0,987**	±0,022	0,873	±0,065	0,877	±0,064	0,920	±0,053	0,955	±0,041
MEQS-MProv	**0,983**	±0,025	0,869	±0,066	0,879	±0,064	0,922	±0,053	0,955	±0,040
MEQF-MaNS	**0,989**	±0,021	**0,913**	±0,055	**0,921**	±0,053	**0,949**	±0,043	**0,966**	±0,035
MEQF-MeNS	**0,981**	±0,027	0,876	±0,065	0,904	±0,058	**0,948**	±0,044	**0,967**	±,035
MEQF-Pond	**0,989**	±0,020	**0,917**	±0,054	**0,924**	±0,052	**0,951**	±0,042	**0,967**	±0,035
MEQF-MProv	**0,984**	±0,024	0,908	±0,057	**0,925**	±0,052	**0,951**	±0,042	**0,967**	±0,035
FR	**0,975**	±0,030	**0,874**	±0,065	**0,887**	±0,062	**0,931**	±0,050	**0,955**	±0,041

A figura 6.10 apresenta um gráfico com os valores da tabela 6.8. Foi utilizado o itinerário MaNS para os previsores MED, MEQS e MEQF. No deadline 500, o previsor FR alcança um resultado melhor que os previsores MED e MEQS.

Assim como ocorreu com as demais cargas simuladas, os resultados com a carga mista revelam que o mecanismo FR apresenta melhores resultados que os previsores *Milestones*, na métrica *Taxa Relativa de Erro*. Isto se deve ao fato que o previsor FR considera todos os possíveis itinerários que a thread distribuída pode percorrer em uma dada ativação nos cálculos de previsão, definindo uma probabilidade melhor em relação aos previsores *Milestones*. Como esta métrica utiliza o valor da probabilidade para calcular o erro, a média do

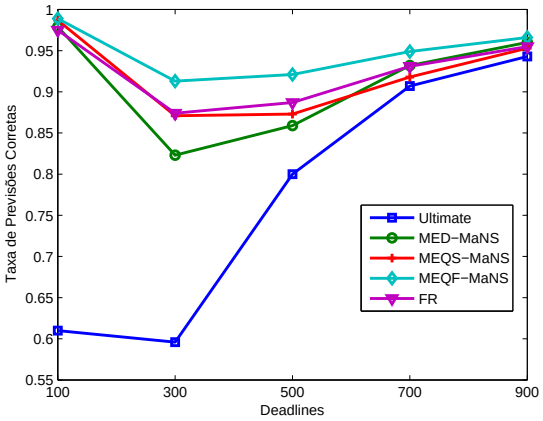

Figura 6.10: Taxa de Previsões Corretas do Mecanismo FR para TDs Carga Mista.

previsor FR é melhor que a dos demais previsores.

Na métrica *Taxa de Previsões Corretas*, o mecanismo FR se manteve com resultados inferiores em relação ao mecanismo MEQF. Esta métrica não utiliza o valor da probabilidade, definida por cada mecanismo. Ela apenas avalia se o valor da probabilidade é menor ou maior que 50% e em função desta comparação, contabiliza os acertos de cada mecanismo.

6.4 Conclusões

Neste capítulo foi apresentado o mecanismo de previsão baseado na folga restante (FR) da thread distribuída. Inicialmente, foi descrita a forma que o mecanismo calcula a probabilidade de uma thread distribuída cumprir seu deadline fim a fim. Para este cálculo, o mecanismo verifica quais itinerários a thread distribuída pode seguir a partir do nodo onde está sendo realizada a previsão. Esta informação é obtida através de uma consulta à estrutura *Parâmetros para Previsão*, que acompanha a thread distribuída conforme ela transpõe os nodos do sistema e é atualizada em cada nodo que a thread distribuída chega para executar. Para cada suposto itinerário, o mecanismo soma os tempos de computação de cada método que pertence à ele e faz a multiplicação desse somatório pela probabilidade do suposto itinerário ser seguido pela thread distribuída. Em função da natureza dinâmica da thread distribuída, um itinerário pode ser percorrido parcialmente, isto é, apenas alguns métodos do itinerário suposto são executados.

Assim que uma thread distribuída finaliza sua execução em um nodo, o mecanismo

de previsão verifica se o tempo de resposta local da thread distribuída é igual ou maior que metade do seu deadline fim a fim. Em caso afirmativo, o mecanismo calcula a probabilidade da thread distribuída cumprir seu deadline. A thread distribuída segue sua execução pelo sistema distribuído, percorrendo um ou mais itinerários supostos. Após concluir sua execução, as métricas taxa de erro relativo e taxa de previsões corretas são usadas para aferir se o mecanismo FR fez uma previsão correta e qual foi a distância entre esta previsão e o que de fato ocorreu.

Nas simulações realizadas, foi possível observar o mesmo comportamento do mecanismo FR nas diferentes cargas simuladas. Com relação à taxa de erro, o mecanismo FR apresentou resultados consistentemente melhores que os mecanismos baseados em *Milestones*. Mesmo usando os intervalos de confiança do mecanismo FR em cada deadline, não houve sobreposição de resultados entre o previsor FR e os previsores *Milestones*. Considerando valores próximos em 1% da taxa de erro do previsor FR em cada deadline, a maioria dos resultados dos previsores *Milestones* não se aproximou dos resultados do mecanismo FR.

Em relação à taxa de previsões corretas, o previsor FR obteve valores menores em relação aos previsores *Milestones*. O previsor *MEQF* obteve as maiores taxas de previsões em todas as cargas e deadlines simulados. Em alguns casos, usando o intervalo de confiança do previsor FR, observou-se um empate entre os mecanismos *Milestones* e FR.

O fato do previsor FR não gerar as maiores taxas de previsões corretas ocorre porque este mecanismo utiliza apenas informações previamente conhecidas de execução passadas, gerando previsões que podem ser equivocadas e que não refletem a verdadeira probabilidade de uma thread distribuída cumprir seu deadline fim a fim.

Para aprimorar os resultados de previsão deste mecanismo seria necessária a utilização de informações sobre o sistema, que tornassem a previsão mais realista e atualizada. Informações sobre, por exemplo, o tamanho da fila do servidor de aperiódicas dos nodos que pertencem aos supostos itinerários que uma thread distribuída pode percorrer, poderiam auxiliar na definição de estimativas de tempos de resposta mais realistas que levariam a previsões mais precisas.

Com o objetivo de refinar o cálculo de previsão do mecanismo FR, no capítulo a seguir será descrito um mecanismo de previsão que utiliza informações sobre o sistema distribuído e que podem ser conhecidas somente em tempo de execução.

Capítulo 7

Mecanismo de Previsão ASQ (*Aperiodic Server Queue Length*)

7.1 Introdução

Até o momento, os mecanismos de previsão de perda de deadline propostos nesta tese utilizam informações previamente conhecidas de execuções passadas. Estas informações são adquiridas no histórico da thread distribuída, que fica armazenado no nodo origem. A partir destas informações, os mecanismos de previsão baseados em *Milestones* definem, estaticamente, antes de cada ativação da thread distribuída, tempos de resposta estimados (*Milestones*) usando um suposto itinerário como base para os cálculos de previsão.

O mecanismo de previsão baseado na Folga Restante (FR) também utiliza informações armazenadas no histórico da thread distribuída. Entretanto, ele utiliza também informações de uma estrutura carregada pela thread distribuída, chamada de *Parâmetros para Previsão*. Mais especificamente, o mecanismo FR pesquisa nesta estrutura todos os possíveis itinerários que a thread distribuída pode executar a partir do nodo em que ela se encontra no momento em que a previsão está sendo realizada.

Os resultados das simulações realizadas com ambos os mecanismos mostram que o previsor FR apresenta menores taxas de erros em relação aos previsores *Milestones*. Isto indica que o fato do mecanismo FR usar informações para fins de previsão, em tempo de execução, melhoram a qualidade das previsões. Esta constatação motivou a pesquisa por mecanismos

mais sofisticados, que pudessem utilizar outras informações sobre o sistema distribuído, conhecidas somente em tempo de execução, fazendo com que as previsões a respeito da perda de um deadline sejam melhores.

Com o intuito de aprimorar os mecanismos de previsão *Milestones* e *FR*, neste capítulo propõe-se o mecanismo chamado ASQ (*Aperiodic Server Queue Length*). Este mecanismo utiliza informações, conhecidas em tempo de execução, referentes à fila do servidor de aperiódicas de cada nodo que faz parte dos itinerários que a thread distribuída poderá visitar. Além disso, o mecanismo ASQ também utiliza informações previamente conhecidas de ativações passadas da thread distribuída. Mais especificamente, o mecanismo considera os tempos de resposta locais efetivos de ativações anteriores dos segmentos da thread distribuída.

A forma como estas informações são relacionadas para a realização da previsão sobre o cumprimento de um deadline fim a fim são descritos na seção 7.2. A seção 7.3 descreve o resultados das simulações realizadas. Na seção 7.4 é realizada uma comparação dos resultados do mecanismo ASQ nas diferentes cargas simuladas. Na seção 7.5 as conclusões são apresentadas.

7.2 Descrição do Mecanismo

O mecanismo ASQ utiliza o tamanho e a composição da fila do servidor de aperiódicas dos nodos que a thread distribuída poderá executar e também o tempo de resposta local efetivo gasto pela thread distribuída em cada um destes nodos. O tamanho da fila se refere à quantidade de segmentos locais de threads distribuídas ativas em um dado nodo e a composição considera os deadlines locais dos segmentos estão na fila. Este mecanismo é definido como:

$$P_k(ASQ) = ((Dff_k - (TRespLocal_k + TRespEsperado_k))/TRespEsperado_k) + \sigma$$

$$Prob_k(ASQ) = \begin{cases} 0 & P_k(ASQ) < 0 \\ P_k(ASQ) & 0 \leq P_k(ASQ) \leq 1 \\ 1 & P_k(ASQ) > 1 \end{cases}$$

onde Dff_k é o deadline fim a fim da thread distribuída k, $TRespLocal$ é o tempo de resposta da thread distribuída k até o momento em que o mecanismo é acionado (tempo de resposta

local efetivo) e $TRespEsperado$ é o tempo de resposta esperado da thread distribuída k a partir do nodo atual até o final de sua execução. O valor σ é usado para ajustar a probabilidade, sendo 0.5 por default.

Para obter o valor de $TRespEsperado$ são considerados todos os caminhos possíveis (itinerários) que esta pode seguir a partir do nodo onde será acionado o mecanismo de previsão.

Em cada itinerário considera-se o tempo de resposta dos nodos que o compõem. O tempo de resposta de cada nodo leva em consideração o tamanho e a composição da fila de cada servidor. A composição da fila do servidor contém os deadlines locais das várias threads distribuídas ativas no sistema. Esta informação é atualizada pelas threads distribuídas, as quais carregam estes dados de um para outro nodo. A figura 7.1 apresenta uma thread distribuída partindo do nodo 2 (*N2*) e levando uma cópia do estado das filas deste nodo para o nodo 3 (*N3*). A cópia para um nodo é realizada caso a informação armazenada nele seja mais antiga que a informação trazida pela thread distribuída.

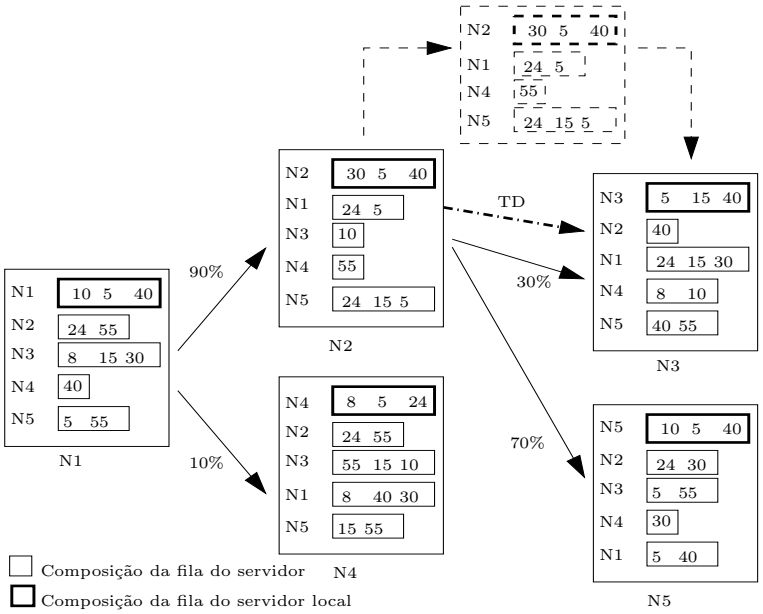

Figura 7.1: Atualização da Composição das Filas dos Servidores de Aperiódicas.

Com o deadline local e a composição da fila do servidor de um nodo é possível definir o provável tamanho de fila que a thread distribuída irá encontrar quando for executar naquele nodo. Se, por exemplo, a fila de um determinado nodo contém segmentos locais com deadlines 10, 25, 51 e 70, e o deadline local da thread distribuída naquele nodo será 30, então o tamanho

da fila para esta thread distribuída naquele nodo é 2 porque ela considera apenas os deadlines menores que o dela (10 e 25).

A thread distribuída utiliza o provável tamanho de fila que ela irá encontrar no próximo nodo a ser executado para relacionar os tempos de resposta e os tamanhos das filas dos servidores, ambos de ativações passadas. O cálculo desta relação é feito através de *Regressão Linear* [34], a qual é definida como:

$$Y_i = \alpha + \beta X_i$$

onde Y_i representa o tempo de resposta estimado da thread distribuída em um nodo e X_i representa o tamanho da fila do servidor de aperiódicas do nodo em questão. As variáveis α e β representam estimativas para a relação entre os tempos de resposta de um determinado segmento local e os respectivos tamanhos das filas dos servidores dos nodos em que este segmento executou nas ativações passadas. Esta relação é obtida com o método dos mínimos quadrados, a partir de um conjunto de observações *(x1,y1), (x2,y2), ..., (xn,yn)*, onde *xi* representa o tamanho da fila do servidor antes do início da execução do segmento local e *yi* representa o tempo de resposta deste segmento. O método dos mínimos quadrados é definido da seguinte forma:

$$\beta = \frac{n\sum(x_i y_i) - (\sum x_i)(\sum y_i)}{n\sum x_i^2 - (\sum x_i)^2}$$

$$Se \quad (\beta < 0) \quad entao \quad \beta = 0;$$

$$\alpha = \frac{\sum y_i - \beta \sum x_i}{n}$$

As estimativas para α e β podem ser calculadas no nodo origem da thread distribuída, antes da sua ativação, já que os valores de x e y observados no passado ficam armazenados neste nodo. Conforme a TD transpõe os nodos do sistema, executando chamadas remotas, ela armazena na estrutura *Parâmetros para Previsão* (seção 4.3.2) alguns dados, como o tempo de resposta local do nodo visitado e o tamanho da fila do servidor quando ela chegou ao nodo. Estes dados são carregados com a thread distribuída até o final de sua execução, quando ela retorna ao nodo origem. Os dados trazidos pela thread distribuída são incluídos no histórico deste nodo, constituindo a estrutura apresentada na figura 7.2.

Após calcular a estimativa do tempo de resposta de cada método que compõe um

TD Segmento Local	Ativações Anteriores	Tamanho Fila do Servidor	Tempo Resposta
A	Ativação 1	10	40
	Ativação 2	20	70
	...		
	Ativação N	25	78
B	Ativação 1	33	85
	Ativação 2	–	–
	...		
	Ativação N	15	56
C	Ativação 1	8	93
	Ativação 2	31	105
	...		
	Ativação N	28	112
...			

Figura 7.2: Dados retornados pela TD após concluir uma ativação no sistema.

possível itinerário a ser seguido pela thread distribuída, é realizado o somatório destas estimativas produzindo uma estimativa do tempo de resposta do itinerário. As estimativas de tempo de resposta dos itinerários são somadas e multiplicadas pela sua probabilidade, definindo o valor da variável $TRespEsperado$.

Utilizando o mesmo exemplo do capítulo anterior, as figuras 7.3 e 7.4 apresentam, respectivamente, os possíveis itinerários que a thread distribuída pode executar e o histórico da thread distribuída, armazenado no nodo origem. O histórico contém os itinerários que a thread distribuída pode executar, os segmentos locais que executam cada método remoto (*coluna sl*), os tempos médios de computação de cada segmento local (indicados pela coluna C), e os respectivos deadlines locais (indicados pela coluna dl). Para fins de escalonamento, o particionamento do deadline é realizado pelo método EQF [6] para o itinerário *Maior Número de Saltos*.

Considerando que o mecanismo de previsão seja acionado no nodo 4, os possíveis itinerários a serem seguidos pela thread distribuída são compostos pelos métodos remotos *M7-M8-M7-M4-M1* ou *M7-M9-M7-M4-M1*, os quais serão executados, respectivamente, pelos segmentos locais 7, 8, 15, 16, 17 ou 7, 9, 18, 19, 20 da thread distribuída. A figura 7.4 mostra que o nodo 4 armazena a composição da fila do servidor de aperiódicas dos nodos 1, 6, 7, 8, além de ter a sua própria fila.

O cálculo da regressão linear deste exemplo envolve a definição dos valores das variáveis

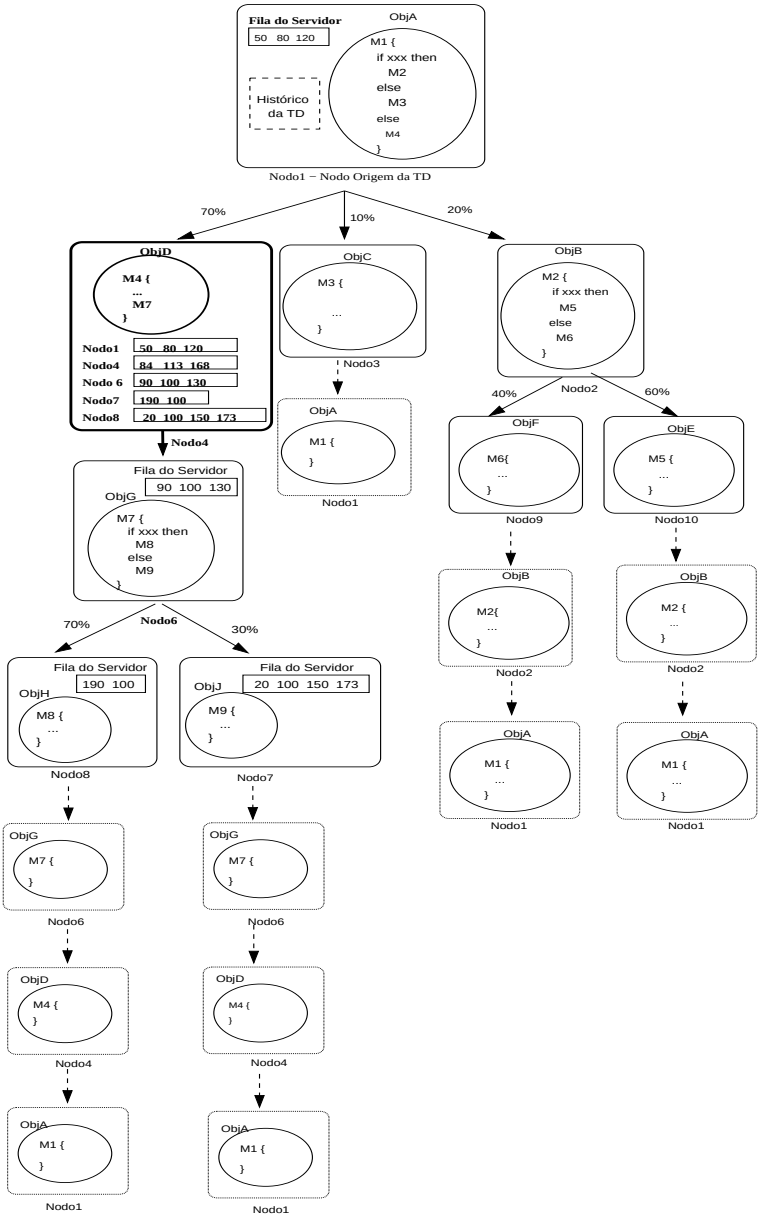

Figura 7.3: Itinerários que a thread distribuída pode executar.

HISTÓRICO DA THREAD DISTRIBUÍDA

Itinerários: Deadline fim a fim = 200ut

I1 = M1−M2−M5−M2−M1 (sl 1−2−5−10−11)

I2 = M1−M2−M6−M2−M1 (sl 1−2−6−12−13)

I3 = M1−M3−M1 (sl 1−3−14)

I4 = M1−M4−M7−M8−M7−M4−M1 (sl 1−4−7−8−15−16−17)

I5 = M1−M4−M7−M9−M7−M4−M1 (sl 1−4−7−9−18−19−20)

sl	C	dl	sl	C	dl
1	40	100	11	13	200
2	10	113	12	7	163
3	15	179	13	28	200
4	5	105	14	4	200
5	20	137	15	13	149
6	30	153	16	9	167
7	10	114	17	17	200
8	5	123	18	25	159
9	25	136	19	34	189
10	41	185	20	12	200

Figura 7.4: Histórico da thread distribuída.

Y7, Y8, Y9, Y15, Y16, Y17, Y18, Y19 e Y20 que representam os tempos de resposta estimados dos segmentos locais 7, 8, 9, 15, 16, 17, 18, 19, 20 respectivamente. Para Y7, por exemplo, o valor de X é 2 porque o deadline local do segmento local 7 é 114 e a fila do servidor do nodo 6 (que hospeda M7) é composta pelos deadlines locais 90, 100 e 130. O valor de X para os demais segmentos locais é, respectivamente, 1, 2, 3, 2, 3, 3, 3, 3.

As estimativas para α e β de cada segmento local são definidas a partir dos valores apresentados na figura 7.5 e as variáveis Y recebem os seguintes valores:

sl	Ativações Anteriores	Tam Fila Servidor	Tempo Resposta
7	Ativação 1	10	140
	Ativação 2	20	140
	Ativação 3	5	160
	Ativação 4	5	152
	Ativação 5	18	250
8	Ativação 1	15	190
	Ativação 2	30	220
	Ativação 3	–	–
	Ativação 4	26	250
	Ativação 5	–	–
9	Ativação 1	–	–
	Ativação 2	–	–
	Ativação 3	2	150
	Ativação 4	–	–
	Ativação 5	10	200
15	Ativação1	12	200
	Ativação2	8	123
	Ativação3	–	–
	Ativação4	18	250
	Ativação 5	–	–
16	Ativação1	5	80
	Ativação2	5	100
	Ativação3	–	–
	Ativação4	13	167
	Ativação 5	–	–
17	Ativação1	11	95
	Ativação2	20	216
	Ativação3	–	–
	Ativação4	8	90
	Ativação 5	–	–
18	Ativação 1	–	–
	Ativação 2	–	–
	Ativação 3	13	150
	Ativação 4	–	–
	Ativação 5	9	110
19	Ativação 1	–	–
	Ativação 2	–	–
	Ativação 3	7	124
	Ativação 4	–	–
	Ativação 5	14	230
20	Ativação 1	–	–
	Ativação 2	–	–
	Ativação 3	19	210
	Ativação 4	–	–
	Ativação 5	6	50

Figura 7.5: Dados usados no cálculo da regressão linear.

$$Y7 = 175,1 + 2,95 \times 2 \qquad Y7 = 181$$

$$Y8 = 155,2 + 2,73 \times 1 \qquad Y8 = 158,01$$

$$Y9 = 137,5 + 6,25 \times 2 \qquad Y9 = 150$$

$$Y15 = 34,5 + 12,35 \times 3 \qquad Y15 = 71,56$$

$$Y16 = 41,87 + 9,625 \times 2 \qquad Y16 = 61,12$$

$$Y17 = -11,66 + 11,17 \times 3 \qquad Y17 = 21,87$$

$$Y18 = 20 + 10 \times 3 \qquad Y18 = 50$$

$$Y19 = 18 + 15,14 \times 3 \qquad Y19 = 63,42$$

$$Y20 = -23,85 + 12,30 \times 3 \qquad Y20 = 13,07$$

O somatório dos valores de Y para os dois possíveis itinerários estão nas variáveis *Itinerário1* e *Itinerário2*. Este somatório considera o tempo de rede (neste exemplo foi considerado 2ut como tempo gasto pela thread distribuída entre dois nodos, logo o tempo de rede em cada itinerário é igual a 10). A variável $TRespEsperado$ recebe os tempos de resposta estimados de

cada itinerário (variáveis *Itinerário1* e *Itinerário2*) e multiplica pela respectiva probabilidade do itinerário ser seguido pela thread distribuída.

$$Itinerario1 = Y7 + Y8 + Y15 + Y16 + Y17 + 10$$
$$Itinerario1 = 181 + 158,01 + 71,56 + 61,12 + 21,87$$
$$Itinerario1 = 503,6;$$
$$Itinerario2 = Y7 + Y9 + Y18 + Y19 + Y20 + 10$$
$$Itinerario2 = 181 + 150 + 50 + 63,42 + 13,07$$
$$Itinerario2 = 467,5;$$
$$TRespEsperado = Itinerario1 \times 0,7 + Itinerario2 \times 0,3$$
$$TRespEsperado = 503,6 \times 0,7 + 467,5 \times 0,3$$
$$TRespEsperado = 492,79;$$

Supondo que o tempo de resposta local da thread distribuída no nodo 4 tenha sido igual a 100, a probabilidade de cumprir o deadline fim a fim, definida pelo mecanismo ASQ, é dada por:

$$P_k(ASQ) = ((200 - (100 + 492,79))/492,79) + \sigma$$
$$P_k(ASQ) = -0,297$$
$$Prob_k(ASQ) = 0;$$

Após a thread distribuída concluir sua execução, a qualidade da previsão realizada pelo mecanismo ASQ é avaliada através das métricas definidas na seção 4.5.

7.2.1 Tamanho da Estrutura *Parâmetros para Previsão*

A estrutura *Parâmetros para Previsão* de uma thread distribuída tende a manter um tamanho equilibrado porque a thread distribuída armazena e descarta dados conforme sua execução avança pelo sistema. Sempre que uma thread distribuída chega em um nodo, ela armazena na estrutura *Parâmetros para Previsão* o tamanho e a composição da fila do servidor. O tempo de resposta da thread distribuída neste nodo também é armazenado nesta estrutura.

Por outro lado, os valores de α e β dos nodos já visitados não são mais necessários e podem ser descartados da estrutura *Parâmetros para Previsão*. A composição dos itinerários da thread distribuída também tende a diminuir porque os nodos visitados não precisam ficar armazenados nesta estrutura. A fila do servidor de aperiódicas do próximo nodo que a thread distribuída irá visitar não precisa ser transportada, porque o próprio nodo sempre possui informações atualizadas sobre ele mesmo.

7.3 Simulações

Com o objetivo de avaliar a qualidade das previsões realizadas pelo mecanismo ASQ, simulações foram realizadas. As condições das simulação foram as mesmas usadas nos previsores *Milestones* e *Folga Restante (FR)*, definidas na seção 5.3.

7.3.1 Simulações com a Carga 1 - *Threads Distribuídas Tipo Pipeline*

Este tipo de thread distribuída executa uma única seqüencia de métodos remotos (um *pipeline*). Por isso, para fins de previsão, elas não utilizam possíveis itinerários de execução. Os resultados das simulações desta carga, segundo a métrica *Taxa Relativa de Erro* ($E(z)$), são apresentados na tabela 7.1.

Neste tipo de carga, o mecanismo de previsão ASQ obteve as menores taxas de erro, em todos os deadlines. A diferença com relação aos demais previsores é significativa na medida em que é possível usar os intervalos de confiança deste previsor em cada deadline e os resultados não se sobrepõem aos resultados dos demais previsores. Considerando, em cada deadline, os resultados próximos em 1% dos demais previsores, percebe-se que nenhum deles se aproxima dos resultados obtidos pelo previsor ASQ.

Comparando os resultados do mecanismo ASQ em cada deadline, observa-se que a menor taxa de erro foi obtida no deadline 100 ($0,014$) e a maior, no deadline 500 ($0,097$). Ocorre que o deadline 100 é considerado um deadline apertado, isto é, existe pouca folga para as threads distribuídas concluírem suas execuções e isso faz com que a maioria delas não consiga cumprir seu deadline fim a fim. O cálculo da previsão indica esta situação definindo uma pequena probabilidade da thread distribuída cumprir seu deadline. Com isso, a taxa de erro, que mede a distância que separa a previsão do que de fato ocorreu, é a menor em

Tabela 7.1: Taxa de Erro do Mecanismo ASQ para Threads Distribuídas tipo *Pipeline*

	Deadline, Taxa de Erro e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,223	±0,012	0,353	±0,014	0,280	±0,023	0,181	±0,027	0,126	±0,024
MED-MaNS	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MED-MeNS	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MED-Pond	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MED-MProv	0,052	±0,006	0,190	±0,010	0,198	±0,012	0,156	±0,018	0,116	±0,019
MEQS-MaNS	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQS-MeNS	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQS-Pond	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQS-MProv	0,043	±0,005	0,166	±0,011	0,173	±0,010	0,160	±0,012	0,125	±0,014
MEQF-MaNS	0,041	±0,005	0,150	±0,010	0,160	±0,009	0,146	±0,011	0,116	±0,014
MEQF-MeNS	0,041	±0,005	0,150	±0,010	0,160	±0,009	0,146	±0,011	0,116	±0,014
MEQF-Pond	0,041	±0,005	0,150	±0,010	0,160	±0,009	0,146	±0,011	0,116	±0,014
MEQF-MProv	0,041	±0,005	0,150	±0,010	0,160	±0,009	0,146	±0,011	0,116	±0,014
FR	0,028	±0,006	0,102	±0,007	0,119	±0,010	0,102	±0,016	0,078	±0,017
ASQ	**0,014**	±0,003	**0,079**	±0,006	**0,097**	±0,007	**0,083**	±0,012	**0,061**	±0,014

relação aos outros deadlines.

Já o deadline 500 é considerado um deadline justo, onde as threads distribuídas têm folga para concluir sua execução, mas esta folga não é suficientemente grande para garantir que ela irá cumprir seu deadline fim a fim mesmo estando em um nodo sobrecarregado. A previsão de perda de deadlines neste caso se torna mais difícil porque apesar da thread distribuída ter folga para cumprir seu deadline, o que realmente vai determinar este cumprimento é o estado dos nodos onde ela irá executar. O previsor ASQ considera a carga dos nodos que a thread distribuída poderá executar, através do tamanho da fila do servidor e por isso sua taxa de erro é a menor em relação aos demais previsores. Entretanto, comparando os resultados do previsor ASQ nos deadlines simulados, percebe-se que no deadline 500, ele obteve a maior taxa de erro. Com deadlines justos, mesmo que o previsor ASQ considere a carga dos nodos, a previsão de cumprir um deadline é mais difícil. O gráfico da figura 7.6 mostra os valores da tabela 7.1.

Considerando a métrica *Taxa de Previsões Corretas* ($PC(z)$), os resultados das simulações com a carga 1 são apresentados na tabela 7.2.

O previsor ASQ não obteve as melhores taxas de previsões corretas, nos deadlines sim-

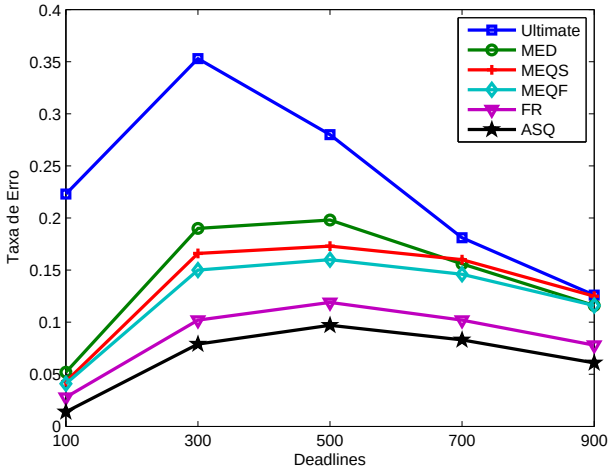

Figura 7.6: Taxa de Erro do Previsor ASQ para Threads Distribuídas tipo Pipeline.

Tabela 7.2: Taxa de Previsões Corretas para Threads Distribuídas tipo *Pipeline*

	Deadline, Taxa de Previsões Corretas (PC) e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,705	±0,089	0,600	±0,096	0,714	±0,089	0,831	±0,073	0,893	±0,061
MED-MaNS	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,880	±0,064	0,906	±0,057
MED-MeNS	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,880	±0,064	0,906	±0,057
MED-Pond	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,880	±0,064	0,906	±0,057
MED-MProv	0,984	±0,024	0,851	±0,070	0,819	±0,075	0,880	±0,064	0,906	±0,057
MEQS-MaNS	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQS-MeNS	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQS-Pond	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQS-MProv	**0,993**	±0,017	0,891	±0,061	0,887	±0,062	0,902	±0,058	0,935	±0,048
MEQF-MaNS	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041
MEQF-MeNS	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041
MEQF-Pond	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041
MEQF-MProv	**0,992**	±0,017	**0,937**	±0,048	**0,929**	±0,050	**0,938**	±0,047	**0,954**	±0,041
FR	0,976	±0,030	0,902	±0,058	0,886	±0,062	0,898	±0,059	0,923	±0,061
ASQ	**0,991**	±0,018	**0,933**	±0,049	**0,923**	±0,052	**0,929**	±0,050	**0,947**	±0,044

ulados. O previsor MEQF se manteve com os melhores resultados e o previsor *Ultimate* com as piores taxas. Entretanto, o previsor ASQ alcança resultados próximos em 1% do melhor previsor (MEQF). Usando o intervalo de confiança (IC) do previsor ASQ, seus resultados empatam com os previsores *Milestones* e *FR*.

As taxas de previsões corretas do previsor ASQ foram melhores que as do previsor FR, indicando que o mecanismo ASQ realiza previsões mais realistas. Considerando apenas os resultados do previsor ASQ nos diferentes deadlines simulados, percebe-se que a melhor taxa de previsão correta ocorreu no deadline 100 e a pior, no deadline 500, justamente pelos motivos explicados anteriormente (previsão para threads distribuídas com deadlines apertados e justos). O gráfico da figura 7.7 mostra os valores da tabela 7.2.

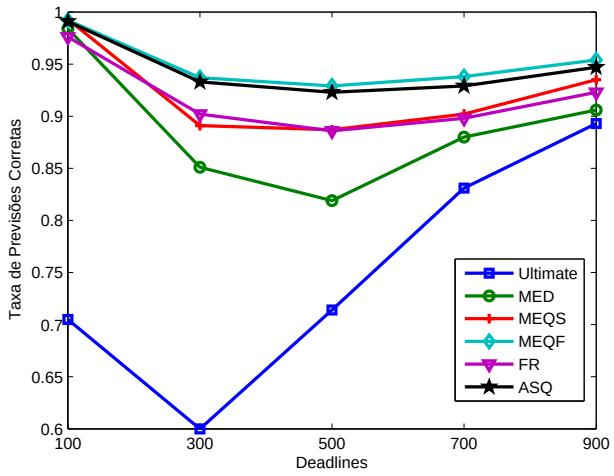

Figura 7.7: Taxa de Previsões Corretas para Threads Distribuídas tipo Pipeline.

Os resultados das simulações com threads distribuídas do tipo *pipeline* revelam que o mecanismo proposto é efetivamente melhor que os mecanismos baseados em *Milestones* e que o mecanismo *FR*. O fato do previsor ASQ considerar a provável carga que a thread distribuída irá encontrar nos próximos nodos em que ela irá executar melhora muito a qualidade das previsões.

Na métrica taxa relativa de erro, o mecanismo ASQ alcança resultados significativamente melhores que os demais mecanismos. Na outra métrica, taxa de previsões corretas, este mecanismo não alcança os melhores resultados, entretanto, ele empata com os demais se forem considerados os intervalo de confiança e os resultados próximos em 1%. A taxa de previsões corretas do mecanismo ASQ é melhor que a do mecanismo FR, em todos os

deadlines simulados.

7.3.2 Simulações com a Carga 2 - *Threads Distribuídas Tipo Árvore Balanceada*

Esta carga representa as diferentes possibilidades de execução das seqüencias de métodos remotos que uma thread distribuída pode percorrer. Os itinerários são usados como uma forma de supor algumas seqüências de métodos possíveis de serem executados pela thread distribuída, mesmo reconhecendo o aspecto dinâmico que ela incorpora no sentido de que os métodos remotos que serão executados pela thread distribuída variam conforme o estado do sistema distribuído em que ela atua. Uma árvore balanceada representa a execução de um mesmo número de métodos remotos, independente do itinerário que a thread distribuída possa seguir. As variações entre as árvores se referem ao número de métodos remotos, tempo de execução de cada método e as possíveis seqüências de execução.

A tabela 7.3 apresenta os resultados das simulações realizadas com esta carga, segundo a métrica Taxa de Erro Relativo ($E(z)$).O previsor ASQ apresenta a menor taxa de erro na maioria dos deadlines simulados, com exceção do deadline 100 com o qual o previsor FR apresenta a menor taxa de erro. Entretanto, considerando os intervalo de confiança destes dois mecanismos em todos os deadlines simulados, é possível afirmar que as taxas de erro geradas por eles empatam.

Considerando as taxas de erro dos outros mecanismos próximas em 1% do previsor ASQ, percebe-se que nenhum deles alcança resultados próximos do mecanismo ASQ.

Analisando os resultados deste previsor em cada deadline simulado, verifica-se que a menor taxa de erro foi obtida no deadline 100 e a maior no deadline 300. Conforme discutido na seção 7.3.1, estes resultados ocorrem em função do deadline 100 ser considerado apertado e o deadline 300 ser considerado justo. A chance de acertar uma previsão de cumprimento de um deadline para uma thread distribuída que possui um deadline justo se torna mais difícil porque se a carga do sistema aumenta após a previsão ter sido realizada, muito provavelmente a thread distribuída irá perder o deadline. Por outro lado, se o sistema não sofre uma sobrecarga após a previsão ter sido realizada, provavelmente a previsão irá se confirmar.

A figura 7.8 mostra um gráfico com os resultados da tabela 7.3. São mostrados apenas

Tabela 7.3: Taxa Relativa de Erro para Threads Distribuídas tipo Árvore Balanceada

	Deadline, Taxa de Erro e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,413	±0,019	0,379	±0,017	0,203	±0,017	0,120	±0,016	0,078	±0,012
MED-MaNS	0,038	±0,005	0,263	±0,011	0,178	±0,011	0,102	±0,011	0,064	±0,009
MED-MeNS	0,102	±0,007	0,301	±0,011	0,186	±0,013	0,110	±0,013	0,070	±0,011
MED-Pond	0,053	±0,005	0,279	±0,011	0,181	±0,012	0,104	±0,012	0,066	±0,010
MED-MProv	0,067	±0,007	0,276	±0,011	0,181	±0,012	0,105	±0,012	0,067	±0,010
MEQS-MaNS	0,051	±0,005	0,223	±0,011	0,190	±0,008	0,120	±0,010	0,072	±0,009
MEQS-MeNS	0,092	±0,005	0,251	±0,012	0,189	±0,009	0,116	±0,010	0,071	±0,009
MEQS-Pond	0,065	±0,005	0,229	±0,011	0,190	±0,008	0,119	±0,010	0,072	±0,009
MEQS-MProv	0,067	±0,005	0,233	±0,011	0,189	±0,009	0,117	±0,010	0,071	±0,009
MEQF-MaNS	0,049	±0,004	0,213	±0,010	0,172	±0,008	0,110	±0,009	0,068	±0,009
MEQF-MeNS	0,099	±0,006	0,252	±0,012	0,175	±0,009	0,109	±0,010	0,068	±0,009
MEQF-Pond	0,064	±0,005	0,220	±0,010	0,171	±0,008	0,109	±0,009	0,068	±0,009
MEQF-MProv	0,069	±0,006	0,227	±0,011	0,171	±0,008	0,108	±0,009	0,068	±0,009
FR	**0,012**	±0,003	0,158	±0,008	0,121	±0,012	0,066	±0,012	0,032	±0,008
ASQ	0,017	±0,003	**0,142**	±0,008	**0,101**	±0,010	**0,050**	±0,009	**0,024**	±0,006

o itinerário MaNS de cada previsor *Milestone*.

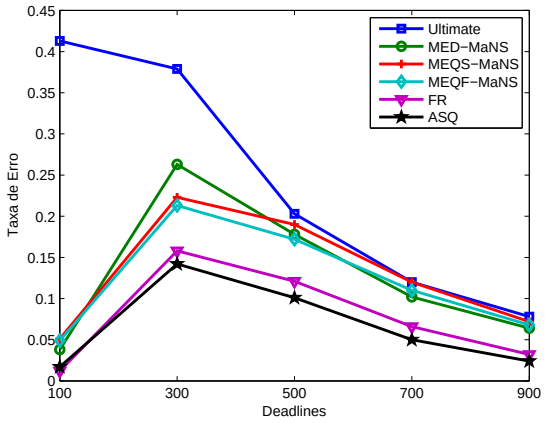

Figura 7.8: Taxa de Erro do Previsor ASQ para TDs tipo Árvore Balanceada.

A tabela 7.4 apresenta os resultados das simulações realizadas com a carga 2, segundo a métrica *Taxa de Previsões Corretas* ($PC(z)$). Conforme descrito nos capítulos anteriores, o previsor MEQF e seus respectivos itinerários obtiveram as maiores taxas de previsões corretas e o previsor *Ultimate* alcançou a menor taxa.

No deadline 100, o previsor ASQ obteve um resultado inferior ao do previsor FR. Nos

demais deadlines, o previsor ASQ obteve taxas de previsões corretas maiores que o previsor FR. Usando o intervalo de confiança do previsor ASQ nos deadlines simulados, observa-se uma situação de empate entre ele e os demais previsores (com exceção do previsor *Ultimate* nos deadlines 100 e 300).

Considerando os resultados próximos em 1% do previsor ASQ, percebe-se que no deadline 100, todos os previsores empatam. No deadline 300, os previsores MEQF-MProv e FR alcançam resultados próximos do previsor ASQ. Já no deadline 500, são os previsores MEQF-MaNS, MEQF-Mens e MEQF-MProv que alcançam resultados próximos do previsor ASQ. No deadline 700, os previsores MEQS-MProv e MEQF em todos os seus itinerários alcançam taxas de previsões corretas próximas em 1% do previsor ASQ. No deadline 900, com exceção do previsor MED-MeNS, todos os demais previsores alcançam resultados próximos do previsor ASQ.

A maior taxa de previsão correta do mecanismo ASQ nos deadlines simulados foi obtida no deadline 100 (*0,985*), seguida pelos deadlines 900, 700, 500 e 300, exatamente como ocorreu nas cargas simuladas anteriormente.

Tabela 7.4: Taxa de Previsões Corretas para Threads Distribuídas tipo Árvore Balanceada

	Deadline, Taxa de Previsões Corretas (PC) e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,482	±0,098	0,602	±0,096	0,817	±0,076	0,905	±0,058	0,937	±0,048
MED-MaNS	**0,983**	±0,025	0,808	±0,077	0,871	±0,066	0,941	±0,046	**0,974**	±0,031
MED-MeNS	0,932	±0,049	0,737	±0,086	0,854	±0,069	0,928	±0,051	0,961	±0,038
MED-Pond	0,978	±0,028	0,778	±0,081	0,862	±0,068	0,940	±0,046	**0,971**	±0,033
MED-MProv	0,949	±0,043	0,782	±0,081	0,865	±0,067	0,937	±0,048	0,967	±0,035
MEQS-MaNS	**0,988**	±0,021	0,868	±0,066	0,883	±0,063	0,940	±0,047	0,967	±0,035
MEQS-MeNS	0,977	±0,029	0,826	±0,074	0,878	±0,064	0,942	±0,046	**0,970**	±0,034
MEQS-Pond	**0,988**	±0,021	0,861	±0,068	0,886	±0,062	0,942	±0,046	**0,969**	±0,034
MEQS-MProv	**0,980**	±0,027	0,848	±0,070	0,885	±0,063	0,943	±0,046	**0,969**	±0,034
MEQF-MaNS	**0,988**	±0,021	**0,904**	±0,058	**0,916**	±0,054	**0,951**	±0,042	**0,972**	±0,032
MEQF-MeNS	0,978	±0,029	0,847	±0,070	0,900	±0,059	**0,953**	±0,042	**0,978**	±0,029
MEQF-Pond	**0,990**	±0,019	0,900	±0,059	**0,919**	±0,053	**0,954**	±0,041	**0,974**	±0,031
MEQF-MProv	**0,981**	±0,027	0,884	±0,063	**0,913**	±0,055	**0,952**	±0,042	**0,974**	±0,031
FR	**0,987**	±0,022	0,850	±0,070	0,879	±0,064	0,934	±0,049	**0,968**	±0,034
ASQ	**0,985**	±0,024	0,884	±0,063	**0,909**	±0,056	**0,953**	±0,041	**0,977**	±0,029

A figura 7.9 apresenta um gráfico com os valores da tabela 7.4. Foi utilizado o itinerário MaNS para os previsores MED, MEQS e MEQF. Observa-se que o previsor MEQF alcança

melhores resultados nos deadlines 300 e 500, e que o previsor ASQ obteve resultados bem próximos do previsor MEQF. O previsor Ultimate apresentou as piores taxas de previsões corretas, seguido pelo previsor MED.

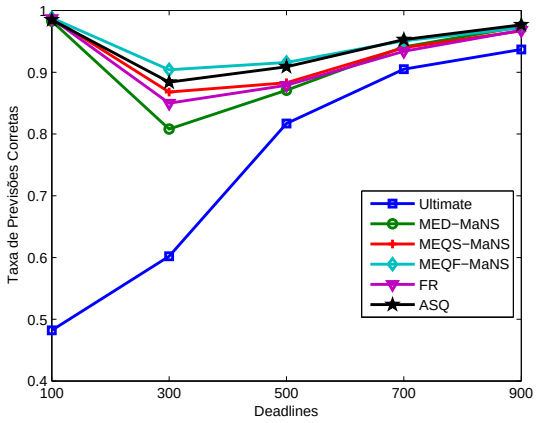

Figura 7.9: Taxa de Previsões Corretas do Mecanismo ASQ para TDs tipo Árvore Balanceada.

As simulações realizadas com árvores balanceadas mostram que o previsor ASQ obteve resultados superiores aos demais mecanismos de previsão na métrica Taxa Relativa de Erro ($E(z)$). Entretanto, usando o intervalo de confiança do previsor ASQ e do previsor FR, observa-se que suas taxas de erro se sobrepõem em todos os deadlines.

Na métrica Taxa de Previsões Corretas ($PC(z)$), o previsor ASQ não obteve as maiores taxas de previsões corretas. Entretanto, seu resultado é superior aos resultados do previsor FR. Conforme ocorreu na outra métrica, se os intervalos de confiança dos previsores ASQ e FR forem aplicados, observa-se que suas taxas de previsões corretas se sobrepõem em todos os deadlines.

7.3.3 Simulações com a Carga 3 - *Threads Distribuídas Tipo Árvore Não Balanceada*

Este tipo de carga também faz uso de supostos itinerários que uma thread distribuída pode executar. Uma árvore não balanceada representa a execução de um número diferente de métodos remotos. As variações entre as árvores refere-se ao número de métodos remotos, tempo de execução de cada método e as possíveis seqüências de execução. A tabela 7.5 apresenta os resultados desta carga, segundo a métrica Taxa Relativa de Erro.

Tabela 7.5: Taxa Relativa de Erro para Threads Distribuídas tipo Árvore Não Balanceada

	Deadline, Taxa de Erro e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,266	±0,016	0,418	±0,013	0,248	±0,014	0,130	±0,012	0,078	±0,012
MED-MaNS	0,033	±0,004	0,247	±0,001	0,213	±0,009	0,130	±0,010	0,082	±0,010
MED-MeNS	0,073	±0,006	0,321	±0,001	0,232	±0,011	0,134	±0,011	0,084	±0,011
MED-Pond	0,036	±0,004	0,268	±0,001	0,218	±0,010	0,131	±0,010	0,083	±0,011
MED-MProv	0,054	±0,006	0,276	±0,001	0,181	±0,010	0,105	±0,010	0,067	±0,011
MEQS-MaNS	0,041	±0,003	0,203	±0,009	0,223	±0,008	0,164	±0,010	0,106	±0,009
MEQS-MeNS	0,065	±0,005	0,232	±0,009	0,226	±0,009	0,157	±0,010	0,098	±0,009
MEQS-Pond	0,050	±0,004	0,205	±0,009	0,221	±0,008	0,162	±0,010	0,105	±0,009
MEQS-MProv	0,053	±0,004	0,210	±0,009	0,220	±0,008	0,160	±0,010	0,104	±0,009
MEQF-MaNS	0,038	±0,004	0,202	±0,009	0,203	±0,007	0,143	±0,009	0,093	±0,008
MEQF-MeNS	0,068	±0,005	0,259	±0,009	0,224	±0,010	0,139	±0,010	0,084	±0,009
MEQF-Pond	0,048	±0,004	0,210	±0,009	0,203	±0,007	0,142	±0,009	0,093	±0,008
MEQF-MProv	0,053	±0,004	0,211	±0,009	0,200	±0,007	0,140	±0,009	0,092	±0,008
FR	0,034	±0,005	0,136	±0,009	0,135	±0,009	0,090	±0,011	0,056	±0,013
ASQ	**0,012**	±0,002	**0,136**	±0,008	**0,118**	±0,008	**0,069**	±0,008	**0,037**	±0,009

O previsor ASQ obtém as menores taxas de erro na maioria dos deadlines simulados, com exceção do deadline 300, onde houve um empate entre os resultados dos previsores ASQ e FR (embora o previsor ASQ tenha um intervalo de confiança menor que o previsor FR). De todas as taxas de erro obtidas pelo previsor ASQ, em cada deadline simulado, o menor valor de taxa foi alcançado no deadline 100, seguido pelos deadlines 900, 700, 500 e 300. Isto confirma a dificuldade de realizar-se previsões corretas em deadlines justos (300 e 500).

Usando o intervalos de confiança de cada taxa de erro de cada deadline do previsor ASQ, percebe-se que o resultado não se sobrepõe à de nenhum outro previsor, com exceção da taxa de erro do previsor FR, no deadline 300. Considerando os resultados dos demais mecanismos, próximos em 1% das taxas de erro do mecanismo ASQ, observa-se que nenhum previsor alcançou resultados próximos (com exceção do previsor FR no deadline 300.)

A figura 7.10 mostra um gráfico com os valores da tabela 7.5. Em todos os deadlines simulados, o previsor ASQ obteve as menores taxas de previsões corretas, com exceção do deadline 300, onde seu resultado empatou com o resultado do previsor FR.

A tabela 7.6 apresenta os resultados das simulações realizadas com a carga 3, segundo a métrica Taxa de Previsões Corretas ($PC(z)$). Em todos os deadlines, o previsor ASQ não

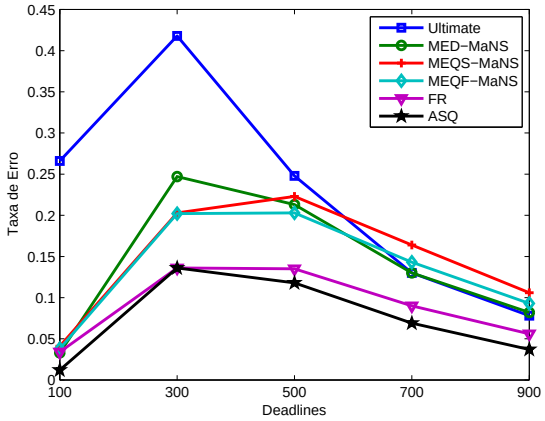

Figura 7.10: Taxa de Erro do Previsor FR para TDs tipo Árvore Não Balanceada.

alcançou as maiores taxas de previsões corretas, sendo que o mecanismo MEQF se manteve com os melhores resultados. Comparando as taxas de previsões corretas dos mecanismos ASQ e FR, observa-se que o previsor ASQ obteve melhores resultados que o FR, em todos os deadlines, afirmando a capacidade do mecanismo ASQ em realizar uma quantidade menor de falsas previsões.

Considerando os resultados obtidos pelo mecanismo ASQ, percebe-se que a maior taxa de previsão correta deste previsor foi alcançada no deadline 100, seguida pelos deadlines 900,700, 500 e 300, como ocorreu nas demais cargas simuladas.

O mecanismo ASQ empata com os previsores *Milestones* e FR se forem usados seus intervalos de confiança. Na comparação da maior taxa de previsões corretas de cada previsor *Milestone*, juntamente com os previsores FR e ASQ, percebe-se que os resultados se sobrepõem.

Comparando os resultados próximos em 1% da taxa de erro do previsor ASQ, é possível verificar que no deadline 100 os previsores FR, MEQS e MEQF (em todos os itinerários) e os previsores MED-MaNS, MED-Pond e MED-MProv alcançaram resultados próximos do previsor ASQ. No deadline 300, apenas o previsor MEQS-Pond obteve uma taxa de erro próxima do previsor ASQ. No deadline 500, apenas o previsor MEQF-MaNS alcançou um resultado próximo do previsor ASQ. Já no deadline 700, os previsores MEQF-MaNS, MEQF-Pond e MEQF-MProv tiveram taxas de erro próximas do ASQ e no deadline 900, todos os previsores alcançaram resultados próximos do mecanismo ASQ, com exceção dos mecanismos FR, MEQS-MaNS, MEQS-Pond e MEQS-MProv.

Tabela 7.6: Taxa de Previsões Corretas para Threads Distribuídas tipo Árvore Não Balanceada

	Deadline, Taxa de Previsões Corretas (PC) e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,688	±0,091	0,519	±0,098	0,770	±0,082	0,898	±0,059	0,951	±0,043
MED-MaNS	0,982	±0,026	0,809	±0,077	0,842	±0,072	0,913	±0,055	0,957	±0,040
MED-MeNS	0,939	±0,047	0,661	±0,093	0,803	±0,078	0,905	±0,057	0,953	±0,042
MED-Pond	0,980	±0,027	0,771	±0,082	0,830	±0,074	0,911	±0,056	0,955	±0,040
MED-MProv	0,954	±0,041	0,771	±0,082	0,835	±0,073	0,911	±0,056	0,956	±0,040
MEQS-MaNS	0,990	±0,019	0,871	±0,066	0,846	±0,071	0,900	±0,059	0,946	±0,044
MEQS-MeNS	0,987	±0,022	0,879	±0,064	0,852	±0,070	0,908	±0,057	0,955	±0,041
MEQS-Pond	0,990	±0,019	0,885	±0,063	0,854	±0,069	0,905	±0,057	0,949	±0,043
MEQS-MProv	0,990	±0,019	0,883	±0,063	0,856	±0,069	0,910	±0,056	0,951	±0,042
MEQF-MaNS	0,991	±0,018	0,907	±0,057	0,903	±0,058	0,941	±0,046	0,966	±0,036
MEQF-MeNS	0,987	±0,022	0,834	±0,073	0,851	±0,070	0,922	±0,052	0,963	±0,037
MEQF-Pond	**0,993**	±0,017	**0,917**	±0,054	**0,911**	±0,056	0,944	±0,045	**0,968**	±0,035
MEQF-MProv	0,991	±0,018	0,908	±0,057	0,910	±0,056	**0,945**	±0,045	**0,968**	±0,035
FR	0,969	±0,034	0,872	±0,065	0,866	±0,067	0,907	±0,057	0,943	±0,046
ASQ	0,990	±0,019	0,893	±0,060	0,895	±0,060	0,939	±0,047	0,963	±0,037

A figura 7.11 apresenta um gráfico com os valores da tabela 7.6. Foi utilizado o itinerário MaNS para os previsores MED, MEQS e MEQF. Percebe-se que os previsores mantiveram o mesmo comportamento que na carga anterior. O previsor ASQ alcançou resultados próximos do previsor MEQF, o qual obteve as melhores taxas de previsões corretas na maioria dos deadlines. Nos deadlines justos (300 a 700), estes previsores mantém uma distância maior dos demais previsores, comparado com os resultados da carga anterior.

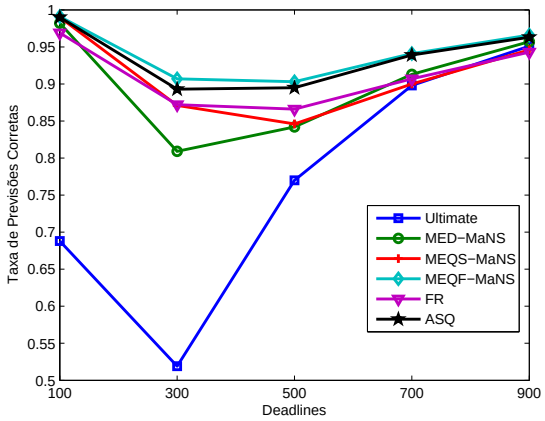

Figura 7.11: Taxa de Previsões Corretas do Mecanismo ASQ para TDs tipo Árvore Não Balanceada.

Assim como aconteceu com as cargas simuladas anteriormente, neste tipo de carga o previsor ASQ obteve as menores taxas de erro em comparação com os demais previsores. A respeito da taxa de previsões corretas, este previsor não alcançou os melhores resultados, embora seu desempenho tenha sido superior ao do mecanismo FR.

7.3.4 Simulações com a Carga 4 - *Threads Distribuídas Carga Mista*

A tabela 7.7 apresenta os resultados das simulações com a carga mista, composta por threads distribuídas do tipo *pipeline*, balanceada e não balanceada, segundo a métrica *Taxa Relativa de Erro*.

O mecanismo ASQ apresenta os melhores resultados, entre todos os mecanismos, em todos os deadlines. A menor taxa de erro ocorreu no deadline 100, seguida pelos deadlines 900, 700, 500 e 300, indicando que a tarefa de prever a perda de um deadline justo é mais difícil.

Usando os intervalo de confiança do previsor ASQ, percebe-se que nos deadlines 100 e 500, os resultados não se sobrepõem aos resultados do previsor FR. Com os demais deadlines, existe sobreposição entre os resultados destes previsores. Considerando os resultados próximos em 1% das taxas de erro do previsor ASQ, observa-se que nenhum outro previsor alcançou resultados próximos.

A figura 7.12 mostra um gráfico com os resultados da tabela 7.7. Para cada previsor *Milestone* foi usado o suposto itinerário MaNS. O gráfico mostra o previsor ASQ com menores taxas de erros que os demais previsores, em todos os deadlines. O previsor *Ultimate* apresenta a maior taxa de erro para os deadlines 100, 300 e 500.

A tabela 7.8 apresenta os resultados das simulações realizadas com a carga 4, segundo a métrica Taxa de Previsões Corretas ($PC(z)$). Nos deadlines 700 e 900, o previsor ASQ obteve as maiores taxas de acertos. Nos demais deadlines, o previsor MEQF manteve as melhores taxas de previsões corretas.

Em todos os deadlines, o previsor ASQ apresentou melhores taxas em relação ao previsor FR. Considerando os resultados do previsor ASQ, pode-se verificar que a maior taxa de previsões corretas ocorreu no deadline 100, seguida pelos deadlines 900, 700, 500 e 300.

Usando os intervalos de confiança do previsor ASQ, é possível afirmar que no deadline

Tabela 7.7: Taxa Relativa de Erro para Threads Distribuídas tipo Carga Mista

	Deadline, Taxa de Erro e intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Ultimate	0,315	±0,017	0,367	±0,016	0,218	±0,020	0,118	±0,016	0,078	±0,014
MED-MaNS	0,047	±0,005	0,234	±0,010	0,184	±0,013	0,110	±0,012	0,074	±0,012
MED-MeNS	0,085	±0,008	0,266	±0,011	0,191	±0,014	0,114	±0,013	0,077	±0,012
MED-Pond	0,055	±0,006	0,245	±0,010	0,186	±0,013	0,111	±0,012	0,075	±0,012
MED-MProv	0,065	±0,007	0,242	±0,010	0,185	±0,013	0,111	±0,012	0,075	±0,012
MEQS-MaNS	0,052	±0,005	0,203	±0,011	0,194	±0,010	0,135	±0,011	0,091	±0,011
MEQS-MeNS	0,076	±0,006	0,221	±0,011	0,193	±0,010	0,132	±0,011	0,089	±0,011
MEQS-Pond	0,061	±0,005	0,206	±0,011	0,193	±0,010	0,135	±0,011	0,091	±0,011
MEQS-MProv	0,062	±0,006	0,209	±0,011	0,191	±0,010	0,133	±0,011	0,091	±0,011
MEQF-MaNS	0,049	±0,005	0,194	±0,010	0,175	±0,009	0,120	±0,010	0,082	±0,011
MEQF-MeNS	0,078	±0,006	0,223	±0,010	0,181	±0,011	0,118	±0,011	0,080	±0,011
MEQF-Pond	0,059	±0,005	0,198	±0,010	0,175	±0,009	0,119	±0,010	0,082	±0,011
MEQF-MProv	0,062	±0,006	0,201	±0,010	0,173	±0,009	0,118	±0,010	0,082	±0,011
FR	0,026	±0,006	0,133	±0,008	0,114	±0,012	0,068	±0,011	0,044	±0,011
ASQ	**0,018**	±0,003	**0,120**	±0,007	**0,097**	±0,010	**0,053**	±0,009	**0,034**	±0,008

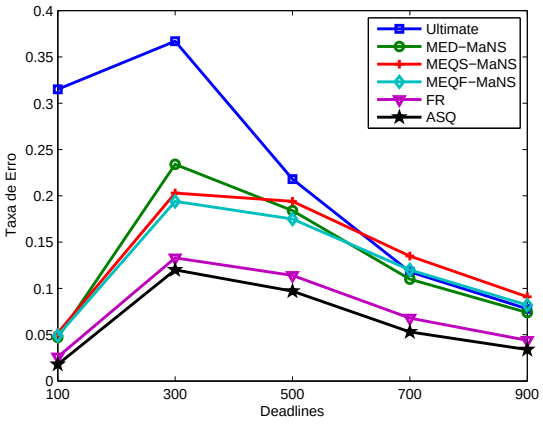

Figura 7.12: Taxa de Erro do Previsor ASQ para TDs Carga Mista.

100 este previsor empata com os demais, sendo que para esta comparação foi utilizada a maior taxa de previsões corretas de cada previsor *Milestone*. No deadline 300, a taxa de previsões corretas do mecanismo ASQ empata com os previsores FR, MEQS-Pond e MEQF-Pond. No deadline 500, o empate ocorre com os previsores FR, MEQS-MProv e MEQF-MProv. Nos deadlines 700 e 900, o empate acontece com os previsores FR, MED-MaNS, MEQS-MeNS, MEQF-MeNS e MEQF-MProv.

Comparando os resultados próximos em 1% dos resultados do mecanismo ASQ, observa-se que no deadline 100, a maioria dos previsores *Milestones* alcançam resultados próximos do previsor ASQ, com exceção dos previsores MED-MeNS, MED-MProv e FR. No deadline 300, a maioria dos previsores *Milestones* não alcança resultados próximos do previsor ASQ, com exceção do previsor MEQF-MProv.

No deadline 500, apenas os previsores MEQF-MaNS, MEQF-Pond e MEQF-MProv alcançam taxas corretas de previsão próximas em 1% do previsor ASQ. A mesma situação aparece no deadline 700, considerando também o previsor MEQF-MeNS com resultado próximo do previsor ASQ. No deadline 900, os previsores MEQF com todos os seus itinerários, o previsor MED com os itinerários MaNS, Pond e MProv e os previsores MEQS-MeNS e FR alcançam taxas de previsões corretas próximas do mecanismo ASQ.

Tabela 7.8: Taxa de Previsões Corretas para Threads Distribuídas - *Carga Mista*

	Deadline, Taxa de Previsões Corretas (PC) e Intervalo de Confiança (IC)									
	100		300		500		700		900	
Mecanismos de Previsão	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Ultimate	0,61	±0,096	0,596	±0,096	0,800	±0,078	0,907	±0,057	0,943	±0,045
MED-MaNS	**0,979**	±0,028	0,823	±0,075	0,859	±0,068	0,932	±0,049	**0,960**	±0,038
MED-MeNS	0,945	±0,045	0,760	±0,084	0,845	±0,071	0,924	±0,052	0,955	±0,041
MED-Pond	0,977	±0,029	0,803	±0,078	0,853	±0,069	0,930	±0,050	**0,959**	±0,039
MED-MProv	0,958	±0,039	0,805	±0,078	0,856	±0,069	0,930	±0,050	**0,958**	±0,039
MEQS-MaNS	**0,987**	±0,022	0,871	±0,066	0,873	±0,065	0,918	±0,054	0,953	±0,041
MEQS-MeNS	**0,981**	±0,027	0,863	±0,067	0,875	±0,065	0,924	±0,052	**0,958**	±0,039
MEQS-Pond	**0,987**	±0,022	0,873	±0,065	0,877	±0,064	0,920	±0,053	0,955	±0,041
MEQS-MProv	**0,983**	±0,025	0,869	±0,066	0,879	±0,064	0,922	±0,053	0,955	±0,040
MEQF-MaNS	**0,989**	±0,021	**0,913**	±0,055	**0,921**	±0,053	**0,949**	±0,043	**0,966**	±0,035
MEQF-MeNS	**0,981**	±0,027	0,876	±0,065	0,904	±0,058	**0,948**	±0,044	**0,967**	±,035
MEQF-Pond	**0,989**	±0,020	**0,917**	±0,054	**0,924**	±0,052	**0,951**	±0,042	**0,967**	±0,035
MEQF-MProv	**0,984**	±0,024	0,908	±0,057	**0,925**	±0,052	**0,951**	±0,042	**0,967**	±0,035
FR	0,975	±0,030	0,874	±0,065	0,887	±0,062	0,931	±0,050	0,955	±0,041
ASQ	**0,987**	±0,023	**0,902**	±0,058	**0,917**	±0,054	**0,952**	±0,042	**0,968**	±0,034

A figura 7.13 apresenta um gráfico com os valores da tabela 7.8. Os resultados dos previsores MEQF e ASQ são próximos em 1%, e o mecanismo MEQF apresenta taxas superiores em relação aos demais mecanismos. É possível observar que os resultados alcançados nesta carga seguem a mesma tendência que aqueles encontrados nas outras cargas simuladas, isto é, para esta métrica todas as cargas apresentaram o mesmo padrão de comportamento.

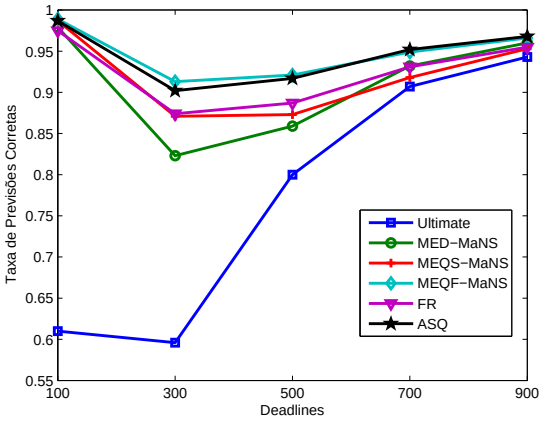

Figura 7.13: Taxa de Previsões Corretas do Mecanismo ASQ para TDs Carga Mista.

As simulações com a carga mista revelam que o mecanismo ASQ apresenta resultados significativamente melhores que os demais previsores na métrica taxa relativa de erro. Entretanto, na métrica taxa de previsões corretas este previsor não alcança os melhores resultados e o previsor MEQF se mantém com as melhores taxas.

7.4 Comparação do Previsor ASQ entre as Cargas Simuladas

A tabela 7.9 apresenta os resultados do previsor ASQ nas quatro cargas simuladas, segundo a métrica taxa relativa de erro. Nenhuma carga obteve as menores taxas de erro em todos os deadlines. No deadline 100, por exemplo, a Carga 3 obteve a menor taxa de erro. Já nos deadlines 300 e 500, as menores taxas foram obtidas na Carga 1 (no deadline 500 a Carga 4 obteve a mesma taxa de erro, porém o intervalo de confiança foi maior). Nos deadlines 700 e 900, a menor taxa de erro foi obtida na Carga 2. A figura 7.14 mostra os resultados da tabela 7.9.

A tabela 7.10 apresenta os resultados do previsor ASQ nas quatro cargas simuladas, segundo a métrica taxa de previsões corretas. Nos deadlines 100 a 500, percebe-se que a

Tabela 7.9: Taxa Relativa de Erro do Previsor ASQ

	Deadline, Taxa de Erro e Intervalo de Confiança (IC)									
	100		300		500		700		900	
Tipo de Carga	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*	*Erro*	*IC*
Carga 1	0,014	±0, 003	**0,079**	±0, 006	**0,097**	±0, 007	0,083	±0, 012	0,061	±0, 014
Carga 2	0,017	±0, 003	0,142	±0, 008	0,101	±0, 010	**0,050**	±0, 009	**0,024**	±0, 006
Carga 3	**0,012**	±0, 002	0,136	±0, 008	0,118	±0, 008	0,069	±0, 008	0,037	±0, 009
Carga 4	0,018	±0, 003	0,120	±0, 007	**0,097**	±0, 010	0,053	±0, 009	0,034	±0, 008

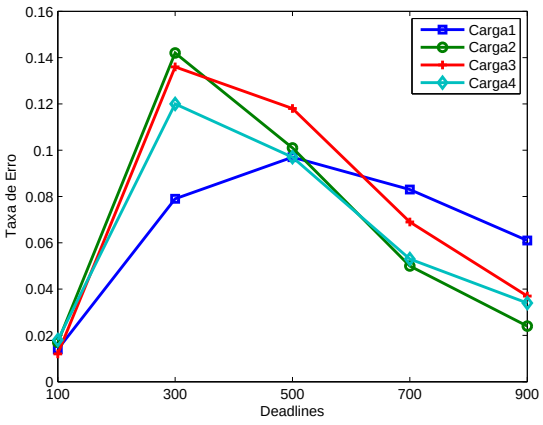

Figura 7.14: Taxa de Erro do Previsor ASQ para TDs tipo Árvores Mistas.

Carga 1 obteve as maiores taxas de previsões corretas e nos deadlines 700 e 900, a Carga 2 obteve os melhores resultados. A figura 7.15 mostra os resultados da tabela 7.10.

Tabela 7.10: Taxa de Previsões Corretas do Previsor ASQ

	Deadline, Taxa de Previsões Corretas (PC) e Intervalo de Confiança (IC)									
	100		300		500		700		900	
Tipo de Carga	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*	*PC*	*IC*
Carga 1	**0,991**	±0,018	**0,933**	±0,049	**0,923**	±0,052	0,929	±0,050	0,947	±0,044
Carga 2	0,985	±0,024	0,884	±0,063	0,909	±0,056	**0,953**	±0,041	**0,977**	±0,029
Carga 3	0,990	±0,019	0,893	±0,060	0,895	±0,060	0,939	±0,047	0,963	±0,037
Carga 4	0,987	±0,023	0,902	±0,058	0,917	±0,054	0,952	±0,042	0,968	±0,034

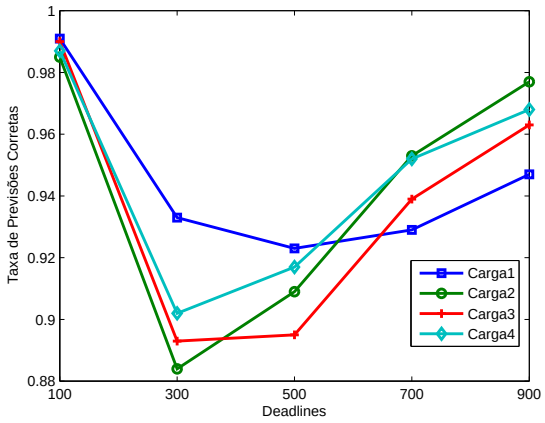

Figura 7.15: Taxa de Previsões Corretas do Previsor ASQ para TDs tipo Árvores Mistas.

7.5 Conclusões

Neste capítulo o mecanismo de previsão de perda de deadline ASQ foi descrito. Este mecanismo faz uso de informações das execuções passadas de uma thread distribuída e de informações do sistema distribuído conhecidas somente em tempo de execução. O cálculo da previsão utiliza regressão linear para gerar um tempo de resposta estimado a partir do nodo em que a thread distribuída se encontra no momento da previsão. Para isso, a thread distribuída consulta o tamanho e a composição das filas dos servidores dos prováveis nodos em que ela poderá executar. Cada nodo armazena a composição da fila do servidor de todos os outros nodos do sistema. Estas informações são trazidas pelas threads distribuídas, na medida em que elas transitam de um para outro nodo. Antes de partir para um próximo

nodo, a thread faz uma cópia destas informações e leva para o nodo em que ela irá executar. Esta cópia é feita para a estrutura Parâmetros de Previsão, que a thread carrega consigo conforme transpõe os nodos do sistema.

Após concluir sua execução em um nodo, o tempo de resposta da thread distribuída é avaliado. Se este tempo for maior ou igual a metade do seu deadline fim a fim, o mecanismo de previsão ASQ é acionado e os cálculos de previsão de perda de deadline são realizados. Quando a thread distribuída finaliza sua execução no sistema, as métricas Taxa Relativa de Erro - *E(z)* e na Taxa de Previsões Corretas - *PC(z)* avaliam a qualidade da previsão realizada. A primeira métrica verifica o quão distante de uma previsão exata foi a previsão realizada. Se a thread distribuída cumpriu o deadline fim a fim, o erro será o faltou para o previsor alcançar uma previsão 100% correta. Senão, o erro será o próprio valor da previsão.

A segunda métrica contabiliza o número de previsões corretas, de acordo com um critério especificado. Este critério define que se o valor resultante do cálculo da previsão for maior que 50% assume-se que a thread distribuída irá cumprir seu deadline fim a fim, senão assume-se que ela não irá cumprir seu deadline. Ao final da execução da thread distribuída, a métrica contabiliza a quantidade de acertos.

Nas simulações realizadas para avaliar a qualidade das previsões geradas por este mecanismo, foi possível observar a mesma tendência de resultados nas diferentes cargas simuladas. Nenhuma carga manteve os melhores resultados em todos os deadlines. Observa-se que para alguns deadlines um tipo de carga obteve melhores resultados que as demais. Esta situação ocorreu nas duas métricas usadas,*E(z)* e *PC(z)*.

Com relação à métrica taxa relativa de erro, foi possível observar que o previsor ASQ produz resultados significativamente melhores que os outros previsores. Na maioria das cargas simuladas, mesmo considerando o intervalo de confiança do mecanismo ASQ, percebe-se que seus resultados não se sobrepõem aos resultados dos demais previsores.

Na métrica taxa de previsões corretas, o previsor ASQ não obteve os melhores resultados na maioria das cargas simuladas. O previsor MEQF manteve as maiores taxas de previsões corretas. Entretanto, se os intervalos de confiança forem utilizados percebe-se que em todas as cargas os resultados do mecanismo ASQ e MEQF se sobrepõem. As taxas de previsões corretas do mecanismo ASQ foram melhores que as taxas dos previsores MED, MEQS e FR.

Um aspecto importante deste mecanismo se refere à validade das informações a respeito das filas dos servidores de aperiódicas armazenadas no nodo em que a previsão é realizada. Se a diferença entre o tempo da última atualização das filas e o tempo atual do sistema for muito grande, significa que elas não refletem o estado atual daqueles nodos. Esta questão pode ser resolvida em tempo de projeto, colocando em cada nodo algum método que seja freqüentemente invocado por alguma thread distribuída.

Nas simulações realizadas esta situação não ocorre porque todos os nodos do sistema são visitados por threads distribuídas com uma freqüência suficiente que mantém as informações atualizadas. Isto é, não existe um nodo que hospeda métodos remotos que são invocados um número de vezes muito menor que os outros métodos hospedados em outros nodos.

Capítulo 8

Trabalhos Relacionados

8.1 Introdução

A previsão a respeito de uma possível perda de deadline é importante na medida que ações podem ser executadas para evitar esta perda ou evitar o desperdício no uso de recursos computacionais por tarefas que provavelmente não irão cumprir seu deadline e conseqüentemente não irão trazer benefícios ao sistema.

Nos capítulos anteriores foram descritos três mecanismos de previsão de perda de deadline para sistemas tempo real implementados a partir de threads distribuídas. O primeiro mecanismo descrito, baseado em *Milestones*, pode ser considerado simples porque ele utiliza apenas informações conhecidas de ativações passadas e define os *Milestones* estaticamente, no nodo origem, antes da thread distribuída iniciar sua execução.

O segundo mecanismo proposto, baseado na *Folga Restante* (FR) da thread distribuída é um pouco mais sofisticado porque considera todos os possíveis itinerários que uma thread distribuída pode executar, além de informações das execuções anteriores da thread distribuída. O uso dos possíveis itinerários de uma thread ocorre em tempo de execução, através da consulta a estrutura *Parâmetros para Previsão*, carregada pela thread distribuída conforme ela visita os nodos do sistema.

O terceiro mecanismo proposto pode ser considerado o mais sofisticado dos três porque relaciona informações conhecidas de ativações passadas com informações conhecidas em tempo de execução, através do uso de regressão linear. O mecanismo utiliza a composição da fila

do servidor de aperiódicas dos nodos que compõem os possíveis itinerários que a thread distribuída pode executar. Esta composição se refere aos deadlines locais das demais threads ativas no sistema. O mecanismo também utiliza a estrutura *Parâmetros para Previsão* para realizar o cálculo da previsão, além de informações armazenadas no nodo onde a previsão está sendo realizada. Este mecanismo faz com que todas as threads distribuídas ativas no sistema colaborem na atualização das informações armazenadas nos nodos do sistema necessárias à previsão.

As simulações realizadas com estes mecanismos mostraram o bom desempenho deles segundo as métricas usadas na avaliação dos resultados. Houve uma crescente melhora destes resultados conforme os mecanismos ganhavam em sofisticação.

A literatura é abrangente em trabalhos que propõem algoritmos que fazem a previsão de tempos de resposta de tarefas e de desempenho de sistemas. Neste capítulo são descritos alguns trabalhos que tratam deste assunto. Na seção 8.2 são descritos algoritmos de previsão na área de sistemas autônomos, sistemas embutidos e de aplicações paralelas. Ainda nesta seção, são descritos trabalhos que propõem algoritmos de previsão para *workflows* e para sistemas distribuídos de tempo real. Na seção 8.3 são apresentadas as conclusões deste capítulo, indicando as principais diferenças dos algoritmos descritos na literatura e os propostos nesta tese.

8.2 Mecanismos de Previsão para Algumas Classes de Sistemas Computacionais

Mecanismos de previsão de tempo de resposta e de desempenho de sistemas são objeto de estudos em várias áreas da computação [35, 36, 32, 37, 28, 38, 39]. A seguir serão descritos alguns trabalhos que propõem algoritmos de previsão para sistemas autônomos (*autonomic systems*), sistemas embutidos, aplicações paralelas, *workflows* e para sistemas distribuídos de tempo real.

8.2.1 Algoritmos de Previsão para Sistemas Autônomos

O trabalho [35] apresenta o projeto e a implementação de um sistema de gerenciamento de serviços que prevê os tempos de resposta de aplicações *web* orientadas por transação. O serviço de previsão é iniciado por um cliente que solicita ao gerente de serviços qual o tempo de resposta de um determinado serviço. O gerente interage com o sistema, calcula o tempo de resposta previsto com base nas condições correntes do sistema e retorna esta informação ao cliente.

O cálculo da estimativa do tempo de resposta modela o atraso da rede e o atraso do servidor. Para modelar o atraso da rede, os autores consideram os parâmetros de uma transação TCP como tempo de inicio da transação, taxa de perda de pacotes, tamanho do pacote a ser enviado, entre outras informações. Para o atraso do servidor, os autores consideram o tempo de espera (chegada da requisição ao servidor até o primeiro pacote de dados ser enviado), *overhead* da CPU associado com a requisição e o tempo de disco (tempo de transferência dos dados do disco para a memória).

Em [36] os autores propõem um suporte para auto-previsão de sistemas de armazenamento distribuído. O objetivo do algoritmo proposto é prever o desempenho de uma determinada carga se os seus dados forem movidos do dispositivo A para o B. O sistema gera informações usadas pelos administradores para fins de análise de desempenho. Estas informações são sobre mudanças na codificação dos dados, adição ou remoção de dados e/ou nodos do sistema.

8.2.2 Algoritmos de Previsão para Sistemas Embutidos e Sistemas de Computação Paralela

O trabalho descrito em [32] considera um sistema embutido que executa várias aplicações diferentes. Uma aplicação é composta por um conjunto de serviços. Tarefas com deadlines não-críticos requisitam um serviço da aplicação. O algoritmo proposto determina a probabilidade de uma tarefa cumprir seu deadline, unindo todos os estados possíveis do sistema em dois: normal ou sobrecarregado.

Um histórico armazena o tempo de resposta médio de cada serviço da aplicação. Quando a tarefa finaliza sua execução, seu tempo de resposta é comparado com o tempo

de resposta médio associado com o serviço que a tarefa utilizou. O novo estado do sistema é definido como normal se o tempo de resposta da tarefa é menor ou igual ao tempo de resposta médio do serviço ou sobrecarregado se o tempo de resposta da tarefa é maior que o tempo médio de resposta do serviço.

O sistema mantém dois históricos, cada um com os tempos de resposta de cada serviço juntamente com a informação sobre o estado do sistema (normal ou sobrecarregado) no momento da chegada da tarefa. Quando uma tarefa chega no sistema e solicita um serviço, o tempo de resposta específico daquele serviço para a carga que o sistema se encontra no momento atual são usados para fins de previsão. Usando o registro histórico como uma Função de Massa de Probabilidade (PMF), a probabilidade condicional da tarefa alcançar seu deadline será calculada, dado que o sistema é identificado como em um dos dois estados possíveis $P(Rk \leq Dk \mid estado\ \ do\ \ sistema)$.

A implementação de uma aplicação como prova de conceito mostrou a viabilidade do método proposto em [32] e também mostrou que o tamanho do histórico que armazena os tempos de resposta dos serviços de uma aplicação não precisam ser grandes devido à dinâmica do sistema. O estudo mostrou que um histórico com três entradas é suficiente para obter resultados similares a um histórico com 100 entradas.

Na área de computação paralela, o trabalho [37] propõe uma técnica de previsão de tempo de execução de uma aplicação que é baseada nos tempos de execução passados de aplicações paralelas similares. A técnica proposta considera aplicações que não possuem restrições temporais.

Técnicas de busca como o algoritmo guloso (*greedy*) e algoritmo genético são usadas para determinar quais características da aplicação produzem a melhor definição de similaridade para fins de previsão. Os autores discutem a dificuldade de encontrar um conjunto apropriado de padrões de aplicações. Se um número pequeno de padrões é definido, muitas aplicações não similares serão agrupadas em um padrão e os resultados da previsão serão insatisfatórios. Por outro lado, se um número grande de padrões é definido, um determinado padrão será formado por poucas aplicações dificultando a geração de previsões corretas.

A técnica apresentada em [37] propõe quatro tipos de previsão, uma que aplica regressão linear nos dados armazenados no histórico, outra que aplica inversão linear, a terceira aplica regressão logarítmica e a quarta faz uma média dos dados contidos no histórico.

8.2.3 Algoritmo de Previsão de Perda de Deadline em *Workflows*

Workflows são abstrações de processos de negócios [28, 30]. Organizações usam estes sistemas de gerenciamento de processos de negócios tanto no nível conceitual quanto no nível de execução, monitorando o progresso da execução do processo e gerando relatórios com estatísticas sobre os processos e os recursos utilizados por eles.

Um *workflow* é composto por atividades e podem ter deadlines. Quando uma atividade perde seu deadline, exceções podem ser lançadas. Tipicamente, a execução destas exceções aumenta o custo dos processos de negócios devido à execução de atividades adicionais, compensação de tarefas concluídas ou a intervenção de gerentes do sistema. Cada atividade tem um custo de lançamento de exceções associado com ela, o qual é definido pelo analista do negócio. Em [28] os autores pesquisam sobre o lançamento antecipado de exceções em *workflows*. Os objetivos do trabalho são minimizar o número de exceções durante a execução de um processo e reduzir o custo associado com exceções quando estas não podem ser evitadas.

Para alcançar estes objetivos, o método proposto em [28] combina dois mecanismos inter-relacionados: ajuste dinâmico de deadline e lançamento de exceções em estágios iniciais. O mecanismo de ajuste dinâmico de deadline reduz o número de exceções durante a execução de um *workflow*, mantendo informações disponíveis de qualquer folga acumulada durante a execução parcial de um processo e usando esta folga para estender os deadlines das atividades restantes. O lançamento antecipado de exceções é baseado na observação de que quando o lançamento de uma exceção é inevitável, existe um benefício em realizar tais ações em estágios iniciais, aqueles em que os deadlines parciais ainda não foram perdidos.

O algoritmo proposto procura prever o lançamento de exceções e decidir se existe benefícios em executá-las antecipadamente. Entretanto, este algoritmo possui restrições relacionadas à arquitetura do *workflow*. O algoritmo aceita apenas workflows com caminhos paralelos não-condicionais (caracterizadas por blocos *AND-split* e *AND-join*). Segundo os autores, a existência de caminhos paralelos condicionais (caracterizados por blocos *OR-splits* e *OR-join*) complica a previsão porque as atividades que serão executadas no futuro não são previamente conhecidas.

Antes de uma atividade iniciar sua execução, o algoritmo verifica se o processo do qual ela faz parte irá perder o deadline. Esta verificação utiliza os tempos de computação

estimados das atividades subseqüentes à atividade em questão. Se o somatório destes tempos de computação excede significativamente o deadline do processo, o algoritmo pesquisa o custo de lançamento de exceções de cada atividade subseqüente à atividade em questão. Se o algoritmo encontra uma atividade com um custo menor que o da atividade em questão significa que existe benefício em esperar a execução desta atividade subseqüente ao invés de executar a exceção imediatamente.

Por outro lado, se o algoritmo não encontra nenhuma atividade sucessora da atividade em questão com um custo de lançamento de exceções menor que o dela, então existe benefício em lançar a exceção antecipadamente, antes da atividade em questão iniciar sua execução.

8.2.4 Algoritmos de Previsão para Sistemas Distribuídos de Tempo Real

Na área de sistemas distribuídos de tempo real, o trabalho descrito em [39] propõe funções de previsão de tempo de resposta de processos periódicos. O cálculo da previsão é realizado *offline* e considera o tempo de execução gasto por cada atividade que compõe o processo em recursos hospedados nos nodos do sistema. O paralelismo do processo e o tempo de rede (tempo gasto no envio de dados para um nodo) também são considerados.

O resultado das funções de previsão é usado para definir quais atividades podem ser atribuídas a outros nodos, de forma que o tempo de execução da atividade diminua e o desempenho geral do processo aumente.

As funções utilizadas para calcular uma estimativa de tempo de resposta são baseadas no período dos processos. Para que estas funções possam ser usadas em sistemas compostos por processos aperiódicos seria necessário uma adaptação de tais fórmulas considerando as restrições temporais típicas de uma tarefa aperiódica.

Em [38], os autores propõem técnicas para prever o tempo de resposta de uma aplicação distribuída de tempo real em um nodo. Os métodos consideram o atraso que a aplicação irá sofrer em função de dois tipos de aplicações: aplicações com a mesma prioridade da aplicação em questão e aplicações com prioridades superiores à da aplicação em questão.

Os métodos propostos diferem dos demais existentes na literatura porque eles utilizam a técnica *Profiling* para determinar o tempo de execução da aplicação em vez de usar a estimativa de pior caso. A técnica *Profiling* utiliza uma descrição dos tempos de execução de

uma aplicação sob diferentes cargas e uma função linear *piecewise*.

Conforme descrito em [38], o tempo de resposta previsto de uma aplicação em um determinado nodo é calculado através da seguinte equação:

$$\lambda pred(A) = C(A) + D_{pred1}(A) + D_{pred2}(A)$$

onde $C(A)$ é o tempo de computação previsto da aplicação A, definido através da técnica *Profiling*, $D_{pred1}(A)$ é o atraso de fila que a aplicação A sofre em função de outras aplicações com a mesma prioridade e $D_{pred2}(A)$ é o atraso de fila que a aplicação A sofre em função de outras aplicações com prioridades mais altas em relação à ela.

O atraso de fila que a aplicação irá sofrer em função de outras aplicações com a mesma prioridade ($D_{pred1}(A)$) e de aplicações com prioridades maiores que a dela ($D_{pred2}(A)$) é definido através de duas técnicas, *Execution Rate* (ER) e *Probabilistic Rate* (PR).

O estudo é realizado sobre a política de prioridades tempo real do sistema operacional *Solaris*, que aloca a CPU para a aplicação de maior prioridade que estiver pronta para executar. Se várias aplicações com a mesma prioridade (mais alta) estiverem prontas para executar, a política de escalonamento *Round-Robin* (RR) é utilizada. Cada nível de prioridade tem uma quantidade de tempo (*quantum*) que é usada pelo RR.

- Técnica *Execution Rate* (ER) para tarefas de mesma prioridade

Na técnica ER, o intervalo de tempo no qual a competição de recursos é considerada é calculado a partir do mínimo múltiplo comum dos períodos das aplicações. O cálculo de $D_{pred1}(A)$, segundo esta técnica, leva em consideração duas informações:

- *RS(A)*: É o requerimento de recursos da aplicação A, calculada da seguinte forma:

  $$RS(A) = S(A) \times (LCM/T(A));$$

  onde S(A) representa quantas vezes, em cada período, a aplicação irá utilizar o *quantum* S definido para aquele nível de prioridade; LCM é o mínimo múltiplo comum dos períodos das aplicações de mesma prioridade que a aplicação A; *T(A)* é o período da aplicação A.

- $RS(A^*)$: É o requerimento de recursos das outras aplicações com a mesma prioridade

da aplicação A, calculada da seguinte forma:

$$RS(A^*) = \sum_{k:a_k \in^* \bigwedge p(a_k)=p(A)} S(a_k) \times (LCM/T(a_k));$$

onde a_k representa cada aplicação com a mesma prioridade da aplicação A; $p(a_k)$ representa a prioridade da aplicação a_k.

Com base nestas duas informações, calcula-se a taxa de execução, conforme a equação abaixo:

$$er(A) = \frac{RS(A)}{RS(A^*) + RS(A)}; \tag{8.1}$$

Usando a taxa de execução ($er(A)$), o atraso de fila que a aplicação irá sofrer em função de outras aplicações com a mesma prioridade ($D_{pred1}(A)$) é calculado da seguinte forma:

$$D_{pred1}(A) = (S(A)/er(A) - S(A)) \times TQ(H, p(A));$$

onde $TQ(H, p(A))$ representa o valor de *quantum* para a prioridade p da aplicação A no nodo H.

- Técnica *Probabilistic Rate* (PR) para tarefas de mesma prioridade

Na técnica PR, o intervalo de tempo no qual a competição de recursos é considerada é definido como o período da aplicação. Inicialmente, a utilização da aplicação em questão ($U(A)$), bem como a utilização das demais aplicações de mesma prioridade ($U(A^*)$) são calculadas. A seguir, a taxa de progresso (pr) é calculada a partir da soma de três probabilidades:

$$\begin{aligned} cp_0(A) &= 1 - (U(A^*) + U(A)) \\ cp_1(A) &= U(A) \\ cp_2(A) &= U(A^*) \times er(A) \\ pr(A) &= cp_0(A) + cp_1(A) + cp_2(A) \end{aligned}$$

onde $cp_0(A)$ representa a probabilidade do recurso não ser completamente usado, $cp_1(A)$ representa a probabilidade de que somente a aplicação A está usando o recurso, $cp_2(A)$

representa a probabilidade que a aplicação A está usando o recurso quando existe competição de outras aplicações de mesma prioridade, a qual é representada por $er(A)$ (calculada usando a técnica ER).

Usando a taxa de progresso ($pr(A)$), o atraso de fila que a aplicação irá sofrer em função de outras aplicações com a mesma prioridade ($D_{pred1}(A)$) é calculado da seguinte forma:

$$D_{pred1}(A) = (S(A)/pr(A) - S(A)) * TQ(H, p(A));$$

- Técnica *Execution Rate* (ER) para tarefas com prioridades mais altas que a aplicação em questão

Na técnica ER aplicada para tarefas com prioridades mais altas que a aplicação em questão, calcula-se o requerimento de recursos para a aplicação, da seguinte forma:

$$RS(A) = C(A) * \frac{LCM}{T(A)}$$

onde $C(A)$ representa o tempo de execução da aplicação A, LCM representa o mínimo múltiplo comum entre os períodos das aplicações e $T(A)$ é o período da aplicação em questão.

Em seguida, calcula-se o requerimento de recursos das demais aplicações, presentes em um determinado nodo, com prioridades mais altas que a aplicação em questão:

$$RS(A^*) = \sum_{k:a_k \in A^* \bigwedge p(a_k) > p(A)} C(a_k) * \frac{LCM}{T(a_k)}$$

A taxa de execução ($er(A)$) (equação 8.1) representa a taxa na qual as requisições da aplicação A para o recurso CPU são reservadas. Esta taxa considera a competição entre a aplicação em questão e as demais que possuem prioridades mais altas. Usando este valor, o atraso de fila que a aplicação sofre em função de outras aplicações com prioridade mais alta ($D_{pred2}(A)$) é calculado como segue:

$$D_{pred2}(A) = \frac{C(A)}{er(A)} - C(A);$$

- Técnica *Probabilistic Rate* (PR) para aplicações com prioridades mais altas que a aplicação em questão

Nesta versão da técnica PR para aplicações com prioridades mais altas que a aplicação em questão, os cálculos da utilização da aplicação em questão ($U(A)$), bem como da utilização das demais aplicações de mesma prioridade ($U(A^*)$) e da taxa de progresso (pr) são realizados da mesma forma que na versão para aplicações de mesma prioridade.

Segundo a técnica PR, o atraso de fila que a aplicação sofre em função de outras aplicações com prioridade mais alta ($D_{pred2}(A)$) é calculado da seguinte forma:

$$D_{pred2}(A) = \frac{C(A)}{pr(A)} - C(A);$$

8.3 Conclusão

A literatura apresenta algoritmos para previsão de desempenho e de tempo de resposta para sistemas computacionais em várias áreas de aplicação, como na área de engenharia de software, sistemas autônomos, sistemas de banco de dados, redes e sistemas multimídia. Como existem métodos de previsão de desempenho em praticamente todas as áreas da computação, não intencionamos esgotar o assunto através da revisão bibliográfica realizada neste capítulo. Nos concentramos em buscar mecanismos de previsão em áreas próximas do objeto de estudo desta tese. Descrevemos alguns mecanismos de previsão para sistemas computacionais encontrados na literatura na área de sistemas autônomos [35, 36], sistemas embutidos [32], aplicações paralelas [37], *workflows* [28] e sistemas distribuídos de tempo real [38, 39].

Nenhum dos algoritmos descritos neste capítulo podem ser diretamente aplicado ao modelo de tarefas proposto nesta tese. Todos eles precisariam de adaptações para incorporar as restrições temporais e os dados usados no modelo proposto. Em [39], por exemplo, as funções que calculam a previsão de tempo de resposta fazem uso do período das tarefas e o modelo de tarefas proposto nesta tese considera tarefas distribuídas aperiódicas.

Em [38], o algoritmo proposto serve para prever o tempo de resposta de uma aplicação distribuída de tempo real em um nodo apenas. O algoritmo não se preocupa em definir o tempo de resposta nos demais nodos onde a tarefa irá executar. No modelo de tarefas proposto nesta tese, esta questão não se resume ao fato de apenas somar as estimativas de tempo de resposta locais. O modelo proposto considera a natureza dinâmica de uma thread distribuída, isto é, suas variações na execução dos possíveis itinerários.

Em [28], a principal limitação do algoritmo proposto para a previsão de perda de deadline se refere ao não-reconhecimento de seqüencias paralelas condicionais de execução, representadas por estruturas do tipo OR. Os mecanismos de previsão propostos nesta tese reconhecem tais estruturas e utilizam informações de ativações passadas da thread distribuída e informações sobre o estado atual do sistema nos cálculos para a definição de um tempo de resposta estimado. O algoritmo descrito em [28] utiliza informações a respeito do tamanho das filas das aplicações que as atividades que compõem um processo irão executar. Especificamente, o algoritmo utiliza o tamanho atual e o tamanho médio da fila de cada aplicação nos cálculos do tempo de computação estimado das atividades. Nesta tese, também utiliza-se o tamanho da fila do servidor de cada nodo que a thread distribuída poderá executar, para fins de previsão. Entretanto, esta informação é associada através de regressão linear com os tempos de resposta de ativações passadas de cada nodo que a thread distribuída visitou.

Em função do exposto acima, é possível afirmar que o modelo de tarefas proposto nesta tese é original como ponto de partida para a previsão de perda de deadline. A originalidade do problema requereu a proposição de algoritmos específicos para ele.

Capítulo 9

Conclusão

Este trabalho se insere na área de sistemas distribuídos de tempo real não-críticos, onde a perda eventual de deadlines é tolerada. O objeto de estudo desta tese foram mecanismos de previsão de perda de deadline. Bons mecanismos de previsão permitem que ações corretivas possam ser executadas a fim de melhorar o desempenho de sistemas tempo real não-críticos. Estas ações podem ser divididas em dois grupos principais, aquelas que buscam evitar que uma tarefa perca seu deadline e aquelas que buscam reduzir o desperdício de recursos do sistema com o processamento de tarefas que provavelmente irão perder seu deadline.

O objetivo geral desta tese foi definir, implementar e avaliar mecanismos de previsão de perda de deadlines em sistemas construídos a partir do conceito de threads distribuídas. Para isso, os seguintes objetivos específicos foram perseguidos:

- Desenvolver uma arquitetura de sistema e um modelo de tarefas para aplicações não críticas implementadas a partir de threads distribuídas;
- Elaborar mecanismos de previsão de perda de deadlines considerando as execuções condicionais das threads distribuídas e a carga computacional do sistema;
- Avaliar, através de simulações, a qualidade das previsões realizadas pelos mecanismos considerando diferentes contextos de execução.

Para atingir os objetivos específicos foi realizada uma revisão da literatura sobre sistemas distribuídos de tempo real e threads distribuídas. Com base nestes conceitos, foi

definida uma arquitetura de sistema que oferece suporte para a implementação e escalonamento de threads distribuídas. Esta arquitetura está presente em cada nodo do sistema e é composta por interceptadores e servidores de aperiódicas que se ocupam em atender as threads distribuídas que chegam no nodo. O método de escalonamento utilizado nesta arquitetura é composto pelo particionamento do deadline fim a fim e escalonamento local.

Nesta tese consideramos a natureza dinâmica de execução da thread distribuída. Esta característica foi usada para a definição de possíveis itinerários de execução, os quais serviram para fins de escalonamento local e para fins de previsão de perda de deadline. Foram, a seguir, definidas as premissas dos mecanismos de previsão, que consideram o momento em que eles são disparados, bem como as métricas usadas para medir a qualidade dos resultados gerados pelos mecanismos propostos.

O primeiro mecanismo de previsão proposto, baseado em *Milestones*, pode ser considerado simples porque utiliza apenas informações conhecidas de execuções passadas da thread distribuída e define estaticamente tempos de resposta estimados (*Milestones*) para cada segmento local da thread distribuída. A implementação deste mecanismo e a análise dos seus resultados, gerados a partir de simulações, conduziram à elaboração dos mecanismos de previsão *Folga Restante* (FR) e *Aperiodic Server Queue* (ASQ).

O mecanismo FR trata, de maneira mais flexível que o mecanismo baseado em *Milestones*, o aspecto dinâmico de execução da thread distribuída. No momento em que o mecanismo é disparado, ele considera todos os possíveis itinerários de execução da thread distribuída a partir do nodo onde é realizada a previsão. A soma ponderada dos tempos médios de execução dos supostos itinerários nos cálculos de previsão produziu melhores resultados que o mecanismo baseado em *Milestones*. Este mecanismo pode ser considerado de complexidade média em termos de implementação porque depende apenas de informações da thread distribuída em questão.

O mecanismo ASQ introduz nos cálculos de previsão informações das demais threads distribuídas ativas no sistema. Estas informações se referem à composição da fila do servidor de aperiódicas de cada nodo que compõem um suposto itinerário que pode ser seguido pela thread distribuída. Antes de partir do nodo em que se encontra, cada thread distribuída faz uma cópia da imagem do estado das filas que aquele nodo possui e carrega para o próximo nodo em que ela irá executar. O cálculo de previsão utiliza informações de ativações passadas da

thread distribuída e o tamanho da fila do servidor de aperiódicas de cada nodo que compõem um suposto itinerário de execução. Estas informações são relacionadas usando regressão linear e definem uma estimativa de tempo de resposta para a thread distribuída. Devido à quantidade de informações necessárias para a implementação deste mecanismo, ele pode ser considerado de complexidade elevada, já que considera informações das demais threads ativas no sistema, além de informações da thread em questão.

Os resultados do mecanismo ASQ, gerados através de simulações, são melhores que os resultados dos mecanismos FR e *Milestones*, mostrando que o uso de informações a respeito da carga dos nodos nos quais uma thread distribuída poderá executar aumenta a capacidade do mecanismo de realizar previsões corretas.

Como em qualquer simulação, os resultados obtidos estão associados as características dos cenários simulados. Permanece como uma questão em aberto a sensibilidade dos resultados obtidos com respeito a alguns fatores, por exemplo, o tamanho da rede de computadores.

O resultado fundamental deste trabalho materializou-se na proposição, implementação e avaliação de três mecanismos de previsão de perda de deadlines para sistemas construídos a partir do conceito de threads distribuídas. De uma forma geral, pode-se citar as seguintes contribuições:

- Elaboração de uma arquitetura de sistema para acomodar threads distribuídas aperiódicas. Esta arquitetura fornece um modelo de tarefas e um método de escalonamento;
- Definição de quatro possíveis itinerários de execução para uma thread distribuída: *Maior Número de Saltos, Menor Número de Saltos, Ponderado* e *Mais Provável*;
- Uso dos fluxos condicionais de execução da thread distribuída para fins de escalonamento e de previsão de perda de deadline;
- Elaboração, implementação e avaliação de três mecanismos de previsão de perda de deadlines: *Milestones*, FR e ASQ.

Não é de conhecimento da autora nenhum outro trabalho na literatura que considere as seqüências condicionais de execução de uma thread distribuída, caracterizadas por estruturas de programação do tipo *Se..Então* (*If..Then*), para fins de previsão.

Para apresentar à comunidade científica de tempo real as idéias e resultados que foram sendo obtidos no decorrer deste trabalho, alguns artigos foram produzidos e submetidos para revisão em simpósios nacionais e internacionais. Foram publicados quatro artigos, sendo dois em simpósios nacionais e dois em congressos internacionais. Estes artigos são listados a seguir:

1. PLENTZ, P. D. M. ; MONTEZ, C. B. ; OLIVEIRA, R. S. . *Prediction of End-to-End Deadline Missing in Distributed Threads Systems.* In: 12th IEEE Conference on Emerging Technologies and Factory Automation, 2007, Patras, Greece. Proceedings of the 12th IEEE Conference on Emerging Technologies and Factory Automation, 2007. p.25-32.

2. PLENTZ, P. D. M. ; MONTEZ, C. B. ; OLIVEIRA, R. S. . *Mecanismos para a Previsão de Perda de Deadline para Threads Distribuídas Tipo Pipeline.* In: Simpósio Brasileiro de Automação Inteligente, 2007, Florianópolis, Santa Catarina. Anais do VIII Simpósio Brasileiro de Automação Inteligente, 2007.

3. PLENTZ, P. D. M. ; OLIVEIRA, R. S. ; MONTEZ, C. B. . *Scheduling of the Distributed Thread Abstraction with Timing Constraints using RTSJ.* In: ETFA'2005 - 10th IEEE International Conference on Emerging Technologies and Factory Automation, 2005, Catania, Italy. ETFA'2005 - 10th IEEE International Conference on Emerging Technologies and Factory Automation, 2005.

4. PLENTZ, P. D. M. ; MONTEZ, C. B. ; OLIVEIRA, R. S. ; FRAGA, J. S. . *Programação Baseada em Threads Distribuída nas Especificações RT-CORBA 2.0 e Distributed RTSJ.* In: Workshop de Tempo Real - WTR, 2003, Natal, RN. SBRC 2003 - 21º Simpósio Brasileiro de Redes de Computadores - Workshop, 2003. p. 47-54.

Com o objetivo de aprimorar os conceitos e os resultados alcançados nesta tese, será confeccionado um artigo para a revista *Computer Communications*. Espera-se que as revisões provenientes desta submissão possam contribuir na melhoria do conteúdo proposto nesta tese.

Este trabalho buscou desenvolver mecanismos de previsão para melhorar o desempenho de sistemas distribuídos de tempo real não-críticos. Os resultados obtidos demonstram a importância dos mecanismos de previsão não somente para a melhoria do tempo de resposta do sistema, mas também como forma de identificar conjuntos de nodos responsáveis por degradar de forma mais acentuada o seu desempenho. Este assunto não foi objeto de análise neste trabalho e mereceria um estudo mais aprofundado. Outros assuntos suscitaram ou se

mostraram de interesse para pesquisas futuras, entre eles podemos citar:

- Investigação de outras formas de acionamento dos mecanismos de previsão propostos, que resultem em previsões mais robustas. Nesta tese os mecanismos são acionados quando a thread distribuída cumpre metade do seu deadline fim a fim, isto é, quando o tempo de resposta parcial da thread distribuída é igual ou maior que metade do deadline. Um modelo mais elaborado de acionamento poderia melhorar a qualidade das previsões;

- Uso de técnicas de inteligência artificial para modelar um mecanismo de previsão, considerando as premissas estabelecidas neste trabalho e a arquitetura de sistema proposta. Modelos de Redes Neurais Artificiais são bastante usados para reconhecimento de padrões e classificação. A elaboração de uma arquitetura de rede neural exige a definição de uma série de questões, como por exemplo, o número de camadas de rede que irão compor a arquitetura, as técnicas de aprendizagem e de treinamento que serão utilizadas. Para um sistema tempo real construído a partir da abstração thread distribuídas, todas estas questões devem considerar as restrições temporais das threads distribuídas.

Referências Bibliográficas

[1] R. S. de Oliveira, *Escalonamento de Tarefas Imprecisas em Ambiente Distribuído*. PhD thesis, Programa de Pós-Graduação em Engenharia Elétrica (PPGEEL), Universidade Federal de Santa Catarina, Brasil, 1997.

[2] J.-M. Farines, J. da Silva Fraga, and R. S. de Oliveira, *Sistemas de Tempo Real*. No. 12, São Paulo: Escola de Computação, Julho 2000.

[3] G. C. Buttazzo, *Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications*. Springer 2nd ed. edition, 2004.

[4] J. W. S. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[5] G. Couloris, J. Dollimore, and T. Kindberg, *Distributed Systems - Concepts and Design*. Addison-Wesley, 2001.

[6] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, pp. 1268–1274, December 1997.

[7] J. J. G. Garcia and M. G. Harbour, "Optimized priority assignment for tasks and messages in distributed hard real-time systems," in *Proc. of the IEEE Workshop on Parallel and Distributed Real-Time Systems*, (Santa Barbara(California)), pp. 124–132, April 1995.

[8] J. Sun, *Fixed-Priority End-to-End Scheduling in Distributed Real-Time Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[9] M. D. Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real-time systems," in *Proceedings of Real-Time Systems Symposium*, vol. 7 of *9*, (San Juan, Puerto Rico), pp. 216–227, December 1994.

[10] D. Marinca, P. Minet, and L. George, "Analysis of deadline assignment methods in distributed real-time systems," *Computer Comunications*, vol. 27, pp. 1412–1423, September 2004.

[11] R. K. Clark, E. D. Jensen, and F. D. Reynolds, "An architectural overview of the alpha real-time distributed kernel," in *Winter USENIX Conference*, pp. 127–146, April 1993.

[12] OMG (Object Management Group), *Real-Time CORBA Specification - Version 1.2*, January 2005.

[13] B. Haumacher, T. Moschny, J. Reuter, and W. F. Tichy, "Transparent distributed threads for java," in *Proceedings of the 5th International Workshop on Java for Parallel and Distributed Computing in conjunction with the International Parallel and Distributed Processing Symposium*, (Nice(France)), April 2003.

[14] OMG (Object Management Group), *Common Object Request Broker Architecture: Core Specification - Version 3.0.3*, March 2004.

[15] P. Li, B. Ravindran, H. Cho, and E. D. Jensen, "Scheduling distributable real-time threads in tempus middleware," in *10th International Conference on Parallel and Distributed Systems*, (New Port Beach(California)), p. 187, July 2004.

[16] Y. Zhang, B. Thrall, S. Torri, C. Gill, and C. Lu, "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*, (San Francisco(California)), pp. 497–506, IEEE Computer Society Washington DC USA, March 2005.

[17] D. C. Schmidt, "The ace orb - tao," (http://www.cs.wustl.edu/ schmidt/TAO.html). Acessado em Agosto de 2007.

[18] Sun Microsystems, *Remote Method Invocation (RMI)*. http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp Acessado em Agosto de 2007.

[19] D. Weyns, E. Truyen, and P. Verbaeten, "Distributed threads in java," in *International Symposium on Distributed and Parallel Computing, ISDPC*, (Iasi(Romania)), pp. 94–104, July 2002.

[20] E. Tilevich and Y. Smaragdakis, "Portable and efficient distributed threads for java," in *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, (New York, NY, USA), pp. 478–492, Springer-Verlag New York, Inc., 2004.

[21] P. D. M. Plentz, R. S. de Oliveira, and C. Montez, "Scheduling of the distributed thread abstraction with timing constraints using rtsj," in *10th IEEE International Conference on Emerging Technologies and Factory Automation*, (Catania(Italy)), pp. 23–30, September 2005.

[22] B. Ravindran, E. Curley, J. S. Anderson, and E. D. Jensen, "On best-effort real-time assurances for recovering from distributed thread failures in distributed real-time systems," in *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, (Santorini Island (Greece)), pp. 344–353, May 2007.

[23] J. S. Anderson and E. D. Jensen, "Distributed realtime specification for java: A status report (digest)," in *Proceedings of the 4th international workshop on Java technologies for real-time and embedded systems*, pp. 3–9, ACM Press, 2006.

[24] JCP (Java Community Process), *Real-Time Specification For Java - Version 1.0.2*, November 2001. Final Approval Ballot.

[25] JCP (Java Community Process), *Distributed Real-Time Specification For Java*. http://jcp.org/en/jsr/detail?id=50 Acessado em Agosto de 2007.

[26] P. D. M. Plentz, C. Montez, and R. S. de Oliveira, "Programação baseada em threads distribuídas nas especificações rt-corba 2.0 e distributed rtsj," in *Workshop de Tempo Real - WTR*, (Natal (RN)), pp. 47–54, Maio 2003.

[27] K. yiu Lam, V. C. Lee, S. lun Hung, and B. C. Kao, "Priority assignment in distributed real-time databases using optimistic concurrency control," in *IEE Proceedings - Computers and Digital Techniques*, vol. 144 of *5*, pp. 324–330, 1997.

[28] E. Panagos and M. Rabinovich, "Reducing escalation-related costs in wfmss," in *NATO Advanced Study Institute on Workflow Management System's and Interoperability*, (Turkey), Springer, Istambul, August 1997.

[29] J. Eder, W. Gruber, and E. Panagos, "Temporal modeling of workflows with conditional execution paths," in *Proceedings of 11th International Conference Database and Expert Systems Applications - DEXA 2000*, (London (UK)), September 2000.

[30] J. H. Son, J. H. Kim, and M. H. Kim, "Hard/soft deadline assignment for high workflow throughput," in *Proceedings of the 1999 International Symposium on Database Applications in Non-Traditional Environments*, p. 359, IEEE Computer Society, 1999.

[31] P. D. M. Plentz, C. Montez, and R. S. de Oliveira, "Prediction of end-to-end deadline missing in distributed threads systems," in *Proceedings of the 12th IEEE International Conference on Emerging Technologies and Factory Automation*, (Patras(Greece)), pp. 25–32, September 2007.

[32] C. Y. Tatibana, C. Montez, and R. S. de Oliveira, "Soft real-time task response time prediction in dynamic embedded systems," in *Proceedings of the 5th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, (Santorini Island, Greece), pp. 1–10, Lecture Notes in Computer Science, May 2007.

[33] L. de Oliveira Rech, *Heurísticas para determinação do itinerário de agentes móveis sob restrições temporais*. PhD thesis, Programa de Pós-Graduação em Engenharia Elétrica (PPGEEL), Universidade Federal de Santa Catarina, Brasil, 2007.

[34] P. A. Barbetta, M. M. Reis, and A. C. Bornia, *Estatística para Cursos de Engenharia e Informática*. Ed. Atlas, 2004.

[35] S. Kirtane and J. Martin, "Application performance prediction in autonomic systems," in *Proceedings of the 44th ACM annual Southeast regional conference*, (Melbourne, Florida), pp. 566–572, ACM, March 2006.

[36] E. Thereska, M. Abd-El-Malek, J. J. Wylie, D. Narayanan, and G. R. Ganger, "Informed data distribution selection in a self-predicting storage system," in *IEEE International Conference on Autonomic Computing - ICAC '06*, (Dublin, Ireland), June 2006.

[37] W. Smith, I. Foster, and V. Taylor, "Predicting application run times with historical information," in *Journal of Parallel and Distributed Computing*, vol. 64 of *9*, pp. 1007–1016, Academic Press, Inc., September 2004.

[38] E.-N. Huh and L. R. Welch, "Adaptive resource management for dynamic distributed real-time applications," in *The Journal of Supercomputing*, vol. 38 of *2*, (Hingham, MA, USA), pp. 127–142, Kluwer Academic, November 2006.

[39] L. R. Welch, A. D. Stoyenko, and T. Marlowe, "Response time prediction for distributed processes specified in cart-spec," in *Control Engineering Practice*, vol. 3 of *5*, pp. 651–664, Elsevier, March 1995.

[40] J. Jonsson and K. G. Shin, "Deadline assignment in distributed hard real-time systems with relaxed locality constraints," in *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97)*, (Washington, DC, USA), p. 432, IEEE Computer Society, 1997.

[41] J. Jonsson, "A robust adaptive metric for deadline assignment in heterogeneous distributed real-time systems," in *Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing*, (Washington, DC, USA), pp. 678–687, IEEE Computer Society, 1999.

Apêndice A

Métodos de Particionamento de Deadlines

O particionamento de deadlines é um tema pesquisado em diversas áreas da computação tempo real. Nas seções a seguir são descritos trabalhos que propõem formas de particionar o deadline de tarefas distribuídas, considerando as características de cada área de aplicação alvo [27, 9, 40, 41].

A.1 Particionamento de Deadlines em Sistemas de Banco de Dados Distribuídos

Usualmente, sistemas de banco de dados distribuídos de tempo real são compostos por transações que possuem restrições temporais. A base de dados é dividida em diferentes nodos, e a competição por recursos, que acontece entre transações, gera forte impacto no desempenho geral do sistema.

Com o objetivo de reduzir os efeitos causados pela competição de recursos, três algoritmos para particionamento do deadline das transações são propostos em [27]. O primeiro deles é o *Static Equal Slack* (SEQS), uma versão estática do algoritmo EQS. No SEQS os deadlines de todas as subtransações são atribuídos apenas uma vez, quando a transação chega no sistema.

O segundo método proposto é chamado de *Number of Data Items* (NL), no qual a

prioridade de uma subtransação é atribuída de acordo com o número de dados acessados pela transação a qual ela pertence. Através da atribuição da prioridade mais alta para a transação que acessa o maior número de dados, a transação pode completar sua execução mais rapidamente.

O terceiro método proposto é chamado *Mixed Method* (MM). Ele inclui as restrições temporais da transação bem como o número de dados acessados por ela, unindo as duas heurísticas citadas acima. Neste método, a inserção da variável *lock_fator(T)* representa o quociente do número de dados acessados pela transação pelo número total de dados requisitados por ela. Com isso, quanto maior o número de dados acessados pela transação, menor é o valor desta variável e, conseqüentemente, menor é o deadline da transação, elevando sua prioridade com relação as demais.

A.2 Particionamento de Deadlines em Sistemas Tempo Real *Hard*

O particionamento de deadlines é parte fundamental em sistemas distribuídos críticos, onde a perda de um deadline pode causar danos irreparáveis ao sistema e ao ambiente que ele controla ou está inserido.

No trabalho descrito em [9] o particionamento de deadlines faz parte do algoritmo de escalonamento fim a fim, o qual envolve a união e coordenação de tarefas e mensagens em uma arquitetura distribuída de memória replicada.

Propõe-se uma combinação de escalonamento *offline* e *online*, sendo que na fase *offline* as restrições de precedência e de comunicação são convertidas em pseudo-deadlines e na fase *online*, o escalonamento opera em paralelo nos nodos envolvidos na computação distribuída.

A fase de processamento *offline* é responsável pela decomposição de processos em tarefas, na determinação do grafo de precedência das tarefas e na atribuição de uma janela de execução para cada tarefa (*slices*). Os escalonadores locais são livres para processar em paralelo tarefas e mensagens dentro de fatias de tempo que tenham sido atribuídas para elas. O conjunto de fatias de tempo atribuídas para todas as tarefas de um processo, juntamente com as fatias de tempo atribuídas para as mensagens trocadas pelo processo é chamada de

template de comunicação. Estas *templates* são determinadas *offline* e armazenadas nos nodos onde o processo é carregado. Elas são o resultado final da fase de pré-processamento, que ocorre *offline*, e contém toda a informação necessária para a execução do escalonamento, garantindo a execução síncrona do conjunto de processos comunicantes. Cada escalonador local recebe uma *template* para cada grupo de comunicação, contendo as fatias de tempo de todas as tarefas do grupo e de todas as mensagens enviadas pelos processos locais.

As fatias de tempo são determinadas através da utilização do conceito de *Caminho Crítico*. Em [9], as métricas utilizadas para definição de um caminho crítico são *Pure Laxity* e *Normalized Laxity*. A primeira é baseada no número de subtarefas que compõem o caminho e a segunda é baseada nos tempos de computação das subtarefas que compõem o caminho. O objetivo destas métricas é maximizar a folga mínima, isto é, dentro do conjunto de caminhos do grafo de tarefas, aquele que possuir a maior folga é escolhido.

Para atribuir folga para as demais tarefas que não pertencem ao caminho crítico, o algoritmo remove iterativamente as tarefas que compõem o caminho crítico criando um novo subgrafo. Então, uma nova busca por um novo caminho crítico acontece e uma nova atribuição de fatias de tempo é realizada. Após N iterações (onde N é o número de tarefas no grafo) todas as tarefas têm suas fatias de tempo definidas.

Os resultados dos experimentos realizados naquele trabalho para avaliar o método proposto mostraram que a métrica *Pure Laxity* apresenta melhores resultados que a métrica *Normalized Laxity* quando o tamanho das tarefas permanece abaixo da folga definida. Estas métricas apresentam resultados insatisfatórios quando o tamanho das tarefas é igual ou maior que a folga definida pelas métricas, mostrando que, neste caso, o método *Slicing* apresenta pouca flexibilidade.

No intuito de transpor as limitações apresentadas pelo método *Slicing*, um algoritmo chamado *Adaptive Slicing Technique* (AST) é proposto em [40]. Este algoritmo define novas métricas, as quais são baseadas nos seguintes conceitos:

- *Execution Time Threshold* (C_{thres}): representa a média dos tempos de computação de todas as subtarefas de um grafo;
- *Virtual Execution Time* (C'): utilizado para atribuir um tempo de execução maior que o tempo de computação real da tarefa.

Com base nos conceitos definidos acima, a métrica *Threshold Laxity Ratio* (THRES) utiliza o *Virtual Execution Time*, em vez de utilizar o tempo de computação real da subtarefa, e define a folga das subtarefas da seguinte maneira:

$$\begin{aligned} C'i &= Ci \quad se \quad Ci < C_{thres} \\ C'i &= Ci(1+\Delta) \quad se \quad Ci \geq C_{thres} \end{aligned}$$

Onde Δ define a quantidade de folga pela qual o tempo de execução real deve ser aumentado para aquelas subtarefas cujos tempos de execução excedem C_{thres}. Nesta métrica o valor de Δ é fixo, limitando seu desempenho.

Para refinar o algoritmo em [40], o valor de Δ pode ser adaptado considerando o *Paralelismo Médio do Grafo de Tarefas*, o qual pode ser definido como a carga de trabalho total do grafo de tarefas dividida pelo tamanho (tempo de computação) do maior caminho no grafo. A métrica que considera este paralelismo é chamada *Adaptive Laxity Ratio* (ADAPT), que define *Virtual Execution Time* como sendo:

$$\begin{aligned} C'i &= Ci \quad se \quad Ci < C_{thres} \\ C'i &= Ci(1+\xi/N_{proc}) \quad se \quad Ci \geq C_{thres} \end{aligned}$$

Onde ξ representa o paralelismo médio do grafo de tarefas e N_{proc} é o número de processadores no sistema. O objetivo desta métrica é utilizar o paralelismo médio do grafo de tarefas pelo número de processadores para compensar o baixo desempenho da métrica *Pure Slicing*, proposta em [9].

Para especificar ainda mais o paralelismo do grafo de tarefas, as métricas *Globally-Adaptive Laxity* (ADAPT-G) e *Locally-Adaptive Laxity* (ADAPT-L) são propostas em [41]. A primeira define o *Virtual Execution Time* como sendo:

$$\begin{aligned} C'i &= Ci \quad se \quad Ci < C_{thres} \\ C'i &= Ci(1+KG\xi/N_{proc}) \quad se \quad Ci \geq C_{thres} \end{aligned}$$

onde KG representa o *Fator Adaptativo Global*.

A segunda métrica, ADAPT-L, define o *Virtual Execution Time* da seguinte maneira:

$$\begin{aligned} C'i &= Ci \qquad\qquad se \quad Ci < C_{thres} \\ C'i &= Ci(1 + KL\psi/N_{proc}) \quad se \quad Ci \geq C_{thres} \end{aligned}$$

onde KL representa o *Fator Adaptativo Local* e ψ representa o conjunto paralelo de Ti (conjunto de tarefas que são candidatas potenciais para executar em paralelo com Ti). Esta métrica adiciona complexidade no algoritmo de particionamento de deadlines fim a fim na medida em que um conjunto paralelo deve ser definido para cada tarefa.

Apêndice B

Threads Distribuídas do tipo Pipeline

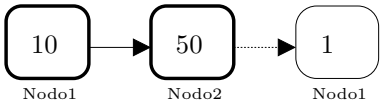

Figura B.1: *Thread Distribuída P1.*

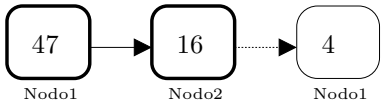

Figura B.2: *Thread Distribuída P2.*

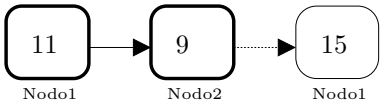

Figura B.3: *Thread Distribuída P3.*

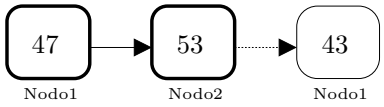

Figura B.4: *Thread Distribuída P4.*

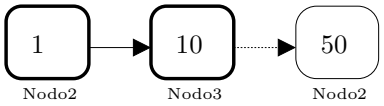

Figura B.5: *Thread Distribuída P5.*

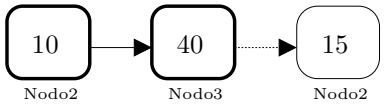

Figura B.6: *Thread Distribuída P6.*

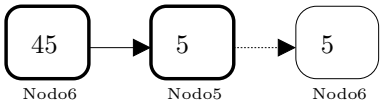

Figura B.7: *Thread Distribuída P7.*

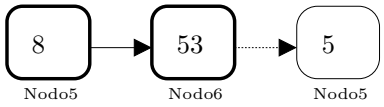

Figura B.8: *Thread Distribuída P8.*

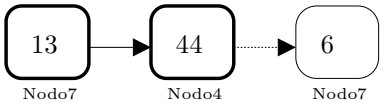

Figura B.9: *Thread Distribuída P9.*

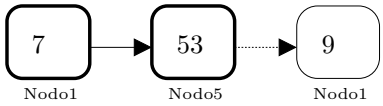

Figura B.10: *Thread Distribuída P10.*

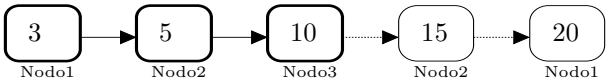

Figura B.11: *Thread Distribuída P11.*

Figura B.12: *Thread Distribuída P12.*

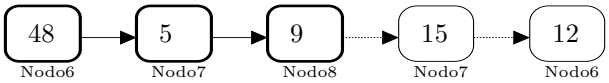

Figura B.13: *Thread Distribuída P13.*

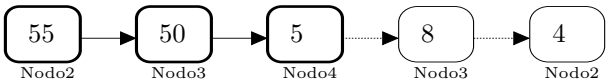

Figura B.14: *Thread Distribuída P14.*

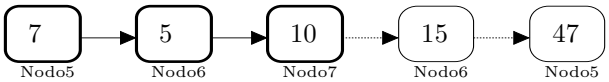

Figura B.15: *Thread Distribuída P15.*

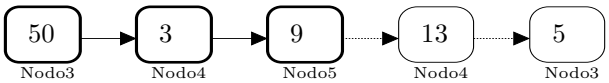

Figura B.16: *Thread Distribuída P16.*

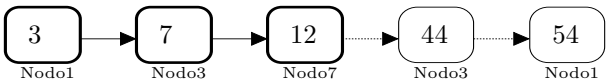

Figura B.17: *Thread Distribuída P17.*

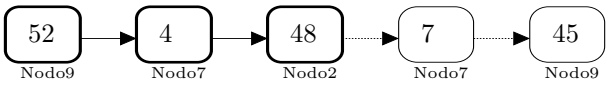

Figura B.18: *Thread Distribuída P18.*

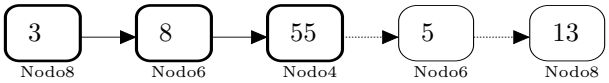

Figura B.19: *Thread Distribuída P19.*

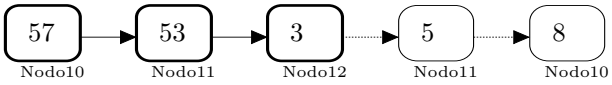

Figura B.20: *Thread Distribuída P20.*

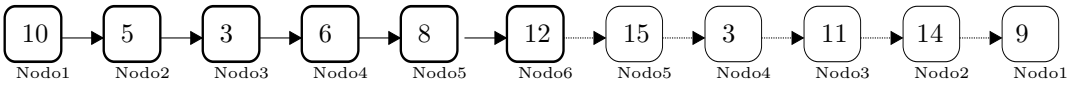

Figura B.21: *Thread Distribuída P21.*

Figura B.22: *Thread Distribuída P22.*

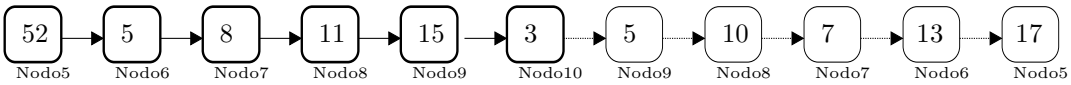

Figura B.23: *Thread Distribuída P23.*

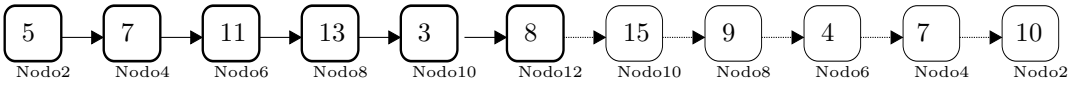

Figura B.24: *Thread Distribuída P24.*

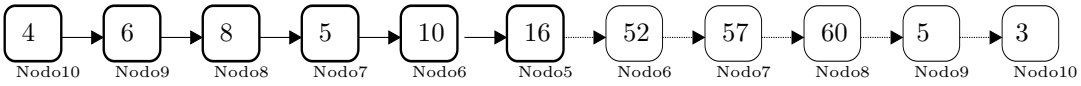

Figura B.25: *Thread Distribuída P25.*

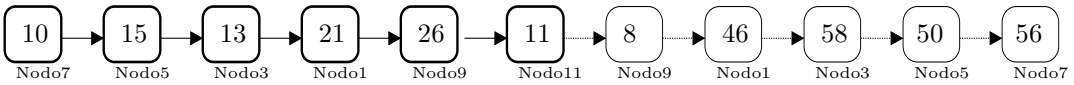

Figura B.26: *Thread Distribuída P26.*

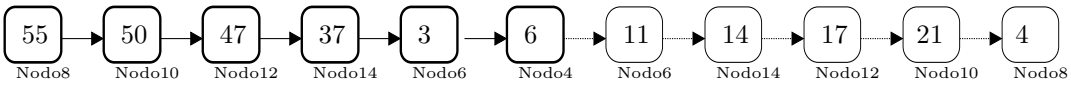

Figura B.27: *Thread Distribuída P27.*

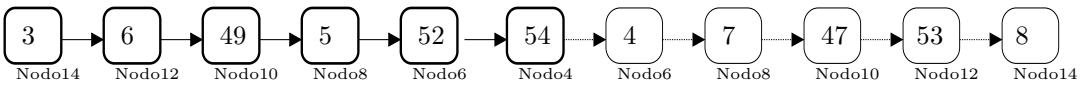

Figura B.28: *Thread Distribuída P28.*

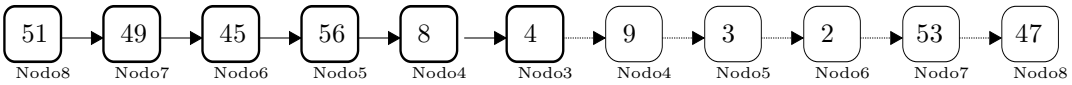

Figura B.29: *Thread Distribuída P29.*

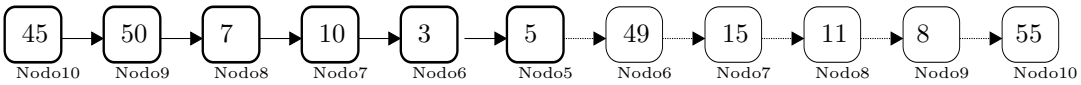

Figura B.30: *Thread Distribuída P30.*

Apêndice C

Threads Distribuídas do tipo Árvore Balanceada

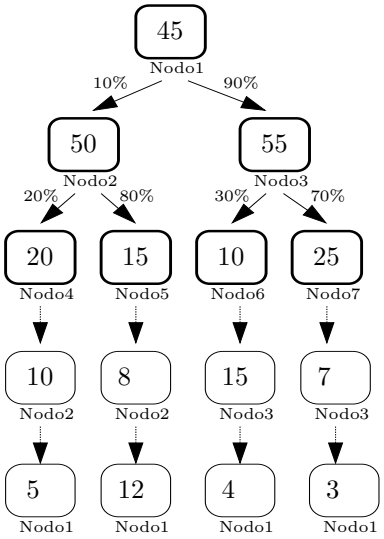

Figura C.1: *Thread Distribuída B1.*

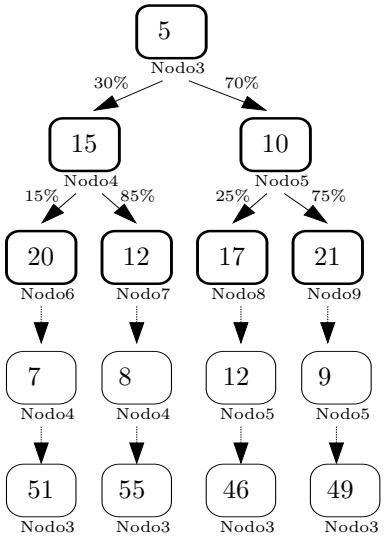

Figura C.2: *Thread Distribuída B2.*

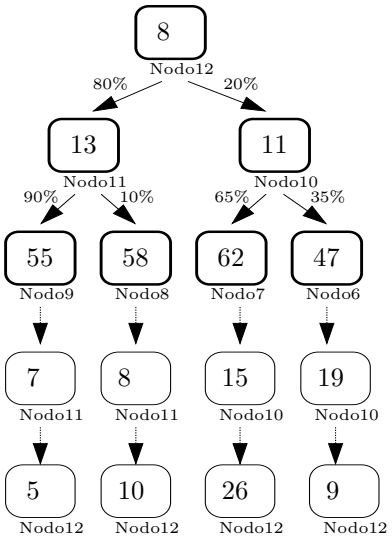

Figura C.3: *Thread Distribuída B3.*

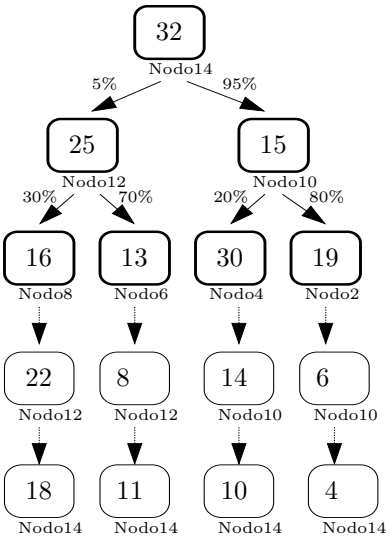

Figura C.4: *Thread Distribuída B4.*

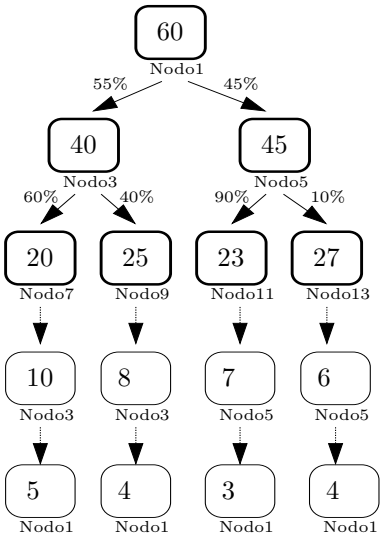

Figura C.5: *Thread Distribuída B5.*

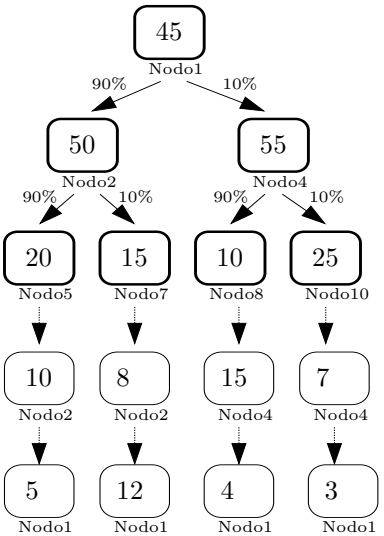

Figura C.6: *Thread Distribuída B6.*

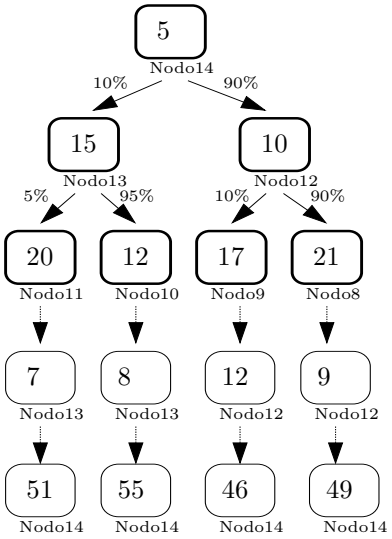

Figura C.7: *Thread Distribuída B7.*

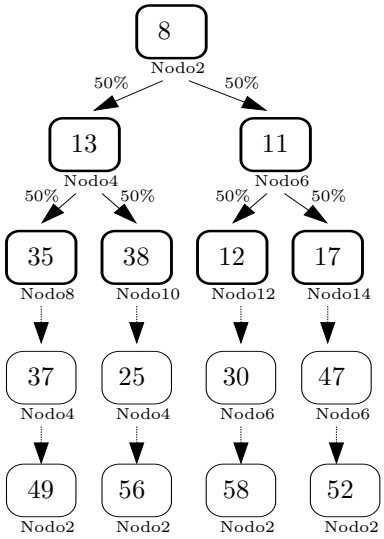

Figura C.8: *Thread Distribuída B8.*

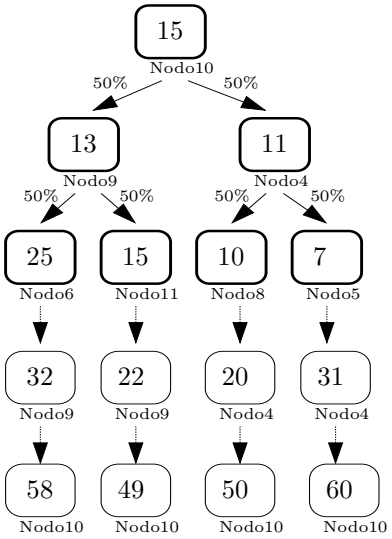

Figura C.9: *Thread Distribuída B9.*

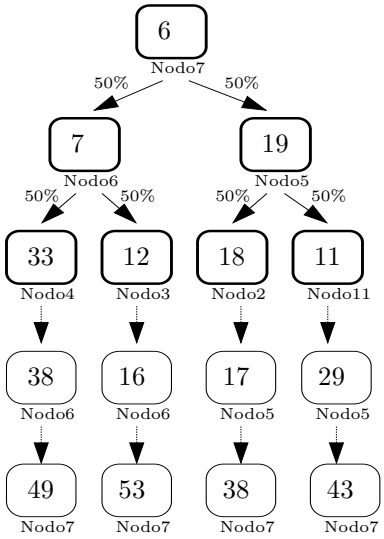

Figura C.10: *Thread Distribuída B10.*

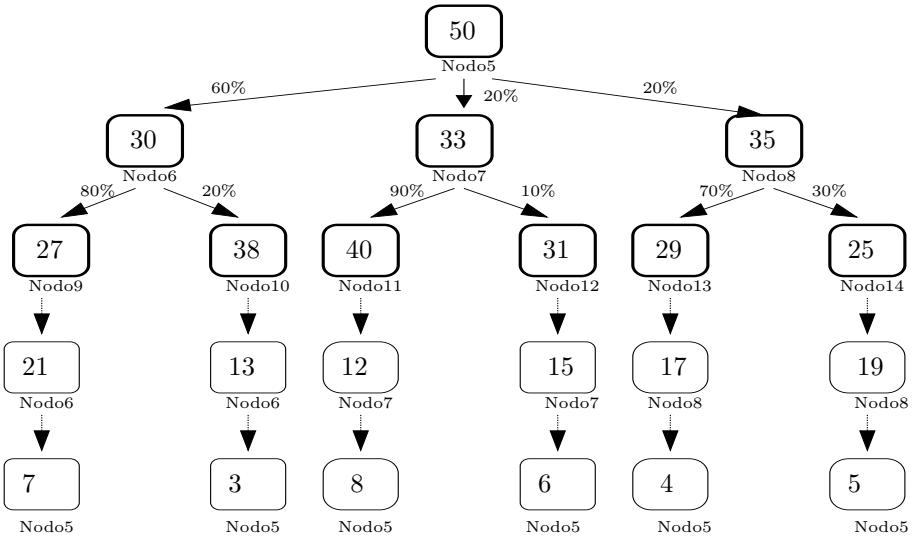

Figura C.11: *Thread Distribuída B11.*

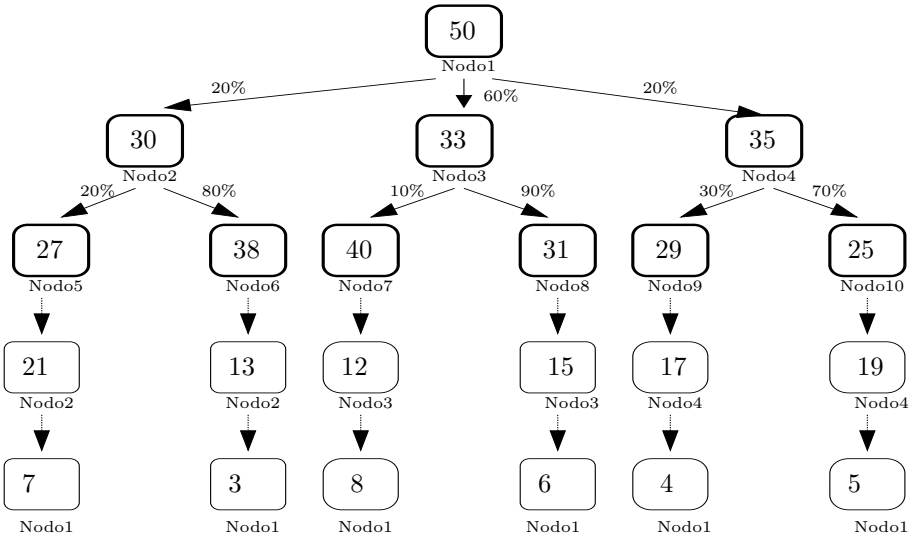

Figura C.12: *Thread Distribuída B12.*

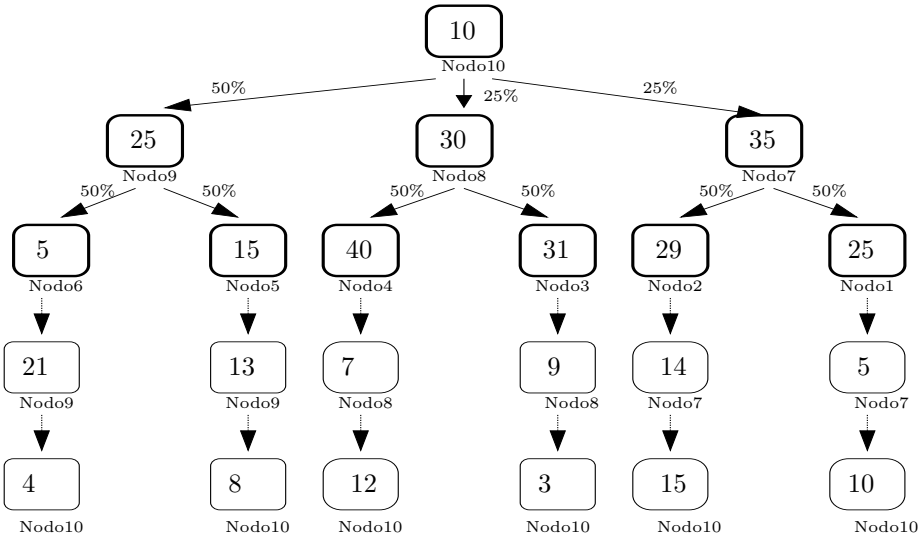

Figura C.13: *Thread Distribuída B13.*

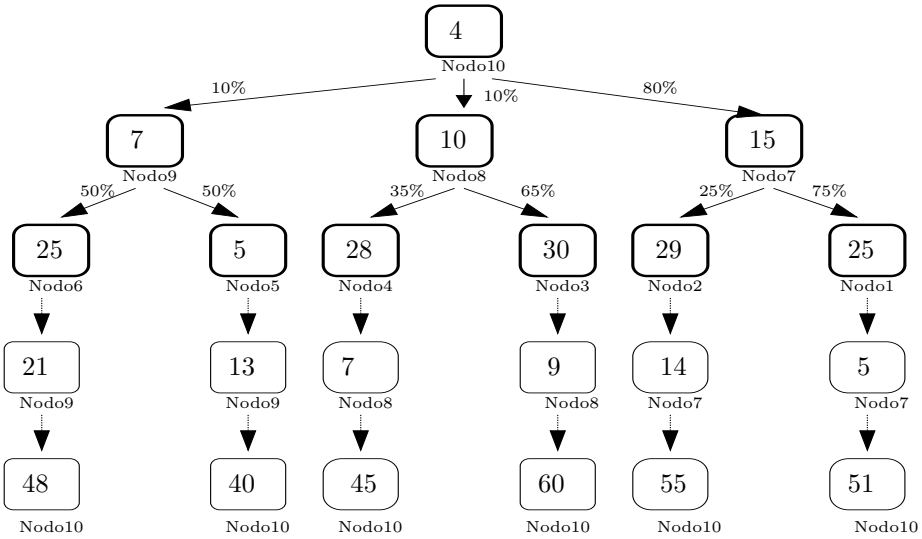

Figura C.14: *Thread Distribuída B14.*

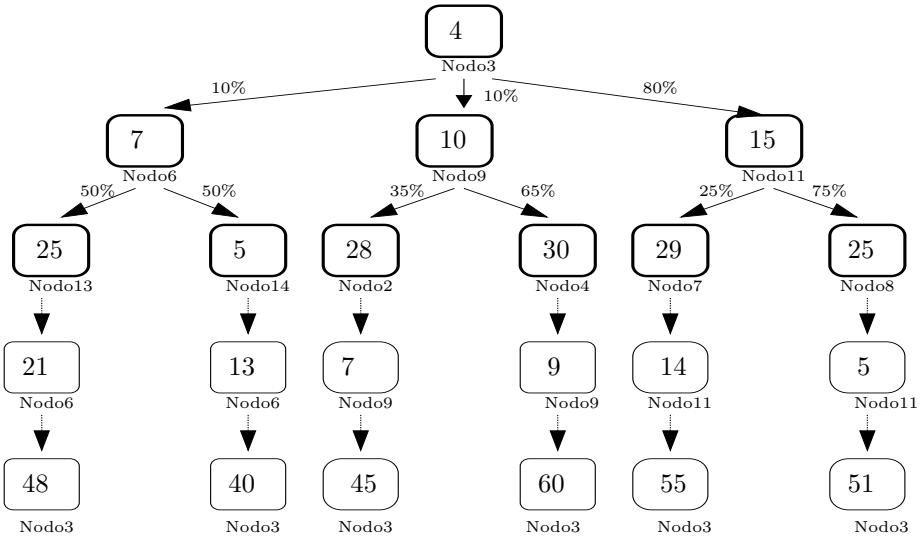

Figura C.15: *Thread Distribuída B15.*

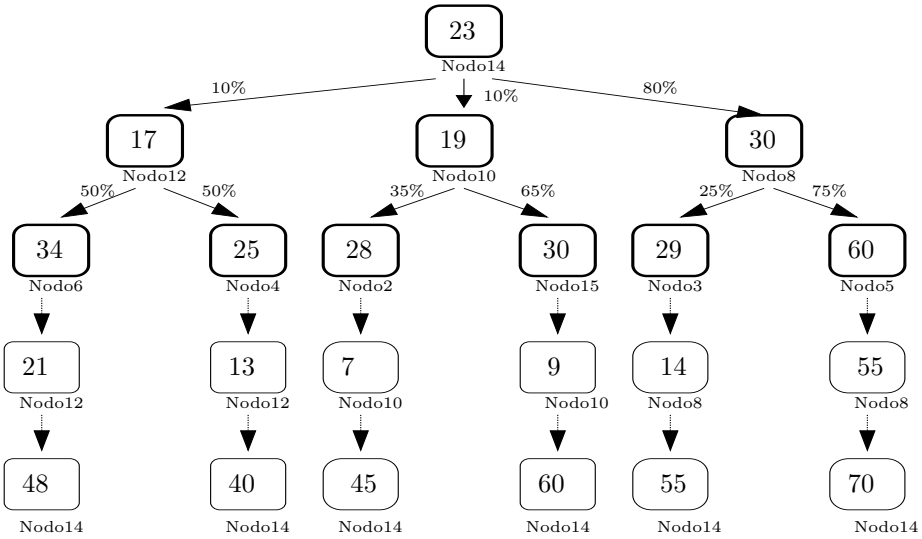

Figura C.16: *Thread Distribuída B16.*

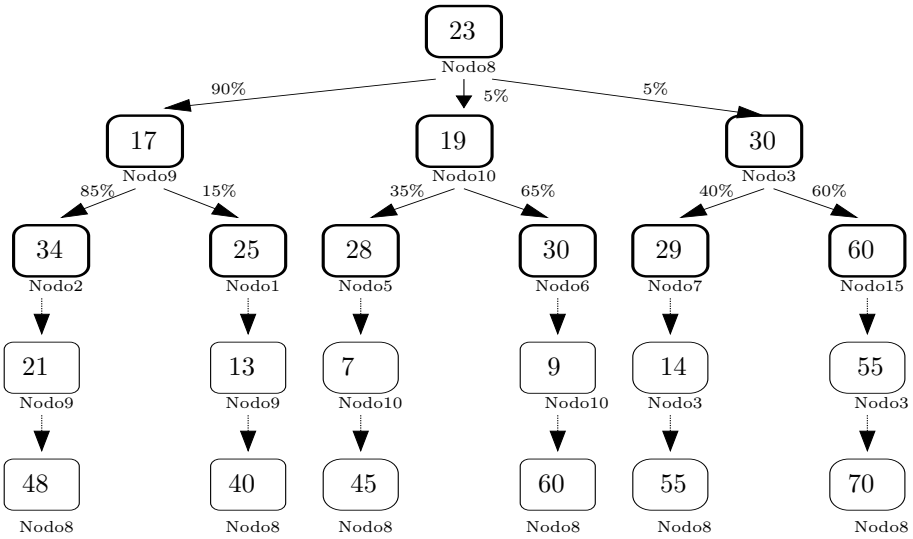

Figura C.17: *Thread Distribuída B17.*

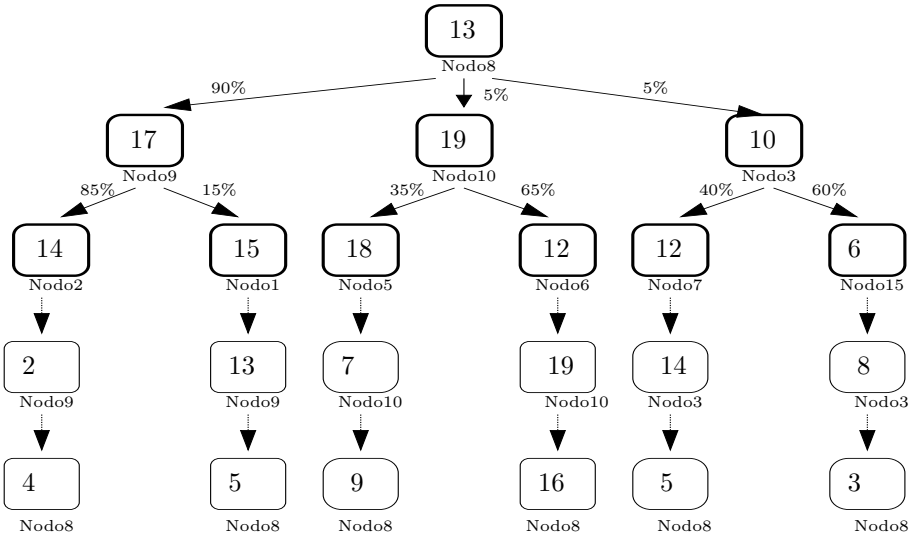

Figura C.18: *Thread Distribuída B18.*

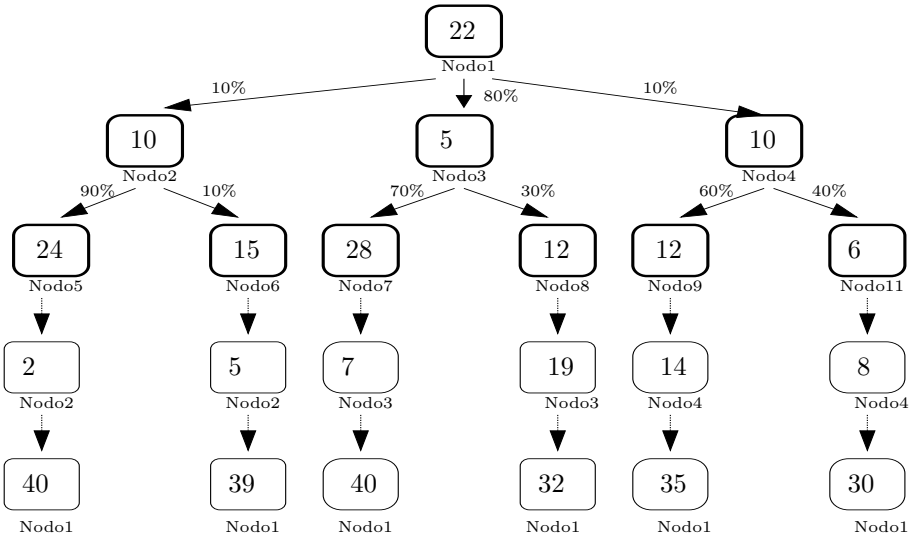

Figura C.19: *Thread Distribuída B19.*

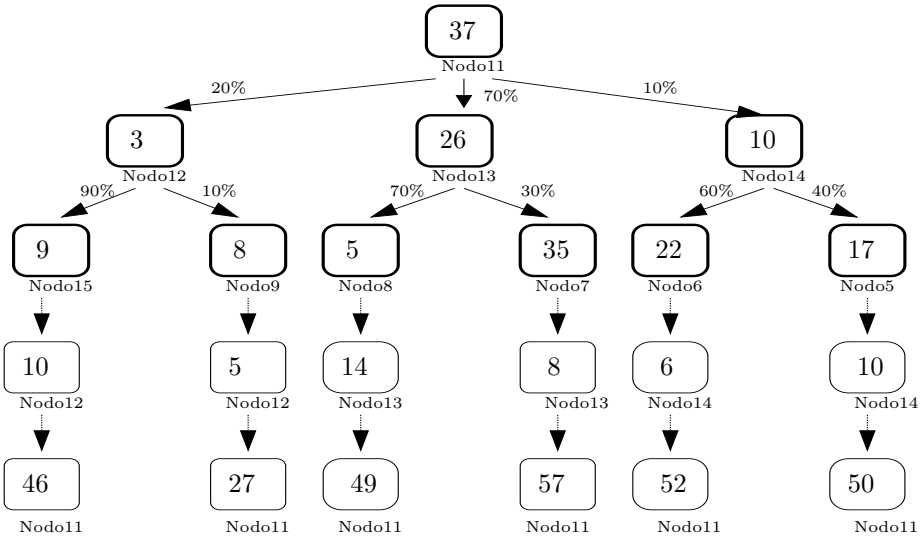

Figura C.20: *Thread Distribuída B20.*

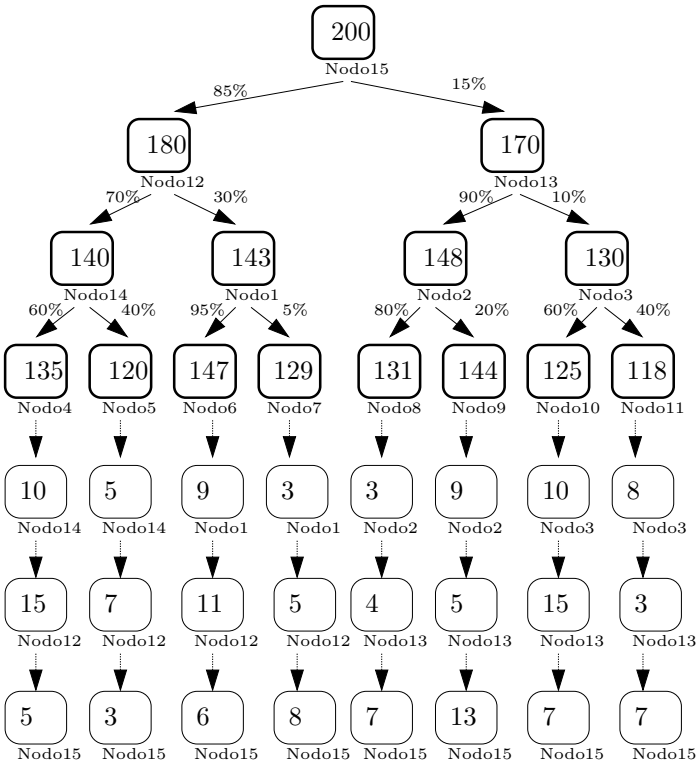

Figura C.21: *Thread Distribuída B21.*

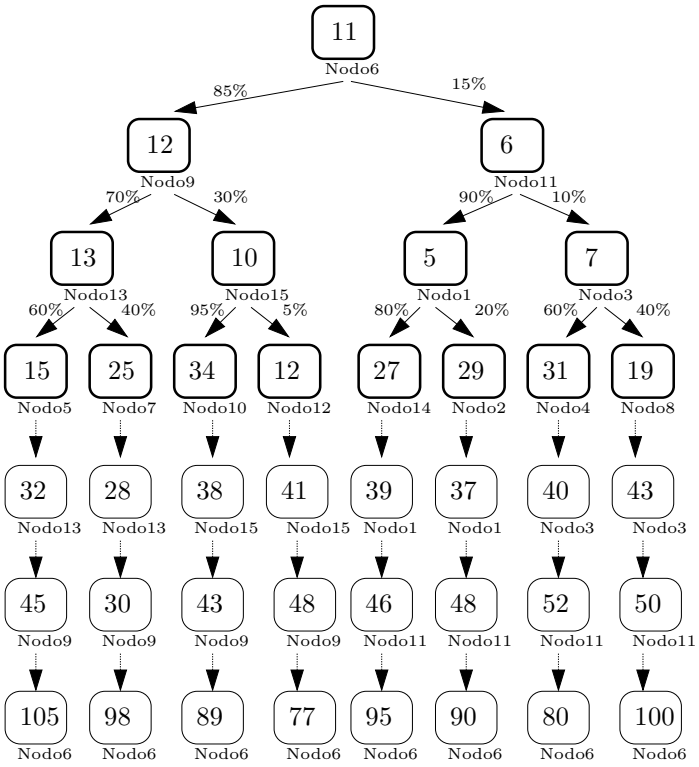

Figura C.22: *Thread Distribuída B22.*

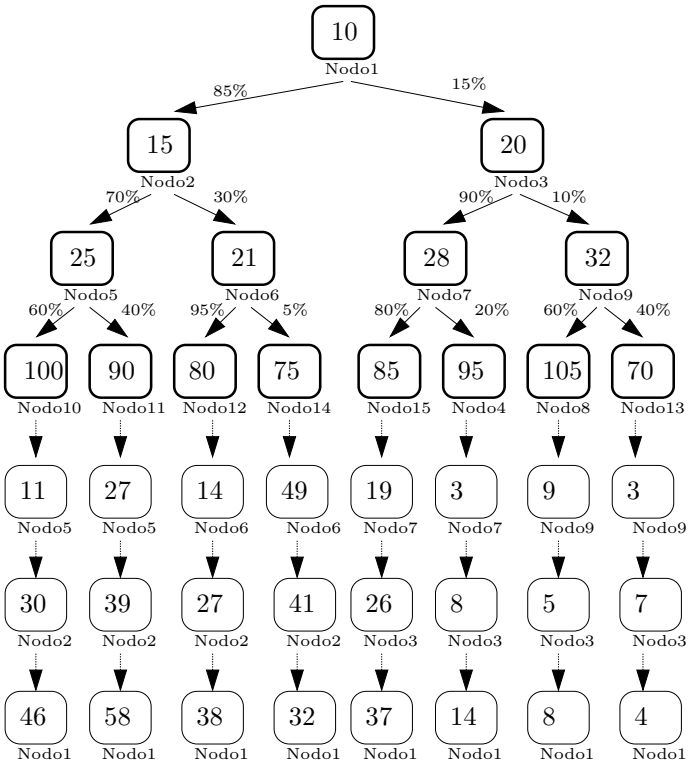

Figura C.23: *Thread Distribuída B23.*

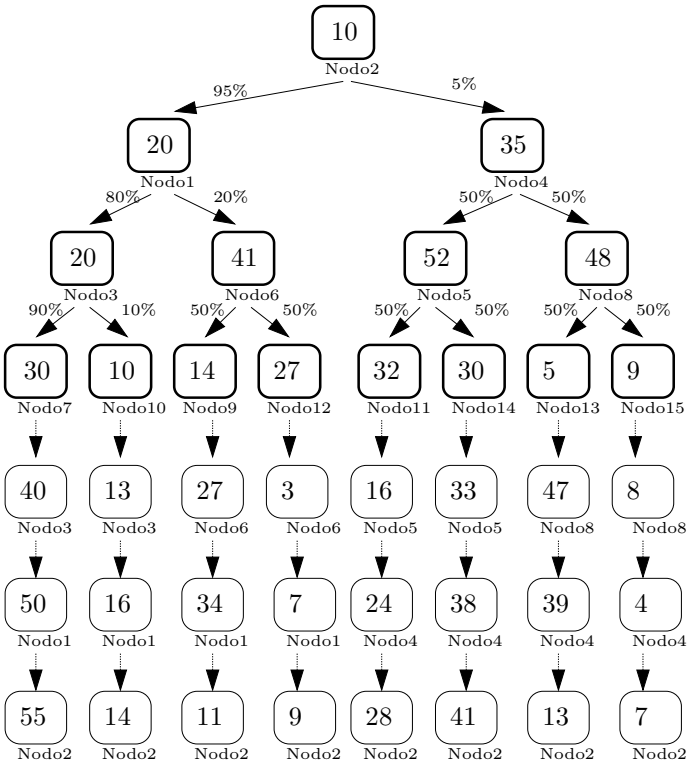

Figura C.24: *Thread Distribuída B24.*

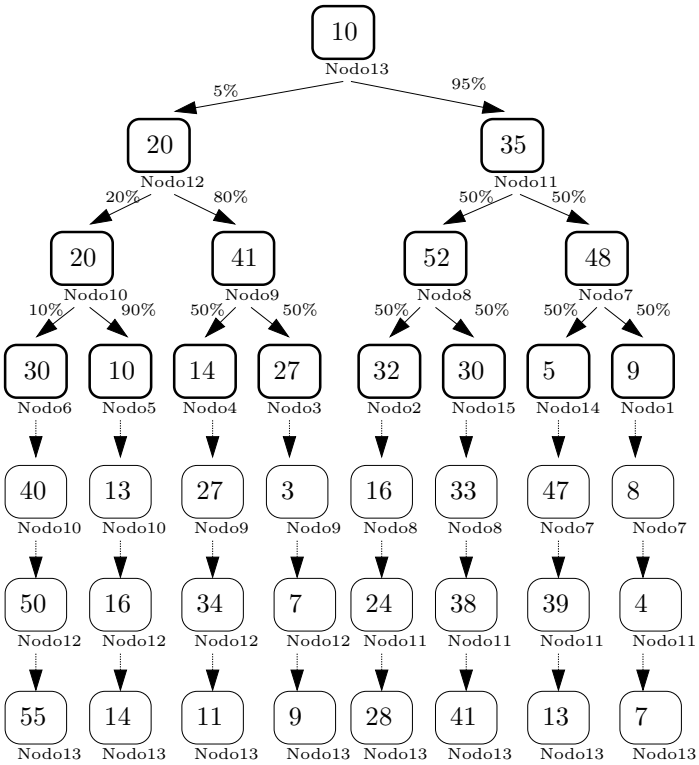

Figura C.25: *Thread Distribuída B25.*

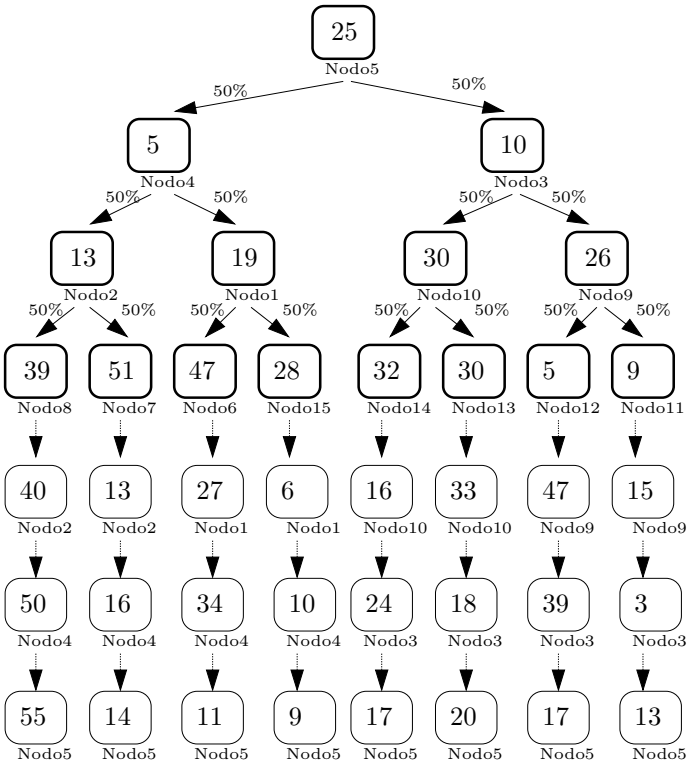

Figura C.26: *Thread Distribuída B26.*

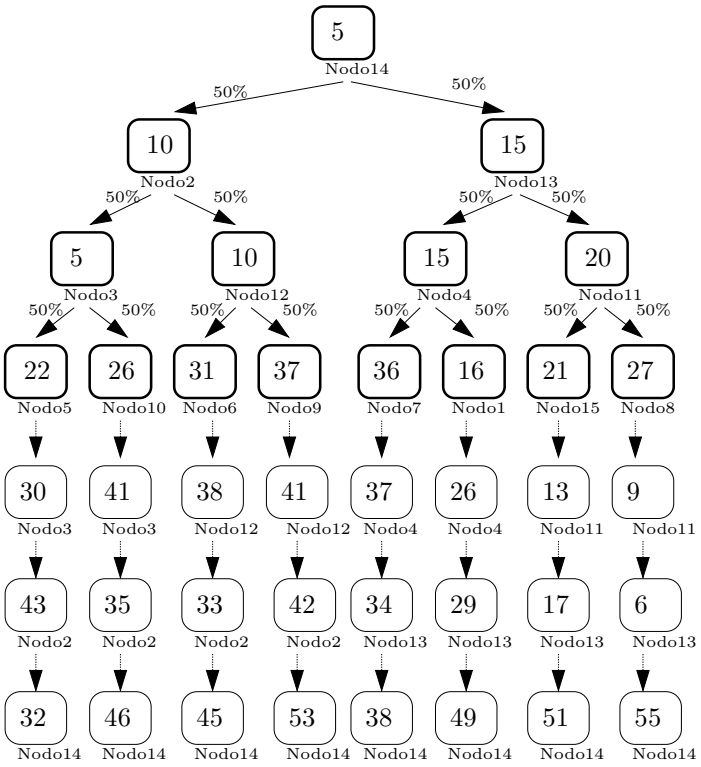

Figura C.27: *Thread Distribuída B27.*

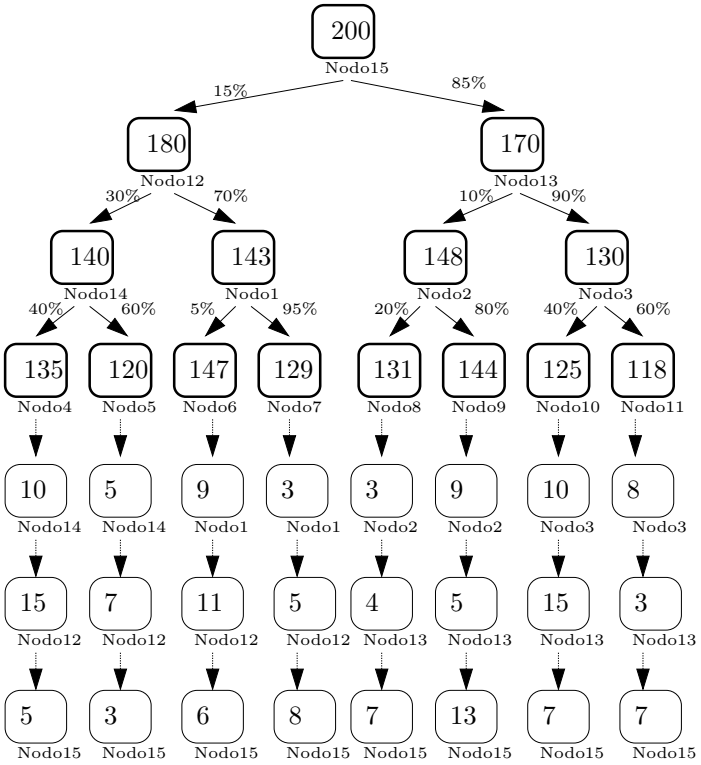

Figura C.28: *Thread Distribuída B28.*

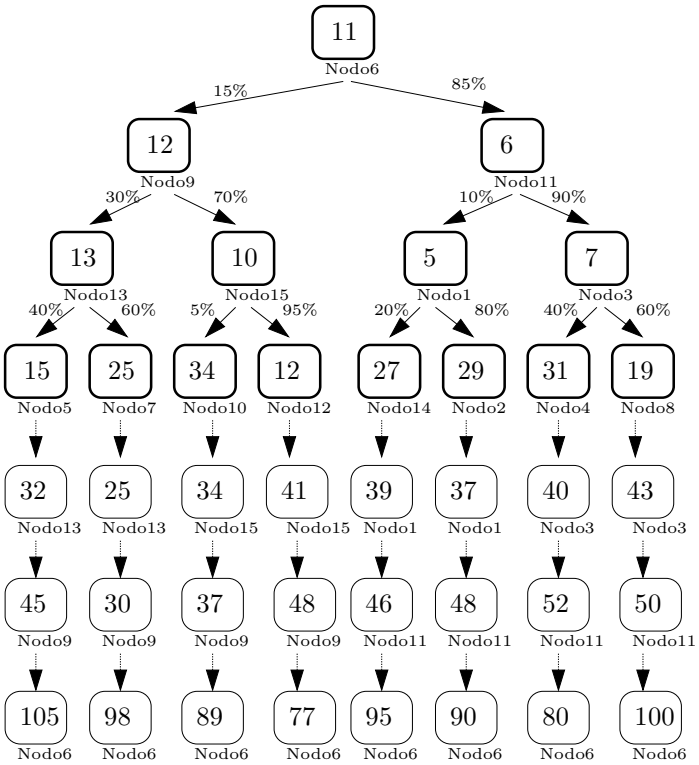

Figura C.29: *Thread Distribuída B29.*

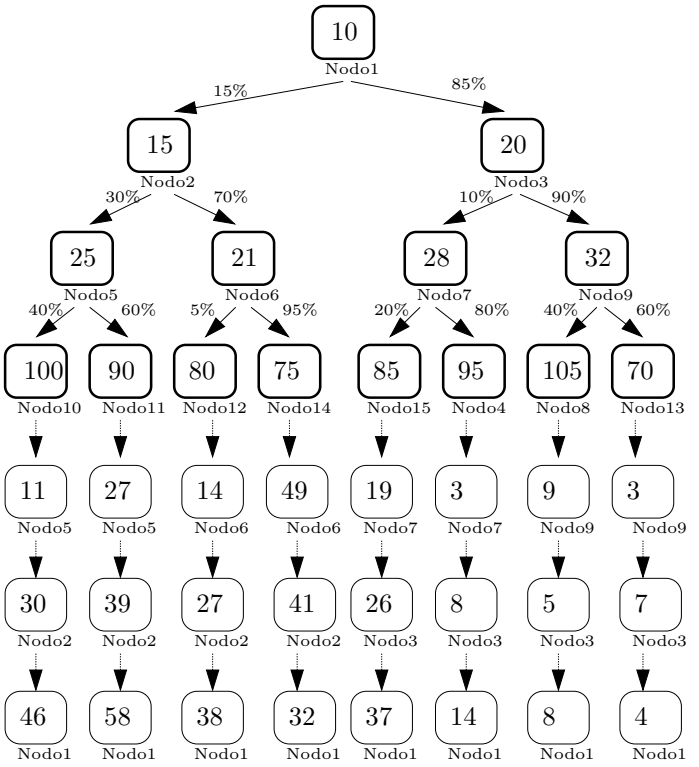

Figura C.30: *Thread Distribuída B30.*

Apêndice D

Threads Distribuídas do tipo Árvore Não Balanceada

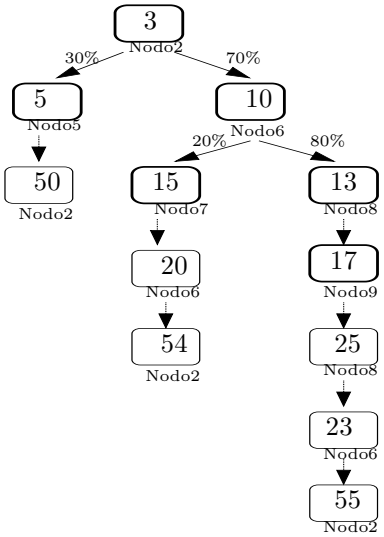

Figura D.1: *Thread Distribuída NB1.*

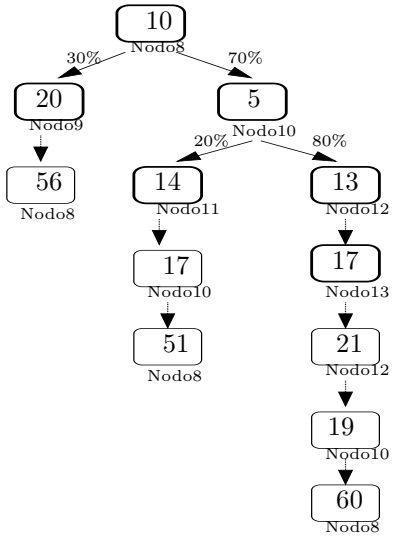

Figura D.2: *Thread Distribuída NB2.*

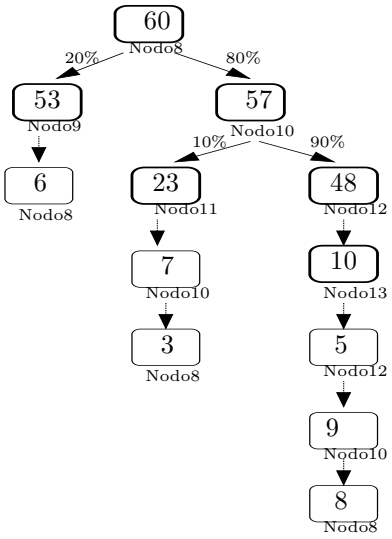

Figura D.3: *Thread Distribuída NB3.*

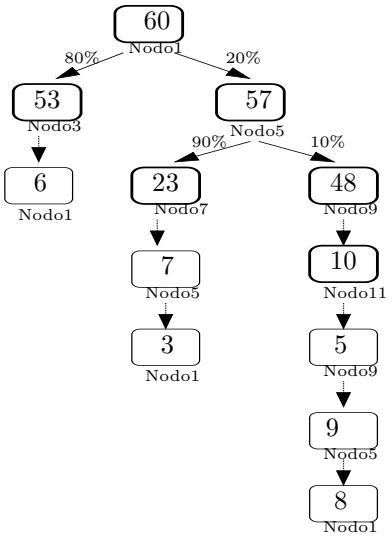

Figura D.4: *Thread Distribuída NB4.*

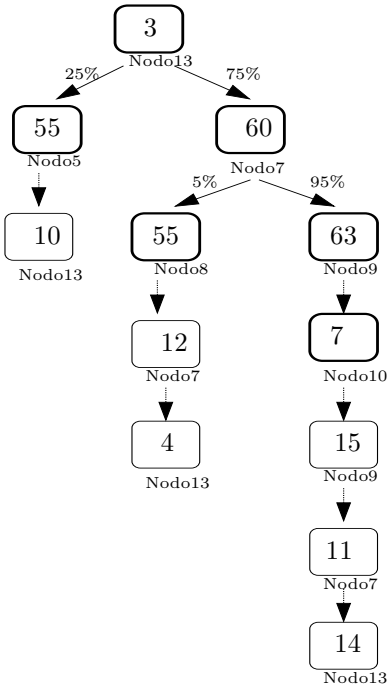

Figura D.5: *Thread Distribuída NB5.*

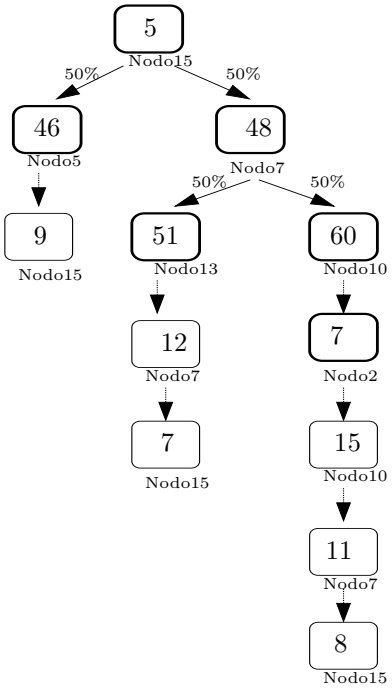

Figura D.6: *Thread Distribuída NB6.*

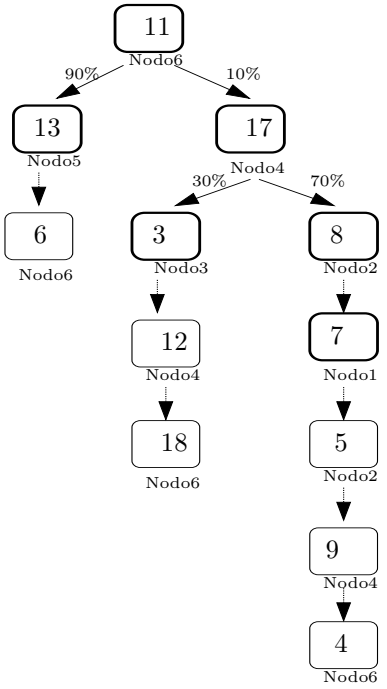

Figura D.7: *Thread Distribuída NB7.*

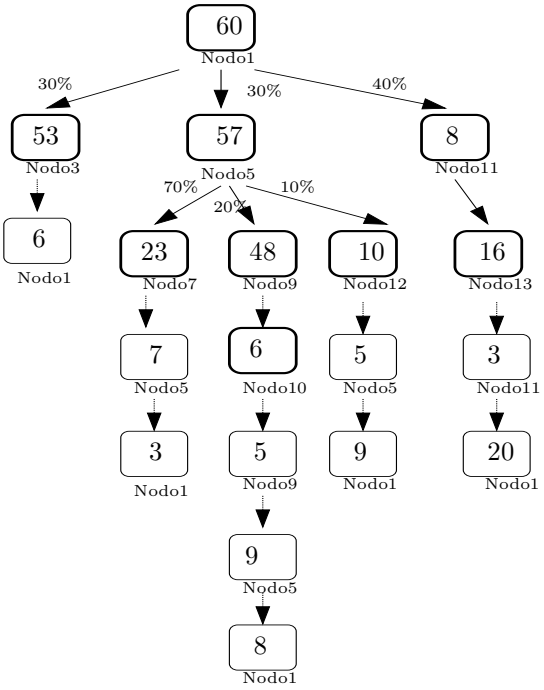

Figura D.8: *Thread Distribuída NB8.*

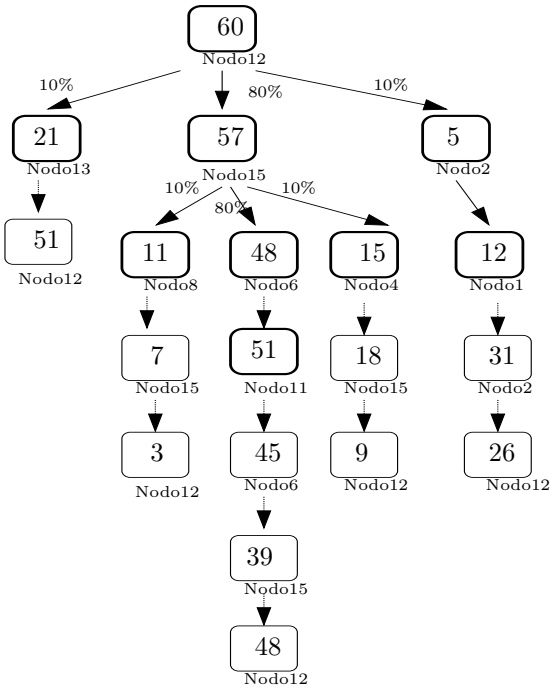

Figura D.9: *Thread Distribuída NB9.*

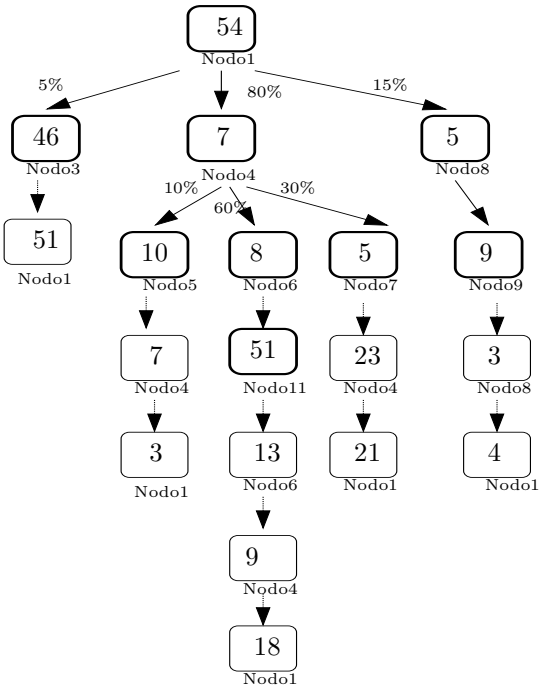

Figura D.10: *Thread Distribuída NB10.*

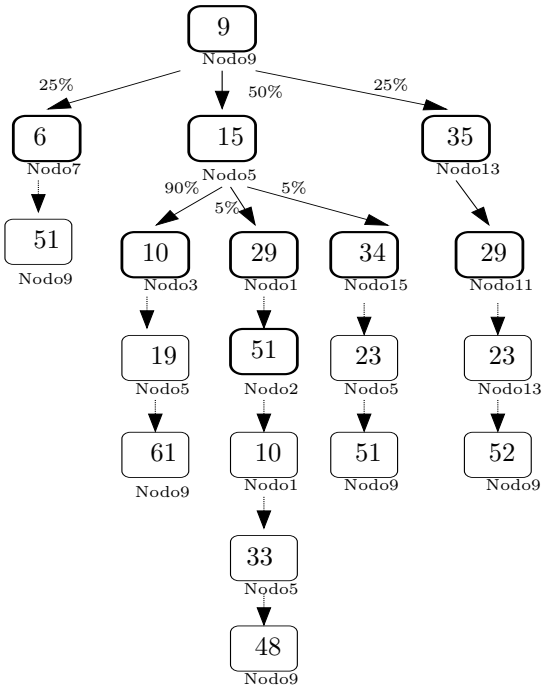

Figura D.11: *Thread Distribuída NB11.*

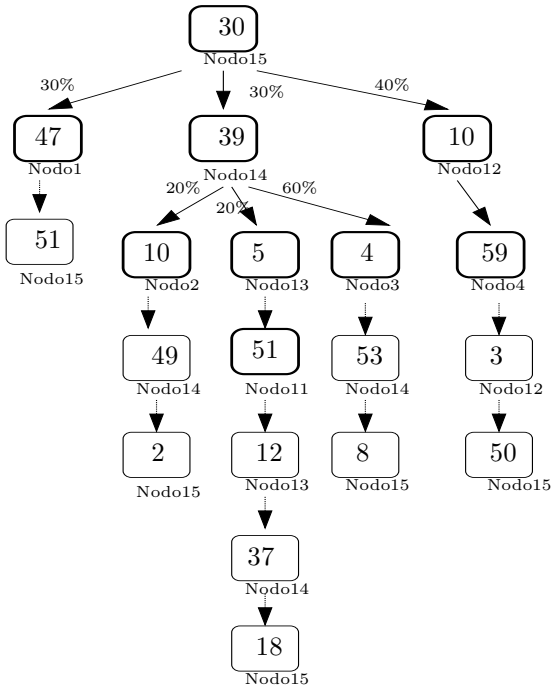

Figura D.12: *Thread Distribuída NB12.*

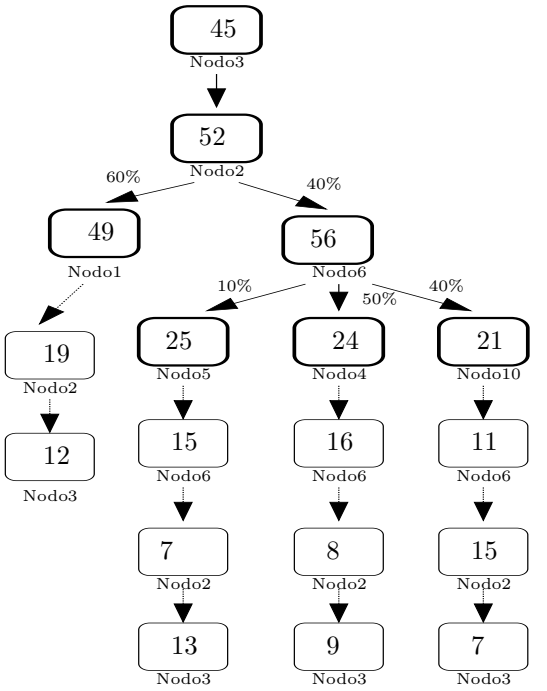

Figura D.13: *Thread Distribuída NB13.*

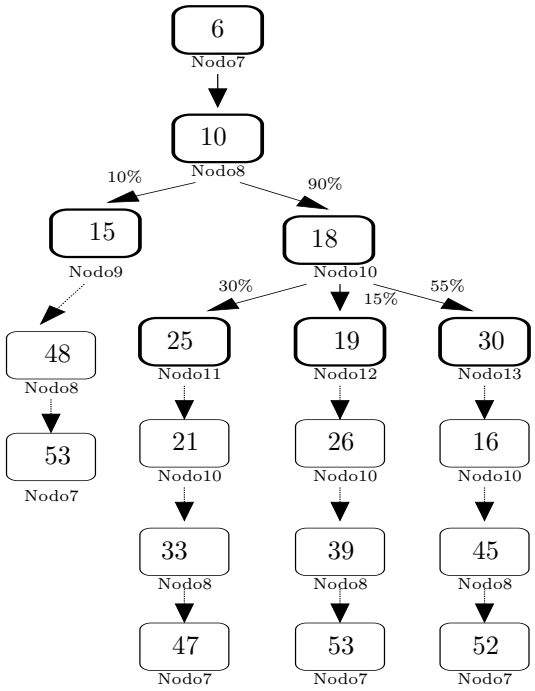

Figura D.14: *Thread Distribuída NB14.*

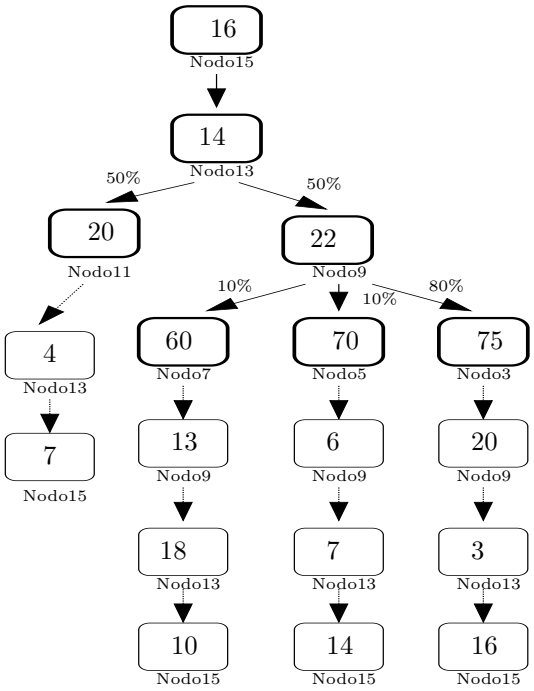

Figura D.15: *Thread Distribuída NB15.*

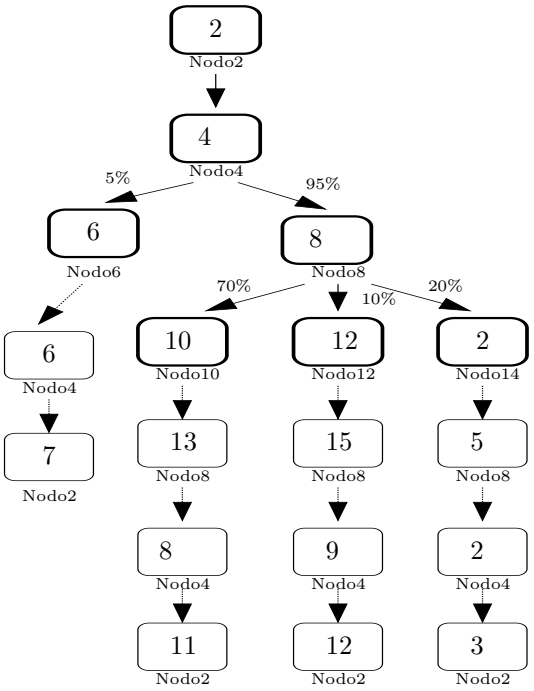

Figura D.16: *Thread Distribuída NB16.*

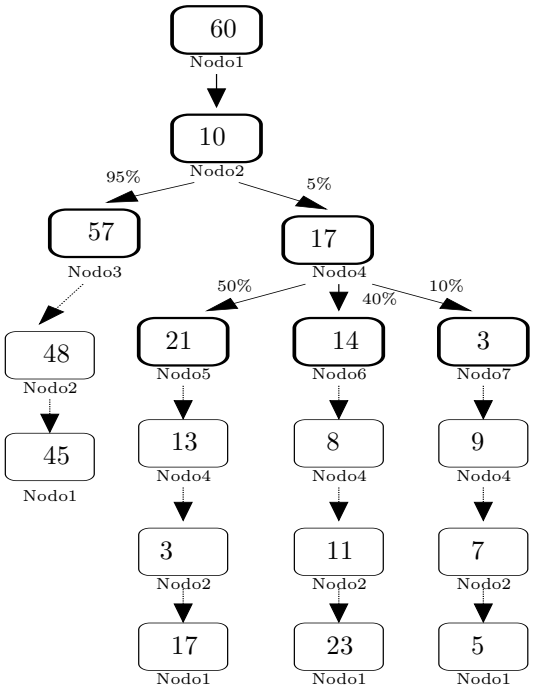

Figura D.17: *Thread Distribuída NB17.*

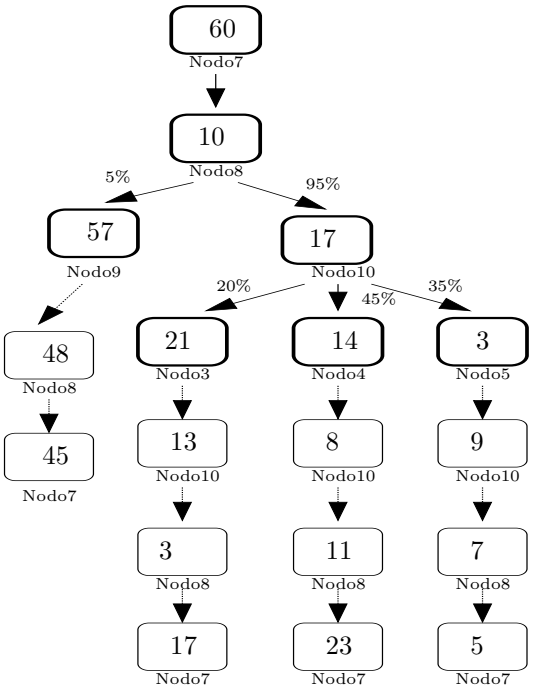

Figura D.18: *Thread Distribuída NB18.*

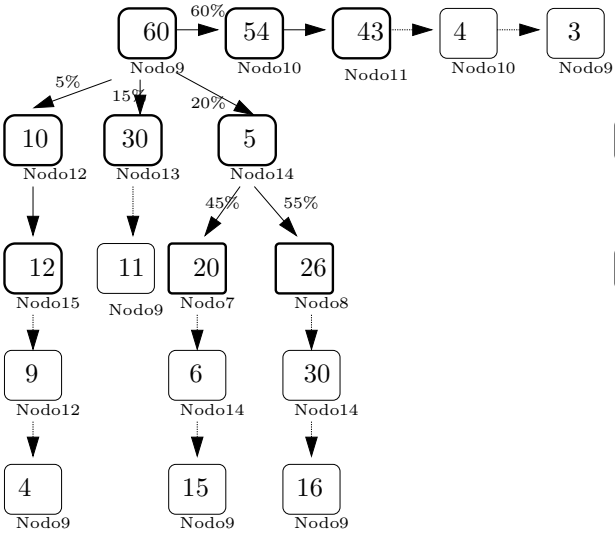

Figura D.19: *Thread Distribuída NB19.*

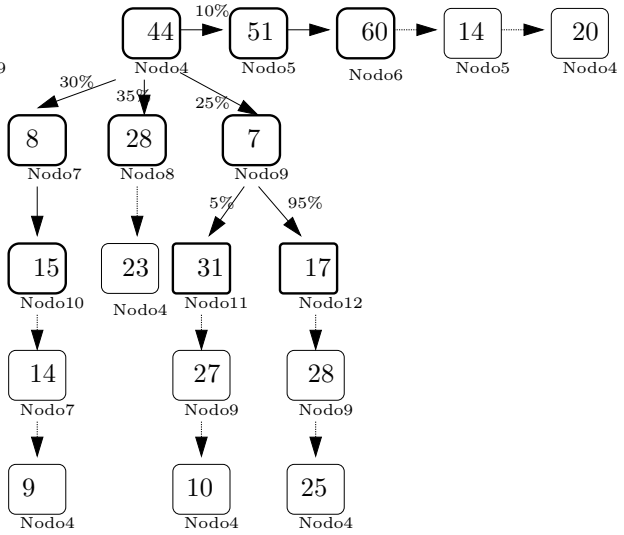

Figura D.20: *Thread Distribuída NB20.*

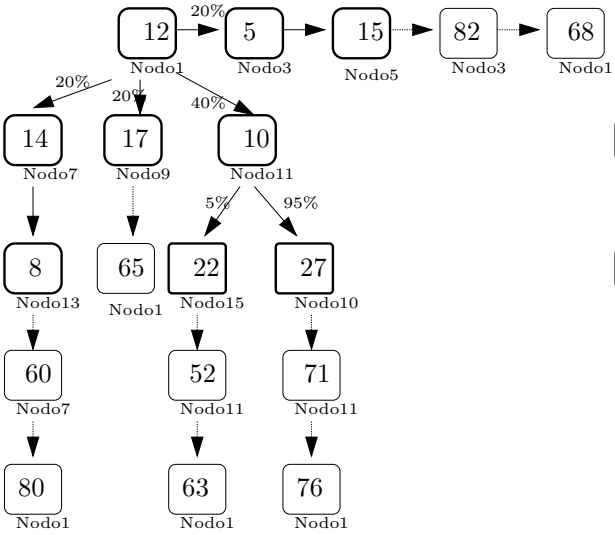

Figura D.21: *Thread Distribuída NB21.*

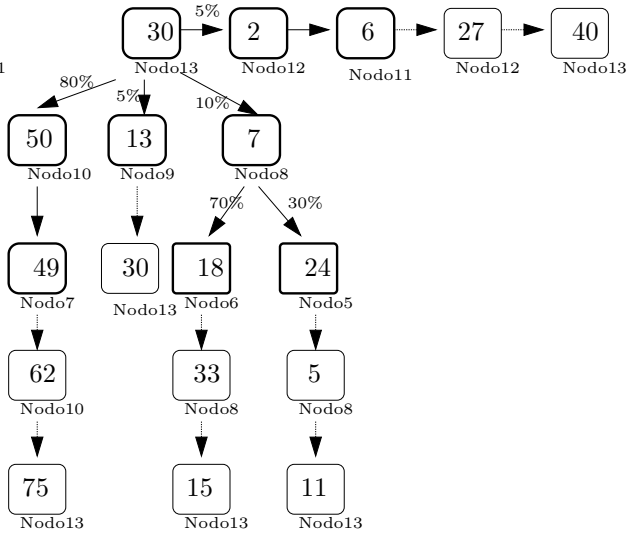

Figura D.22: *Thread Distribuída NB22.*

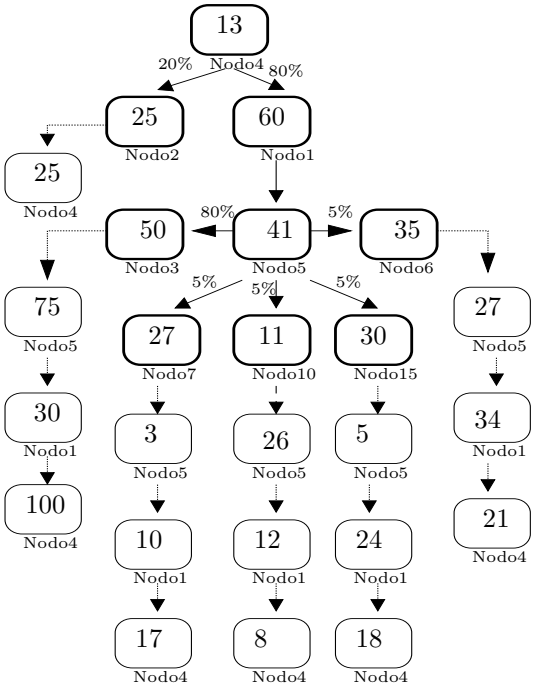

Figura D.23: *Thread Distribuída NB23.*

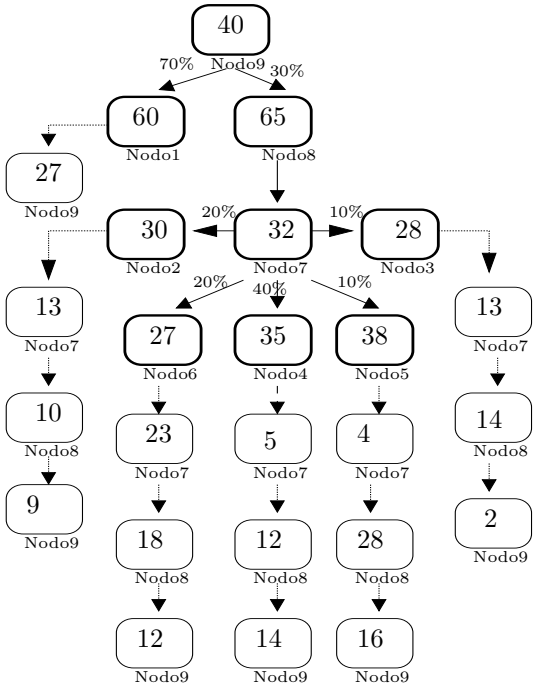

Figura D.24: *Thread Distribuída NB24.*

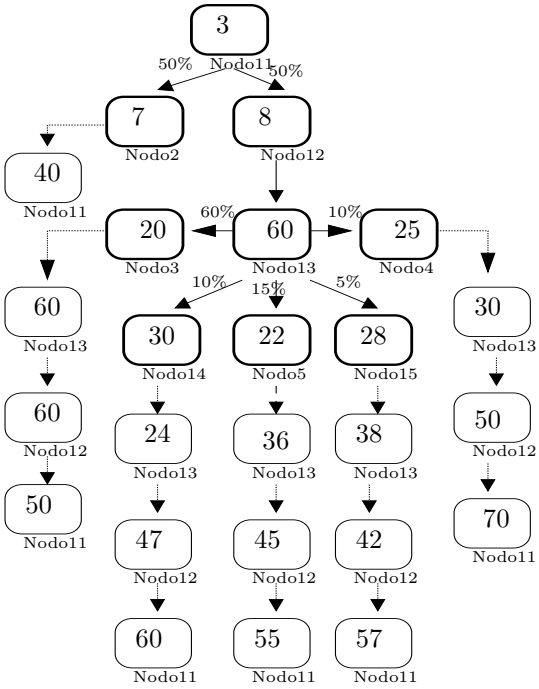

Figura D.25: *Thread Distribuída NB25.*

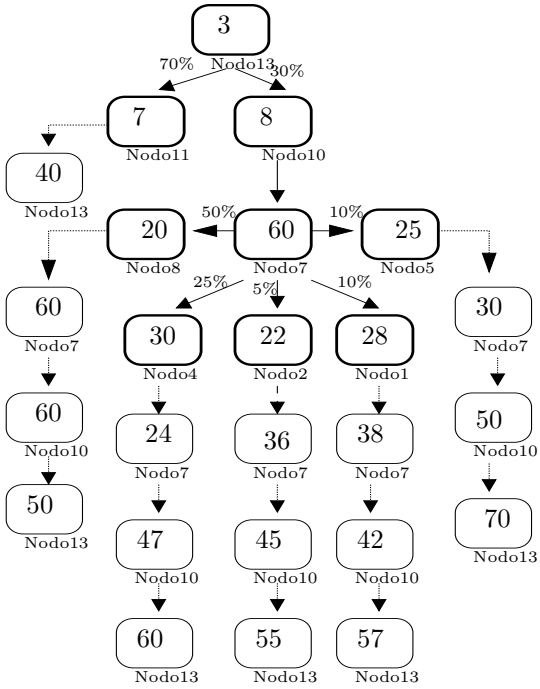

Figura D.26: *Thread Distribuída NB26.*

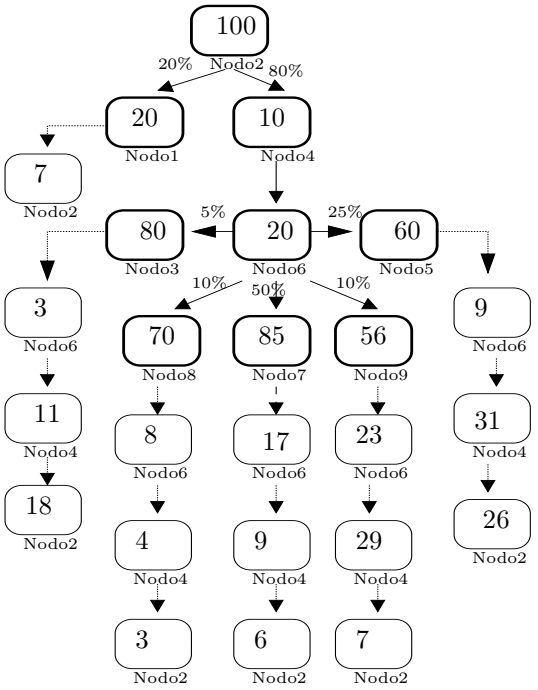

Figura D.27: *Thread Distribuída NB27.*

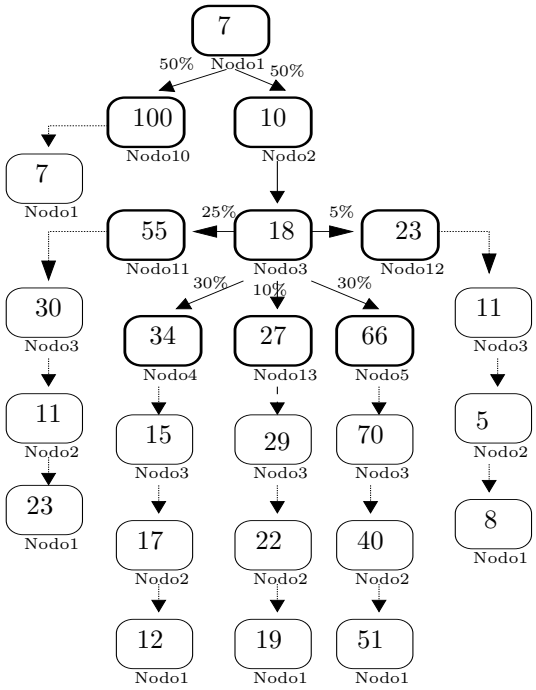

Figura D.28: *Thread Distribuída NB28.*

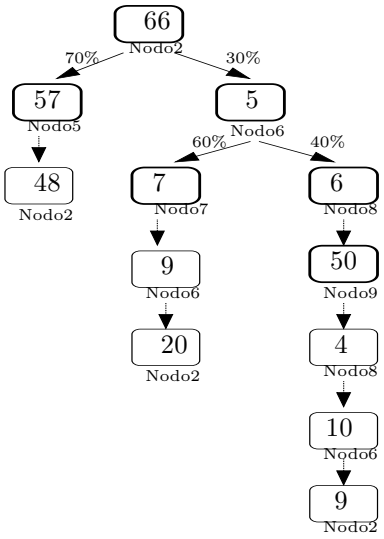

Figura D.29: *Thread Distribuída NB29.*

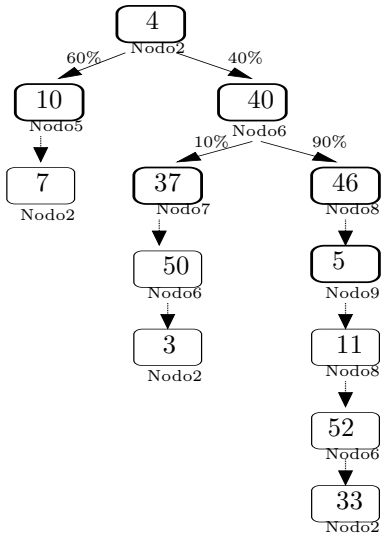

Figura D.30: *Thread Distribuída NB30.*