# 4
# Experimentos e Avaliação de Desempenho

Este capítulo é dedicado à descrição e à discussão dos experimentos envolvendo a provisão de reserva de recursos no nível do usuário. Nas próximas seções, são apresentados testes que comprovam a funcionalidade e o grau de confiabilidade das reservas do ReservationSuite, assim como experimentos para determinar a escalabilidade do servidor e a sobrecarga gerada pelas técnicas de interposição e interceptação de código.

Neste capítulo, também é descrito um estudo de caso em que o ReservationSuite é utilizado para controlar o uso de recursos de uma aplicação multicamadas que faz uso intensivo dos recursos de processamento, largura de banda de disco e de rede. Como o consumo desses recursos está diretamente ligado a medidas de desempenho de alto nível da aplicação, é preciso estabelecer meios de mapear requisitos de QoS de alto nível em montantes de reserva de CPU, largura de banda de disco e rede atribuídos a cada componente da aplicação. Para isso, apresentamos um método simples de decomposição de QoS, o qual também é descrito e avaliado neste capítulo. Projetado para ter uma implementação simples, esse método não tem como meta propor uma solução que represente o estado da arte para a decomposição de QoS, mas sim mostrar que, a partir de técnicas bastante simples, é possível predizer com razoável precisão os parâmetros de reserva a serem empregados no ReservationSuite e, assim, garantir qualidade de serviço às aplicações.

## 4.1
## Infraestrutura de Testes

Os experimentos relativos à provisão de QoS foram realizados com o objetivo de determinar a eficácia das reservas impostas pelo ReservationSuite e a sobrecarga que o gerenciamento dessas reservas geram no servidor da ferramenta. Os testes foram realizados em dois computadores cujas configurações são descritas na tabela 4.1.

Tabela 4.1: Computadores utilizados nos testes de eficácia e desempenho do ReservationSuite.

| Computador | Processador | Memória | SO | Kernel |
|---|---|---|---|---|
| máquina I | Intel Centrino Duo 1,66 GHZ | 1G | Ubuntu 9.04 | 2.6.24 |
| máquina II | Intel Core 2 Duo 2.10 GHZ | 3G | Ubuntu 9.04 | 2.6.28 |

## 4.2 Testes de Eficácia da Reserva de Processamento

Para avaliar a eficácia das reservas de processamento, foram executadas duas aplicações CPU-intensivas no processador I da máquina I descrita na tabela 4.1 – uma aplicação sem reserva, isto é, sendo atendida no modelo de melhor esforço, e uma aplicação sendo atendida com a reserva de CPU provida pelo ReservationSuite e sem a propriedade de *work-conserving*, ou seja, mesmo que haja processamento ocioso na máquina, a reserva desse recurso para o processo não será expandida. O resultado do experimento é apresentado na figura 4.1. Inicialmente, somente a primeira aplicação é executada. Assim, ela ocupa quase a totalidade de processamento da CPU. Próximo a 80 segundos, a aplicação com reserva é iniciada com uma requisição de 800 ms a cada 1000 ms, ou seja, com reserva de 80% da CPU. Então, é possível perceber que a aplicação com reserva tem a sua taxa de processamento garantida, enquanto a aplicação de melhor esforço passa a utilizar apenas o processamento ocioso presente na máquina (cerca de 20% da CPU). Aos 140 ms, a primeira aplicação é finalizada, mas a aplicação com reserva não utiliza a totalidade de CPU, pois está configurada como *non-work-conserving*. Aos 210 ms, essa propriedade é alterada e o processo passa a utilizar os 80% de processamento garantidos pela sua reserva mais o montante de processamento que se encontrar ocioso na CPU. Assim, o processo com reserva passa a consumir quase 100% da CPU onde executa.

A análise dos experimentos mostra que a reserva de processamento é realizada com eficácia no ReservationSuite. Por atribuir altas prioridades aos processos com reserva e, sendo essas prioridades variáveis determinantes no escalonamento de processos do sistema operacional, o ReservationSuite consegue garantir a execução de seus processos com a qualidade requisitada mesmo em casos onde há processos externos à ferramenta sendo executados.

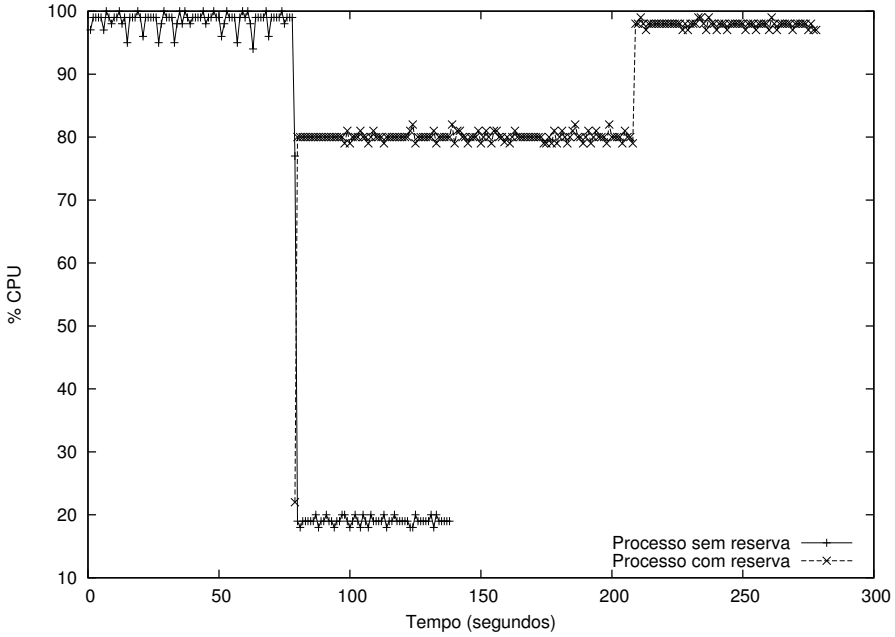


Figura 4.1: Consumo de CPU de aplicações com e sem reserva de processamento.

## 4.3 Escalabilidade do Servidor

Os testes de escalabilidade foram realizados no computador II da tabela 4.1. O objetivo dos testes foi estimar o quanto o servidor consome de recursos para atender a um número crescente de clientes. Como reservas de largura de banda de disco e rede são realizadas por uma biblioteca ligada dinamicamente à própria aplicação sendo executada, a escalabilidade do servidor não é influenciada por elas. Assim, na realização dos experimentos de escalabilidade, foram utilizadas requisições de execução de aplicações limitadas a CPU somente.

Nesses experimentos, o servidor ReservationSuite foi inicializado para gerenciar os dois processadores da máquina, enquanto as aplicações (carga de trabalho) consistiram em processos de espera ocupada. Cada aplicação requisitou ao servidor uma reserva de 20 ms a cada 1000 ms, isto é, 2% de CPU cada. Com esse valor de reserva, dado que a máquina II é duo-processada, idealmente, é possível executar até 100 aplicações paralelamente. Tão pequena porcentagem de reserva foi escolhida para criar o maior número de alarmes no menor tempo possível para o CPUReserve. Isso porque o CPUReserve obtém o tempo de execução de um processo a partir do arquivo /proc/pid/stat. Esse

tempo é dado em *jiffies*, uma medida que, no ambiente de testes utilizado, corresponde a 10 ms. Porém, o uso de 10 ms para a fatia de tempo causa algumas inconsistências quando um processo já executou os seus 10 ms do ciclo, mas o SO ainda não atualizou o seu tempo de execução no arquivo /proc. Com reservas de 2% de processamento, o servidor trabalha mais consistentemente e próximo à sua capacidade máxima.

Os experimentos mostraram que o CPUReserve, independentemente de qual política de escalonamento é utilizada, consome, por si só, isto é, sem gerenciar nenhum processo cliente, tecnicamente 0% de CPU e 1 MB de memória. A fim de impedir que páginas de memória utilizadas pelo servidor sejam enviadas para a área de *swap*, utilizamos a chamada *mlock* para fixar parte do espaço do endereço virtual do servidor a 32 MB de memória RAM dedicada a ele.

A figura 4.2 apresenta o consumo de CPU do servidor com um número crescente de clientes. Nessa figura, cada ponto consiste em uma média de 180 medidas de consumo de recursos colhidas em 3 diferentes execuções. As duas curvas do gráfico são referentes a implementações da política de alarme mais próximo (política em que processos cujos alarmes expiram mais cedo têm maior prioridade de execução) nas linguagens de programação C e Lua.

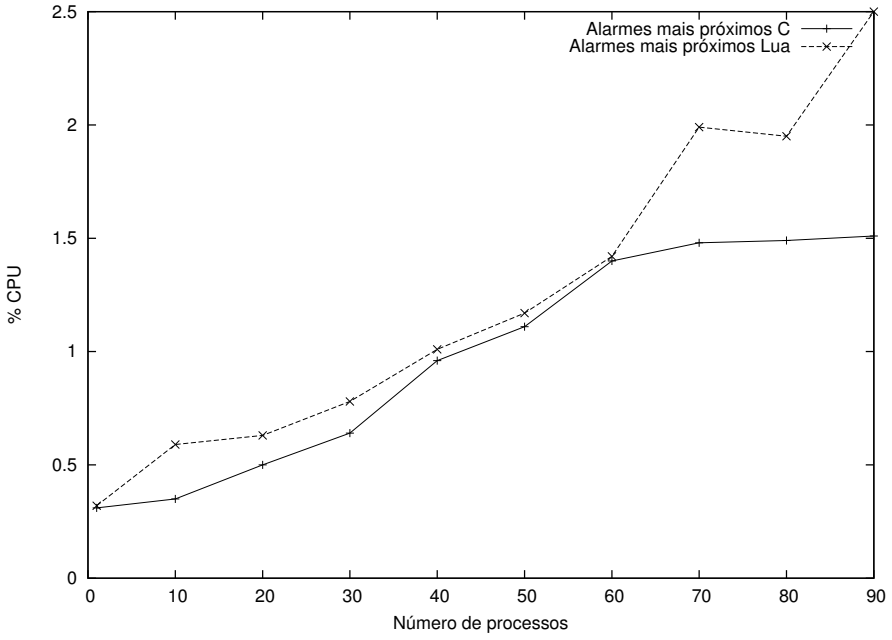


Figura 4.2: Uso de CPU do servidor ReservationSuite com um número variável de clientes.

Como é possível notar, o consumo de processamento da política C cresce

linearmente com o número de clientes. O desempenho da política implementada em Lua, inicialmente, é muito similar ao da política C, mas é possível notar uma acentuação da curva de consumo de processamento quando o número de clientes é maior que 60. Atribuímos esse fato, principalmente, à crescente interferência que o gerenciador de memória de Lua gera no desempenho do servidor ReservationSuite quando o número de processos controlados por nossa ferramenta aumenta.

Em Lua, por padrão, o gerenciamento de memória é realizado de maneira automática. Assim, tarefas de alocação e dealocação de memória podem ser realizadas de maneira transparente ao programador. Nesses casos, não há meios simples de se determinar a quantidade de memória que um processo está utilizando e nem em que momentos o processamento utilizado pelo processo irá variar por conta da manipulação do espaço de memória. Conclui-se então que políticas escritas em Lua tornam o escalonamento mais fácil de ser flexibilizado, mas podem gerar degradações de desempenho quando comparadas às políticas implementadas em C.

O desempenho expresso na figura 4.2 é melhor que o desempenho anteriormente documentado [25]. O ganho observado é decorrente de melhorias nas funções de monitoramento e de mudanças nas estruturas de dados utilizadas no gerenciamento dos processos.

## 4.4 Testes de Eficácia da Reserva de Largura de Banda de Disco

Para avaliar a eficácia das reservas de largura de banda de disco, foram executadas duas instâncias do *benchmark* `iozone`[1] na máquina II descrita na tabela 4.1 – uma instância sem reserva de disco, isto é, sendo atendida no modelo de melhor esforço, e uma instância sendo atendida com a reserva de disco provida pelo ReservationSuite. Ambas as aplicações realizavam escritas constantes no disco da máquina (comando `iozone -s 4g -i 0 -f fileX.tmp`). O resultado do experimento é apresentado na figura 4.3. Inicialmente, somente a primeira aplicação é executada. Como é possível notar, a aplicação tem uma taxa de escrita oscilante, mas que, em média, fica em torno de 50000 KB/segundo. Próximo dos 50 segundos, a segunda aplicação é iniciada com uma reserva de 30000 KB/segundo. No entanto, são precisos cerca de 70 segundos para que essa reserva comece a ser garantida. Esse comportamento acontece durante o período em que a *cache* de arquivos está carregada com dados da primeira aplicação. Quando a *cache* de arquivos é carregada com dados da segunda aplicação e esses dados começam a ser escritos em disco, a

[1] www.iozone.org

reserva passa a ser garantida, mesmo quando a primeira aplicação é finalizada, indicando a eficácia na limitação de acesso ao disco. No instante próximo a 230 segundos, o processo com reserva requisita uma adaptação para acessar o disco a 40000 KB/segundo e tem o seu pedido atendido pelo servidor.

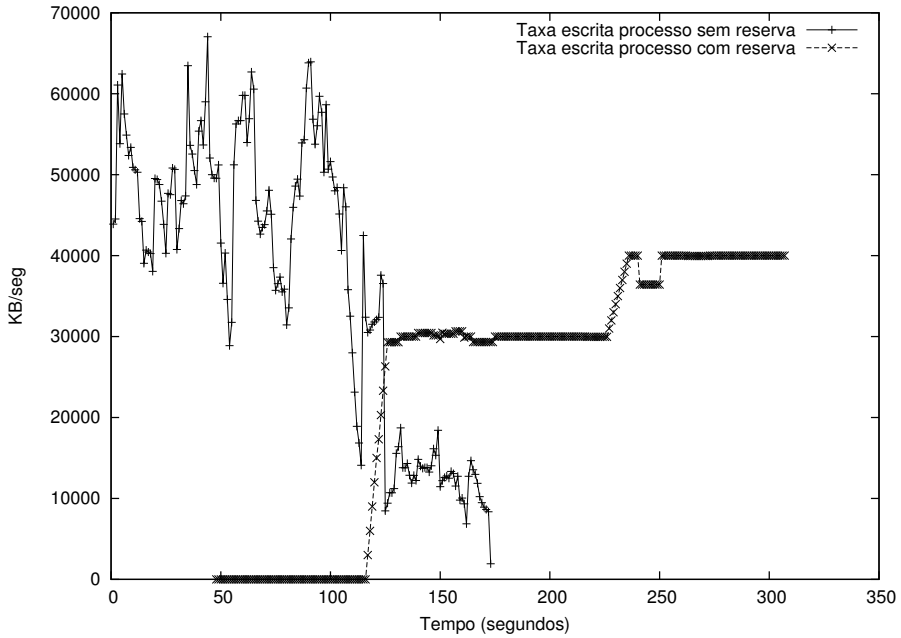


Figura 4.3: Taxa de escrita de aplicações com e sem reserva de largura de banda de disco.

Em um ambiente com necessidades mais fortes de garantia de desempenho, é papel do controlador de admissão recusar o pedido de reserva do novo processo de modo a evitar períodos de inanição tal como o observado na figura 4.3.

É importante notar que o retardo na execução da aplicação com reserva não acontece em situações de leitura do disco (*read*) por que esse tipo de requisição é realizada de maneira síncrona. A figura 4.4 apresenta a execução de duas aplicações com leitura intensa de disco (*script* com laço infinito de `cat fileX.tmp > /dev/null`). Enquanto a primeira aplicação é a única que concorre ao acesso ao disco, ela lê a mídia em uma média próxima a 50000 KB/segundo. Aos 71 segundos, a segunda aplicação é iniciada com reserva de 30000 KB/segundo e, praticamente no mesmo instante, ela já tem as suas requisições de leitura atendidas. A partir desse momento, a aplicação sendo atendida por melhor esforço passa a ler dados a uma taxa próxima a 15000 KB/segundo.

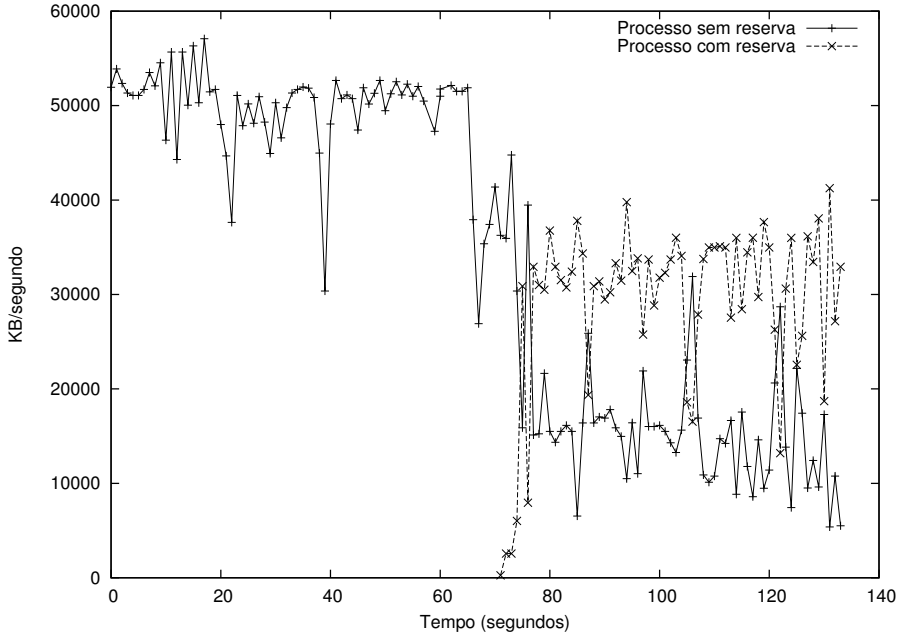


Figura 4.4: Taxa de leitura de aplicações com e sem reserva de largura de banda de disco.

Operações de escrita que não fazem uso de *cache* (*flag O_DIRECT* da chamada para abertura de arquivos *open* habilitada) também não sofrem retardos na execução. A desabilitação do uso de *cache*, porém, insere sincronismo nas operações de acesso ao disco o que, muitas vezes, resulta em significativa perda de desempenho, principalmente em casos onde a aplicação acessa um grande volume de dados sequenciais e não implementa uma *cache* própria. No caso do `iozone` executando operações de escrita com e sem o uso de *cache* (comandos `iozone -s 4g -i 0 -f fileX.tmp` e `iozone -s 4g -i 0 -I -f fileX.tmp`, respectivamente), por exemplo, nota-se que o desempenho do *benchmark* varia drasticamente conforme apresentado na tabela 4.2. Com o uso de *cache* e sem sincronismo, o *benchmark* apresenta uma média da taxa de escrita em disco de 48747 KB/segundo. Ao desabilitar a *cache* de arquivos, essa média cai para 18860 KB/segundo, ou seja, cerca de 40% da média com *cache*.

Tabela 4.2: Taxas de escrita do iozone com e sem o uso de *cache*.

| Comando | Taxa de escrita (KB/s) |
|---|---|
| Com cache | 48747 |
| Sem cache | 18860 |

A fim de se reduzir retardos de execução de aplicações com E/S, a desabilitação do uso da *cache* pode ser uma boa prática. Para contornar as perdas de desempenho, poderiam ser implementadas *caches* em nossa ferramenta de reserva, de maneira similar à descrita no trabalho de Wachs *et al.* [52]. Uma outra opção, de menor interferência no desempenho das aplicações, consiste em garantir que todos os processos sendo executados na máquina servidora (à exceção dos processos do SO) sejam disparados a partir do ReservationSuite e somente até que a saturação do disco seja atingida.

Quando todos os processos são disparados a partir do ReservationSuite, a nossa ferramenta de controle de acesso ao disco suspende o atendimento de requisições de aplicações que já utilizaram suas cotas de largura de banda de disco para o período. Assim, outras aplicações com necessidade de acesso à mídia de armazenamento podem ser atendidas. Para garantir que todas as reservas sejam atendidas nas frequências especificadas durante o pedido de reserva, no ReservationSuite, novas reservas de largura de banda de disco são aceitas somente até que o valor da propriedade DISK_RESERVATION_LIMIT seja alcançado. Esse valor, configurável pelo usuário de nossa ferramenta, deve ser estimado no ponto de saturação do disco, ou seja, na taxa de *bytes* lidos/escritos quando o disco é continuamente acessado (100% de utilização).

Para mostrar a importância de se estimar corretamente o ponto de saturação da mídia de disco, realizamos um experimento sem o uso do nosso controlador de admissão. Nesse experimento, a cada 20 segundos, uma instância do *benchmark* `iozone`, com os mesmos parâmetros do experimento da figura 4.3 (`-s 4g -i 0 -f fileX.tmp`), é iniciada com uma requisição de taxa de escrita de 3000 KB/segundo. Pretendemos mostrar que quanto mais saturado o disco estiver, maior será a frequência em que violações na taxa de escrita esperada acontecem. A figura 4.5 apresenta o resultado do experimento em dois gráficos. Para preservar a visibilidade, ambos os gráficos apresentam as taxas de acesso ao disco somente para os três primeiros *benchmarks*, iniciados nos instantes 0, 20 e 40.

Na figura 4.5, o gráfico maior ilustra a taxa de escrita dos três processos até o instante 140, ou seja, até o momento em que os três processos competiam com outros 5 processos pelo acesso ao disco. Pela análise do gráfico, é possível notar que, até os 140 segundos de execução, à exceção de duas grandes variações na qualidade de serviço esperada para o processo 2 e de outras pequenas oscilações, as reservas de largura de banda de disco são rapidamente atendidas e respeitadas para os três processos.

O gráfico no topo esquerdo da figura 4.5 apresenta a continuidade do experimento até os 240 segundos, quando havia 13 processos em execução.

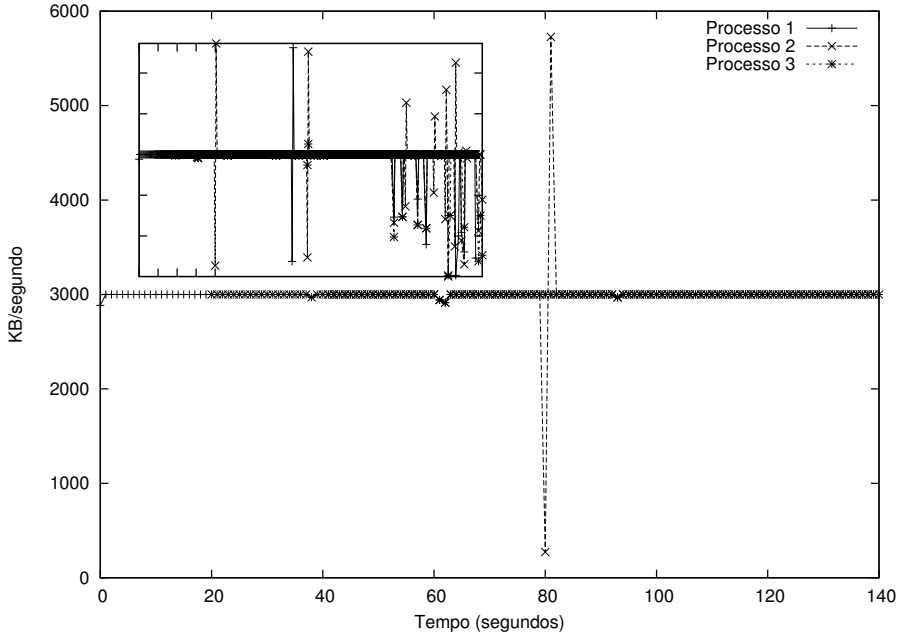


Figura 4.5: Taxa de escrita de aplicações com reserva de largura de banda de disco.

Nota-se que, como esperado, à medida que novas instâncias do `iozone` foram sendo criadas, as oscilações na taxa de escrita dos processos passaram a ser maiores e mais frequentes, um indício de que o disco utilizado estava entrando em estado de saturação.

A análise do experimento realizado mostra que não se pode determinar, com exatidão, o número de processos que podem ser atendidos sem causar saturação no disco, pois esse número é dependente da confiabilidade das reservas que se deseja obter. Para exemplificar, considere uma situação em que as reservas devem ser atendidas 90% das vezes. De acordo com o gráfico maior da figura 4.5, poderíamos configurar o ponto de saturação do disco a 24000 KB/segundo (8 processos com taxa de 3000 KB/segundo cada). Em uma outra situação em que não se espera que as reservas sejam cumpridas em mais de 50% das vezes, de acordo com o gráfico menor da figura 4.5, o ponto de saturação poderia ser até maior que 39000 KB/segundo (13 processos com taxa de 3000 KB/segundo cada).

No Linux, o envio e o recebimento de dados para o disco e para a rede podem ser feitos de maneira síncrona ou assíncrona. Operações síncronas de E/S não podem ser divididas em suboperações pelo ReservationSuite porque, muitas vezes, os dados ainda não se encontram disponíveis para o envio/recebimento e a espera por eles poderia gerar inanição no processo.

Assim, operações de envio/recebimento são realizadas com uma única chamada ao sistema (*system call*) relativa à operação sendo tratada. Quando um pedido de envio/recebimento de dados ocorre em um período em que o montante de *bytes* a ser enviado/recebido para rede ou disco é superior ao montante de *bytes* disponíveis para serem enviados/recebidos naquele período, o servidor do ReservationSuite envia/recebe esses dados de uma única vez e desconta o montante excedente do próximo período do processo. Assim, algumas aparentes "sobreutilizações" de recursos podem ocasionalmente acontecer.

## 4.5 Testes de Eficácia da Reserva de Largura de Banda de Rede

Para mostrar a eficácia da reserva de largura de banda de rede, foi necessário instrumentar o código da ferramenta de reserva de rede para que ela indicasse o número de *bytes* lidos e enviados por cada processo do sistema, pois não há no sistema operacional Linux um comando que informe tal dado; apenas a taxa de transferência para todos os processos ativos no SO é disponibilizada. Além disso, no experimento realizado, não foram utilizadas aplicações externas ao ReservationSuite, pois o Linux não provê mecanismos de priorização para envio/recebimento de *bytes* na rede. Ainda assim, tal como acontece com a largura de banda de disco, o acesso à rede pode ser facilitado em casos onde a ferramenta ReservationSuite possui controle sobre todos os processos acessando intensamente a rede.

A figura 4.6 apresenta o desempenho de uma única aplicação *web* (servidor *web* Tomcat) servindo a uma carga de trabalho gerada pelo *benchmark* `TPC-W`[2]. Nessa figura existem duas linhas: uma que representa a taxa de envio de *bytes* pela interface de rede, a qual é monitorada pelo sistema operacional, e a taxa de envio de *bytes* monitorada pelo ReservationSuite. Como é possível notar, sem o limite de uso da rede, a aplicação do experimento envia dados para a interface de rede a uma taxa variável com média de 43500 B/segundo. Aos 53 segundos, essa aplicação sofre uma adaptação a qual impõe um uso da largura de banda de rede a uma taxa de 10000 B/segundo. A partir desse momento, a taxa de acesso à rede é fielmente respeitada pela aplicação.

No experimento com o servidor Tomcat é importante observar a robustez do ReservationSuite ao gerenciar uma aplicação composta por dezenas de *threads*, todas requisitando concorrentemente operações de envio de dados na rede. Essa mesma robustez foi evidenciada em testes com o banco de dados `MySQL` para a limitação do uso de disco e rede pelo TPC-W. Detalhes desses testes serão descritos adiante neste capítulo.

[2]http://www.tpc.org/tpcw/.

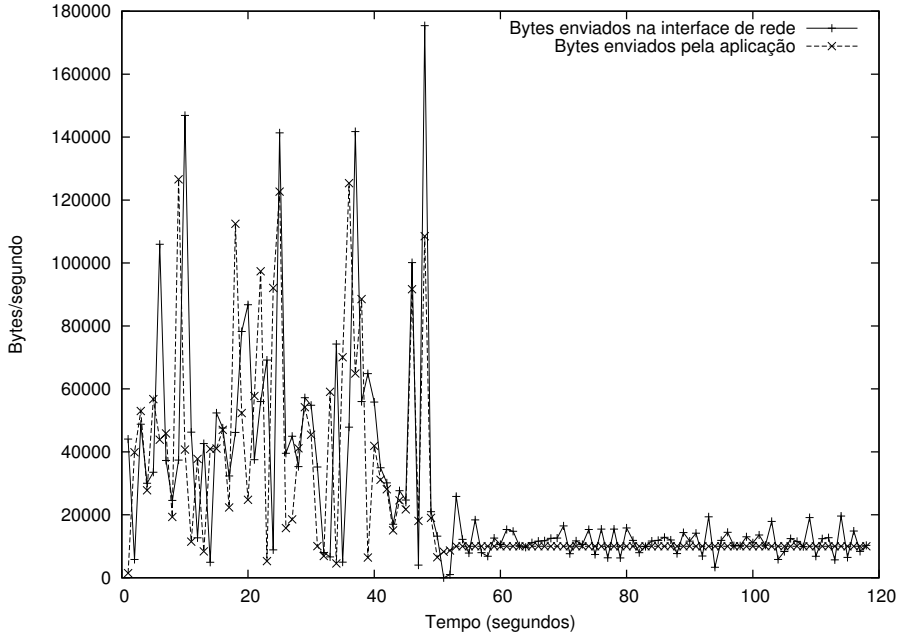


Figura 4.6: Taxa de envio de dados de aplicação com reserva de largura de banda de rede.

## 4.6 Sobrecarga da Interposição de Funções

Analisando o gráfico da figura 4.7, correspondente ao consumo de CPU da execução dos *benchmarks* apresentados na figura 4.3, é possível notar que a interposição de funções para o controle de E/S de dados acarreta uma carga de processamento extra na aplicação. Em especial, no período em que o processo com reserva de disco encontra-se com uma taxa de escrita de 30000 KB/segundo (período iniciado próximo dos 140 segundos), é possível verificar que ele ocupa uma porcentagem de processamento similar à que o processo sem reserva ocupava no início do experimento, sendo que, nesse período, a taxa de escrita do processo sem reserva foi, em média, de 48860 KB/segundo.

A carga de trabalho gerada pela interposição de funções é consequência dos procedimentos de controle de *bytes* lidos/escritos a cada operação de acesso ao disco. Na interposição da operação de *write*, por exemplo, é preciso verificar se o descritor de arquivos passado como parâmetro da função é pertencente a um *socket* ou a um arquivo comum. Se ele pertencer a um arquivo, o pedido de escrita será tratado com limites de acesso ao disco, se ele pertencer a um *socket*, o pedido será tratado com limites de acesso à rede. Somente para essa distinção de descritores, duas chamadas ao sistema são feitas: *fstat* e *fcntl*.

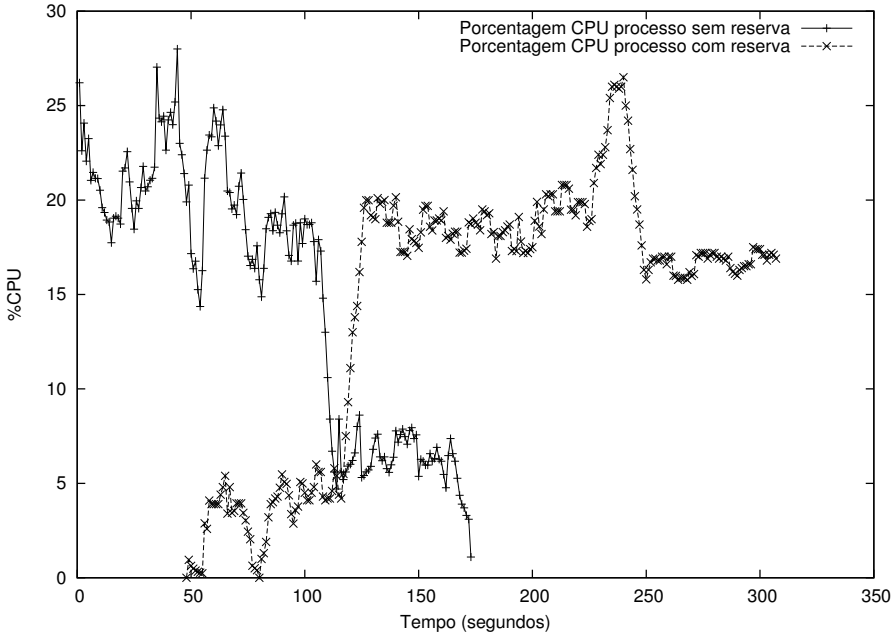


Figura 4.7: Consumo de CPU de aplicações com e sem reserva de largura de banda de disco.

Além delas ainda é preciso invocar a chamada *gettimeofday* para determinar se um novo período corrente de acesso ao disco ou à rede já foi iniciado.

Considerando que a operação de *write* é a mais sobrecarregada das funções sobrescritas, ela serve como um limite superior na determinação da sobrecarga imposta pela interposição no controle de E/S de dados. Assim, novamente foram realizados experimentos com o *benchmark* `iozone` configurado para executar somente operações de escrita. O objetivo dos experimentos foi medir o tempo de execução desse *benchmark* e o consumo de CPU quando não são utilizadas técnicas de controle de E/S, quando a interposição é utilizada e quando há a interceptação utilizando *ptrace* (técnica descrita na seção 3.2.2). Os resultados são apresentados na figura 4.8.

Pela análise da figura 4.8, nota-se que o tempo de execução do *benchmark* sem técnicas de controle de E/S é, em uma média de 30 execuções, igual a 45841 ms. O mesmo *benchmark* executa com 3 segundos a mais quando utiliza a técnica de interposição, enquanto utilizando a técnica de interceptação são necessários 5 segundos adicionais.

A figura 4.9 apresenta o consumo de CPU para cada uma dessas execuções. Considerando o consumo de processamento durante os diferentes tempos de execução apresentados nessa figura, nota-se que a interposição de funções acarreta uma sobrecarga de até 50% no consumo de CPU do

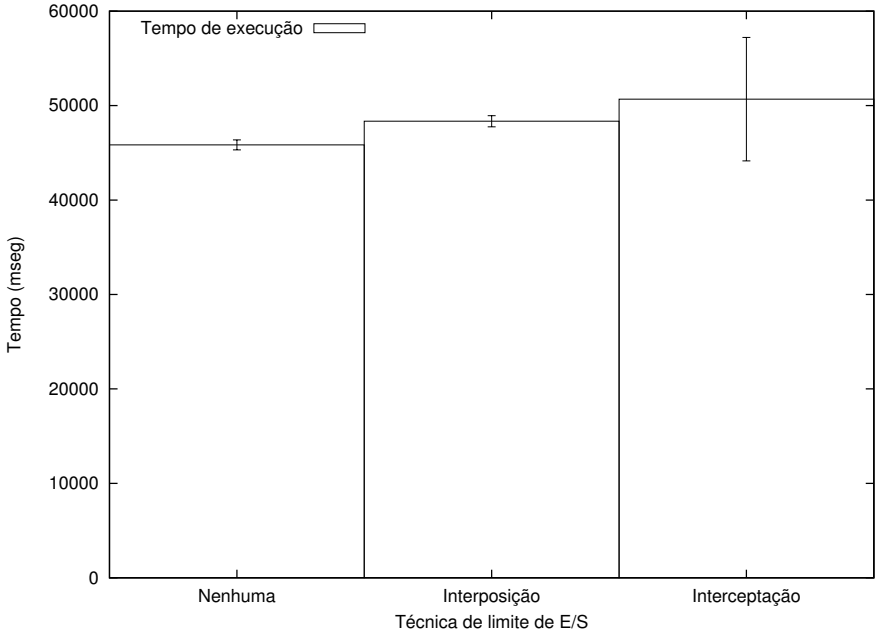


Figura 4.8: Tempo de execução de aplicações sem limite de uso de disco, com limite imposto por interposição e com limite imposto por interceptação.

*benchmark*. A interceptação, por sua vez, acarreta uma sobrecarga muito maior ao considerarmos que o processo que realiza o controle de E/S é externo ao *benchmark*, sendo representado pelo processo `Interceptador` na legenda da figura 4.9. A soma dos consumos de CPU do processo `iozone` e do processo interceptador é de 74.64%, cerca de 4.4 vezes maior que o consumo do processo sem controle de E/S.

Ainda que 50% de sobrecarga possa representar um acréscimo muito alto de processamento em uma aplicação, esse valor é justificável ao considerarmos os benefícios que o uso de controle de reservas pode trazer a um ambiente altamente compartilhado. Além disso, poucas aplicações acessam tão intensamente o disco tal como o *benchmark* utilizado. Assim, é esperado que essa sobrecarga seja atenuada em situações em as aplicações acessem menos intensamente o disco rígido.

## 4.7 ReservationSuite e Máquinas Virtuais

Durante a fase de criação de uma máquina virtual, muitas vezes é possível especificar o quanto de memória, armazenamento e processamento que a máquina conterá. Com essa especificação, monitores de máquinas virtuais são

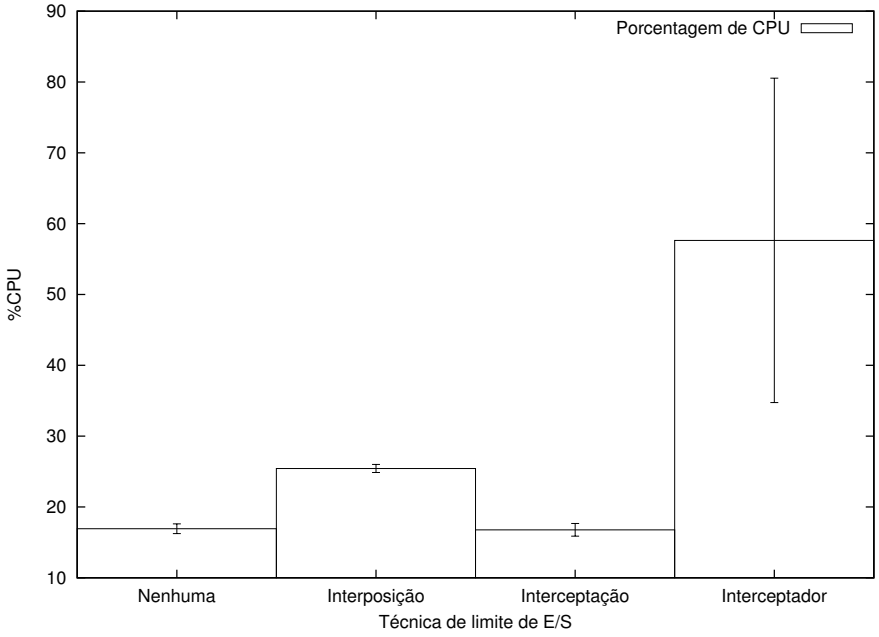


Figura 4.9: Consumo de CPU de aplicações sem limite de uso de disco, com limite imposto por interposição e com limite imposto por interceptação.

capazes de prover certo isolamento de desempenho entre os diferentes domínios presentes no *hardware* através da multiplexação dos recursos das máquinas.

O custo para o gerenciamento de máquinas virtuais é alto e pode inviabilizar a prática de isolar a execução de aplicações colocando-as em diferentes máquinas virtuais [11]. Por outro lado, a execução de mais de uma aplicação por máquina virtual deixa de garantir o isolamento de desempenho. Como alternativa, é possível visualizar ambientes de isolamento híbridos onde coexistem máquinas virtuais e gerenciadores de reserva no nível do usuário.

A figura 4.10 apresenta um ambiente híbrido contendo uma hierarquia de reservas, onde uma máquina física é representada com duas máquinas virtuais. Na primeira delas, existe uma instância do ReservationSuite responsável por gerenciar de forma mais fina as reservas de recursos entre as aplicações de uma mesma máquina virtual. Nessa máquina, é possível que diversas aplicações executem concorrentemente, sem que o desempenho entre elas seja afetado. Pode-se, por exemplo, especificar que a primeira máquina será executada com 50% de uma CPU e uma aplicação dessa máquina (App1, por exemplo) executará com 30% desses 50%, ou seja, com aproximadamente 16% da capacidade de processamento de uma CPU. Já na segunda máquina virtual, o isolamento de desempenho é garantido se somente uma aplicação

for executada, banalizando assim a criação de novas máquinas virtuais quando novas execuções forem solicitadas.

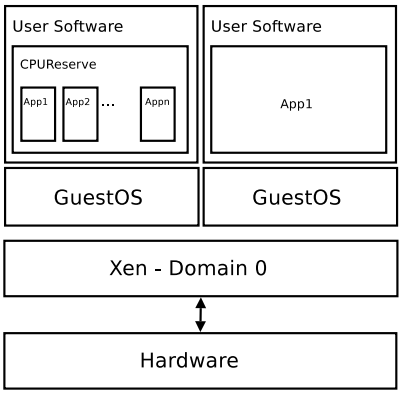


Figura 4.10: Hierarquia de reserva de processamento em máquinas virtuais.

A fim de validar a proposta de um ambiente híbrido, um cenário como o apresentado pela figura 4.10 foi construído utilizando-se o KVM (*Kernel based Virtual Machine*) [71] e o núcleo Linux 2.6.24. Nesse cenário, o Reservation-Suite foi utilizado tanto para inicializar as máquinas com as respectivas reservas de processamento, quanto para isolar o desempenho das aplicações dentro de cada domínio. Os experimentos mostram que a ferramenta de reserva estudada pode ser facilmente utilizada em diferentes níveis de granularidade de código.

## 4.8 Um Estudo de Caso sobre Decomposição de QoS

Tão importante quanto a reserva e a garantia de uso de recursos é a estimativa do montante de recursos necessários para se atingir uma meta de desempenho. Quando feita com precisão, a estimativa de consumo é capaz de aumentar o compartilhamento e de garantir qualidade de serviço às aplicações. Para isso, é necessário traçar um perfil de uso de recursos da aplicação de acordo com a qualidade de serviço esperada. Considere, por exemplo, uma aplicação de vídeo em que a qualidade de exibição é determinada pelo número de quadros exibidos por segundo. Nesse caso, é preciso determinar uma relação entre o número de quadros, considerado o parâmetro de alto nível, e os montantes mínimos de CPU, memória, disco e rede, considerados parâmetros de baixo nível, necessários para se atingir o desempenho desejado.

Assim como descrito por Dodonov [72, 73], existem diferentes técnicas capazes de extrair o perfil de consumo de recursos de aplicações. Essas técnicas vem sendo utilizadas de diferentes formas durante as últimas décadas. Ainda em 1989, por exemplo, Devarakonda e Iyer propuseram uma técnica estatística

para a predição dos consumos de CPU, entrada e saída de arquivos e memória de programas executáveis, os quais eram identificados pelas semelhanças de suas chamadas ao sistema (*system calls*) realizadas [74].

Diferentemente dos experimentos conduzidos por Devarakonda e Iyer, muitos trabalhos precisam determinar os montantes de recursos utilizados não por um programa, mas por uma requisição feita a uma aplicação, fato que agrega dificuldades à predição [75, 76]. Urgaonkar descreve experimentos nos quais as aplicações são monitoradas e têm o seu uso de recursos e a variância medidos em intervalos de tempo constantes. Com as informações coletadas, distribuições de probabilidade são geradas e utilizadas na tomada de decisões que influenciam no atendimento de requisitos de QoS [75].

Outros trabalhos de relevância na literatura, tais como os de Zhang [76] e de Wood [77], têm relatado o uso de regressão linear [78] nos dados de utilização de recursos coletados em execuções prévias. No entanto, esses trabalhos tipicamente analisam as necessidades de um único tipo de recurso, mais tradicionalmente a CPU.

Em sua tese de doutorado, Aron [79] faz previsões de múltiplos recursos de processamento, largura de banda de disco e uso de memória para servidores *web*, mas simplifica o problema da predição ao assumir que a qualidade de serviço está linearmente relacionada à utilização desses recursos. Dependendo do tipo de carga de trabalho sendo gerenciada, essa suposição não pode ser considerada. No trabalho citado, Aron ainda utiliza a média de utilização de recursos como métrica para a predição de desempenho. Tal métrica, como exposto no trabalho de Mi [80] e discutido adiante neste capítulo, não tende a ser uma boa métrica para aplicações cujas cargas de trabalho apresentam grandes variações no consumo de recursos.

Como nossa aplicação alvo é multicamadas, um novo problema na determinação de uso de recursos surge: como distribuir as capacidades dos recursos entre as diferentes camadas de *software* de forma que o resultado final de desempenho atenda às necessidades da aplicação. A solução de tal problema, chamado de decomposição de QoS, tipicamente envolve, além da determinação do perfil de consumo da aplicação, a modelagem analítica de sua arquitetura (por exemplo, através de rede de filas) e posterior análise para a satisfação de restrições.

Não foi encontrado na literatura nenhum estudo de decomposição de QoS que envolvesse simultaneamente os recursos computacionais tratados pelo ReservationSuite. Porém, acreditamos que uma prática que estime o uso desses recursos é essencial para a verificação das funcionalidades de garantia e limitação de consumo de processamento, disco e rede propostas pelo nosso

conjunto de ferramentas de reserva.

Na literatura consultada para o levantamento de trabalhos relacionados à predição de desempenho, muitos trabalhos utilizam a predição para determinar o planejamento de capacidade de servidores que hospedam aplicações multicamadas [76, 77, 81, 82, 83, 84], mas apenas o trabalho descrito por Chen utiliza a predição no problema de decomposição de QoS [81]. Nesse trabalho, somente o mapeamento de requisitos ligados ao processamento de dados foi considerado. Assim, o problema de decomposição de QoS atacado consistia em encontrar uma combinação de reservas de CPU que, imposta às camadas da aplicação, permitisse que o objetivo de alto nível do usuário fosse garantido. Por seu problema tratar apenas o recurso de processamento, Chen propôs encontrar as possíveis combinações de reserva pelo método de força bruta. No entanto, em experimentos envolvendo mais tipos de recursos, essa solução pode não ser adequada por necessitar de muito esforço computacional.

Nenhum método de regressão para a determinação dos parâmetros de QoS foi utilizada no trabalho de Chen. Em vez disso, para parametrizar reservas utilizadas na instanciação de máquinas virtuais, foram criadas tabelas que associam o limite de consumo com as taxas de QoS obtidas. No entanto, os autores alertam que técnicas tais como a regressão linear poderiam ser utilizadas para estimar esses valores.

Dadas as características dos trabalhos envolvendo a previsão de desempenho e o fato de que não foram encontrados trabalhos que realizassem a decomposição de qualidade de serviço em reservas de recursos de processamento, disco e rede, decidimos propor um novo método que realizasse de maneira satisfatória a decomposição de QoS. Inspirada na mescla dos trabalhos de Urgaonkar e de Zhang [76, 83], esse método utiliza medidas de consumo de recursos das aplicações gerenciadas como variáveis preditoras de regressões lineares.

Em nossas regressões, temos por objetivo utilizar apenas medidas de consumo de recursos que sejam providas pelo próprio sistema operacional. Tais medidas, apesar de não proverem informação a respeito dos diferentes tipos de carga de trabalho gerada por uma aplicação, apresentam a vantagem de serem obtidas sem a necessidade de instrumentação do código original da aplicação. Em uma aplicação *web*, por exemplo, não precisamos de informações a respeito de cada tipo de navegação feita pelo usuário. Todos os tipos de navegação, tais como requisições de compra e busca de informações, são tratadas sem distinção.

Como em nossos experimentos três recursos foram utilizados no mapeamento de requisitos de QoS, algoritmos de força bruta para a combinação de reservas poderiam ser muito custosos. Desse modo, optamos por um método de

estudo do comportamento da aplicação quando a capacidade de processamento do servidor de entrada (servidor *web*) sofre variação. Esse método, assim como descrito no trabalho de Chen, tende a garantir valores mínimos de reserva de recursos.

### 4.8.1 Aplicação Alvo

Por se tratar de uma aplicação multicamadas complexa e que faz uso intensivo de todos os recursos gerenciados pelo ReservationSuite, o TPC-W [85] foi a aplicação escolhida para os experimentos envolvendo a decomposição de QoS. Entre outros tipos de aplicações sintéticas de múltiplas camadas, escolhemos o TPC-W porque ele é utilizado em muitos trabalhos relacionados ao nosso [76, 77, 81, 84].

O TPC-W consiste em um *benchmark* de transações *web*, cuja carga de trabalho simula as transações de uma loja virtual de livros do sítio *amazon.com*. Uma instalação do TPC-W pode abranger até 3 máquinas físicas: uma para o servidor *web*, uma para o servidor de aplicação (podendo esse último servidor ser hospedado juntamente com o primeiro) e outra para o banco de dados.

No TPC-W, o número de sessões de usuários — representadas pela emulação de navegadores (*browsers*) — é mantido constante durante sua execução. Assim, sempre que uma sessão é finalizada, uma outra é iniciada imediatamente. Para cada navegador emulado, o *benchmark* define estatisticamente o tamanho da sessão do usuário, os tempos de resposta desse usuário e as consultas geradas pela sessão.

As sessões do TPC-W incluem atividades (interações) de navegação e requisição realizadas a partir de 14 diferentes páginas *web*. Atividades de navegação, tais como leitura de texto das páginas, busca e seleção de produtos, são atendidas utilizando-se recursos mínimos do servidor *web*, enquanto atividades de requisição, tais como *login*, registro, pedido e confirmação de compra, são atendidas por meio de acessos frequentes ao banco de dados. De acordo com as porcentagens de atividades de navegação e requisição realizadas, pode-se classificar uma sessão como sendo de *navegação*, de *compra* ou de *pedido*. Sessões de *navegação* são sessões onde o usuário passa 95% do tempo navegando e apenas 5% do tempo fazendo pedidos; sessões de *compra* são sessões onde o usuário passa 80% do tempo navegando e 20% do tempo fazendo pedidos; e sessões de pedidos são sessões onde o usuário passa metade do tempo navegando e a outra metade fazendo pedidos.

Por gerar mais carga de processamento na camada de banco de dados do que os demais tipos de sessão, sessões de pedido foram escolhidas para serem

utilizadas nos experimentos. Considerando-se esse tipo de sessão, pudemos emular um número máximo de 300 navegadores, pois com um número maior que esse, os processadores da camada de banco de dados tendiam a ficar saturados muito rápido.

Por padrão, o TPC-W disponibiliza dois tipos de métricas de desempenho: o número de interações *web* completadas a cada segundo durante todo o tempo de execução do *benchmark* (*Web Interactions per Second* - WIPS) e os tempos de resposta de cada interação realizada (*Web Interaction Response Time* - WIRT). Apesar da métrica WIRT ser mais expressiva que a métrica WIPS, ela é mais difícil de ser obtida em aplicações reais, pois necessita que o código da aplicação seja instrumentado a fim de diferenciar os tipos de interações realizados pela aplicação. Assim, optamos pelo uso da métrica WIPS que, devido à sua granularidade mais grossa que a da WIRT, é mais simples de ser obtida.



### 4.8.2 Método Adotado

Uma forma natural de se modelar o TPC-W é relacionar, para cada camada da aplicação, o montante de recursos computacionais gastos para o tratamento de uma única interação *web*. A partir desse montante, é possível obter valores, tais como o tempo de serviço e o número de visitas, para, de acordo com um modelo de rede de filas, descobrir o número de interações a serem concluídas quando diferentes parametrizações de reserva são utilizadas. Essa abordagem, no entanto, tende a torna-se muito complexa quando novos elementos além do processamento são inseridos na análise.

A complexidade introduzida pelo uso de múltiplos recursos em um modelo de rede de filas é parcialmente causada pelo aumento do número de centros de serviço e o consequente acréscimo de informações necessárias para a análise do modelo. Assim, optamos por um método mais simples e limitado que expressa o consumo de recursos a partir da limitação de processamento da primeira camada do *benchmark*, no caso, o servidor *web* e de aplicação.

Em nosso método, para cada uma das taxas de limitação impostas ao servidor *web*, as frequências de acesso ao disco e à rede são observadas juntamente com o consumo de processamento da camada executora do banco de dados e o desempenho, em WIPS, obtido pelo TPC-W. Esses valores funcionam como preditores (*dataset*) em diversas regressões lineares simples (do termo inglês *Simple Linear Regression - SLR*). Para cada reserva de recurso a ser parametrizada, há uma regressão correspondente que relaciona valores de reserva com os respectivos números de interações *web* por segundo (WIPS)

atingidos. A ideia é que a dependência e a comunicação entre as camadas possam ser extraídas sem que modelos de interação entre os componentes da aplicação sejam necessários.

A figura 4.11 representa o método de parametrização que propomos. A partir do número de navegadores emulados (N), o parametrizador deve estimar os montantes de recursos necessários para se atingir o número de interações *web* por segundo (WIPS) desejadas. Cada regressor da figura é responsável por estimar o valor de reserva de um recurso. No nosso caso, trabalhamos com o TPC-W sendo executado em duas camadas e consumindo três tipos de recursos controlados pelo ReservationSuite. Assim, existem 6 regressores necessários para parametrizar a execução do *benchmark*. Toda parametrização deve ser estimada baseada no nível de confiança das predições. Esse nível tem por função corrigir possíveis erros de previsão de reserva que estimem para menos o consumo de recursos necessários para se atingir a meta de desempenho dada em WIPS.

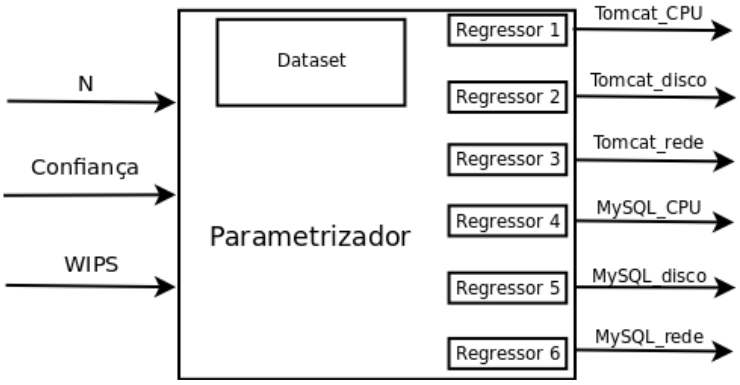


Figura 4.11: Representação do método de parametrização.

A figura 4.12 ilustra os detalhes do ajuste de erro aplicado ao regressor 1 da figura 4.11. A partir de um *dataset* previamente criado em um cenário controlado, o regressor deriva o valor da variável de resposta que, no caso, corresponde ao montante de reserva de processamento a ser destinado ao servidor Tomcat (Tomcat_CPU). Esse valor é então incrementado de acordo com o erro observado durante a fase de teste da regressão e de acordo com o nível de confiança desejado para a predição.

O intervalo de confiança das predições, por sua vez, é dado pela fórmula [78]:

$$pred\_conf = pred + t(conf, n)s_e\sqrt{1 + \frac{1}{n} + \frac{(x_p - \overline{x}^2)}{\sum x^2 - n\overline{x}^2}}$$

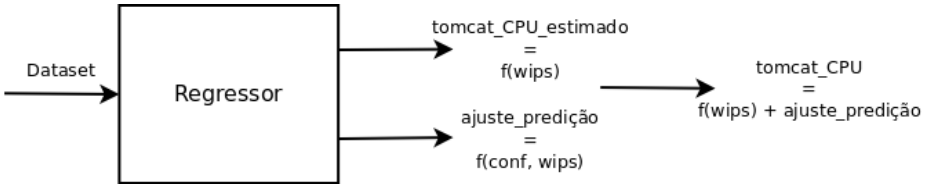


Figura 4.12: Detalhes do funcionamento de um regressor.

onde $s_e$ corresponde ao desvio padrão dos erros, *t(conf,n)* ao nível de confiança, $n$ ao número de observações da predição e $x$ aos valores observados (variáveis preditoras). O objetivo do emprego dessa fórmula é contornar erros de predição que estimem montantes de recursos inferiores aos necessários.

A partir da fórmula anterior nota-se que o desvio padrão da predição é mínimo no centro da faixa de valores medidos (x=$\overline{x}$) e aumenta conforme o valor de $x$ se distancia desse centro. Esse comportamento é apresentado na figura 4.13 [78]. Se valores muito além dos utilizados na definição do modelo de regressão forem utilizados, a variância e o intervalo de confiança da predição serão altos e, consequentemente, a acurácia da predição será baixa.

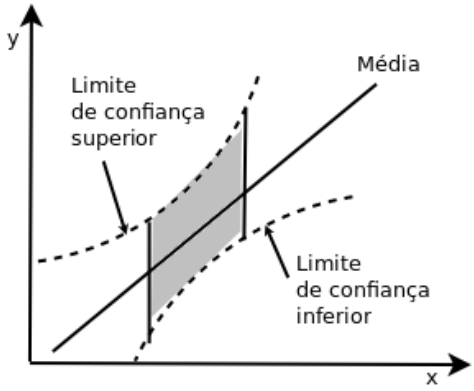


Figura 4.13: Intervalo de confiança para predições.

Segue ainda que, quanto maior o nível de confiança, maior a chance da reserva gerar um desempenho satisfatório, mas, por outro lado, maior a chance de haver uma subutilização do uso dos recursos nos casos em que a predição gerar valores de reserva maiores que os necessários. O uso do método apresentado será descrito e avaliado na Seção 4.8.3.

### 4.8.3 Experimentos

Os experimentos relativos à decomposição de QoS foram realizados em um cenário composto por três máquinas físicas idênticas: uma máquina para a execução do Tomcat, uma máquina para a execução do MySQL, e uma terceira para a emulação dos navegadores, assim como apresentado na figura 4.14. As máquinas utilizadas apresentavam processador Intel Pentium D 3.4 GHz dual core, com 2 G de memória e interligados por uma rede de 1 Gbit de velocidade.


Figura 4.14: Infraestrutura utilizada nos experimentos de decomposição de QoS.

Na fase de geração de dados para a formação do *dataset*, um ambiente foi projetado de modo que apenas o processamento da máquina onde o Tomcat executava foi limitado (servidor *web* e de aplicação). Os valores de reserva escolhidos para a camada do Tomcat foram 3, 5, 8, 10, 13, 15 e 18% da capacidade total de processamento da máquina, montantes que geraram uma vazão de 5 a 40 WIPS (interações *web* por segundo). A partir de reservas maiores que 18%, o processamento reservado passava a ficar ocioso e o número de WIPS obtido permanecia constante, tanto pela saturação do disco da camada de banco de dados, como também pelo alcance do limite de interações concorrentes geradas pelo TCP-W ao emular o número de 300 navegadores. Para cada uma das taxas de reserva configuradas, as frequências de acesso ao disco e à rede das camadas eram observadas juntamente com o consumo de processamento da camada executando o MySQL. Esses dados foram então processados e colocados em arquivos conforme o padrão especificado pela ferramenta de mineração de dados utilizada, o Weka [86].

Cada parâmetro de reserva de cada camada da aplicação precisa de um arquivo de dados de execução para guiar o aprendizado das funções de predição. O código a seguir apresenta parte de um arquivo de dados cujo objetivo é guiar o aprendizado do consumo de processamento necessário ao Tomcat de acordo com o número de WIPS desejado.

```
@relation cpu_tomcat
```

```
@attribute wips real
@attribute cpu real
@data
7.609   30
7.428   30
7.571   30
7.388   30
7.665   30
...
```

Nas primeiras linhas do arquivo são apresentadas algumas propriedades da regressão. Primeiramente, o nome da relação que será apresentada no arquivo é definida em `cpu_tomcat`. Em seguida, são apresentados os atributos da relação. Nesse caso, temos dois atributos reais, um expressando o número de WIPS observado e outro expressando o tempo de CPU, em milissegundos, destinado ao Tomcat a cada segundo para atingir o desempenho dado em WIPS. Por fim, são apresentadas colunas com as instâncias a serem estudadas pela regressão: na primeira coluna desse arquivo o número de WIPS é informado e na segunda coluna é informado o montante de CPU dedicado ao Tomcat para se atingir aquela taxa de interações *web*. No exemplo de arquivo apresentado, todas as medidas de WIPS foram obtidas com o Tomcat sendo limitado a 30 ms de CPU a cada 1000 ms, ou seja, a reserva de processamento do Tomcat foi parametrizada em 3% da capacidade total da máquina executora.

Cada taxa de processamento do Tomcat apresenta 14 entradas nos arquivos de aprendizado. Esses números são provenientes de duas execuções do *benchmark*, cada execução gerando 7 pontos no plano da regressão. Assim, cada *dataset* utilizado em nossos experimentos apresenta 98 pares (WIPS, taxa de processamento), visto que são analisados 7 valores de uso de processamento para o Tomcat, cada um apresentando 14 medidas de WIPS.

Estatisticamente, arquivos de aprendizado devem apresentar dados que, por serem obtidos por amostragem, devem ser representativos da população sendo estudada. Assim, quanto mais diversificado for um *dataset*, mais representativo ele tende a ser. Consequentemente, mais precisas serão as regressões baseadas nesse *dataset*. Apesar de nossos *datasets* serem pequenos, contendo apenas 98 entradas, acreditamos que eles sejam suficientemente expressivos para nossos propósitos, visto que foram construídos a partir da observação do consumo de recursos em ambientes com capacidades computacionais variadas (porcentagem de processamento dedicado variou de 3 a 18%).

A figura 4.15 apresenta a variação no uso de CPU do Tomcat quando o mesmo não tem o seu consumo de processamento limitado. Nesse caso, a

média do uso de processamento dessa aplicação é de 17.3% com desvio padrão médio de 7%.

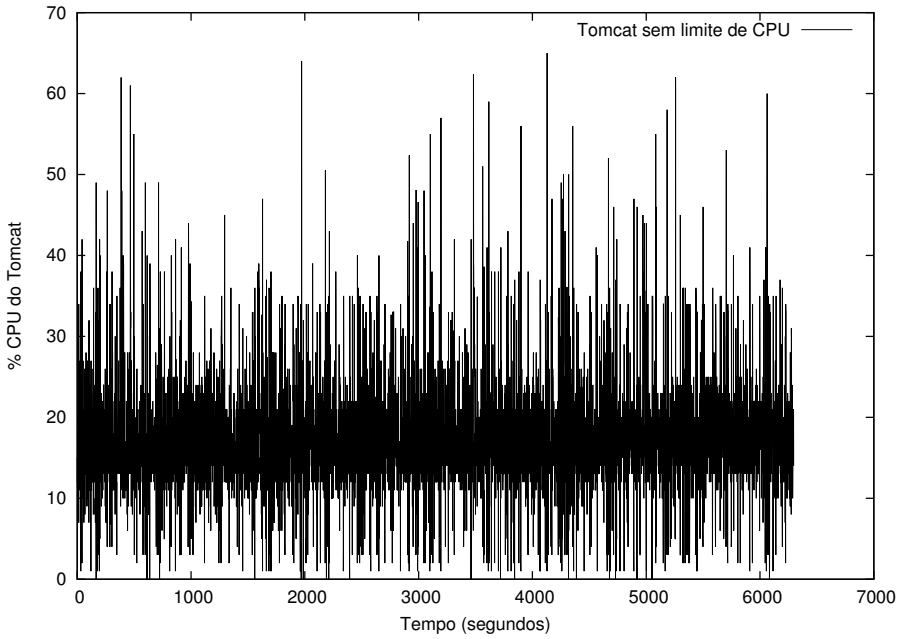


Figura 4.15: Consumo de CPU do Tomcat quando essa aplicação não tem o processamento limitado.

A figura 4.16, por sua vez, apresenta a variação no uso de CPU do Tomcat quando o mesmo é limitado a utilizar apenas 8% da capacidade de processamento da máquina. Nessa figura, são apresentados os resultados de dois treinos com o Tomcat limitado a 8% de processamento. Nota-se que as curvas se sobrepõem. A média do uso de processamento observada é de 9.1%. É possível notar que essa média varia menos do que na figura 4.15, com um desvio padrão médio de 1.9%. Esse decréscimo na variação é consequência do uso de nossa ferramenta de controle de uso de processamento. Infelizmente, não foi possível reduzir essa variação a um número mais próximo de zero. Acreditamos que essa impossibilidade ocorra por causa do grande número de *threads* da aplicação sendo gerenciada. Quando o servidor do ReservationSuite envia um sinal para suspender o processamento do Tomcat, por exemplo, há um retardo no recebimento de tal sinal em algumas das *threads* da aplicação, principalmente quando a máquina hospedeira é multiprocessada. Além disso, pode acontecer de uma *thread* em execução ainda não ter tido a sua criação detectada pelo servidor do ReservationSuite e, assim, não receber o sinal de suspensão de execução naquele período.

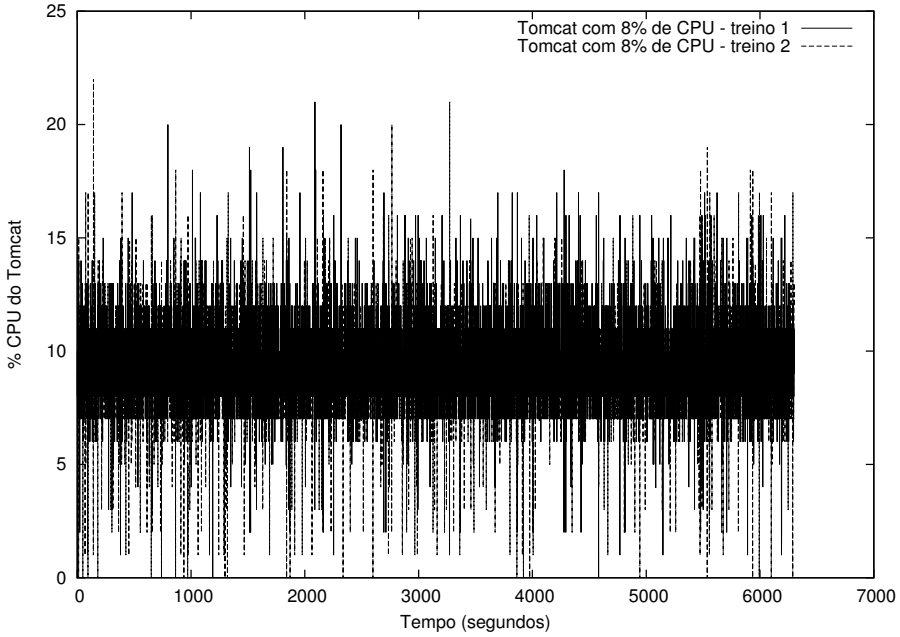


Figura 4.16: Consumo de CPU do Tomcat quando essa aplicação é limitada a 8% do processamento total da máquina.

De maneira geral, os consumos dos recursos de processamento, disco e rede para o Tomcat e para o MySQL apresentam uma convergência com significativa variância, a qual é determinada principalmente pela maneira como a carga de trabalho (perfil de navegação) é gerada no TPC-W. Devido a essa alta variância, o uso da média de consumo de recursos como uma medida para garantir a qualidade de WIPS no TPC-W mostrou-se inadequado. A análise do consumo dos recursos no experimento baseado em médias revelou que as reservas não foram integralmente utilizadas, o que resultou no não cumprimento da meta de finalização de interações *web* por segundo. Durante os experimentos, notou-se que, em diversos instantes, o processamento da máquina hospedeira encontra-se totalmente ocioso. Esse fato revela que existem gargalos de desempenho criados por predições mal conduzidas. O problema acontece porque dado que o tráfego de requisições no TPC-W se dá em rajadas, ao limitar as reservas de recursos ao valor médio verificado durante a geração de dados, as requisições submetidas em momentos de grande demanda não são atendidas no tempo verificado na fase anterior. Quando a demanda de recursos aumenta e a reserva é totalmente utilizada, o processo o qual excedeu a sua fatia de uso é suspenso até o próximo período, retardando assim a resposta à camada requisitante do TPC-W.

### Predições Baseadas em 5-Percentil

Para contornar o problema da grande variação de demanda de serviço, decidimos utilizar uma outra medida de uso de recursos que atendesse melhor às requisições nos momentos de grande número de pedidos. A solução encontrada foi utilizar uma estimativa pessimista e basear as predições na média dos 5% maiores índices de consumo para cada tipo de recurso. Assim, ignoramos os 95-percentil inferiores das medições realizadas na fase de geração de dados. Com essa política, novos experimentos foram conduzidos para satisfazer as metas de 5, 10, 15, 20, 25, 30 e 35 WIPS. Os parâmetros de reserva para a meta de 15 WIPS são apresentados na segunda coluna da tabela 4.3. Salvo a estimativa de consumo de processamento do Tomcat que permaneceu inalterada nesse novo experimento, esses parâmetros são bem maiores que os utilizados na predição baseada na média de consumo de recursos. Esse fato é consequente da grande variação no uso de recursos pelo TPC-W.

| WIPS | Média Perc. | Utilização |
|---|---|---|
| $Tomcat\_CPU$ | 7% | 10% |
| $Tomcat\_disco$ | 703277 (B) | 478006 (B) |
| $Tomcat\_rede$ | 1009849 (B) | 697273 (B) |
| $MySQL\_CPU$ | 134% | 40% |
| $MySQL\_disco$ | 98844765 (B) | 21944782 (B) |
| $MySQL\_rede$ | 343337 (B) | 170885 (B) |

Tabela 4.3: Estimativas de reserva para se atingir 15 WIPS a partir de predições baseadas no 5-percentil de consumo de recursos e a utilização de recursos quando as reservas são implantadas.

A figura 4.17 apresenta o consumo de processamento do Tomcat quando as reservas do TPC-W foram parametrizadas com os valores percentis da tabela 4.3. Diferentemente do que ocorreu nos experimentos das regressões baseadas em médias, nesse experimento, a reserva de processamento do Tomcat foi, em média, integralmente utilizada (utilização média de 10% de acordo com a tabela 4.3), ou seja, os retardos introduzidos pelo limite de uso de recursos do ReservationSuite não foram demasiados de forma a gerar gargalos de desempenho. O resultado dessa boa utilização de processamento na camada pertencente ao Tomcat garantiu que os números de interações *web* estipuladas antes da execução dos experimentos fossem alcançados, assim como ilustrado na figura 4.18.

A figura 4.18 apresenta as médias de WIPS obtidas a partir de regressões baseadas em valores médios, percentis e percentis com um nível de confiança de 90% e as põe em comparação com as médias que deveriam ser obtidas em uma predição perfeita (regressão ideal). A partir dessa figura, nota-se que os

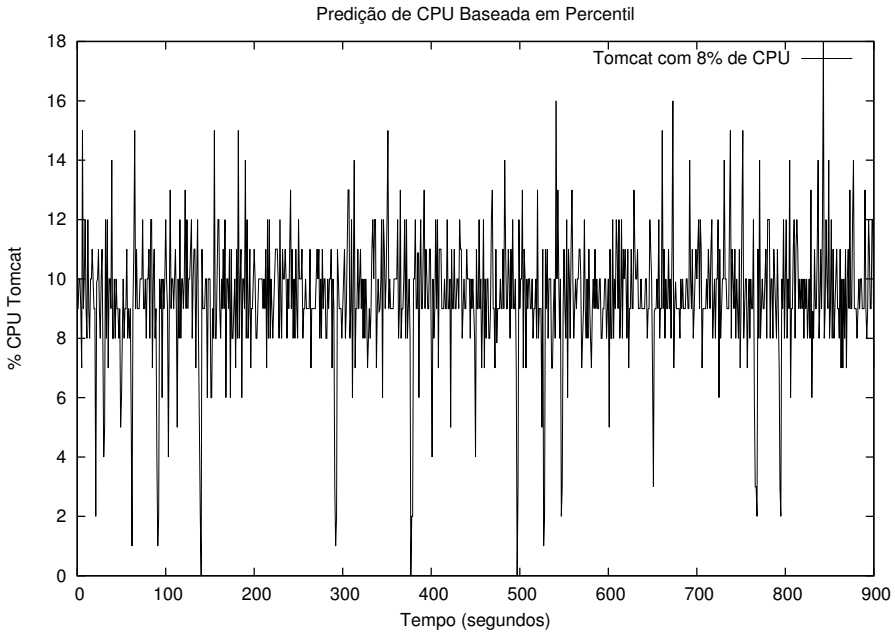


Figura 4.17: Consumo de processamento do Tomcat quando os recursos são limitados de acordo com os valores médios da tabela 4.3.

números de WIPS obtidos com o valor percentil poderiam ser até menores que não comprometeriam a qualidade esperada, ou seja, as reservas poderiam ter o montante de recursos requisitados reduzido.

O resultado da regressão baseada em percentil com nível de confiança ficou muito próximo ao resultado que não utilizou esse artifício. No entanto, a parametrização gerada quando o intervalo de confiança é utilizado é significativamente maior, assumindo os valores de 8%, 821709 B e 1166582 B para o processamento, a largura de banda de disco e a largura de banda de rede, respectivamente, para o Tomcat; e os valores de 151%, 113261334 B, 375419 B para o processamento, a largura de banda de disco e a largura de banda de rede, respectivamente, para o MySQL. Calibrar as predições com nível de confiança de 90% é, nesse caso, desnecessário visto que o número de interações aumenta muito pouco comparado ao montante de recursos que devem ser alocados para atingir o nível de confiança.

Diferentemente da reserva de processamento do Tomcat, as demais reservas garantidas pelo ReservationSuite ao TPC-W foram subutilizadas em alguns momentos porque as mesmas não poderiam ser decrementadas, já que deveriam estar disponíveis sempre que um pico de demanda de serviço acontecesse. Esse fato pode ser comprovado verificando-se a utilização das reservas, a qual é exposta na terceira coluna da tabela 4.3.

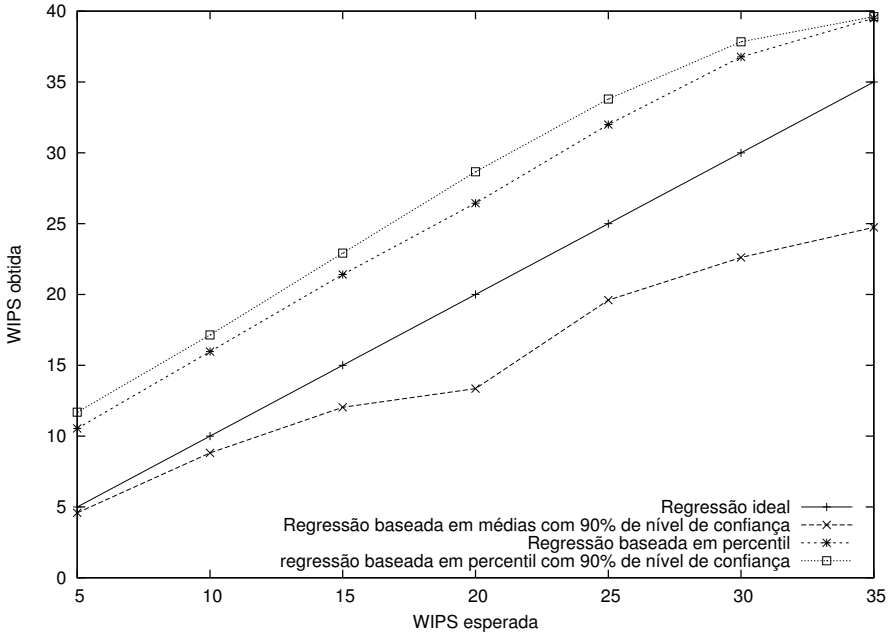


Figura 4.18: Médias de WIPS obtidas a partir das regressões baseadas em valores médios e em percentis.

É importante frisar que, como exposto no início deste capítulo, o objetivo dos nossos experimentos não é prover um método de regressão altamente preciso, mas mostrar que, com o uso de uma técnica bastante simples, é possível parametrizar reservas no ReservationSuite de forma satisfatória utilizando apenas medidas de desempenho providas de forma nativa pelo sistema operacional, sem requerer alterações no código da aplicação. No entanto, novos estudos que determinem medidas mais precisas para a provisão de QoS poderiam ser conduzidos. Em especial, a facilidade de adaptação de reservas provida pelo ReservationSuite poderia ser utilizada por políticas que, baseadas nos padrões de uso de recursos de uma aplicação, aumentassem/diminuíssem o montante de reserva de recursos à medida que novas requisições fossem feitas/atendidas. Em aplicações com picos de utilização de recursos, tal como o TPC-W, essas medidas poderiam ser capazes de garantir a qualidade de serviço esperada, ainda que reduzindo algumas reservas a fim de ceder recursos temporariamente a aplicações com maiores necessidades. Em casos extremos, um esquema de *overbooking*, em que se reserva mais recursos do que o que há disponível, poderia ser implementado. Em todos os casos envolvendo adaptação de reserva, porém, é preciso que a política de aumento/decréscimo de reservas seja fortemente ligada ao controlador de admissão do sistema para que degradações de

desempenho não ocorram além das taxas previstas.

## 4.9 Uma Nova Avaliação da Eficácia das Reservas de E/S

Na Seção 4.4, apresentamos alguns experimentos para a verificação da eficácia das reservas de largura de banda de disco. Porém, nesses experimentos foram executadas somente aplicações de acesso intenso e contínuo ao disco, sem que nenhuma aplicação de acesso mais aleatório, tal como o TPC-W, fosse utilizada. Assim, ainda é preciso determinar o comportamento das reservas de largura de banda de disco em aplicações que acessam essa mídia de armazenamento de forma aleatória e, possivelmente, de forma concorrente com outras aplicações de uso mais intensivo da largura de banda de disco.

Para avaliar a eficácia das reservas do TPC-W em cenários onde a infraestrutura de execução é compartilhada com cargas externas de acesso ao disco, alguns experimentos foram realizados com o TPC-W sendo inicializado a partir do ReservationSuite e com reservas infinitas de processamento, largura de banda de disco e de rede. Com essa configuração, garante-se que nenhum gargalo de desempenho é inserido no *benchmark*. A ideia é que o TPC-W tenha acesso aos recursos computacionais sempre que for necessário de forma que o desempenho obtido com ou sem carga externa seja o mesmo.

Como carga externa foi utilizado um processo que lia constantemente um arquivo de 4 G de tamanho. A escolha de uma carga dessa característica é importante porque satura o disco ao não conseguir armazenar todo o arquivo em *cache*. Essa carga também maximiza a concorrência pelo acesso ao disco ao utilizar uma operação de E/S que, assim como as operações do TPC-W, será priorizada pelo mecanismo de *ionice* do Linux quando executada a partir do ReservationSuite. Uma carga que realize operações de escrita em disco não é capaz de influenciar de tal maneira a execução de outras aplicações do mesmo servidor físico, pois operações de escrita não são tratadas de forma diferenciada pelo escalonador de E/S do Linux, como mencionado na Seção 3.2.2.

Nos primeiros experimentos, as utilizações de disco geradas pelo TPC-W e pela carga externa foram medidas. Notou-se que a camada do Tomcat tem utilização de disco próxima a zero (muitas operações de leitura com dados, muitas vezes, contidos em *cache*), enquanto a camada do MySQL tem utilização média de 12%. A carga externa gera uma utilização de 20% para cada dezena de Mbytes lidos por segundo e 100% de utilização quando executada sem limitação de uso de largura de banda de disco. Assim, ao requisitar uma taxa de leitura de aproximadamente 50 Mbytes/segundo ou maior, essa aplicação causa a saturação do disco.

Na figura 4.19 são apresentados os resultados de experimentos em que as camadas do Tomcat e do MySQL são executadas sem carga externa e com diferentes cargas externas parametrizadas com largura de banda de disco igual a 10, 20, 30, 40 e 50 Mbytes/segundo. As duas linhas horizontais desenhadas no gráfico representam o desvio padrão médio de 10 medições no número de WIPS obtido pelo TPC-W quando essa aplicação foi executada sem a interferência de uma carga externa.

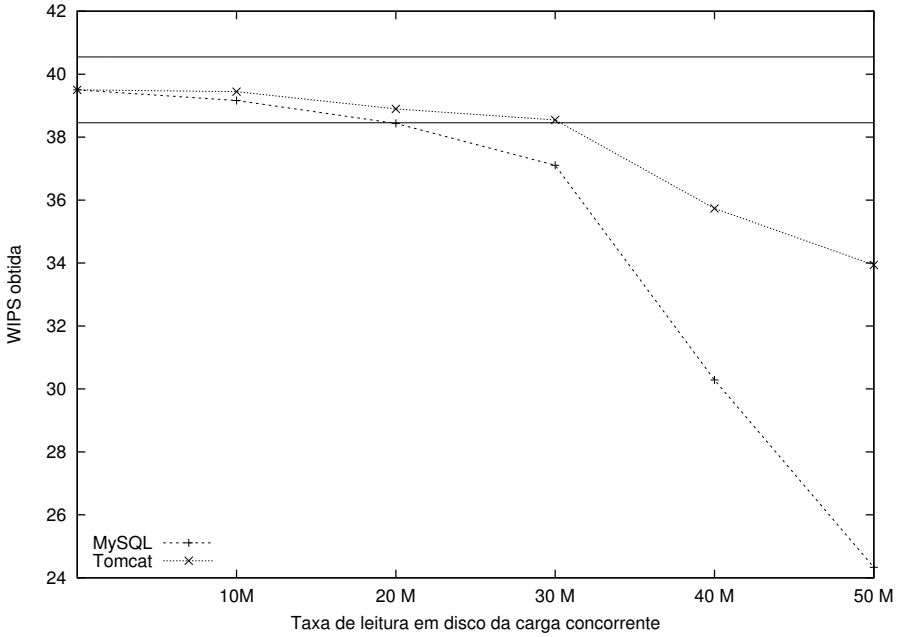


Figura 4.19: Vazão obtida pelo TPC-W ao ter o Tomcat e o MySQL executados concorrentemente com outra carga de trabalho.

Pela figura 4.19 nota-se que, com carga externa limitada até 30 Mbytes/segundo, o Tomcat sofre uma degradação de desempenho tão leve que os valores de WIPS obtidos estão incluídos no intervalo de desempenho esperado o qual abrange o número de WIPS obtido quando não há carga externa no sistema. A degradação sofrida pelo MySQL quando essa camada é executada concorrentemente com a carga externa também é mínima até o uso de 20 Mbytes por segundo pela carga externa.

A partir de 30 Mbytes/segundo para o Tomcat e 20 Mbytes/segundo para o MySQL, a utilização do disco passa a ser disputada com a carga externa, causando um gargalo no acesso ao disco. Consequentemente, o desempenho do TPC-W passa a degradar mais intensamente. O comportamento observado confirma a importância do papel de um controlador de admissão juntamente

com o ReservationSuite. Recursos computacionais que encontram-se saturados de reservas não devem ter novos pedidos de uso aceitos, caso contrário, não será possível garantir a qualidade de serviço esperada para as aplicações.

É importante notar também que a mesma carga externa executada sem o uso do ReservationSuite não foi capaz de interferir significativamente no resultado gerado pelo TPC-W porque as operações da carga externa apresentavam, nesse caso, menor prioridade de acesso ao disco do que o Tomcat e o MySQL.

## 4.10 Considerações Finais

Este capítulo apresentou experimentos para a verificação das funcionalidades providas pelo ReservationSuite, assim como para a exposição das limitações desse conjunto de ferramentas. Nesses experimentos foram expostas algumas formas de contornar tais limitações a fim de garantir um nível de confiabilidade aceitável das reservas de recursos computacionais.

Os resultados dos experimentos mostram que o ReservationSuite é eficaz na tarefa de limitar e garantir o consumo de recursos a aplicações. As conclusões apontam que o controle de admissão de novas reservas no ReservationSuite deve ser acompanhado de mecanismos que determinem a saturação dos dispositivos de E/S. Estabelecer valores de saturação é uma tarefa difícil, principalmente em mídias tais como os discos rígidos, e, por esse motivo, essa tarefa deve ser realizada por módulos complementares à nossa ferramenta.

As sobrecargas de trabalho causadas pela interposição de funções podem ser evitadas em situações em que o servidor estiver gerenciando apenas uma aplicação ou em situações em que as aplicações em execução, garantidamente, não geram contenção no acesso ao disco/rede. Nesses casos, pode-se deixar a aplicação consumir o montante de largura de banda de disco e rede que for necessário.

A dificuldade de se controlar o acesso à mídia de armazenamento tende a ser atenuada com o uso de unidades de estado sólido (do termo inglês *solid state drive* - SSD) as quais, por não possuírem partes móveis, tal como a cabeça do disco, apresentam tempo de acesso reduzido. No entanto, o alto custo de aquisição de unidades de estado sólido ainda deve motivar o estudo de alternativas de controle de acesso ao disco rígido por mais alguns anos. Além disso, mesmo com a existência dos SSDs, nosso controle de acesso à mídia de armazenamento ainda se aplica em situações em que essa mídia representa um gargalo de desempenho.

Este capítulo também apresentou um estudo de caso sobre a decom-

posição de QoS. O objetivo desse estudo consistiu em mostrar que o ReservationSuite apresenta reservas eficazes no gerenciamento de consumo de recursos de aplicações multicamadas que, assim como o TPC-W, fazem uso intensivo e paralelo dos recursos de processamento e E/S de dados. Para isso, foi preciso definir um método para a decomposição de QoS que previsse o consumo de recursos de processamento, largura de banda de disco e largura de banda de rede, e que fosse simples de ser implementado, resultando em estimativas bem sucedidas, mas sem a expectativa de que a solução proposta representasse o estado da arte na decomposição de QoS.

Analisar o quanto nosso método de decomposição se aproxima dos trabalhos mais relevantes desse tema consiste em uma tarefa difícil, dado que apenas o trabalho de Chen [81] descreve um método completo para a decomposição de QoS. Ainda assim, o trabalho de Chen apresenta algumas diferenças. Primeiro, esse trabalho trata somente a decomposição de QoS em recursos de processamento. Segundo, ele caracteriza o tempo de resposta obtido por cada tipo de transação quando diferentes montantes de recursos são alocados, e não o número de WIPS obtido por um tipo de combinação de transações quando se varia o montante de recursos alocados, tal como o nosso método propõe. Dados a respeito dos tipos de interações presentes no TPC-W são mais informativos e, por esse motivo, geram previsões mais precisas. Chen relata pouco a respeito dos resultados obtidos em seu trabalho, mas com base no que há documentado, pode-se dizer que a previsão de tempo de resposta apresenta erros menores que 30%, enquanto a combinação de reservas gera uma subutilização de processamento menores que 20%.

A necessidade de simplicidade na definição do método guiou algumas decisões. Primeiro, mesmo sabendo que extrair um perfil de consumo de recursos próprio para cada tipo de interação do TPC-W poderia gerar predições precisas, optamos por caracterizar todos os tipos de interação com um único perfil de uso de recursos. Essa prática nos garantiu que o código das camadas do TPC-W não precisariam ser alterados, tal como aconteceu no trabalho de Chen. A segunda decisão foi não utilizar rede de filas para modelar a arquitetura do TPC-W. Decidimos, em vez disso, estudar o impacto que a variação de desempenho de uma camada da aplicação gera nas demais através da aplicação de limites de consumo de processamento à camada de entrada do TPC-W. Os dados gerados nos sucessivos ajustes de limites alimentavam arquivos utilizados na estimativa de reservas através da técnica de regressão. Uma das vantagens do uso da nossa modelagem está na direta definição dos recursos mínimos das camadas de *software* a partir do estudo da parametrização de um único recurso da camada de entrada da aplicação,

ou seja, com a nossa abordagem, a fase de parametrização de reservas é simplificada por não resultar em problemas de otimização combinatória que, nesse caso, envolveriam um grande número de variáveis.

Os resultados dos experimentos com o método empregado na previsão de QoS mostram que é possível estimar parâmetros de reserva com precisão sem o uso de técnicas avançadas de planejamento de capacidade. Nos experimentos realizados, o uso do 5-percentil de consumo de cada recurso, em duas execuções prévias do TPC-W, foi suficiente para garantir a qualidade de serviço esperada e parametrizar as reservas de uso de processamento, disco e rede. Pode-se dizer que os valores de reserva encontrados a partir dessa média consistem em estimativas sobrestimadas (frouxas), visto que nenhuma das fatias de reserva foi integralmente utilizada. Uma forma de calibrar as estimativas é utilizar índices que expressem a dispersão em torno da média do uso de recursos no desenvolvimento de novas políticas de previsão de desempenho. Políticas com índices de dispersão podem aumentar a precisão das predições, reduzindo o montante de recursos subutilizados. Melhorias também podem ser obtidas com o aumento do número de pontos preditores na regressão linear, pois quanto mais pontos são utilizados na fase de aprendizado da função preditora, maior a precisão das predições. Porém, é importante dizer que, mesmo com o ajuste de estimativas de reserva, devido à característica de carga de trabalho do TPC-W, com períodos de rajadas onde ocorrem grandes variações no número de requisições por segundo, as reservas de recursos tendem a ser subutilizadas para poderem atender às requisições dos clientes na vazão desejada.

Este capítulo encerra as discussões sobre a viabilidade de se implementar mecanismos de reserva de recursos no nível do usuário e sobre a eficácia desses mecanismos em determinados contextos. A partir do estudo de caso apresentado, foi possível identificar duas novas limitações do ReservationSuite. A primeira dessas limitações diz respeito à dificuldade de se enviar sinais às *threads* de uma aplicação. Muitas vezes, o monitor da nossa ferramenta de reservas ainda não detectou a criação de uma nova linha de execução e assim não envia um sinal de interrupção de execução a ela. Outras vezes, o monitor até envia um sinal de parada à *thread*, mas, por limitações do próprio sistema operacional quando executado em máquinas multiprocessadas, esse sinal demora a chegar até o destino.

Uma outra limitação evidenciada em nosso estudo de caso está relacionada à dificuldade de se controlar o envio/recebimento de dados dos dispositivos de E/S quando as operações são realizadas de maneira síncrona. Bloquear uma seção crítica até que a *thread* executando a seção envie/receba os dados pelos quais espera, pode gerar inanição das outras linhas de execução da

aplicação. Assim, optamos por realizar dentro da seção crítica somente o débito de *bytes* a serem lidos/escritos. Saindo dessa seção, as operações síncronas são realizadas de maneira atômica.

No próximo capítulo, são apresentadas uma síntese das conclusões do estudo realizado, as contribuições desse estudo e alguns temas para trabalhos futuros.