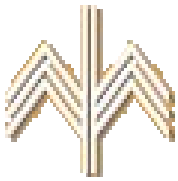

Universidade Federal do Rio de Janeiro
Instituto de Matemática
Núcleo de Computação Eletrônica

ORBITA: UMA ESTRATÉGIA DE BALANCEAMENTO DE CARGA PARA TAREFAS COM RESTRIÇÕES TEMPORAIS

Dissertação de Mestrado

Aluno:
Luís Fernando Orleans

Orientador:
Prof. Dr. Carlo Emmanoel Tolla de Oliveira

Rio de Janeiro

2007

ORBITA: UMA ESTRATÉGIA DE BALANCEAMENTO DE CARGA PARA TAREFAS COM RESTRIÇÕES TEMPORAIS

Luís Fernando Orleans

Dissertação submetida ao corpo docente do Instituto de Matemática e do Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro - UFRJ, como parte dos requisitos necessários à obtenção do grau de Mestre em Informática.

Aprovada por:

Prof._____ (Orientador)
Carlo Emmanoel Tolla de Oliveira, Ph.D.

Prof._____
Vanessa Braganholo Murta, D. Sc.

Prof._____
Alexandre Sztajnberg, D.Sc.

Rio de Janeiro

2007

ORLEANS, LUIS FERNANDO.

ORBITA: Uma Estratégia de Balanceamento de Carga para Tarefas com Restrições Temporais / Luís Fernando Orleans. Rio de Janeiro: UFRJ / IM / DCC, 2007.

71 pp.
Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro - UFRJ, IM / DCC, 2007.

Orientador: Prof. Carlo Emmanoel Tolla de Oliveira

1. Balanceamento de carga. 2. Qualidade de Serviço 3. Restrições temporais.
I. IM/NCE/UFRJ. II. Título (série).

Para Helen,
meu outro mundo.

Agradeço primeiramente a Deus, sem o qual nada seria possível. Sou muito agradecido por tudo que me foi proporcionado, pelas oportunidades que me foram concedidas, pelas pessoas e lugares que conheci e amigos que fiz.

À minha família que, mesmo nas diversas adversidades que passamos durante a elaboração deste trabalho, nunca deixou de me apoiar e incentivar para continuar até o fim. Em especial, gostaria de agradecer à minha mãe, meu irmão, meu padrasto e meus tios Luiz Paulo, Susie e Solange por serem, em todos os momentos, meu porto seguro.

À minha noiva Helen, fiel companheira de todas as horas. Obrigado por ter estado sempre a meu lado, não deixando nunca me sentir sozinho. Seu amor foi uma profunda fonte de inspiração para mim.

Ao meu primo Ricardo, médico da família, por cuidar muito bem de nossa saúde e pelo exemplo de perseverança para atingir os objetivos.

Aos meus amigos Dione, Fábio e Guilherme, irmãos que pude escolher.

Gostaria, por fim, de agradecer ao professor Pedro Furtado, da Universidade de Coimbra – Portugal, por ter me ajudado na elaboração deste trabalho, desde sua concepção até a implementação. Foram horas de reuniões e discussões, onde tive o privilégio de ouvir e aprender com um dos mais conceituados doutores na área de bancos de dados.

"*Só posso fazer uma coisa de cada vez,*
mas posso evitar fazer muitas coisas ao mesmo tempo."

Ashleigh Brilliant

RESUMO

ORLEANS, Luís Fernando. **ORBITA: Uma Estratégia de Balanceamento de Carga para Transações com Restrições Temporais**. Orientador: Prof. Carlo Emmanoel Tolla de Oliveira. Rio de Janeiro: UFRJ/IM, 2007. Dissertação (Mestrado em Informática).

Um dos pontos-chave na elaboração de projetos de sistemas paralelos consiste na forma como as requisições enviadas pelos clientes são distribuídas entre os servidores de modo a obter a máxima utilização destes, ou seja, suas políticas de balanceamento de carga. Apesar de este ser um tema largamente estudado, onde as soluções propostas são bem aceitas pela comunidade acadêmica, a adição de restrições temporais, também chamadas neste trabalho de *deadlines*, às requisições traz à tona uma nova perspectiva para o problema de balanceamento de carga: como distribuir as tarefas de modo a minimizar a taxa de quebra de *deadlines* (ou contratos). Usando um simulador, este estudo mostra que a variabilidade das durações das tarefas representa um papel crucial neste novo problema, identificando as requisições mais longas como os elementos mais críticos. A simulação mostra que as estratégias atuais, mesmo as dinâmicas, não conseguem obter uma taxa de quebra de contratos aceitável pois tratam da mesma forma tarefas grandes e pequenas. Com isso, este trabalho propõe um novo algoritmo de balanceamento de carga, chamado ORBITA, que inclui um mecanismo de classificação das tarefas de acordo com suas durações e um controle de admissão para tarefas grandes. O resultado da etapa da simulação mostra que este algoritmo supera seus concorrentes, apresentando maior taxa de tarefas que terminam a execução dentro da janela de tempo especificada. Sua superioridade é ainda mais evidente quando o sistema se encontra sob alta utilização. Por fim, foi desenvolvido um protótipo atestou a veracidade dos resultados colhidos durante a simulação. Este protótipo, desenvolvido em Java, realiza balanceamento de carga em bancos de dados paralelos e foi testado com o *benchmark* TPC-C. As experiências confirmaram todas as conclusões anteriores, mostrando que, a fim de minimizar a taxa de quebra de contratos, deve-se entender previamente a variabilidade da carga de trabalho do sistema.

ABSTRACT

ORLEANS, Luís Fernando. **ORBITA: Uma Estratégia de Balanceamento de Carga para Transações com Restrições Temporais**. Orientador: Prof. Carlo Emmanoel Tolla de Oliveira. Rio de Janeiro: UFRJ/IM, 2007. Dissertação (Mestrado em Informática).

A key point in parallel systems design is the way clients requests are forwarded and distributed among the servers, trying to obtain the maximum throughput from them or, in other words, the load-balancing policy. Although it is a largely studied theme, with well accepted solutions, the inclusion of temporal constraints, also denoted as deadlines in this work, to the requests brings new complexities to the load-balancing problem: how to distribute the tasks and minimize the miss rate. Using a simulator, this dissertation attests that the workload variability plays a crucial role in this problem, identifying the big requests as the most critical elements. The simulation process also shows that even dynamic load-balancing algorithms are not able to reach an acceptable miss rate, since they handle both short tasks and big tasks the same way. Hence, we propose a new load-balancing algorithm, called ORBITA, which has a request identification and classification mechanism and an admission control module as well, restricting the number of big tasks within the system. The simulation results shows that this algorithm outperforms its competitors, which means that it has a bigger rate of tasks that end within the deadline, specially when the system is under high load. A prototype was also built in order to check the veracity of the previous phase. The experiments were run against a benchmark tool, TPC-C, and all the results confirmed the previous assumptions, leading to the conclusion that it is a good practice to understand the system's workload in order to minimize the miss rate.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ACID: Atomicidade, Consistência, Isolamento, Durabilidade

FCFS: First Come, First Serve

LWR: Least-Work-Remaining

MPL: Multiprogramming Level

OLTP: Online Transaction Processing

ORBITA: On-demand Restriction for Big Tasks

QoS: Quality of Service

RR: Round-Robin

SGBDR: Sistema Gerenciador de Banco de Dados Relactionais

SITA-E: Size-Based Interval Task Assignment for Equal Load

SQL: Structured Query Language

TAGS: Task Assignment by Guessing Size

TPC-C: Transaction Processing Council - C

TPM: Transactions Per Minute

1 Introdução

Sistemas responsáveis por tratar requisições enviadas por clientes remotos apresentam uma perda de desempenho sempre que submetidos a altas utilizações – ocasionada pelos altos números de *page faults* e interrupções para tratamento de novas tarefas (DYACHUK, DETERS, 2007). A simples adição de novos servidores (ou nós) ou substituição por máquinas mais novas e com maior capacidade de processamento, apesar de ser uma solução simples, não é necessária para garantir quaisquer políticas de qualidade de serviço que possam ser estabelecidas entre o provedor de serviços e seus contratantes, pois um simples aumento na taxa de chegadas de novas requisições – ocasionada pelo aumento no número de clientes – faz com que o problema de excessivos *page faults* e interrupções retorne. Fica claro, assim, que é necessário um estudo mais profundo sobre como projetar um sistema paralelo/distribuído de modo que seu desempenho seja imune às variações da taxa de chegadas de novas requisições. Este trabalho visa realizar este estudo.

1.1 Balanceamento de carga

De acordo com o (DEVINE et al, 2005),

> "a política de balanceamento de carga – a tarefa de associar trabalhos a processadores – é uma escolha crítica em ambientes de simulação paralelos. Tal política é responsável por maximizar o desempenho da aplicação mantendo o processador o mais ocupado possível e diminuindo a comunicação entre processos."

Durante este trabalho, será considerado apenas balanceamento de carga para computadores, pois estamos levando em consideração apenas sistemas que processam requisições de clientes. As políticas de balanceamento de carga podem ser divididas em, segundo (DEVINE et al, 2005):

- **Estáticas:** quando não são levados em conta os estados dos servidores. O balanceador de carga atua como um objeto cego, simplesmente despachando a requisição para um servidor pré-determinado. Um exemplo de algoritmo de balanceamento de carga estático é o *Round-Robin*, onde as requisições são distribuídas pelos computadores obedecendo uma ordem circular.

- **Dinâmicas:** nesta classe, o balanceador de carga mantém algumas informações sobre os estados dos servidores e, com isso, consegue tomar uma decisão mais

inteligente sobre qual servidor deve receber a próxima requisição. Um exemplo desta política é o *Least-Work-Remaining* (LWR), que despacha as requisições sempre para o servidor que possuir o menor número de tarefas pendentes.

Existem vários estudos sobre balanceamento de carga que visam, principalmente, dois objetivos distintos: minimizar os tempos de resposta das requisições que chegam ao sistema ou maximizar a taxa de vazão. (HARCHOL-BALTER, 2002) apresenta uma solução, chamada *Task Assignment by Guessing Size* (TAGS, ver seção 2.2), para modelos onde o tamanho (ou duração) das tarefas não é conhecido *a priori*, o processo de chegadas das tarefas é Poisson, o tamanho das tarefas, que são não-preemptivas, segue uma distribuição de Pareto.

Ainda com o tamanho das tarefas desconhecido, mas o tamanho das tarefas seguindo uma distribuição exponencial, (NELSON, PHILIPS, 1989) e (NELSON, PHILIPS,1993) sugerem que o algoritmo *Least-Work-Remaining*, onde as requisições são encaminhadas para o servidor com menor utilização, é ótimo.

Em (HARCHOL-BALTER, CROVELLA, MURTA, 1999), é possível ver que o algoritmo *Size-Based Task Assignment with Equal Load* (SITA-E) possui um desempenho superior às políticas de balanceamento de carga tradicionais, como *Round-Robin*, *Weighted Round-Robin*, *Least-Work-Remaining* e *Shortest-Queue*. Neste caso, os tamanhos das tarefas devem ser conhecidos *a priori* e devem seguir uma distribuição de Pareto, o que significa possuir uma variabilidade alta. Quando a duração das tarefas segue uma distribuição que apresente uma variabilidade baixa, como é o caso da distribuição exponencial, o algoritmo SITA-E apresenta um desempenho inferior ao *Least-Work Remanining.*

Para fins de comparação, os três algoritmos citados (TAGS, SITA-E e LWR) são testados na fase de simulação e apenas os dois últimos, que apresentaram os melhores desempenhos nesta etapa, são incluídos na fase do protótipo.

1.2 Qualidade de serviço

(FURTADO, SANTOS, 2007) define qualidade de serviço (QoS – *Quality of Service*, em inglês) como:

> Um tema muito comum em pesquisas de sistemas distribuídos. A principal justificativa para a utilização de políticas de QoS para a internet se baseia no fato de a grande rede ser, em sua natureza, um ambiente de melhor esforço. Enquanto isso, alguns tipos específicos de aplicações, como transmissão de

> vídeo em tempo-real possuem requisitos que precisam ser satisfeitos.

Em sistemas paralelos, diversos servidores processam as requisições (tarefas ou transações) que lhes são encaminhadas de acordo com alguma política de balanceamento de carga. Estes servidores, em princípio, não fornecem nenhuma garantia sobre os tempos de resposta das requisições, seguindo uma abordagem conhecida como *melhor esforço* (WYDROWSKI, ZUKERMAN, 2002). Em situações de pico, onde as requisições chegam a uma alta taxa, esta prática pode levar a um cenário onde cada tarefa demore dezenas de vezes mais do que ocorreria em um servidor menos sobrecarregado.

Uma das maneiras de oferecer algum tipo de QoS é tentar garantir que os tempos de resposta das requisições emitidas por clientes de uma aplicação não ultrapassem um limite máximo, chamado de *deadline*. Os deadlines podem ser ser classificado de dois tipos: suaves ou rígidos. Segundo (GHAZALIE, BAKER, 1995),

> Um deadline rígido é aquele que deve ser sempre satisfeito, enquanto um deadline suave representa somente um tempo de resposta médio desejável. Se uma tarefa possuir ambos os tipos de deadlines, o suave deve apresentar um tempo de resposta inferior ao estabelecido no rígido.

Em sistemas comerciais pode ser desejável garantir um tempo de resposta máximo para cada requisição e este não deve ser demasiado longo. Para que isto seja possível, é necessário haver um estudo prévio acerca das durações das tarefas, de modo que a variabilidade do tamanho das tarefas seja reduzida. Segundo (HARCHOL-BALTER, CROVELLA, MURTA, 1999), o tempo de duração das tarefas segue uma distribuição de Pareto, o que significa que existirão poucas tarefas que comporão aproximadamente metade da carga do sistema. Este fenômeno é altamente indesejável pois, em situações de pico de acesso, tais tarefas podem comprometer o desempenho do sistema inteiro.

Outro fator que pode influenciar na duração das tarefas é o número de requisições executando simultaneamente no mesmo servidor. Para contornar este problema, (SCHROEDER, HARCHOL-BALTER, 2006) propôs uma arquitetura onde o nível de multiprogramação (MPL) limitava o número de trabalhos concorrentes em um único servidor. Ainda em (SCHROEDER, HARCHOL-BALTER, 2006), foi realizado um estudo de modo a encontrar o número mínimo para o MPL, onde a taxa de vazão do sistema fosse ótimo. Mas o artigo não abordou transações com tempo máximo de execução, o que é fundamental para este estudo.

1.3 Controle de admissão

Segundo (AMZA, COX, ZWAENEPOEL, 2005),

> controle de admissão é um adicional muito importante para qualquer algoritmo de balanceamento de carga, onde existe um limite pré-determinado para o número de execuções simultâneas em um repositório em particular. Limitando a carga tem o efeito de diminuir os efeitos causados por situações de pico, ocasionadas por rajadas de requisições que poderia causar uma condição de sobrecarga no banco de dados.

Normalmente, as tarefas que encontram, no momento de suas chegadas, o sistema saturado, são colocadas em uma fila de espera, normalmente com uma política de retirada baseada na ordem de chagada (primeiro a chegar, primeiro a ser servido, ou FCFS, em inglês). Uma outra abordagem, utilizada por servidores de bancos de dados é a rejeição da requisição, com o conseqüente envio de uma notificação de erro para o cliente.

O principal objetivo desta dissertação é estudar como os algoritmos de balanceamento de carga se comportam em momentos de alta utilização quando restrições de qualidade de serviço são incorporadas ao sistema. Será mostrado que, quando existem restrições temporais para as execuções das tarefas, os algoritmos tradicionais não conseguem obter uma taxa de vazão (aqui definida como número de requisições finalizadas antes de o deadline ser atingido por minuto) ao mesmo tempo justa e satisfatória. Será mostrado, ainda, que dos algoritmos já existentes, o de melhor desempenho (SITA-E, explicado na seção 2.3) realiza uma diferenciação das tarefas, classificando-as em “longas” ou “curtas”. Contudo, apesar de possuir uma taxa de vazão das requisições que atendam às restrições de QoS relativamente alta quando comparada com as outras abordagens, esta é composta basicamente de tarefas curtas ou, visto de outra forma, não pode ser considerada uma solução justa. Na visão deste trabalho, a taxa de vazão deveria seguir a proporção de tarefas curtas e longas que compõem a carga de trabalho entrante no sistema para que possa ser considerada uma solução de balanceamento de carga justa. Por fim, será proposto um algoritmo, chamado ORBITA, que é uma extensão do SITA-E, onde preocupações com as restrições temporais são adicionadas. O algoritmo ORBITA possui um mecanismo de controle de admissão, que restringe o número de tarefas longas sendo executadas nos servidores – número este calculado em tempo de execução.

Este trabalho abrange a criação de uma ferramenta, chamada Midas, capaz de realizar

balanceamento de carga entre servidores de bancos de dados relacionais paralelos. Projetada para ser uma ferramenta de fácil adoção para sistemas em produção, o Midas funciona como um interceptador dos diálogos entre clientes e bancos de dados, de acordo com o padrão de projeto *Proxy*. As atualizações nos dados são feitas em todos os servidores, enquanto as leituras são distribuídas de acordo com a política de balanceamento de carga. A fim de simplificar a construção do Midas, todas as atualizações são síncronas, ou seja, o cliente que enviou a operação fica bloqueado até que esta seja executada em todos os servidores.

1.4 Metodologia

Para avaliar o desempenho da proposta de estratégia para balanceamento de carga, será utilizada uma metodologia que inclui a utilização de um simulador para comparar os desempenhos dos algoritmos, a construção de um protótipo de acordo com os resultados obtidos no simulador em um ambiente paralelo simples, com apenas dois servidores. Uma ferramenta de *benchmark* conhecida, o TPC-C, será utilizada para simular um sistema transacional na fase do protótipo, que servirá para corroborar os resultados obtidos durante a fase de simulação.

1.5 Resultados esperados e contribuição científica

Ao término deste trabalho, são esperados resultados e uma ferramenta que sejam de valia para administradores de sistemas que necessitam acrescentar restrições temporais às requisições de forma rápida e transparente às aplicações, sem a necessidade de alterações em grande parte do código fonte. A ferramenta Midas, que será apresentada no capítulo 4, terá como função reduzir a complexidade advinda desta tarefa e, após a sua introdução no sistema, este deverá apresentar comportamento previsível e tempos de resposta que não ultrapassem o valor máximo pré-estabelecido.

A conclusão deste trabalho traz, como contribuição científica para a comunidade acadêmica, uma nova política de balanceamento de carga, capaz de reduzir o número de requisições que perdem deadlines. Os conceitos aqui apresentados podem ser utilizados tanto para pesquisas na área de banco de dados, como para sistemas de tempo real e redes de computadores.

1.6 Estrutura da dissertação

O restante da dissertação está estruturado da seguinte forma: o capítulo 2 faz uma revisão dos conceitos que são fundamentais para o total entendimento deste trabalho apresentando os algoritmos de balanceamento de carga existentes, discutindo suas vantagens e desvantagens em ambientes onde existem deadlines e alta taxa de chegada de requisições. Este capítulo aponta as causas do desempenho insatisfatório destes algoritmos. O capítulo 3 apresenta a proposta deste trabalho, o algoritmo de balanceamento de carga ORBITA. O capítulo 4 fornece detalhes da implementação do Midas, uma implementação do ORBITA para bancos de dados paralelos. O capítulo 5 compara, através de simulação, os desempenhos dos algoritmos de balanceamento de carga apresentados no capítulo 2 com o desempenho do ORBITA O capítulo 6 apresenta os resultados obtidos com o Midas, quando executado em conjunto com o TPC-C, uma ferramenta de *benchmark* para bancos de dados. Por fim, o capítulo 7 apresenta as conclusões deste trabalho e sugere possíveis pesquisas futuras.

2 Algoritmos existentes de balanceamento de carga

Balanceamento de carga é um conceito fundamental na construção de sistemas escaláveis. Apesar de diversos algoritmos de balanceamento de carga terem sido propostos, um dos mais utilizados na prática é também um dos mais simples de todos – Round-Robin (RR). Este algoritmo realiza um balanceamento de carga estático, onde as requisições são encaminhadas para os servidores de uma maneira circular, sem maiores considerações.

Por outro lado, o algoritmo Least-Work-Remaining (LWR) produz um balanceamento de carga dinâmico, pois as requisições são enviadas para o servidor com a menor utilização no momento – tarefas na fila de espera ou tarefas em execução. Este algoritmo é considerado neste trabalho pois é, supostamente, a melhor alternativa quando as durações das tarefas seguem uma distribuição exponencial (NELSON, PHILIPS, 1989) e (NELSON, PHILIPS, 1993).

De acordo com (HARCHOL-BALTER, 2002) e (HARCHOL-BALTER, CROVELLA, MURTA, 1999), o principal problema com o algoritmo LWR é a ausência de um controle sobre a durações das tarefas, pois misturam no mesmo servidor tarefas curtas e longas. Esta fraqueza se torna mais evidente quando a duração das requisições segue uma distribuição *heavy-tailed*, com uma minúscula fração das tarefas sendo responsável por mais da metade do processamento total.

A seguir, os algoritmos utilizados no trabalho são listados e seus desempenhos são exibidos.

2.1 Least-Work-Remaining (LWR)

Este algoritmo envia a próxima requisição para o servidor menos utilizado, aqui definido como aquele que possui o menor número de tarefas executando no momento.

Pseudo-código

```
for each task that arrives:
        next_server := least_utilized(server_list)
        send (task, next_server)
```

O LWR é um exemplo de algoritmo de balanceamento de carga dinâmico, pois leva

em consideração o estado de cada servidor na hora de despachar uma nova requisição. Intuitivamente, quando a variabilidade das durações das tarefas é pequena, ou seja, quando as tarefas têm em sua maioria durações semelhantes, este algoritmo possuirá um desempenho ótimo – pois o algoritmo tende a manter as utilizações de todos os servidores próximas umas das outras. Contudo, quando a variabilidade das durações das tarefas é alta, como acontece em distribuições *heavy-tailed* (ver seção 5.1), o desempenho do LWR diminui. Isto é causado pelo fato de existirem tarefas muito demoradas e outras muito curtas dividindo um mesmo servidor – pois deve-se levar em conta que as tarefas curtas levarão muito mais tempo para executar se estiverem dividindo o recurso com outras tarefas que são muito longas.

Figura 1: Comportamento do LWR quando as durações das tarefas são similares.

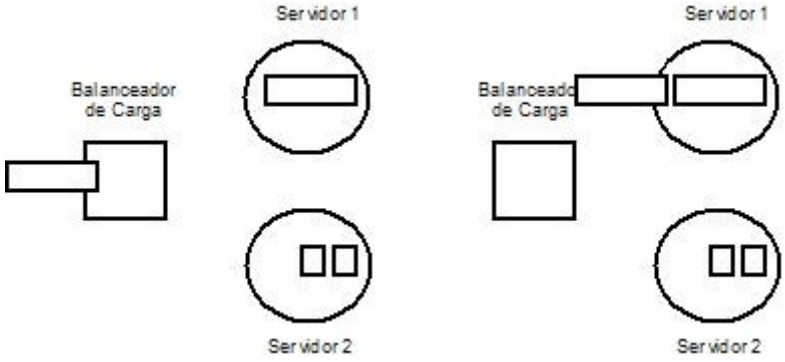

Figura 2: Possível comportamento do LWR quando as durações das tarefas seguem uma distribuição heavy-tailed.

As figuras 1 e 2 mostram como o algoritmo LWR se comporta quando a carga de

trabalho segue uma distribuição de baixa variabilidade (1) e quando segue uma distribuição que apresenta alta variabilidade (2). Neste segundo caso, o algoritmo apresenta um comportamento indesejável, pois

2.2 Task Assignment by Guessing Size (TAGS)

Este algoritmo envia todas as requisições para o primeiro servidor. Caso o tempo de resposta de alguma destas tarefas ultrapasse um determinado valor, ou seja, a tarefa é classificada como uma tarefa grande, ela é cancelada no primeiro servidor e reiniciada no segundo servidor.

Pseudo-código

```
for each task that arrives:
        send (task, first_server)
        schedule_dispatch_to_second_server(task)
```

Levando-se em conta que as durações das tarefas seguem uma distribuição *heavy-tailed*, o TAGS possui como principal vantagem sobre o LWR o fato de tentar evitar que tarefas pequenas tenham seus desempenhos prejudicados pelas tarefas grandes. Pode-se notar que este algoritmo já apresenta uma preocupação em dividir as tarefas em grandes ou pequenas, mesmo sem possuir um estudo prévio sobre as durações das tarefas.

Intuitivamente, o TAGS é uma opção mais adequada ao LWR quando:

1. A duração das tarefas segue uma distribuição *heavy-tailed.* Caso contrário, o algoritmo LWR é ótimo, como mostrado em (NELSON, PHILIPS, 1989) e (NELSON, PHILIPS,1993).

2. O sistema é estável, ou seja, a utilização do sistema é menor do que 1. Caso contrário, o número de tarefas enviadas para o primeiro servidor, grandes ou pequenas, o sobrecarregaria, enquanto que no LWR esta carga seria distribuída entre todos os nós.

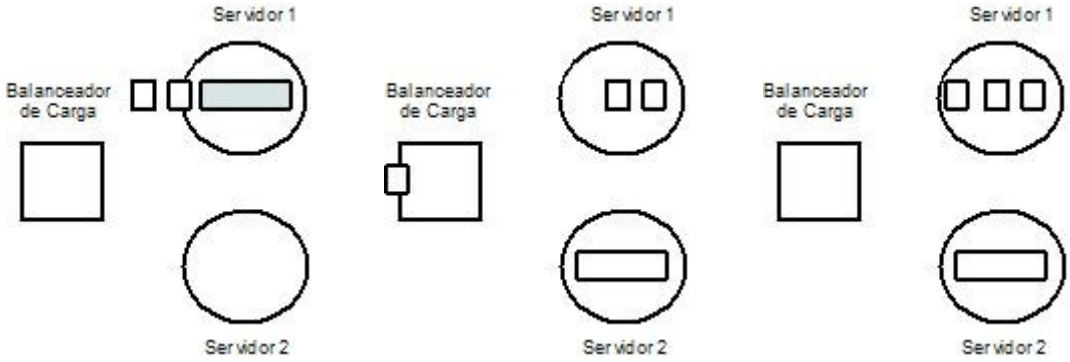

Figura 3: Comportamento do TAGS quando o sistema é dito estável.

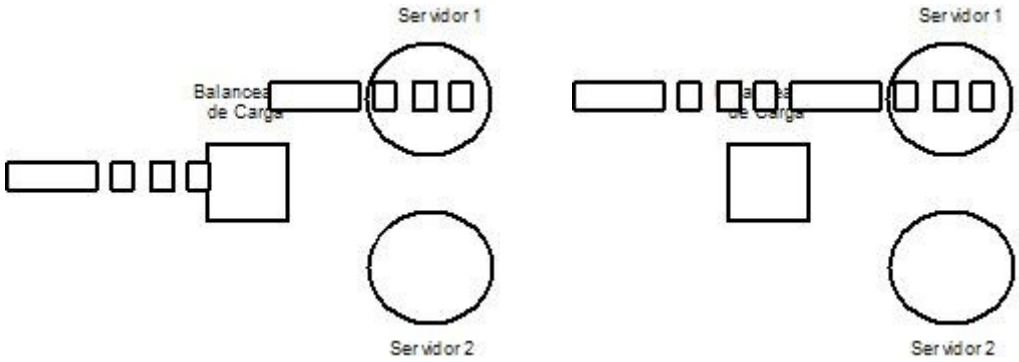

Figura 4: Comportamento do TAGS quando o sistema é dito instável. O servidor 2 fica ocioso, enquanto o servidor 1 fica completamente sobrecarregado.

As figuras 3 e 4 mostram como o algoritmo TAGS se comporta diante de diferentes utilizações. Na figura 3, considerando um sistema estável, é mostrado uma tarefa grande sendo cancelada no primeiro servidor (retângulo em cinza) e sendo reiniciada no segundo servidor. Enquanto isso, é mostrado, ainda, uma tarefa qualquer que acabou de chegar no sistema sendo despachada para o primeiro servidor. Já na figura 4, onde o sistema é dito instável, todas as tarefas são encaminhadas para o primeiro servidor, sobrecarregando-o. Como efeito secundário, o segundo servidor fica ocioso, até que uma tarefa seja cancelada e reiniciada nele.

Por fim, é importante salientar que, quando o sistema se encontra sob altíssima utilização, é possível que uma tarefa pequena seja cancelada e reiniciada no segundo servidor. Isto ocorreria pelo fato de haver muitas requisições concorrentes, fazendo com que o tempo de serviço de cada uma delas cresça incontrolavelmente – fazendo até com que tarefas pequenas demorem tanto para executar que o sistema as interpretaria como grandes.

2.3 Size Interval Task Assignment with Equal Load (SITA-E)

Este algoritmo, que precisa saber a duração de cada tarefa, divide as requisições em classes, de acordo com o tempo de duração. Desta forma, os servidores são reservados para executar somente requisições pertencentes a um tipo de classe. Assim como o TAGS, este algoritmo possui um excelente desempenho quando as durações das tarefas seguem uma distribuição *heavy-tailed*, onde uma minúscula fração é responsável por mais da metade da carga total do sistema.

Pseudo-código

```
for each task that arrives:
        if task is a big task
         server := server_list ->second_server
        else
         server := server_list ->first_server
        send (task, server)
```

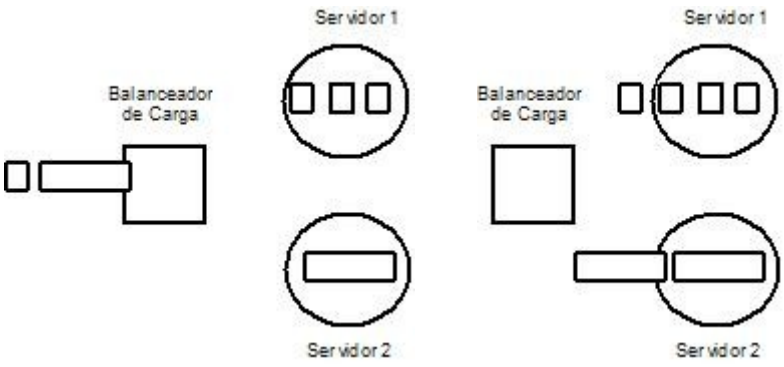

Figura 5: Comportamento do SITA-E quando a carga de trabalho é heavy-tailed.

A figura 5 mostra o desempenho do algoritmo SITA-E quando submetido a uma carga de trabalho de alta variabilidade. De fato, este algoritmo é recomendado para este tipo de carga de trabalho pois, esta possuir baixa variabilidade, o segundo servidor, dedicado ao tratamento de requisições grandes, pode ficar ocioso. Neste caso, o algoritmo LWR possuiria um desempenho superior ao SITA-E, uma vez que o LWR não permite que nenhum servidor

fique sem trabalhos.

Um outro problema do SITA-E que é importante frisar é a admissão não controlada de tarefas grandes no sistema. Desta forma, caso ocorra uma “rajada” de tarefas grandes sendo submetidas pelos clientes, todas elas serão direcionadas para o segundo servidor. Isto pode ser altamente indesejável em sistemas onde existam restrições temporais, já que a execução de diversas tarefas em um mesmo servidor fará com que cada requisição seja executada por um tempo maior do que se estivesse executando sozinha.

3 ORBITA

A proposta deste trabalho, conforme mencionado anteriormente, é a criação de um algoritmo de balanceamento de carga que seja robusto, isto é, que seja capaz de manter o mesmo desempenho até quando submetido a altíssimas utilizações, e que seja capaz de respeitar restrições temporais adicionadas por políticas de qualidade de serviço. Desta forma, chegou-se ao ORBITA, que significa, em inglês, *On-demand Restriction for Big Tasks* (restrições para tarefas grandes sob demanda).

Este algoritmo é uma melhoria do SITA-E. Tal como seu antecessor, divide as tarefas em classes de acordo com suas durações. As tarefas curtas são sempre admitidas no sistema, sendo encaminhadas para o primeiro servidor. Por outro lado, as tarefas longas devem passar por um controle de admissão que, por sua vez, tenta minimizar ao máximo a perda de deadlines. Para isso, calcula dinamicamente o número máximo de tarefas que podem ser executadas simultaneamente no mesmo servidor, tendo em vista o modelo de *processor-sharing* linear adotado. Caso seja constatado que requisição recém-chegada ocasionará a perda de *deadlines*, esta é recusada pelo sistema.

É possível calcular o número máximo de tarefas que podem executar concorrentemente sem que o *deadline* de nenhuma delas seja quebrado. Para isto, basta que o balanceador de carga divida o valor do *deadline* pelo valor da duração da **maior** tarefa executando no servidor. Caso o modelo de *processor-sharing* linear seja adotado, como neste trabalho, pode-se afirmar que o número inteiro do resultado da divisão corresponde ao MPL máximo para que nenhum *deadline* seja quebrado. Se o MPL do servidor for igual ou superior ao MPL máximo, a requisição que acabou de chegar deve ser recusada neste nó. Caso existam outros servidores dedicados ao tratamento de requisições longas, a tarefa pode ser aceita por outro nó, desde que sua aceitação não ultrapasse o valor do MPL máximo obtido naquele servidor.

É importante lembrar que essa abordagem de recusa de tarefas já é realizada por diversos tipos de servidores, sejam eles servidores web, de aplicação ou de banco de dados. Contudo, existe uma diferença crucial: enquanto um servidor recusa as tarefas que excedam o número máximo de conexões abertas, o ORBITA rejeita as conexões que possivelmente ocasionarão perda de *deadlines*.

Pseudo-código

```
for each task that arrives:
        if task is a small task
         server := server_list ->first_server
         send(task, server)
        else
         server := server_list ->second_server
         bigger_task := bigger_running_task(server)
         bigger_task := MAX(bigger_task, duration(task))
         max_ce := LOWER_BOUND(deadline/bigger_task)
         if number_of_running_tasks(server) >= max_ce
          NOT_ADMITT(task)
         else
          send (task, server)
```

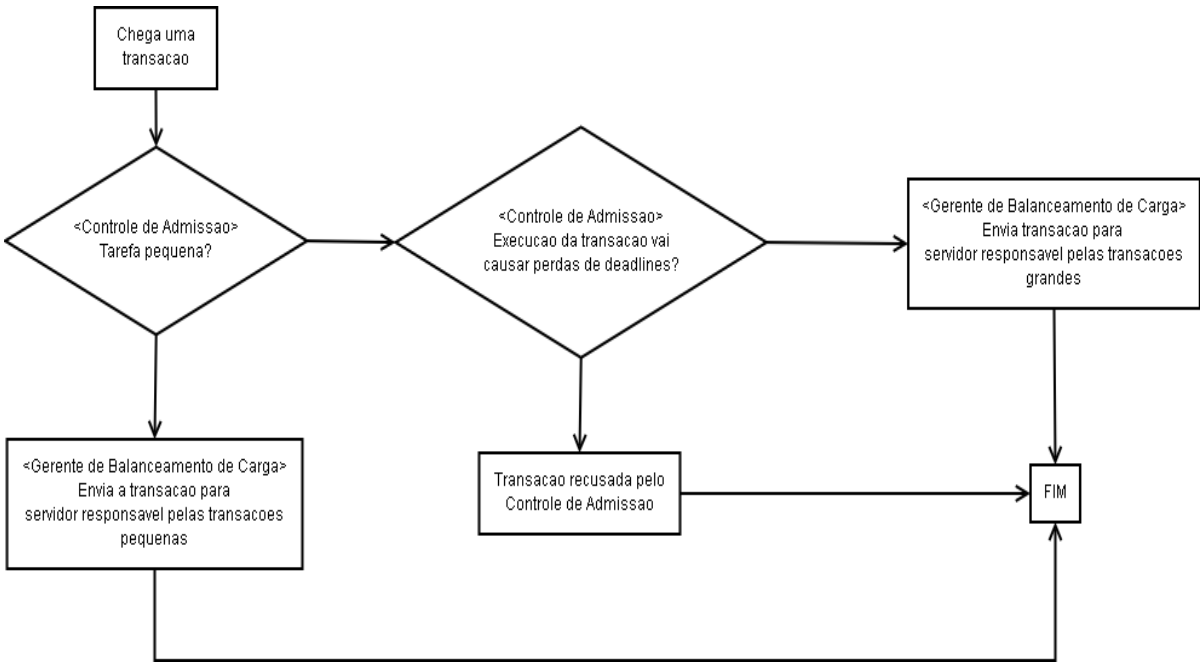

Figura 6: Fluxograma de uma chegada de transação com o algoritmo ORBITA.

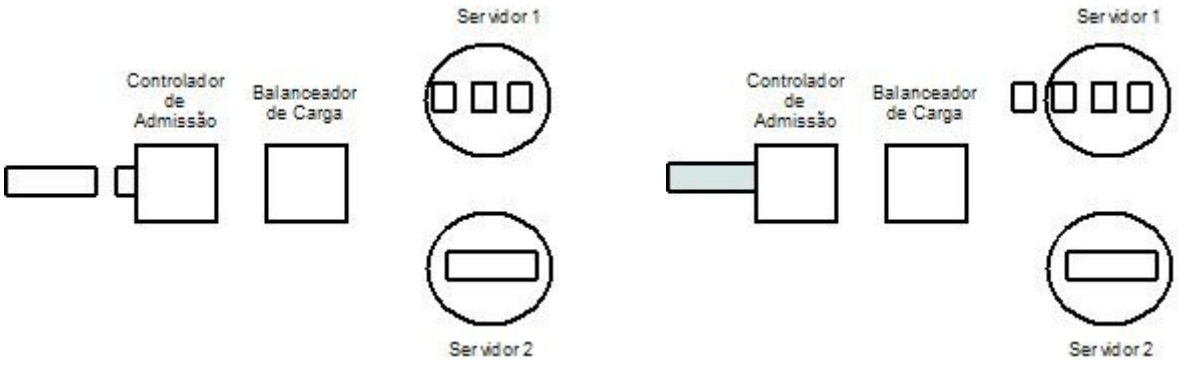

Figura 7: Exemplo de comportamento do algoritmo ORBITA.

A figura 7 mostra que o ORBITA, além de possuir um componente responsável por realizar o balanceamento de carga, possui um módulo adicional, cuja tarefa é controlar a admissão de tarefas grandes. Na figura, chegam duas requisições: uma curta e outra longa. A primeira passa direto pelo controle de admissão e o balanceador de carga a despacha para o primeiro servidor. A segunda tarefa, contudo, é grande. Logo, o controlador de admissão verifica se a admissão da tarefa fará com que alguma das requisições grandes já executando ultrapassem o deadline – o que é confirmado e a nova tarefa, rejeitada (marcada em cinza).

A tabela 1 faz uma comparação entre cada uma dos algoritmos utilizados no trabalho. É interessante notar que apenas o ORBITA possui as três melhorias consideradas como importantes.

Tabela 1: Comparação entre os algoritmos utilizados no trabalho.

	Algoritmo Dinâmico	***Controle de Admissão***	***Diferenciação das requisições***
TAGS	Não	Não	Não
LWR	Sim	Não	Não
SITA-E	Não	Não	Sim
ORBITA	Sim	Sim	Sim

Para finalizar, é mostrado a seguir um exemplo “passo-a-passo” de como o ORBITA se comporta. Para isso, considerou-se que uma tarefa com duração menor do que 1 segundo é considerada pequena e que o *deadline* foi estabelecido em 10 segundos. Por fim, considere que um modelo de *processor-sharing* linear, o que significa que as durações aumentam linearmente de acordo com o número de tarefas executando concorrentemente.

Tabela 2: Exemplo de comportamento do algoritmo ORBITA.

Tempo (s)	***Evento***	***Ação***
1	Chegada de req. 0.5 seg	Encaminhar ao servidor 1
1.2	Chegada de req. 0.5 seg	Encaminhar ao servidor 1
1.4	Chegada de req. 5 seg	Servidor 2 vazio, logo não ocasionará perda de *deadlines*. Encaminhar ao servidor 2
1.6	Chegada de req. 0.5 seg	Encaminhar ao servidor 1
1.8	Chegada de req. 0.5 seg	Encaminhar ao servidor 1
2.0	Chegada de req. 5 seg	Cálculo do MPL máximo: 10/5 = 2. Existe somente 1 requisição no servidor, logo a admissão não ocasionará perda de *deadlines*. Encaminhar ao servidor 2.
2.2	Chegada de req. 0.5 seg	Encaminhar ao servidor 1
2.4	Chegada de req. 5 seg	Cálculo do MPL máximo: 10/5 = 2. Já existem 2 requisições no servidor, logo a admissão ocasionará perda de *deadlines*. Rejeitar tarefa.
...	***...***	***...***
9.4	Saída da primeira req. 5 seg	-
9.6	Chegada de req. 8 seg	Cálculo do MPL máximo: 10/8 = 1. Já existe 1 requisição no servidor, logo a admissão ocasionará perda de *deadlines*. Rejeitar tarefa.

3.1 Discussão

O algoritmo ORBITA foi criado, assim como seu predecessor, o SITA-E, para cenários onde a duração das tarefas siga uma distribuição *heavy-tailed*, ou seja, com alta variabilidade e onde o número de tarefas grandes é pequeno. O mecanismo de controle de admissão do ORBITA tem a responsabilidade de rejeitar as tarefas longas que possivelmente ocasionarão perda de *deadlines*, quebrando, assim, restrições de qualidade de serviço.

Intuitivamente, em cenários de baixa e média utilizações, o algoritmo apresentará uma taxa de rejeição relativamente baixa, pois o número de tarefas grandes é muito pequeno. Porém, se pensarmos em um sistema instável, onde a taxa de chegadas de novas tarefas é superior à soma da capacidade de processamento dos servidores, o número de tarefas rejeitadas será maior. Porém, a taxa de vazão das tarefas que terminaram de executar dentro da janela de tempo permitida permanecerá inalterada, devido ao mecanismo de controle de admissão.

É interessante notar que o ORBITA preocupa-se somente em restringir o número de tarefas grandes. Será mostrado em capítulos posteriores que, mesmo quando o sistema está sob altíssima utilização, todas as tarefas pequenas conseguem executar dentro do *deadline*. Isto não ocorre por acaso. Como as tarefas pequenas são de fato muito rápidas, elas conseguem terminar suas execuções mesmo quando executadas simultaneamente com outras.

4 Midas

Foi desenvolvida uma ferramenta para estudo de caso, chamada Midas. Em linhas gerais, esta ferramenta implementa o padrão de projeto *Proxy* (GAMMA et al, 2002) e possui como objetivo prover diversos serviços de QoS para transações realizadas em Sistemas Gerenciadores de Bancos de Dados Relacionais (SGBDRs) de forma transparente. O Midas, desenvolvido em Java, atua como um interceptador das transações e, com isso, consegue adicionar as funcionalidades necessárias para implementar corretamente o algoritmo ORBITA. De modo a verificar que os resultados apresentados anteriormente estejam corretos, os algoritmos LWR e SITA-E também foram implementados.

4.1 Arquitetura

A arquitetura do Midas está fortemente ligada ao padrão de projeto *Proxy* pois, desta forma, é capaz de adicionar funcionalidades a uma aplicação de forma transparente. Adicionalmente, o Midas faz uso do padrão de projeto *Singleton* (GAMMA et al, 2002), para garantir que existirá somente um objeto em memória responsável pelo balanceamento de carga. A figura 8 mostra, de forma simplificada, a arquitetura do Midas. A principal classe é a ConnectionQoS, uma implementação do padrão de projetos *Proxy*. Ela intercepta as requisições enviadas pelo cliente e nelas aplica as políticas de balanceamento de carga, controle de admissão e replicação de dados.

Quando um cliente adquire uma conexão para o banco de dados, na verdade, ele recebe uma instância do *Proxy* que implementa a mesma interface da conexão real. A partir daí, sempre que um cliente iniciar uma transação, o *Proxy* consulta o *Singleton* para saber se a transação pode executar naquele momento ou não. Se puder executar, um identificador da transação é adicionado a uma lista de transações permitidas e todas as requisições seguintes são executadas sem passar pelo controle de admissão novamente. Caso a transação não possa executar, uma exceção chamada *QoSException* é lançada, avisando que o sistema está saturado.

O Midas é dividido em três módulos que podem evoluir de forma independente, cada um de acordo com uma determinada política. São eles: controle de admissão, replicação de dados e balanceamento de carga.

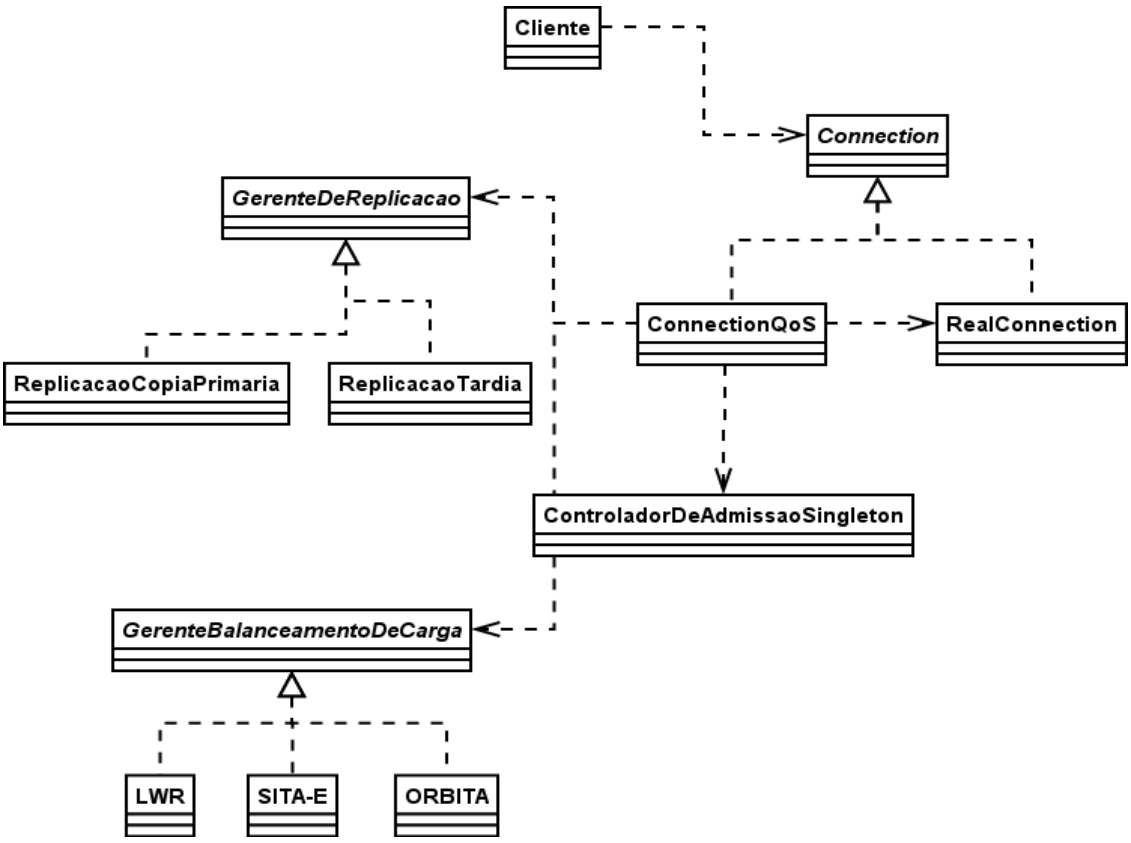

Figura 8: Diagrama de classes do Midas.

4.1.1 Módulo de controle de admissão

Este módulo é responsável por definir qual a medida a ser utilizada para considerar o sistema saturado. Dentre as possíveis, estão:número de transações que podem ser executadas de forma concorrente, a taxa de perda de deadlines, o somatório das durações das transações (previamente computado), etc..

Este módulo deve, ainda, decidir o que fazer com as transações que chegam e não podem ser executadas, ou seja, se elas são descartadas de imediato ou se aguardam execução em uma fila de espera. Ambas as políticas estão implementadas na ferramenta mas, para ser fiel ao modelo de simulação, apenas o modelo de rejeição de transações foi utilizado neste estudo. Experiências com uma arquitetura contendo fila de espera foram feitas, mas os

resultados não foram nada interessantes. Isto ocorre pois, para sistemas que sofrem com mau desempenho ocasionado pela sobrecarga de requisições, a fila de espera cresce indefinidamente e todas as requisições ultrapassam seus deadlines antes mesmo de entrarem em execução. Diversos algoritmos para retirar tarefas da fila foram propostos, principalmente por pesquisadores na área de sistemas de tempo real, mas a implementação de tais algoritmos está fora do escopo deste trabalho, que é averiguar políticas de balanceamento de carga que respeitem restrições temporais das requisições.

De toda maneira, é possível utilizar filas de espera no Midas, bem como aprimorar o algoritmo de escalonamento para tentar evitar a perda de deadlines (o algoritmo padrão é FCFS, ou seja, as requisições são atendidas na ordem que chegam).

Uma última consideração sobre o módulo de controle de admissão é a possibilidade de este realizar diferenciação de serviço entre as transações. Desta forma, pode-se definir quais os tipos de transação que serão submetidos ao controle de admissão. Esta diferenciação é realizada no ORBITA, onde as tarefas consideradas pequenas são executadas imediatamente, enquanto as tarefas grandes têm suas aceitações no sistema restringida. Para realizar esta etapa, é preciso que cada transação possua um identificador. Quando um cliente envia a primeira requisição SQL para o banco de dados, deve informar antecipadamente o identificador da transação que está sendo iniciada para que o Midas saiba como tratá-la.

4.1.2 Módulo de replicação de dados

O módulo de replicação de dados, apesar de não ser o foco principal do trabalho, é de suma importância para que o balanceamento de carga funcione corretamente, pois é responsável por manter a consistência em todos os servidores, permitindo que as consultas retornem sempre os dados mais atualizados possível. Existem diversos algoritmos de replicação de dados (WIESMANN, SCHIPER, 2005), (WIESMANN et al, 2000), (AKAL et al, 2005), muitos deles implementados de forma nativa pelos SGBDs. O Midas, entretanto, oferece uma alternativa externa à tecnologia de armazenamento, permitindo utilizar servidores heterogêneos.

Dentre os modelos de replicação, estão:

- **Tranca distribuída:** as atualizações de dados (inserções, modificações e exclusões) são enviadas para todos os nós de forma síncrona, enquanto as consultas são distribuídas entre os servidores de acordo com a política de balanceamento de

carga. Esta forma de replicação possui diversos problemas, estando entre os principais a sobrecarga de rede causada pelos excessivos *broadcasts*, e o nivelamento do desempenho do sistema com o servidor mais lento.

- **Replicação de cópia primária:** neste algoritmo, todas as transações responsáveis por atualizações de dados são encaminhadas para um único servidor. Ao término da confirmação da transação (*commit*), uma única mensagem contendo a transação inteira é enviada por *broadcast* para os demais servidores de forma assíncrona, o que causa um ganho de desempenho. As transações que não atualizam dados (transações somente de leituras), podem ser balanceadas entre os nós, com a penalidade de que os dados podem estar desatualizados, ou seja, os dados foram alterados no servidor primário mas a mensagem de alteração ainda não foi recebida pelo servidor onde a consulta foi realizada.

- **Replicação tardia:** a replicação tardia tem sido proposta como uma alternativa às técnicas de replicação imediata. Como todos os nós aceitam transações que alteram dados, este modelo tem como principal problema a possibilidade de haver conflitos de atualização de dados, o que viola duas das propriedades fundamentais das transações (consistência e isolamento das propriedades ACID). Sempre que o mesmo dado for atualizado em mais de um servidor por transações diferentes, deve ser executada uma *rotina de reconciliação*, responsável por devolver o banco de dados a um estado consistente.

Estes e outros modelos de replicação de dados são comparados em (WIESMANN, SCHIPER, 2005), onde o algoritmo de replicação tardia se mostra o de melhor desempenho (no artigo citado, o autor não leva em consideração colisões de atualizações, nem o tempo extra que deveria ser despendido pelo mecanismo de reconciliação). Ainda em (WIESMANN, SCHIPER, 2005), os autores concluem que a política de balanceamento de carga possui um impacto profundo no desempenho de bancos de dados distribuídos, principalmente quando a carga de trabalho possui um grande número de consultas.

Para fins de estudo de caso, o algoritmo escolhido para a replicação de dados foi a replicação de cópia primária pois, desta forma, podemos analisar o desempenho do ORBITA em um cenário onde os dados estão sempre consistentes, apesar de possivelmente desatualizados.

4.1.3 Módulo de balanceamento de carga

Este módulo, que é o principal tema deste trabalho, apresenta uma maneira simples e flexível de utilizar diversos algoritmos de balanceamento de carga. Através do padrão de projeto *Strategy* (GAMMA et al, 2002), o Midas é capaz de utilizar as mais diversas políticas de balanceamento de carga. Os algoritmos utilizados nas experiências são o LWR, o SITA-E e o ORBITA.

4.2 Diferenciação dos tipos de transações

Para tornar possível o Midas implementar corretamente o algoritmo ORBITA, foi necessário criar um mecanismo para realizar a diferenciação das transações. Em princípio, a ferramenta foi planejada para que fosse possível identificar todas as transações através dos comandos SQL que as compõem, sempre em tempo de execução. Desta forma, seria possível adotar o Midas em sistemas que já estivessem em produção, sem a necessidade de nenhuma outra configuração adicional. Porém, como tal tarefa de implementação mostrou exigir mais esforços para implementação do que o inicialmente planejado, foi necessário adotar uma solução simplificada, onde os clientes devem informar ao Midas o tipo de transação (pré-definidas em um arquivo texto separado) que irão submeter naquele momento.

5 Simulação

Com o intuito de verificar o desempenho dos algoritmos sob condições de stress do sistema, foi criado um simulador que possui como variáveis:

- O número de servidores;
- A taxa de chegadas de requisições, segundo uma distribuição exponencial;
- Distribuição da duração das tarefas, segundo uma distribuição de Pareto;
- Tempo máximo que uma requisição pode executar (deadline);
- Algoritmo de balanceamento de carga;
- Duração máxima para uma tarefa ser considerada pequena.

Nesta fase de análise preliminar, um sistema com apenas dois servidores idênticos foi simulado. O tempo de deadline foi ajustado para 20 segundos. Requisições chegam segundo uma distribuição exponencial cuja média varia de 1 a 10 requisições por segundo e suas durações seguem um distribuição de Pareto, onde o fator de variabilidade *a* varia de 0.1 a 0.3. A menor duração de uma tarefa é 0.001 segundo e a maior é 10 segundos. O modelo de concorrência é compartilhamento de processador (*processor sharing*) linear, o que significa que se uma tarefa demorar 10 segundos para executar sozinha em um servidor, levará 20 segundos se dividir o recurso com outra requisição, 30 segundos se dividir o recurso com mais 2 tarefas e assim por diante. Para cada algoritmo, foi executada uma rodada de simulação que durou 6 horas em um sistema vazio. Para eliminar a fase transiente, onde os dados ainda não estão consistentes para serem colhidos, as informações coletadas durante a primeira hora de simulação foram descartadas. Apenas os dados colhidos nas 5 horas seguintes foram considerados para a elaboração dos gráficos.

***Tabela 3**: Porcentagem das durações geradas, de acordo com o fator de variabilidade (a) da distribuição de Pareto e dos intervalos de duração.*

a	[0, 1[	[1, 2[	[2, 3[	[3, 4[	[4, 5[	[5, 6[	[6, 7[	[7, 8[	[8, 9[	[9, 10[	10
0,1	53,88%	6,97%	5,17%	4,90%	4,42%	4,64%	3,93%	4,24%	4,25%	3,74%	3,85%
0,2	76,02%	4,75%	3,18%	2,94%	2,30%	1,90%	1,87%	1,95%	1,79%	1,56%	1,75%
0,3	87,90%	2,91%	1,76%	1,38%	1,12%	0,98%	0,89%	0,80%	0,78%	0,75%	0,73%

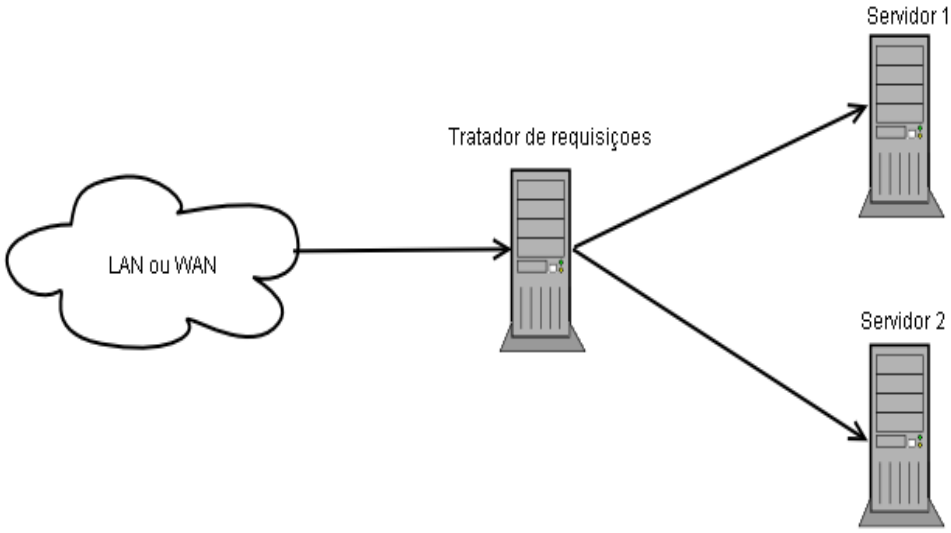

Figura 9: Arquitetura simulada

5.1 Geração de cargas de trabalho

A geração de cargas de trabalho é um tema amplamente estudado onde, em (HARCHOL-BALTER, CROVELLA, MURTA, 1999), defende-se que as durações das tarefas que chegam a um sistema distribuído seguem alguma distribuição que possua alta variabilidade (heavy-tailed). De um modo geral, tais distribuições são da forma

$$\Pr\{X > x\} \sim x^{-a},$$

onde $0 < a < 2$. A distribuição deste tipo mais simples é a *distribuição de Pareto*, que possui como função probabilidade de massa

$$f(x) = ak^a x^{-a-1}, a, k > 0, x > k$$

e função distribuição acumulada:

$$F(x) = 1 - (k \div x)^a$$

Onde *a* é o expoente da *power law* (também chamado neste trabalho de **fator de variabilidade** de uma distribuição *heavy-tailed*) e *k* é o menor valor possível de uma amostra. Tarefas que sigam uma distribuição como essa possuem as seguintes propriedades:

1. Quanto mais tempo uma tarefa estiver sendo executada, maior é a probabilidade de ela continuar em execução.

2. Variância infinita (e se $a \leq 1$, média infinita).

3. A propriedade de que uma fração muito pequena (<1%) do conjunto total será responsável por uma fração muito grande (metade) da carga do sistema.

Quanto menor for o valor de *a*, maior será a variabilidade da distribuição. Em (HARCHOL-BALTER, DOWNEY, 1997), é mostrado que esta distribuição está presente em vários ambientes de computação, incluindo ambientes de pesquisa e administrativos.

Distribuições como essa parecem se encaixar em vários das medidas de sistemas de computação. Como exemplo, temos:

- Requisitos de CPU de processos UNIX, medidos na UC Berkley: $a \sim 1$ (HARCHOL-BALTER, DOWNEY, 1997);
- Tamanho de arquivos transferidos pela internet: $1.1 \leq a \leq 1.3$ (CROVELLA, BESTAVROS, 1997)

Distribuições *heavy-tailed* possuem como propriedade uma variância infinita, o que não é possível acontecer em um sistema como o simulado, onde as tarefas possuem tempos máximos de execução. O nosso interesse é gerar um número de tarefas pequenas muito maior do que de tarefas grandes, de modo que uma função MOD (duração_gerada MOD duração_máxima) foi utilizada para distribuir de forma igualitária as durações que excedessem o valor máximo, no caso 10 segundos. A tabela 3, mostrada anteriormente, exibe as porcentagens das durações geradas dentro de cada intervalo, de acordo com o fator de variabilidade *a*. Note que quanto maior o valor de *a*, menor é a variabilidade – a maioria das durações geradas fica dentro do intervalo [0,1[.

5.2 Modelos de simulação

Geradores de carga de trabalho podem ser classificados com base em um modelo de sistema fechado, onde novas submissões de trabalho são disparadas ao término da execução de outros trabalhos (seguido de um "tempo para pensar"), ou em um modelo de sistema aberto, onde a chegada de novos trabalhos independe do término de tarefas anteriormente submetidas. As figuras 1 e 2 ilustram os dois tipos de modelos.

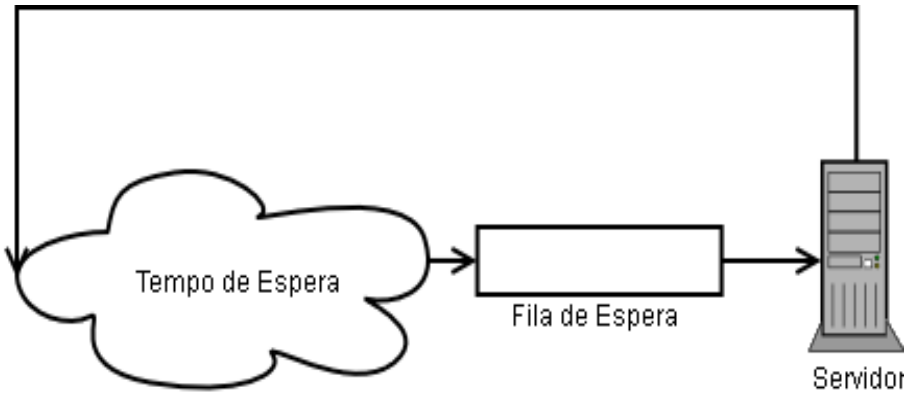

Figura 10: Modelo de sistema fechado. Número de clientes fixo com um "tempo para pensar" antes do envio da próxima requisição.

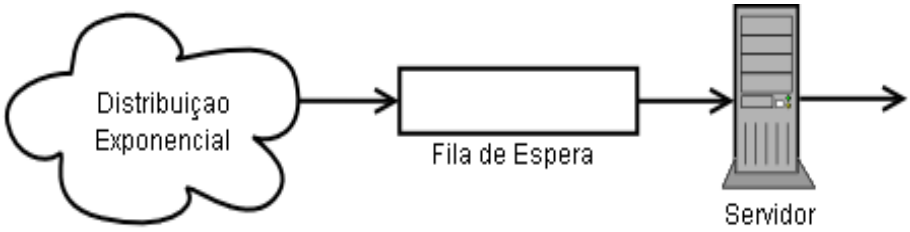

Figura 11: Modelo de sistema aberto. As requisições chegam ao sistema de acordo com alguma distitribuição estatística e, após serem processadas pelo servidor, deixam o sistema.

Em linhas gerais, os modelos de sistema fechado possuem um número fixo de clientes que enviam requisições para o sistema e esperam sua completa execução. Após a tarefa anterior ser executada, o cliente espera um tempo, conhecido como "tempo para pensar" (*think time,* em inglês) para submeter uma nova requisição. Por outro lado, modelos de sistema aberto não possuem um número fixo de clientes. As requisições chegam de acordo com alguma distribuição estatística, onde a mais comumente utilizada é a distribuição exponencial. Após a execução da tarefa, o cliente deixa o sistema.

Em (SCHROEDER, WIERMAN, HARCHOL-BALTER 2006), é mostrado que a escolha de diferentes modelos de geração da carga de trabalho para simulações possuem resultados completamente distintos, mesmo se os sistemas estiverem sob a mesma utilização e possuírem a mesma demanda de serviço. Tal estudo chega a conclusão que os tempos de resposta para sistemas fechados é, normalmente, menor do que para sistemas abertos e,

conforme o MPL (ver seção 2.2) aumenta, o desempenho de sistemas fechados se aproxima do apresentado pelos modelos que seguem um processo de Poisson, onde o tempo entre as chegadas de novos clientes é de acordo com uma distribuição exponencial.

5.3 Resultados

Os gráficos exibidos nas figuras 12 e 13 exibem resultados onde são consideradas todas as tarefas, independente da classe em que se encontram (curtas ou longas). Pode-se notar que o ORBITA possui um desempenho igual ou superior em relação aos outros algoritmos para quase todas as taxas de chegadas de requisições. Quando a variabilidade é maior (figura 12), o ganho com o uso da estratégia proposta, ORBITA, é ainda mais evidente, enquanto o algoritmo SITA-E possui um desempenho intermediário e LWR e TAGS continuam a apresentar as menores taxas de vazão.

O motivo para que o ORBITA e o SITA-E possuam uma taxa de vazão maior do que os seus concorrentes está na divisão das tarefas em classes e da reserva ser servidores específicos para cada uma delas. Com isso, as requisições curtas, que representam a maior parte da carga de trabalho, executam em um servidor dedicado, sendo elas as responsáveis pelo bom desempenho dos algoritmos. O ORBITA ainda possui uma atuação superior ao SITA-E, pois seu controle de admissão evita que as tarefas grandes ultrapassem o deadline.

De todos os algoritmos testados, o TAGS foi o que apresentou a menor taxa de tarefas concluídas dentro do tempo permitido, com um valor inferior ao dos demais concorrentes. Isto ocorre pelo fato de todas as tarefas serem sempre encaminhadas para o mesmo servidor. As tarefas grandes acabam sendo canceladas e reiniciadas no segundo servidor, mas o tempo de execução não é reiniciado, fazendo com que praticamente todas as tarefas grandes não consigam executar dentro da janela de tempo permitida. Pior ainda: as tarefas pequenas têm seus desempenhos prejudicados por serem executadas juntamente com as grandes. Para cenários onde a carga é muito alta, muitas dessas requisições curtas são mortas. Isto pode ser visto nas figuras 14 e 15, que mostra a taxa de tarefas que ultrapassam o *deadline*.

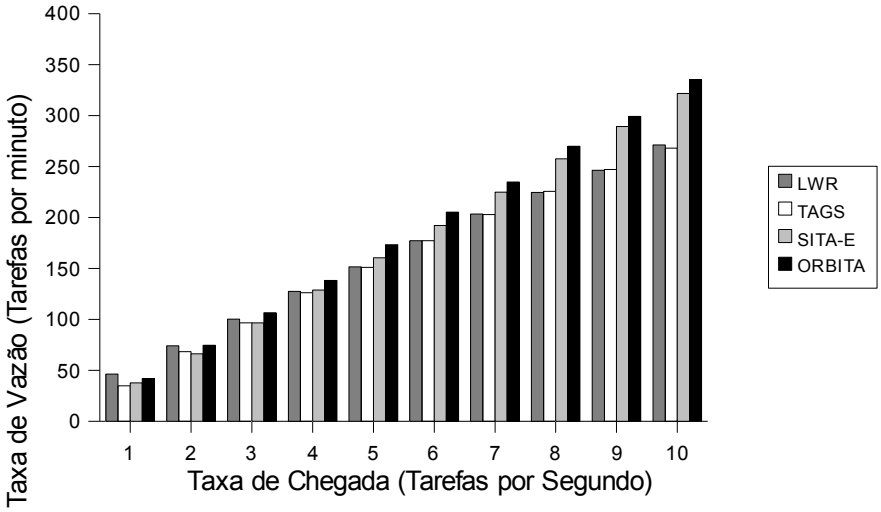

Figura 12: Taxa de vazão total com fator de variância 0.1.

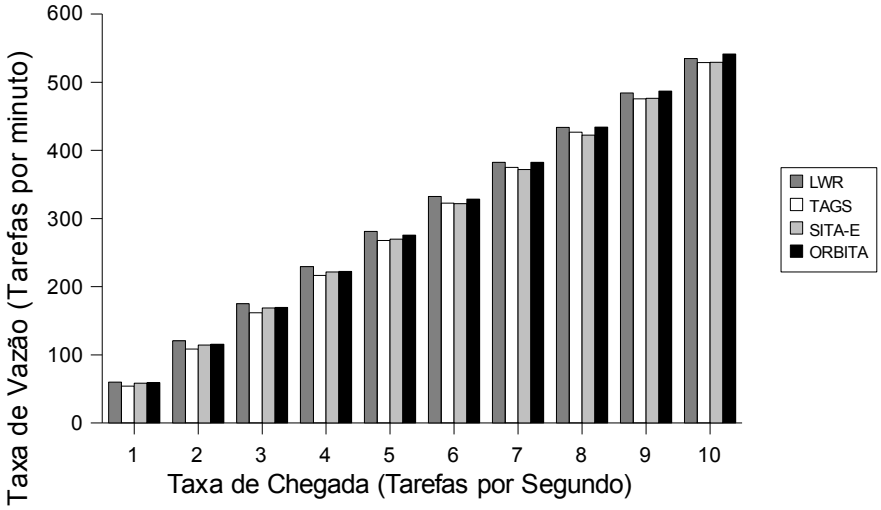

Figura 13: Taxa de vazão total com fator de variância 0.3.

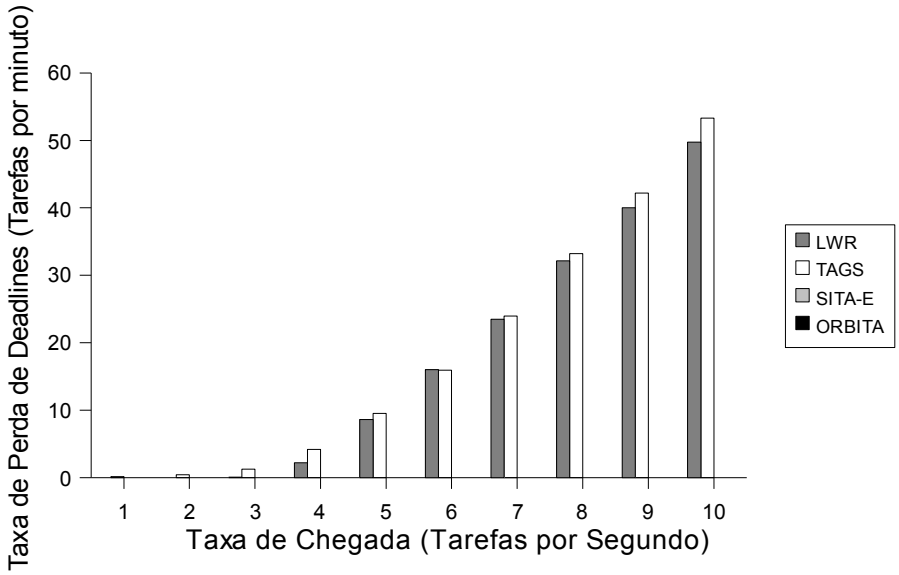

Figura 14: Taxa de perda de deadlines com fator de variância 0.1

O algoritmo LWR possui um desempenho similar ao ORBITA quando a taxa de chegadas de tarefas é baixa, notadamente abaixo de 3 tarefas por segundo. Contudo, para valores acima deste patamar, o algoritmo começa a sentir os efeitos de misturar tarefas grandes com tarefas curtas.

As figuras 16 e 17 mostram a taxa de vazão somente das tarefas pequenas para cada algoritmo. Tanto o SITA-E quanto o ORBITA possuem um servidor dedicado a atender somente este tipo de requisição, o que faz com que o número de deadlines quebrados seja praticamente inexistente. Por outro lado, nos algoritmos LWR e TAGS este número existe e cresce de acordo com a taxa de chegada de requisições. É importante notar nestes gráficos aqui exibidos que quanto maior a variabilidade, maior é o número de tarefas que ultrapassam o tempo limite. Isto ocorre pela maior presença de requisições longas, como pôde ser visto na tabela 3.

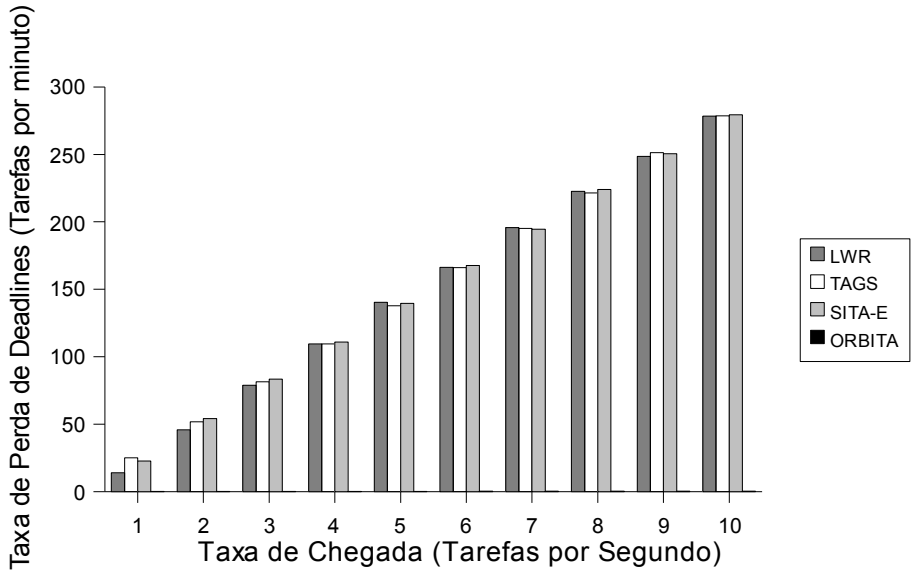

Figura 15: Taxa de perda de deadlines com fator de variância 0.3

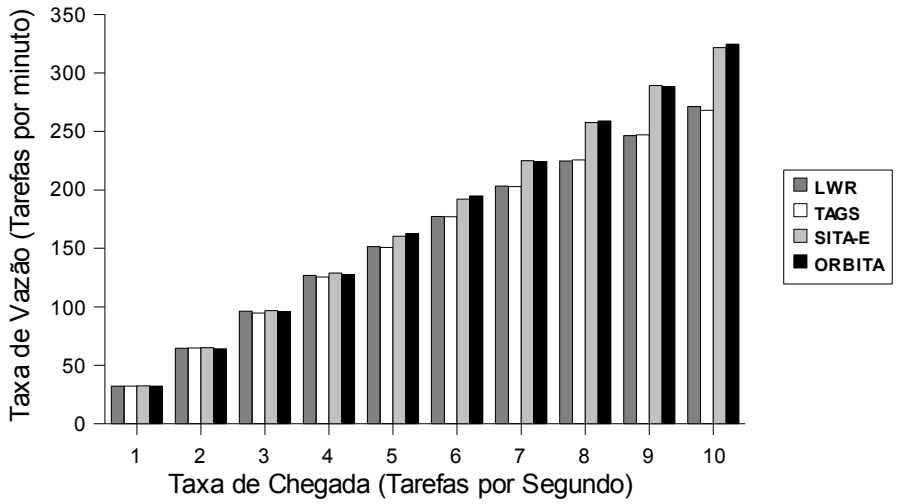

Figura 16: Taxa de vazão de tarefas pequenas e fator de variância 0.1.

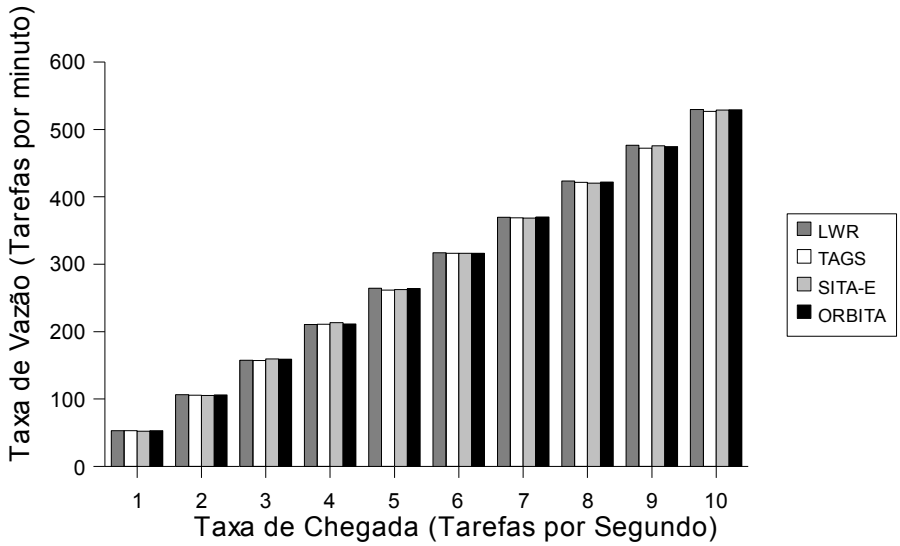

Figura 17: Taxa de vazão de tarefas pequenas e fator de variância 0.3

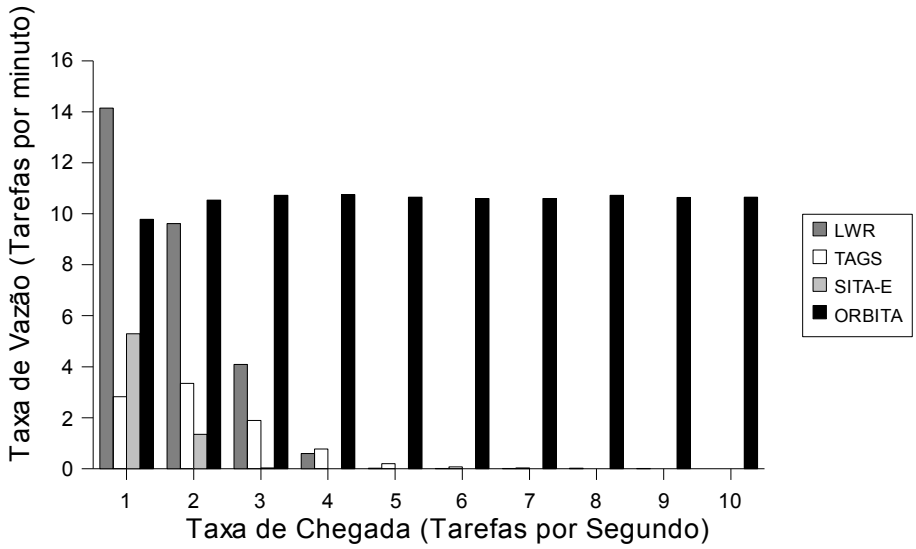

Figura 18: Taxa de vazão de tarefas grandes e fator de variância 0.1

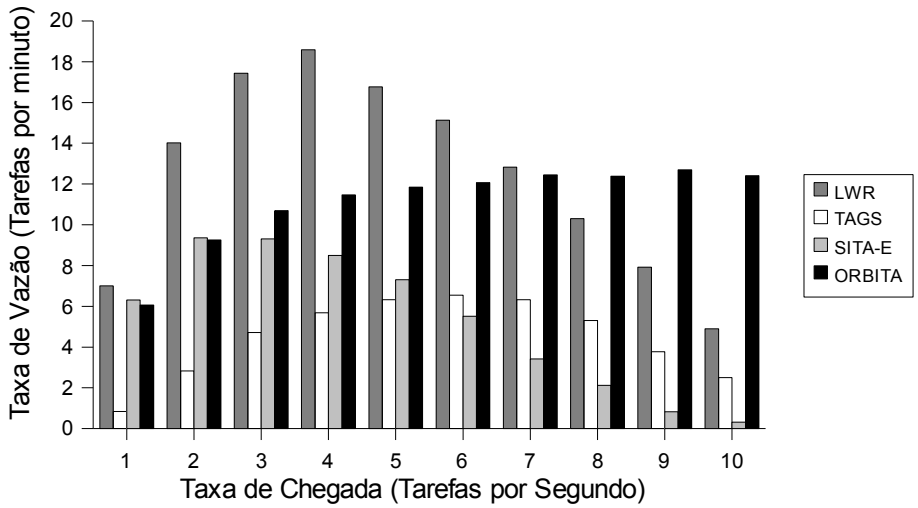

Figura 19: Taxa de vazão de tarefas grandes e fator de variância 0.3

As figuras 18 e 19 exibem o desempenho dos algoritmos para as tarefas longas e, aqui, pode ser notada a grande superioridade do ORBITA em relação aos seus concorrentes, principalmente para cenários com alta variabilidade entre as durações das tarefas. O ORBITA se mostra um algoritmo robusto, capaz de atender tarefas grandes mesmo sob altas taxas de chegadas de requisições, enquanto as outras alternativas têm suas taxas de vazão comprometidas.

A figura 19 mostra que o LWR supera o ORBITA no intervalo compreendido entre 1 e 7 requisições por minuto. Porém, se forem analisados os histogramas das tarefas que terminaram dentro do deadline de cada um destes dois algoritmos (tabelas 4 e 5 a seguir), pode-se notar que o ORBITA é mais igualitário, uma vez que requisições de todas as durações são executadas – enquanto no LWR, para altas taxas de chegada, apenas as tarefas menores não ultrapassam o deadline.

Tabela 4: Histograma da taxa de vazão (em tpm) do LWR com fator de variância 0.3.

Tx	[0, 1[	[1, 2[	[2, 3[	[3, 4[	[4, 5[	[5, 6[	[6, 7[	[7, 8[	[8, 9[	[9, 10[	10
1	52,85	1,77	1,09	0,78	0,65	0,5	0,57	0,45	0,41	0,43	0,36
2	106,56	3,62	2,16	1,66	1,39	1,2	0,98	0,97	0,82	0,63	0,58
3	157,52	5,16	3,08	2,54	1,91	1,34	1,15	0,78	0,69	0,46	0,33
4	210,77	7,31	4,4	2,86	1,58	0,96	0,57	0,36	0,25	0,16	0,13
5	264,27	9,19	4,51	1,74	0,7	0,27	0,17	0,09	0,05	0,02	0,02
6	317,21	10,58	3,47	0,79	0,19	0,06	0,02	0,02	0	0	0
7	369,7	11,01	1,63	0,16	0,02	0,01	0	0	0	0	0
8	423,7	9,71	0,56	0,03	0	0	0	0	0	0	0
9	476,46	7,76	0,16	0	0	0	0	0	0	0	0

Tabela 5 Histograma da taxa de vazão (em tpm) do ORBITA com fator de variância 0.3.

Tx	[0, 1[	[1, 2[	[2, 3[	[3, 4[	[4, 5[	[5, 6[	[6, 7[	[7, 8[	[8, 9[	[9, 10[	10
1	53,06	1,43	0,91	0,69	0,53	0,5	0,44	0,41	0,41	0,37	0,37
2	106,07	2,31	1,29	1,11	0,87	0,81	0,64	0,65	0,57	0,55	0,45
3	159,01	2,6	1,38	1,23	1,03	0,97	0,82	0,71	0,68	0,68	0,59
4	211,17	2,92	1,73	1,2	1,05	0,95	0,85	0,8	0,68	0,64	0,65
5	263,86	2,9	1,85	1,24	1,02	1,08	0,86	0,72	0,79	0,65	0,73
6	316,49	3,02	1,75	1,3	1,12	0,94	0,94	0,9	0,71	0,67	0,72
7	370,23	3,21	1,98	1,35	1,11	0,97	0,88	0,75	0,76	0,73	0,71
8	422,01	3,08	1,75	1,47	1,07	1,05	0,9	0,87	0,81	0,7	0,68
9	474,35	3,27	1,98	1,28	1,24	1,08	0,9	0,83	0,74	0,7	0,68

5.4 Discussão

Neste capítulo, quatro algoritmos de balanceamento de carga foram analisados: TAGS, LWR, SITA-E e ORBITA. Foi mostrado que a variabilidade das tarefas que compõem a carga de trabalho é um fator determinante para o desempenho de um sistema e que, para minimizar a perda de deadlines, uma política de balanceamento de carga deve ser capaz de classificar as requisições que chegam de acordo com suas durações. Foi mostrado, ainda, que os algoritmos que não realizam essa classificação (TAGS e LWR) possuem os piores desempenhos quando em cenários que apresentam variabilidade e taxa de chegadas de requisições altas. Isto ocorre pelo fato de elas tratarem da mesma maneira tarefas grandes e pequenas, fazendo com que as requisições curtas, ao terem de compartilhar o processador com outras tarefas muito longas, fiquem em execução por um tempo maior do que o permitido, sendo, assim, canceladas.

Já os algoritmos que realizam essa diferenciação (SITA-E e ORBITA) possuem um desempenho superior no cenário citado, pois as tarefas pequenas conseguem sempre executar

dentro do tempo limite. Para esses, a preocupação passa a ser com a execução das grandes, as quais o número de executores simultâneas deve ser restrito. O algoritmo SITA-E, que não realiza controle de admissão para as tarefas grandes, possui uma taxa de quebra de deadlines bem maior do que a apresentada pelo ORBITA, que realiza esse procedimento. Na verdade, o ORBITA possui um mecanismo de controle de admissão otimizado, pois somente aceita uma requisição se houver um servidor destinado às tarefas grandes capaz de atendê-la, sem que cause quebras de deadlines.

No capítulo seguinte serão apresentados os resultados obtidos através da utilização do Midas com o TPC-C. É importante salientar que o Midas utilizou a mesma arquitetura proposta neste capítulo, com dois servidores homogêneos. Isto para que seja possível atestar a veracidade dos dados mostrados neste capítulo.

6 Configurações das experiências

Conforme mencionado, a ferramenta TPC-C foi utilizada para gerar as transações. Contudo, o consórcio TPC fornece apenas uma especificação, não uma implementação. Desta forma, foi utilizada uma versão gratuita desta especificação, a jTPCC, disponível em http://jtpcc.sourceforge.net. O TPC-C foi utilizado em sua versão original, que implementa um modelo fechado, e em uma versão levemente modificada, onde as transações são geradas de forma a seguir uma distribuição exponencial. Note que essa última versão se aproxima mais do modelo utilizado na simulação apresentada no capítulo anterior e, segundo (SCHROEDER, WIERMAN, HARCHOL-BALTER 2006) é um modelo que se aproxima mais de um sistema real baseado em internet.

6.1 TPC-C

Aprovado em Julho de 1992, o TPC Benchmark C é uma ferramenta de benchmark para processamento de transações online (OLTP). O TPC-C é composto por um total de 5 transações diferentes, que são executadas concorrentemente por clientes emulados. O esquema relacional proposto pelo TPC-C é composto de 9 tabelas, com diferentes números de registros. A principal métrica da especificação é o número de transações executadas até o fim por minuto (*tpm*).

O TPC-C simula um ambiente computacional fechado completo, onde uma população de usuários executam transações em um banco de dados. A ferramenta é centrada nas atividades principais (transações) de um ambiente de entrada de pedidos. Tais transações são:

- Entrada de um novo pedido;
- Entrega de um pedido;
- Registro de pagamentos;
- Status de um determinado pedido; e
- Consulta ao estoque;

No modelo de negócios do TPC-C, um fornecedor (também chamado de empresa) opera um número de armazéns e seus respectivos distritos de venda. O TPC-C foi projetado para simular uma expansão da empresa e existe a possibilidade de se criar novos armazéns. Entretanto, certos requisitos de consistência devem ser mantidos conforme o modelo é

incrementado: cada armazém é responsável por 10 distritos de venda e cada distrito serve 3000 clientes. Um operador de um distrito de vendas pode selecionar, a qualquer momento, uma das 5 operações (ou transações) oferecidas pelo sistema.

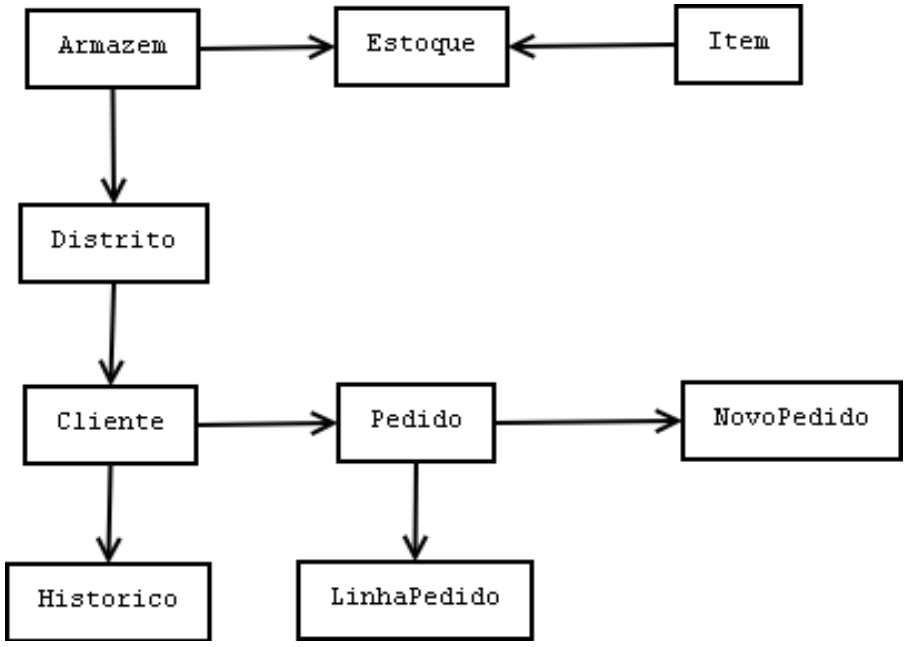

Figura 20: Diagrama entidade-relacionamento simplificado das tabelas utilizadas pelo TPC-C

A transação mais freqüente (em torno de 45% do total de transações geradas) consiste na entrada de um novo pedido que, em média, é composto de 10 itens diferentes. Cada armazém tenta manter o estoque para os 100.000 itens de catálogo da empresa e preencher os pedidos de acordo com a disponibilidade. Entretanto, na realidade, um armazém não possuirá todas os itens necessários para preencher todos os pedidos. Então, o TPC-C requisita que aproximadamente 10% de todos os pedidos sejam complementados por outros armazéns. A segunda transação mais freqüente da especificação consiste no registro de pagamento recebido de um cliente, acontecendo 43% das vezes. As outras três transações acontecem com a mesma frequência:4%.

6.2 Ambiente de testes

Conforme mencionado anteriormente, a arquitetura da aplicação foi constituída por 2 servidores homogêneos, cada um com a seguinte configuração:

- Pentium II MMX com 400 MHz e 256 MB de RAM;

- Disco rígido IDE de 60 GB;
- Distribuição Debian do sistema operacional Linux (Kernel 2.6);
- Sistema de arquivos ext3;
- Servidor de banco de dados PostgreSQL 8.1;
- Banco de dados utilizado pela ferramenta TPC-C, criado com 10 armazéns e possuindo 1.11 GB;

A utilização de servidores com baixa capacidade de processamento foi necessária para que fosse possível causar degradação de desempenho com um número de clientes relativamente baixo. A configuração do computador utilizado para simular os clientes segue abaixo:

- Pentium IV com 3.2 GHz e 2GB de RAM;
- Distribuição Debian do sistema operacional Linux (Kernel 2.6);
- Máquina virtual Java 5.0, disponibilizada pela Sun Microsystems;
- Ferramenta de benchmark jTPCC, utilizando a ferramenta de QoS Midas;

Cada cliente corresponde a uma *Thread* Java, enviando uma ou mais transações para sistema, de acordo com o modelo adotado (aberto ou fechado). Todos os computadores estavam conectados por uma rede Ethernet full-duplex de velocidade de transmissão 100Mbits/s.

6.2.1 Medidas das durações de cada transação

Conforme descrito na seção 6.1, o TPC-C possui 5 tipos diferentes de transações. Como o ORBITA precisa conhecer a duração de cada tipo de transação para realizar a etapa de diferenciação de serviços, foi necessário obter uma estimativa da duração de cada transação. Assim, foi realizada uma etapa preliminar às experiências, onde cada transação foi executada sozinha no banco de dados por 150 vezes. Seus tempos de resposta foram medidos e uma média foi calculada em cima desses valores, tendo sido descartados os 5 valores mais altos e os 5 valores mais baixos. A tabela a seguir mostra os valores das médias do tempo de resposta de cada transação, bem como suas classificações em transações curtas ou longas.

Tabela 6: Média das durações de cada transação e suas respectivas classificações.

	Média de duração (ms)	***Classificação***
Novo Pedido	320.11	Longa
Entrega de Pedido	497.76	Longa
Pagamento	123.48	Longa
Consulta ao estoque	70.35	Curta
Status do pedido	115.06	Curta

6.3 Composição da carga de trabalho

A configuração padrão do TPC-C possui a seguinte composição de carga de trabalho (doravante denominada *composição padrão*):

- 45% das transações geradas são do tipo *Novo Pedido*;
- 43% das transações geradas são do tipo *Pagamento*; e
- 4% para cada tipo de transação restante.

Estes valores, obtidos pelo consórcio através de medidas retiradas de sistemas reais, são incompatíveis com a composição da carga de trabalho apresentada na seção 3.2, onde o número das transações rápidas é predominante.

Desta forma, o trabalho analisará, ainda, um outro tipo de composição de carga de trabalho: *composição heavy-tailed*, que apresenta a seguinte porcentagem

- 5% das transações geradas são do tipo *Entrega de pedido*;
- 95% das transações geradas são do tipo *Consulta ao estoque*; e
- 0% para as demais transações.

Esta composição de transações, apesar de possuir um fraco valor semântico, serve perfeitamente como ilustração para a robustez do ORBITA perante as demais estratégias de balanceamento de carga.

6.4 Outras configurações

As configurações restantes aparecem na tabela 7.

Tabela 7: Demais configurações das experiências realizadas.

Parâmetro	***Valor***
Taxa de chegadas de transações para o modelo aberto	10 transações por segundo
"Tempo para pensar" para o modelo fechado	Distribuição exponencial com média 8 segundos e valor máximo permitido de 80 segundos.
Número de clientes para o modelo fechado	100
Tempo de cada rodada de simulação	20 minutos
Tempo descartado no início da simulação (fase transiente)	5 minutos
Deadline	5 segundos

Cada rodada de simulação foi executada em um banco de dados recém-criado, com o sistema completamente vazio. Desta forma, o banco de dados possuía o mesmo estado para todas as experiências executadas.

6.5 Resultados

Os resultados obtidos com as experiências corroboram as constatações feitas no capítulo 3, onde o ORBITA aparecia como a melhor política de balanceamento de carga para evitar perda de deadlines.

6.5.1 Modelo aberto

Composição padrão da carga de trabalho

Primeiramente, iremos analisar o desempenho dos algoritmos em um modelo aberto, onde as transações chegam segundo uma distribuição exponencial. A utilização deste modelo ficou um pouco prejudicada, pois o custo para simular altas taxas de chegadas através da criação e destruição *threads* é muito alto e mesmo a utilização de um *pool de threads* não se mostrou suficiente para simular a carga especificada com precisão, pois a taxa de chegadas de novas requisições foi sempre menor do que a definida no início de cada experiência. Isto

ocorre, provavelmente, devido ao tempo gasto com a troca de contexto e tratamento de interrupções causados pela contínua chegada de novas requisições. Com isso, não foi possível atingir a média especificada para taxas de chegadas acima de 14. De toda maneira, os resultados mostram de forma clara como os algoritmos se comportam dentro de cenários de alta utilização.

Em princípio, o que mais chama a atenção na figura 21 é o desempenho do algoritmo LWR: nenhuma transação grande consegue terminar sua execução dentro do tempo máximo permitido, mesmo quando a taxa de chegadas é baixa. Como as transações longas possuem comandos de modificação de dados (cláusulas SQL dos tipos *insert*, *update*, e *delete*), provavelmente este desempenho ruim ocorre por causa da existência de trancas que cada uma das transações requer para manter o nível de isolamento. Esta foi uma característica não levada em consideração no modelo de simulação, uma vez que é uma característica própria de ambientes de bancos de dados paralelos ou distribuídos.

A aceitação das transações grandes no sistema de forma não-controlada prejudica, ainda, o desempenho das transações pequenas, como pode ser constatado na figura 22. Isto ocorre pois o algoritmo LWR mistura, dentro de um mesmo servidor, transações das duas classes. Desta forma, as trancas obtidas pelos comandos de modificação de dados interferem diretamente na execução das transações curtas.

Por outro lado, os algoritmos SITA-E e ORBITA possuem taxas de vazão superiores. Isto ocorre, principalmente, pela divisão das transações. Nas transações grandes, o ORBITA possui um desempenho superior causado pelo controle do número de transações grandes. De fato, conforme a taxa de chegadas aumenta, a taxa de vazão do ORBITA da figura 21 se mantém praticamente inalterada. Nesta mesma figura pode-se observar que o desempenho do SITA-E começa a degradar no caso de taxas de chegadas elevadas. Em comparação com o resultado da simulação (figura 19), a queda de desempenho do SITA-E é menos acentuada, pois a transação mais longa (Entrega de pedido) representa apenas 4% da carga de trabalho total.

Em relação às transações pequenas, ORBITA e SITA-E possuem taxas de vazão similares, pois possuem exatamente a mesma política de tratamento para tal classe: executam em um servidor dedicado.

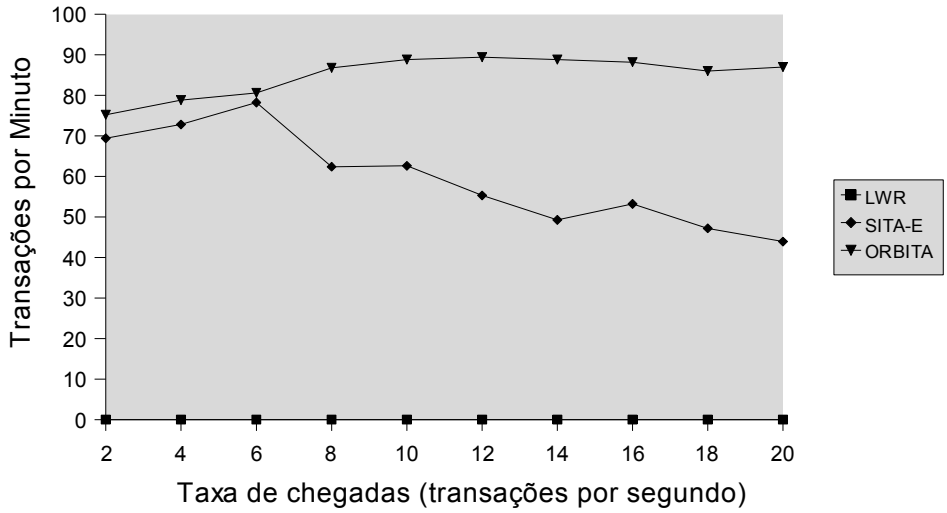

Figura 21: Taxa de vazão das tarefas grandes quando submetidas à carga de trabalho padrão

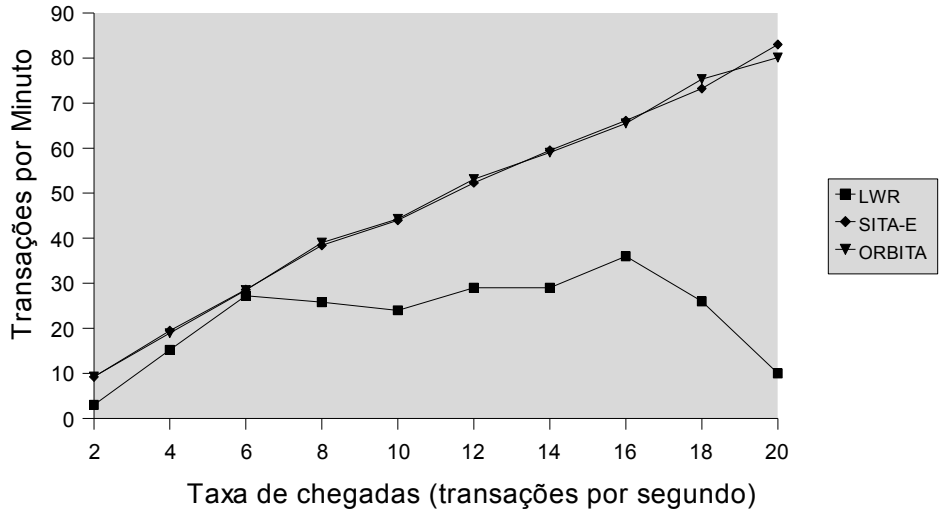

Figura 22: Taxa de vazão das tarefas pequenas quando submetidas à carga de trabalho padrão

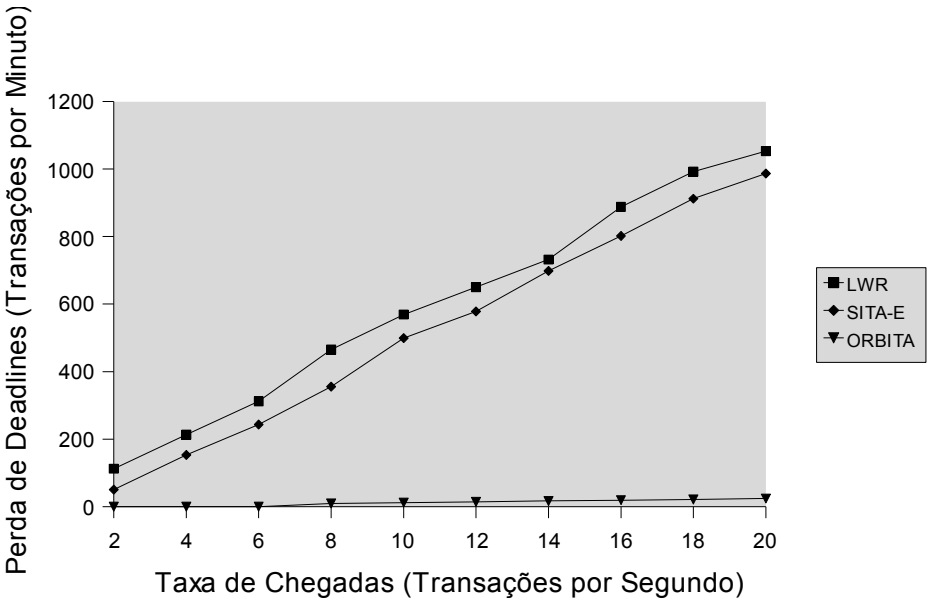

Figura 23: Taxas de perda de deadlines dos algoritmos para carga de trabalho padrão.

Composição heavy-tailed da carga de trabalho

Na composição *heavy-tailed*, temos resultados semelhantes com os que foram obtidos na etapa de simulação. Mais uma vez, pode-se observar que o algoritmo LWR possui um desempenho muito ruim, não conseguindo terminar nenhuma transação longa dentro do deadline, mesmo que estas representem apenas 5% da carga de trabalho total. Isto ocorre devido à própria natureza da distribuição *heavy-tailed*, onde a variabilidade das durações das transações é muito alta. Isto faz com que a presença de poucas tarefas longas no sistema seja capaz de saturá-lo por completo, impedindo que as transações consigam terminar suas execuções dentro do tempo limite. Quando analisamos o desempenho do LWR em relação às transações pequenas, o resultado é ainda pior do que no caso da carga de trabalho padrão: para elevadas taxas de chegada (a partir de 14) a taxa de vazão dentro do deadline é igual a 0. Isto ocorre, novamente, pelo fato de o LWR misturar transações dos dois tipos (grandes e pequenas) dentro dos mesmos servidores. Neste caso, a interferência das transações grandes é ainda maior do que no caso anterior, visto que a transação do tipo "Entrega de pedido" é a que realiza o maior número de trancas e, conseqüentemente, a que demora mais.

Os algoritmos SITA-E e ORBITA, mais uma vez, despendem o mesmo tratamento para as transações pequenas e, por isso, apresentam desempenhos similares. Quando analisamos as taxas de vazão das tarefas grandes, mais uma vez vemos que o ORBITA se

mostra muito mais robusto do que SITA-E, devido ao seu mecanismo de otimização do número de transações concorrentes. Enquanto no SITA-E, nenhuma transação grande consegue terminar dentro do deadline a partir de 8 chegadas por segundo, o ORBITA consegue manter a taxa de vazão destas tarefas de forma quase constante.

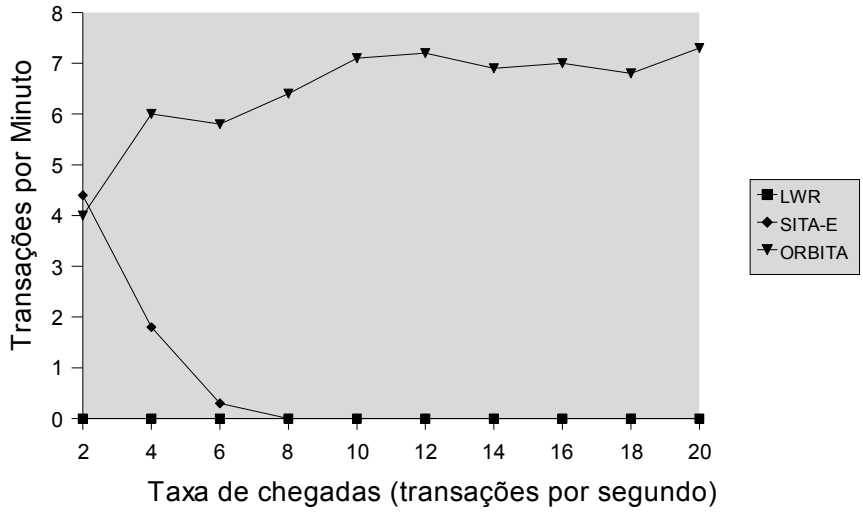

Figura 24: Taxa de vazão das tarefas grandes quando submetidas à carga de trabalho heavy-tailed

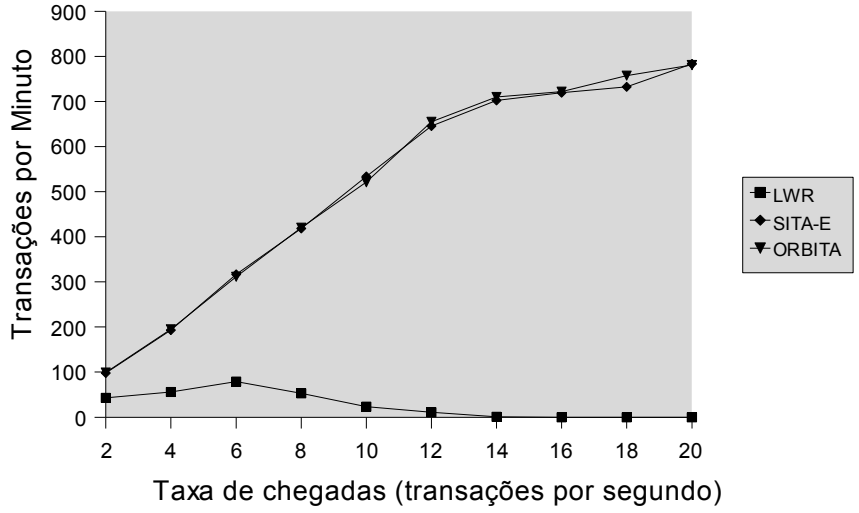

Figura 25: Taxa de vazão das tarefas pequenas quando submetidas à carga de trabalho heavy-tailed

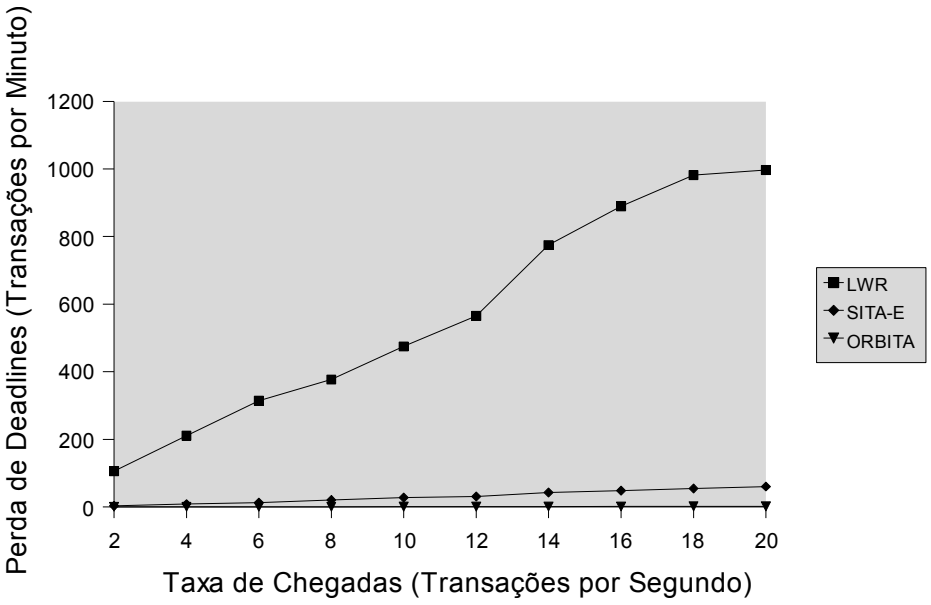

Figura 26: Taxas de perda de deadlines dos algoritmos para carga de trabalho heavy-tailed.

Um último gráfico, que complementa as informações contidas nas figuras anteriores, deve ser analisado, para que seja possível entender a natureza da robustez do ORBITA. Para ser possível manter a taxa de vazão das transações grandes, o ORBITA é obrigado a descartar as transações que causariam degradação do desempenho. O Midas, em particular, lança uma exceção do tipo *QoSException*, indicando que a transação foi rejeitada. É importante salientar que este tipo de controle já é feito pelos fabricantes de SGBDs, onde existe um parâmetro indicando o número máximo de conexões abertas possível. A diferença entre as abordagens consiste no fato de que, no caso do ORBITA, este número é calculado de forma inteligente, de acordo com os tipos de transações que estão executando e da transação que acabou de chegar, enquanto que na implementação nativa, este número é fixo e não leva em consideração a heterogeneidade da carga de trabalho.

Analisando as linhas da figura 27, podemos ver que a carga de trabalho padrão possui uma taxa de rejeição maior do que a correspondente na composição *heavy-tailed*. Isto ocorre pelo fato de a carga de trabalho padrão ser composta por 92% de transações longas, contra apenas 5% da *heavy-tailed*.

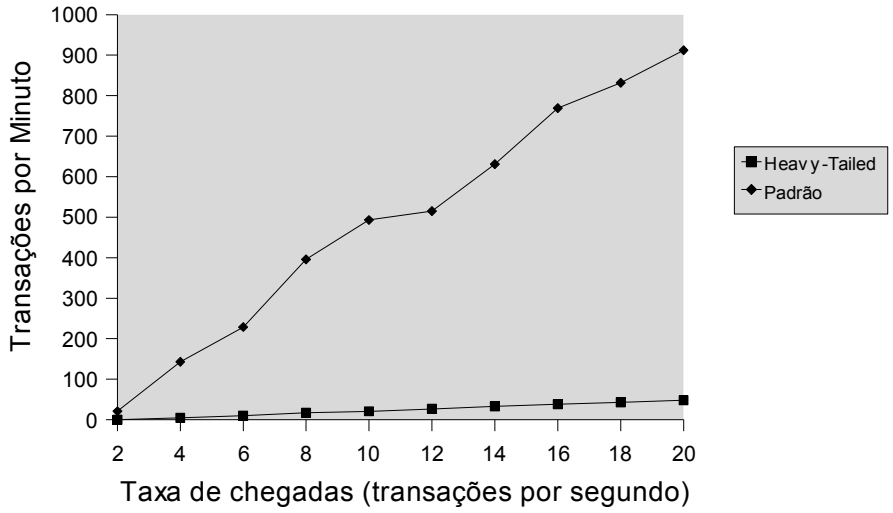

Figura 27: Taxa de transações rejeitadas pelo ORBITA para o modelo aberto.

6.5.2 Modelo fechado

Como uma forma de analisarmos de forma completa o desempenho da implementação do ORBITA no Midas, a ferramenta foi testada com o modelo fechado, padrão da especificação TPC-C. Foram simulados 100 clientes, que enviam comandos SQL para banco de dados concorrentemente. Após finalizar cada transação (o envio de um comando *commit* ou *rollback*), o cliente fica inativo por um tempo aleatório, segundo uma distribuição exponencial com média 8 segundos e máximo 80 (valores da especificação do TPC-C). Mais uma vez, foram utilizadas as cargas de trabalho de composição padrão e *heavy-tailed*.

Composição padrão da carga de trabalho

O algoritmo LWR continua tendo um desempenho muito ruim, onde todas as transações grandes não conseguem terminar suas execuções dentro do deadline. Mais uma vez, isto ocorre pelo fato de algoritmo LWR não realizar distinção entre tarefas grandes e pequenas, utilizando apenas o número execuções pendentes em cada servidor. Entretanto, as transações pequenas conseguem terminar, mesmo em número muito reduzido – mesmo neste cenário, onde as transações pequenas acontecendo apenas 8% das vezes, a taxa de vazão é muito pequena quando comparada com os outros algoritmos.

As políticas SITA-E e ORBITA, por outro lado, possuem desempenhos superiores. O

SITA-E, por não realizar controle de admissão sobre as tarefas grandes, tem uma taxa de vazão (dentro do deadline) deste tipo de transações bem menor do que o ORBITA, que realiza esta etapa. Para as transações pequenas, os resultados destas duas políticas são basicamente iguais, uma vez que tratam da mesma forma este tipo de tarefa.

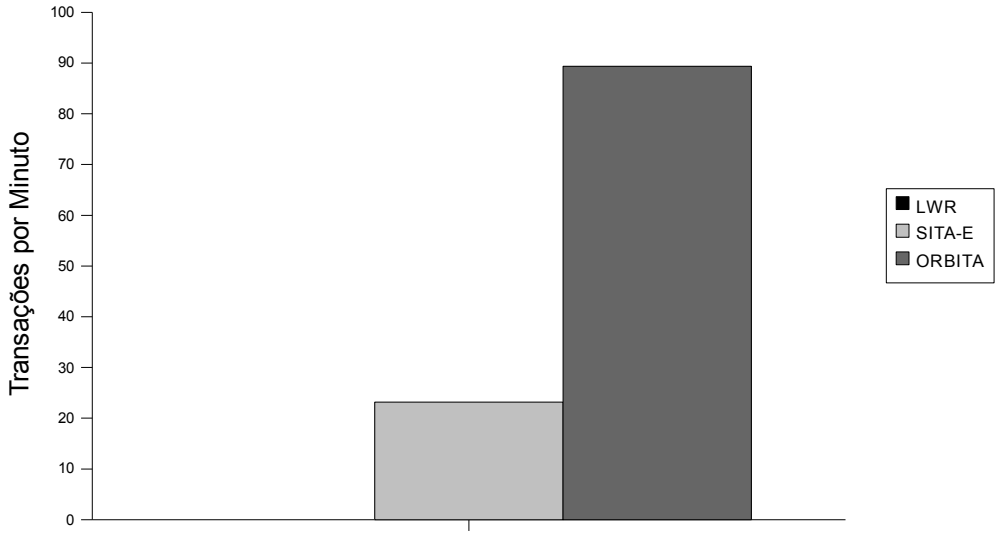

Figura 28: Taxa de vazão das transações grandes quando submetidas à carga de trabalho padrão.

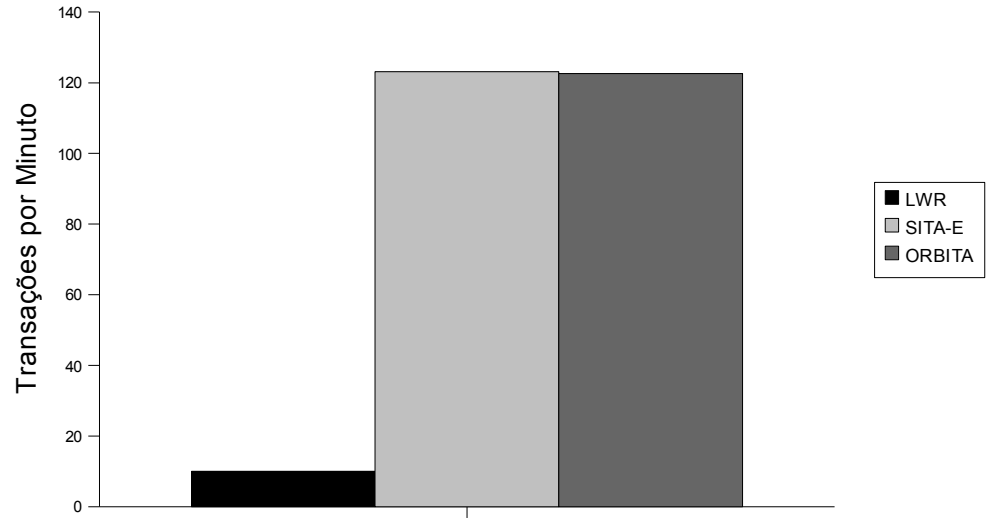

Figura 29: Taxa de vazão das transações pequenas quando submetidas à carga de trabalho padrão.

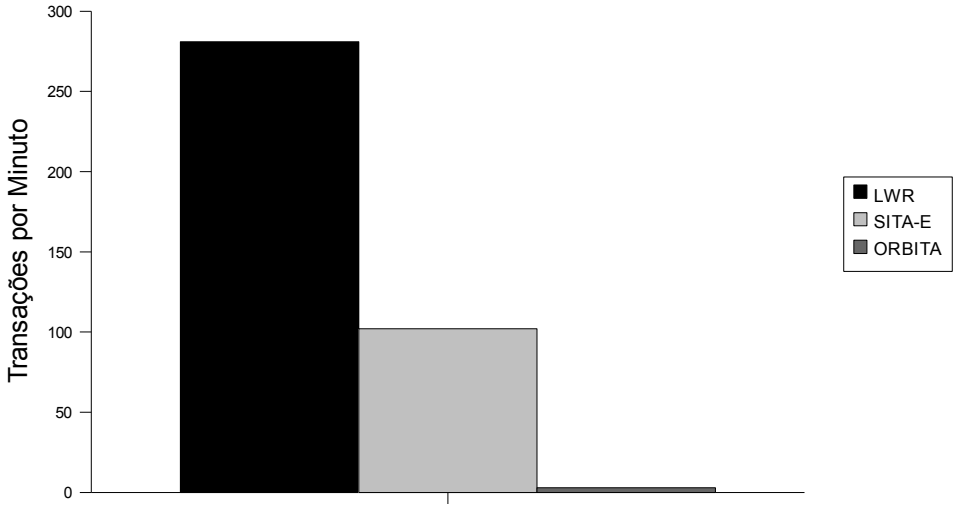

Figura 30: Taxa de perda de deadlines das transações pequenas quando submetidas à carga de trabalho padrão.

Composição heavy-tailed da carga de trabalho

Nesta última análise, podemos observar como o ORBITA mantém basicamente o mesmo desempenho quando submetido a modelos fechados ou a modelos abertos. O desempenho deste é praticamente o mesmo para cenários do modelo aberto onde a taxa de chegadas de transações novas é muito grande. O algoritmo SITA-E possui uma taxa de vazão muito ruim para as transações grandes, o que leva a crer que o número de requisições simultâneas foi alto. Comparando os desempenhos do algoritmo SITA-E da figura 31 com o da figura 24, pode-se ver que a taxa de vazão é bem menor no modelo fechado do que na média do modelo aberto. A explicação para isso está no fato de o número de clientes ser alto (100) e o tempo entre os envios de duas transações ser padrão. Para diminuir a carga no servidor, deve-se utilizar menos clientes ou aumentar o "tempo para pensar".

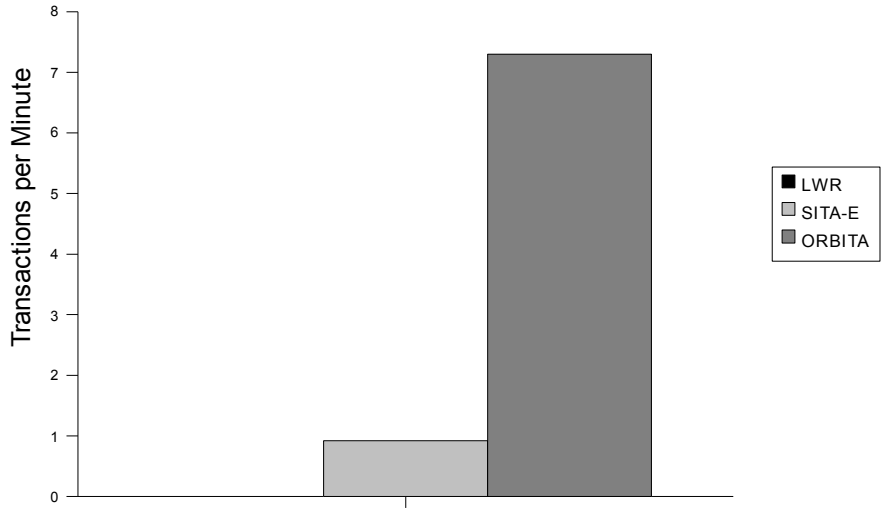

Figura 31: Taxa de vazão das transações grandes quando submetidas à carga de trabalho heavy-tailed.

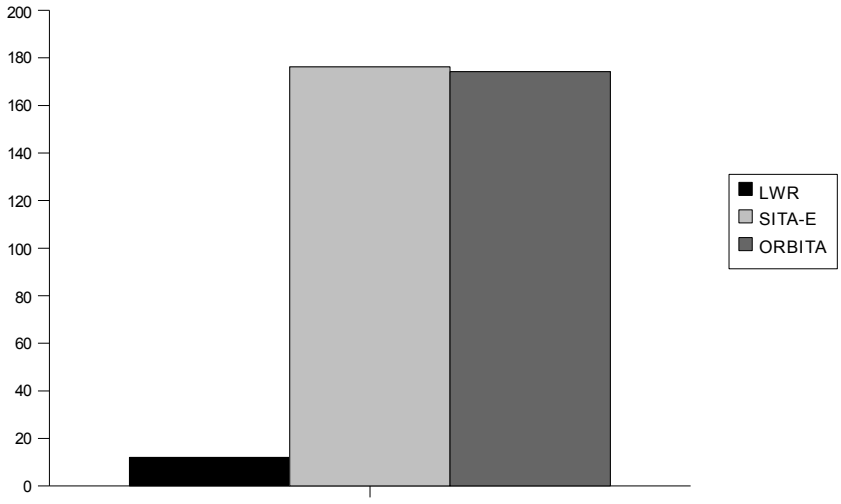

Figura 32: Taxa de vazão das transações grandes quando submetidas à carga de trabalho heavy-tailed.

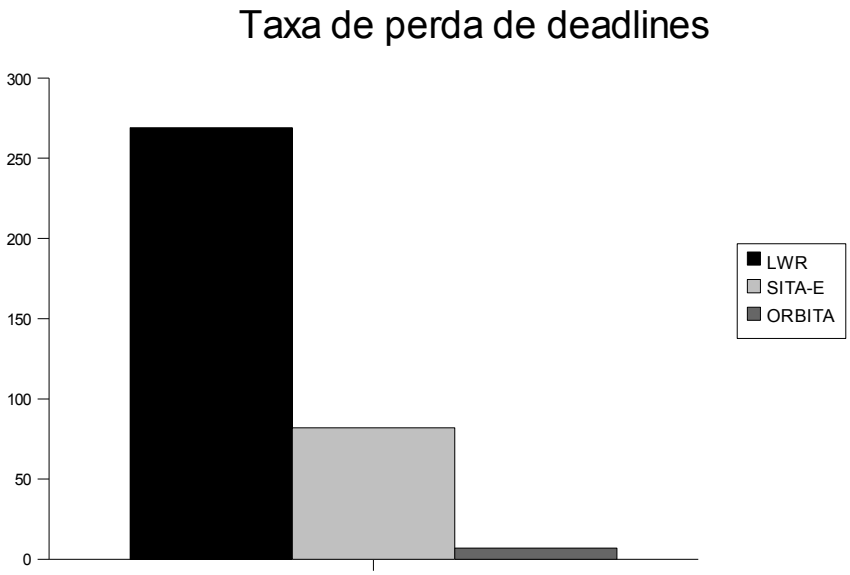

Figura 33: Taxa de perda de deadlines das transações quando submetidas à carga de trabalho heavy-tailed.

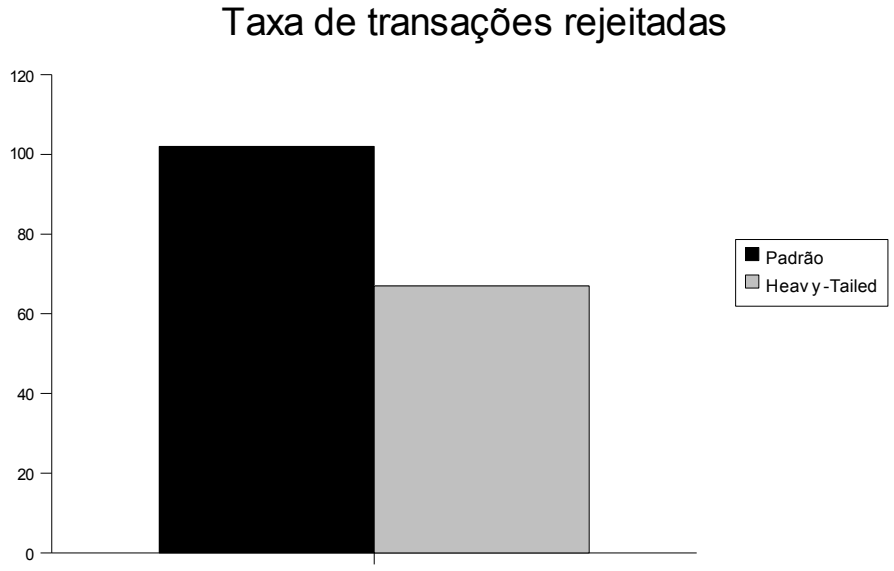

Figura 34: Taxa de transações rejeitadas pelo ORBITA para o modelo fechado.

6.6 Discussão

Neste capítulo foi apresentado o Midas, uma ferramenta que implementa técnicas de qualidade de serviço para bancos de dados relacionais. Dentre as funcionalidades que a ferramenta possui estão:

- Propagação de alterações dos dados, utilizando a técnica de replicação de cópia primária;
- Controle de admissão, limitando o número de clientes simultâneos;
- Diferenciação de serviços, onde a ferramenta é capaz de classificar uma transação que acabou de chegar no sistema em "longa" ou "curta"; e
- Balanceamento de carga, possuindo os algoritmos *Least-Work-Remaining* (LWR), *Size Interval Task Assignment with Equal Load* (SITA-E) e *On-Demand Restriction for Big Tasks* (ORBITA).

Com a utilização da ferramenta de benchmark jTPC-C, foi possível comparar o desempenho dos algoritmos de balanceamento de carga em um banco de dados paralelo com diferentes configurações de cargas de trabalho. As primeiras experiências foram feitas utilizando um modelo de chegadas de requisições aberto, obedecendo uma distribuição exponencial. Logo após, foi utilizado um modelo fechado, onde o número de clientes é fixo e varia-se apenas o tempo entre duas requisições consecutivas feitas por um mesmo cliente. As composições das cargas de trabalho também foram duas: a composição padrão da especificação TPC-C e uma configuração Heavy-Tailed, onde o número de transações muito curtas era muito maior do que o número de transações longas.

Os resultados mostraram que o algoritmo LWR, apesar de sua natureza dinâmica, possui um péssimo desempenho, principalmente quando submetido a uma carga de trabalho heavy-tailed. Os motivos para isso são dois:

1) mistura de tipos de transações nos servidores, onde as transações longas interferem nas execuções das tarefas curtas. Isto é acentuado em um ambiente como o de um banco de dados, devido à presença de trancas solicitadas pelas operações de modificação de dados;e
2) falta de um controle de admissão na entrada das transações longas.

Chega-se à conclusão de que a afirmação 1 é verdadeira pois, se os gráficos de desempenho dos algoritmos LWR e SITA-E forem comparados, pode-se ver que o desempenho deste último apresenta um valor muito superior ao primeiro. A principal diferença entre estes dois algoritmos está no fato de que o SITA-E possui um mecanismo que realiza uma divisão das transações, classificando-as em longas ou curtas.

Relativamente à afirmação 2, pode-se chegar à conclusão que ela também é verdadeira

quando compara-se os desempenhos do SITA-E e do ORBITA. Este último algoritmo, além de realizar a mesma etapa de classificação das tarefas que o SITA-E faz, aplica um mecanismo de controle de admissão otimizado para as tarefas grandes, achando o nível de multi-programação (MPL) de forma dinâmica. Desta forma, a taxa de vazão das tarefas grandes é superior no ORBITA. As tarefas pequenas, mesmo ocorrendo de forma abundante, não constituem uma classe que represente preocupação, uma vez que nenhuma das tarefas pequenas terminou a execução após o *deadline*.

7 Conclusão e trabalhos futuros

Em sistemas com requisitos de qualidade de serviço, pode ser desejável que as requisições submetidas tenham um tempo máximo de execução. Para isso, o sistema deve possuir, dentro de sua arquitetura, preocupações com estas restrições temporais, visando sempre minimizar a perda desses deadlines.

Este trabalho apresentou um algoritmo de balanceamento de carga, chamado ORBITA, que tenta minimizar o número de tarefas perdidas, aplicando um mecanismo de diferenciação de serviços adicionado a um controle de admissão. O principal princípio do algoritmo, para obter uma taxa de vazão superior a dos seus concorrentes, é a classificação das requisições em tarefas grandes e pequenas, de acordo com suas durações médias. Esta divisão é fundamental, pois evita que tarefas pequenas percam seus deadlines por estarem sendo executadas em conjunto com requisições muito grandes. De fato, este trabalho mostrou que, se as durações das requisições segue uma distribuição *heavy-tailed*, então pode-se separar um servidor somente para a execução das tarefas pequenas. Mesmo chegando com uma taxa muito alta, elas nunca perdem seus deadlines, por serem muito rápidas. Por outro lado, as requisições demoradas demandam um maior cuidado em seus tratamentos. Por possuírem pouco tempo ocioso antes de atingir o tempo máximo permitido, deve-se restringir o número de requisições grandes sendo executadas ao mesmo tempo, o que leva as tarefas que chegam e encontram o sistema saturado serem descartadas.

Nos resultados apresentados pela simulação preliminar, foi analisado que o ORBITA, algoritmo proposto neste trabalho, e o SITA-E possuem desempenhos superiores, pois fazem a distinção entre as tarefas, classificando-as de acordo com suas durações. As outras políticas, LWR e TAGS apresentaram taxas de vazão menores, principalmente nos cenários onde a taxa de chegada de novas requisições era muito alta e a variabilidade das durações geradas também era alta. O ORBITA se mostrou mais robusto do que o SITA-E por permitir que mesmo as tarefas grandes conseguissem executar dentro do tempo permitido. E, apesar de o LWR ter apresentado uma taxa de vazão maior do que o ORBITA para casos de baixa variabilidade e baixa taxa de chegadas, o algoritmo proposto ainda se mostrou como uma alternativa melhor quando foi utilizado um cenário com altas taxas de chegada.

Em seguida, foi apresentada a ferramenta Midas, que foi construída para averiguar a veracidade dos resultados obtidos na fase anterior. A ferramenta, construída como um

interceptador de transações de SGBDRs, foi testada com a utilização de uma aplicação de benchmark conhecida, o TPC-C, em duas versões: uma que simula um modelo aberto de geração de tarefas, onde as chegadas de novas transações segue uma distribuição exponencial, e outra que simula um modelo fechado, que é o padrão da especificação. Em ambas as configurações, o ORBITA obteve melhores resultados do que seus concorrentes, principalmente quando foram comparadas as taxas de vazão das transações longas: quanto mais elevadas eram as taxas de chagada de transações (no modelo aberto), tanto o LWR como o SITA-E não eram capazes de garantir que nenhuma transação desta classe terminasse dentro do deadline. Contudo, isto não ocorreu com o ORBITA, que foi capaz de manter praticamente inalteradas as taxas de vazão. Isto ocorre devido ao mecanismo que o algoritmo proposto possui, no qual as transações longas somente são aceitas se estas não causarem perdas de deadlines.

Durante a elaboração deste trabalho, foram submetidos diversos artigos a conferências internacionais, sendo que 3 deles foram aceitos. São eles:

- *Optimization for QoS on Web-Service-Based Systems with Tasks Deadlines*, aceito para publicação na 3a. *International Conference on Autonomic and Autonomous Systems,* realizada em junho de 2007. Este artigo apresenta um estudo preliminar sobre como diminuir a taxa de quebra de *deadlines.*

- *Task assignment on parallel QoS systems,* aceito para publicação na *8th International Conference on Web Information Systems Engineering,* a ser realizada em Dezembro de 2007. Este artigo apresenta um algoritmo, chamado *Task Assignment by Isolating Big Task* (TAIBT), que é uma primeira tentativa de diminuir a taxa de quebra de restrições temporais utilizando um controle de admissão.

- Por fim, o artigo *Fair Load-Balancing on Parallel Systems for QoS*, foi aceito para publicação na *International Conference on Parallel Processing*, realizada em setembro de 2007. Este artigo apresenta o ORBITA.

7.1 Trabalhos futuros

Como trabalhos futuros, poderia ser realizado um estudo sobre como as transações poderiam ser classificadas em tempo de execução. Atualmente, o Midas necessita que, ao início de cada transação, seja informado o tipo da mesma (Novo Pedido, Nível de Estoque, etc) para poder realizar os algoritmos de balanceamento de carga corretamente.

Adicionalmente, poderia ser estudada uma forma de estimar, em tempo de execução, a duração de cada comando SQL, mesmo que em números aproximados. Desta forma, seria possível realizar o balanceamento de carga com uma granularidade bem fina, para cada requisição enviada pelo cliente. É bom manter em mente que esse não é um problema trivial, uma vez que em bancos de dados replicados, é importante que o nível de isolamento seja respeitado para todas as transações e, cada uma das requisições contendo comandos SQL deve encontrar o banco de dados no mesmo estado da anterior.

Bibliografia

AKAL, F. et al, Fine-Grained Replication and Scheduling with Freshness and Correctness Guarantees. *31st Very Large Databases Conference*, p. 565 – 576, Trondheim, Norway, 2005.

AMZA, C., COX, A.L., ZWAENEPOEL, W. A Comparative Evalution of Transparent Scaling Techniques for Dynamic Content Servers. *21st International Conference On Data Engineering*, 2005.

BARKER, K. et al, A Load-Balancing Framework for Adaptative and Asynchronous Applications, *IEEE Journal Transactions on Parallel And Distributed Systems*, v.15, n.2, fevereiro 2004.

BARKER, K.; CHERNIKOV, A.; CRHISOCHOIDES, N.; PINGALI, K. A Load Balancing Framework for Adaptive and Asynchronous Applications. *IEEE Transactions on Parallel and Distributed Systems*, v. 15, n. 2, 2004

BHATTI, N.; FRIEDRICH, R. Web server support for tiered services. *IEEE Network*, v.13, n. 5, p. 64–71, set 1999.

BHOJ; RMANATHAN; SINGHAL. Web2K: Bringing QoS to Web servers. *Tech. Rep. HPL-2000-61*, HP Labs, May 2000.

CARDELLINI, V.; COLAJANNI, M.; YU, P.S. The State of the Art in Locally Distributed Web-Server Systems. *ACM Computing Surveys*, v. 34, p.263 – 311, 2002.

CHEN, X; MOHAPTRA, P; CHEN, H. An admission control scheme for predictable server response time forWeb accesses. In *Proceedings of the 10th World Wide Web Conference*, Hong Kong, maio 2001.

CHERKASOVA; PHALL Session-based admission control: A mechanism for peak load management of commercial Web sites. *IEEE Req. on Computers*, v.51, n.6, Jun 2002.

CROVELLA, M.; BESTAVROS, A. A.Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking,* p.835-836, 1997.

DAUDJEE, K.; SALEM, K., Lazy Database Replication with Snapshot Isolation, *30th Very Large Databases Conference*, p. 715 – 726, Seoul, Korea, 2006.

DEVINE, D.K. et al, New challenges in dynamic load balancing,*ADAPT '03: Conference on Adaptive Methods for Partial Differential Equations and Large-Scale Computation*, v. 52, n. 2-3, p. 133 – 152, fevereiro 2005

DYACHUK, D.; DETERS, R., Optimizing Performance of Web Service Providers, *IEEE 21st International Conference on Advanced Information Networking and Applications*, p. 46 – 53, Niagara Falls, Ontario, Canada, maio 2007.

ELNIKETY, S.; NAHUM, E.; TRACEY, J; ZWAENEPOEL, W. A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites, *WWW2004: The*

Thirteenth International World Wide Web Conference, New York City, NY, USA, maio 2004.

FURTADO, P.; SANTOS, C. , Extensible Contract Broker for Performance Differentiation, *International Workshop on Software Engineering for Adaptive and Self-Managing Systems* (2007), Minneapolis, USA.

GAMMA, E. et al, Padrões de Projeto: Soluções Reutilizáveis de Softwares Orientados a Objetos, Porto Alegre: Bookman, 2002.

GHAZALIE, T. M.; BAKER, T. P. Aperiodic Servers In A Deadline Scheduling Environment. *Real Time Systems Journal*. v 9, n. 1, p. 31 – 67, jul 1995.

GOKHALE, S. S.; LU, J., Performance and Availability Analysis of E-Commerce Site, *30th Annual International Computer Software and Applications Conference*, Chicago, setembro 2006.

HARCHOL-BALTER, M. Task assignment with unknown duration.. *Journal of the ACM,* 2002

HARCHOL-BALTER, M.; CROVELLA, M.: MURTA, C.:On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, v.59 n. 2, 204-228, 1999.

HARCHOL-BALTER, M.; DOWNEY, A. Exploiting process lifetime distributions for dynamic load-balancing. *ACM Transactions on Computer Systems*,1997.

KNIGHTLY, E.; SHROFF, N. Admission Control for Statistical QoS: Theory and Practice. *IEEE Network*, v. 13, n. 2, pp. 20-29, 1999.

NELSON, R.; PHILIPS, T. An approximation for the mean response time for shortest queue routing with general interarrival and service times. *Performance Evaluation,* p.123-139, 1993.

NELSON, R.; PHILIPS, T. An approximation to the response time for shortest queue routing. *Performance Evaluation Review,* p.181-189, 1989

ORLEANS, L.F., FURTADO, P.N., Fair Load-Balancing on Parallel Systems for QoS, *ICPP International Conference on Parallel Processing,* p. 22, setembro 2007

ORLEANS, L.F., FURTADO, P.N., Optimization for QoS on Web-Service-Based Systems with Tasks Deadlines, *ICAS - Third International Conference on Autonomic and Autonomous Systems,*, p. 6, junho 2007

PRADHAN, P.; TEWARY, R; SAHU, S; CHANDRA, A. An observation-based approach towards self managing Web servers. In *International Workshop on Quality of Service*, Miami Beach, FL, maio 2002.

SCHROEDER, B.; HARCHOL-BALTER, M. Achieving class-based QoS for transactional workloads. *IEEE International Conference on Data Engineering*, 2006.

SCHROEDER, B.; WIERMAN, A.; HARCHOL-BALTER, M. Open Versus Closed: A Cautionary Tale. *Network System Design and Implementation*, San Jose, CA. pp 239-252, 2006

SERRA, A.; GAÏTI, D.; BARROSO, G.; BOUDY, J. Assuring QoS Differentiation and Load-Balancing on Web Servers Clusters. *IEEE Conference on Control Applications*, p. 8.85-890, 2005.

WIESMANN, M. et al, Understanding Replication in Databases and Distributed Systems, 20th IEEE International Conference on Distributed Computing Systems , p. 464, 2000

WIESMANN, M., SCHIPER, A. Comparison of Database Replication Techniques Based on Total Order Broadcast. *IEEE Journal Transactions On Knowledge And Data Engineering*, v. 17, n. 4, p. 551 – 566, abril 2005.

WIKIPEDIA the free encyclopedia. Disponível em <http://en.wikipedia.org/wiki/Load_balancing_%28computing%29>. Acesso em 08/07/2007.

WIKIPEDIA the free encyclopedia. Disponível em <http://en.wikipedia.org/wiki/Quality_of_service>. Acesso em 15/07/2007.

WYDROWSKI, B.;ZUKERMAN, M., QoS in Best-Effort Networks, *IEEE Communications Magazine*, dezembro 2002

XIONG, M. et al, Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics, *IEEE Journal Transactions On Knowledge And Data Engineering*, v.14, n.5, p. 1155 – 1166, setembro 2002.