Classes de Serviço em Servidores Web Apache Através de Escalonamento Adaptativo e Controle de Admissão

Tiago Semprebom
Prog. Pós Graduação em Eng. Elétrica
Univ. Federal de Santa Catarina
55 48 3331-7650
tisemp@das.ufsc.br

Rômulo de Oliveira
Depto de Automação e Sistemas
Univ. Federal de Santa Catarina
55 48 3331-7677
romulo@das.ufsc.br

Carlos Montez
Depto de Automação e Sistemas
Univ. Federal de Santa Catarina
55 48 3331-7650
montez@das.ufsc.br

ABSTRACT
In this paper, adaptive scheduling based on dynamic priorities and imprecise computation is used in web servers. The goal is to keep proportional differentiation services. An approach with admission control has its properties of controllability and predictability evaluated. Another contribution is to show how to use and implement this approach in Apache web servers.

RESUMO
Neste artigo, técnicas de escalonamento adaptativo com prioridades dinâmicas e computação imprecisa são empregadas em servidores web. O principal objetivo é manter diferenciação de serviços proporcional. Um controle de admissão é adicionado à abordagem e suas propriedades de controlabilidade e previsibilidade são avaliadas. Outra contribuição do artigo é o de mostrar como implementar essas abordagens utilizando Apache.

Categories and Subject Descriptors
D.4.7 [**Organization and Design**]: Distributed systems, Interactive systems, Real-time and embedded systems.

General Terms
Measurement, Performance, Experimentation, Standardization.

Keywords
QoS, Differentiated Services, Web servers, Apache.

1. INTRODUÇÃO
Recentemente, têm surgido diversos trabalhos que buscam oferecer qualidade de serviço em servidores web [2-9]. Essas pesquisas complementam outras que atuam na infraestrutura da rede [1]. A impossibilidade de delimitar a carga desses servidores leva a situações de sobrecargas transientes, nas quais os tempos de resposta das requisições ultrapassam valores aceitáveis para usuários humanos. Nesses casos, quando os agentes de usuário (ex. navegadores web) implementam valores para *timeout*, todos os recursos (espaço em fila, tempos de processador, acesso a disco, etc) envolvidos na requisição acabam sendo desperdiçados.

Grande parte das pesquisas no tema propõe controle de admissão das requisições [2,3,9,14]. O objetivo é, em situações de sobrecarga, reduzir a carga e evitar desperdício de recursos, impedindo o efeito dominó dos cancelamentos de requisições. Nesse tema também existe interesse em aplicar técnicas de controle (*feedback scheduling* [3,9]) para implementar o controle de admissão.

Além dessas, outras abordagens propõem diferenciação de serviços nos atendimentos às requisições. Na impossibilidade de se oferecer um serviço diferenciado individual, visando a escalabilidade, essa abordagem sacrifica a QoS individual de cada requisição, lidando com um número reduzido e conhecido de classes de serviço. Para cada requisição ao servidor, classifica-se e agrega-se requisições, oferecendo-se diferenciação de serviços por classes de serviço.

O mecanismo usado para diferenciar os serviços dessas classes pode ser através de atribuição de prioridades [5], do controle do tamanho da fila de requisições, ou do número de *threads* ou de processos que executam as requisições, ou mesmo atuando na alocação de recursos no nível do sistema operacional [6].

Além do controle de admissão, com objetivo de controlar a carga de servidores, outras abordagens podem adotar técnicas como a do escalonamento adaptativo por computação imprecisa [2]. No caso de servidores web pode-se implementar páginas web com duas versões: uma versão completa (precisa) e outra parcial (imprecisa). Um servidor implementado segundo esse modelo, em uma situação de sobrecarga temporária, pode responder a uma requisição de cliente com uma resposta parcial, mas que seja suficiente para que o cliente tenha a percepção que foi atendido, ainda que, com uma qualidade reduzida. Acessos a servidores implementados segundo esse modelo apresentam a propriedade de "polimorfismo temporal", onde os resultados dependem das condições de carga do servidor.

Em abordagens onde se aplica controle de admissão, um servidor que simplesmente descarte quase todas requisições tende a ter tempos de resposta pequenos. Um situação semelhante pode ocorrer com abordagens que adotam o modelo de computação imprecisa e executam quase todas as requisições de forma imprecisa. Por esse motivo, o tempo de resposta das requisições não é uma métrica adequada para avaliar a abordagem, pois sozinha não consegue mensurar a situação de uma execução imprecisa ou de uma requisição descartada.

Este artigo propõe e avalia abordagens de escalonamento adaptativo por computação imprecisa com e sem controle de admissão. *Deadlines* absolutos são atribuídos às requisições, comparados com seus tempos de resposta e os resultados servem para guiar às políticas da abordagem de escalonamento. Diferentemente de muitos trabalhos [6,7,14,15] em que a métrica usual é o tempo de resposta, o conceito de valor cumulativo [12] é adotado. Um controle de admissão é incorporado na abordagem PCV (*Proportional Cumulative Value Attribution*) [8] e seu comportamento é avaliado com experimentos.

Este texto está dividido em 6 seções. A seção 2 trata sobre técnicas de diferenciação de serviço e computação imprecisa. Na seção 3 são descritos o modelo de adaptação e suas políticas. Na seção 4 o modelo tem sua implementação descrita em um servidor web Apache. Na seção 5 o modelo é avaliado através de simulações e de um protótipo. Na seção 6 são colocados os comentários finais.

2. DIFERENCIAÇÃO DE SERVIÇOS E COMPUTAÇÃO IMPRECISA EM SERVIDORES WEB

O crescimento do comércio eletrônico, e de outros serviços oferecidos pela Internet, tornou premente a necessidade de identificar, classificar e atender as demandas de clientes de forma diferenciada. Técnicas atuais de negócios como o CRM (*Customer Relationship Management*) propõem abordagens nesse sentido.

Essa abordagem é importante em momentos onde a demanda ultrapassa a capacidade. No entanto, é desnecessária nas outras situações pois todas as requisições deverão ser atendidas dentro de prazos satisfatórios.

Em muitos casos, lidar com estados individuais de cada requisição pode ser inviável. No caso de servidores web, estes podem possuir um grande número de clientes, fazendo requisições aperiódicas, o que torna difícil a manutenção da informação do estado de cada cliente. Esse problema de escalabilidade pode ser resolvido de forma similar à arquitetura DiffServ [10], agregando requisições de clientes em classes de serviço.

No caso de servidores da Internet, uma diferenciação de serviços pode ser baseada nas identidades dos clientes; nas suas localizações; no tipo dos conteúdos acessados; ou nas identidades das pessoas que criaram esses conteúdos. Está fora do escopo deste trabalho tratar como é implementada essa classificação, a qual pode ser efetuada por um módulo no servidor, ou ter origem em um datagrama IP previamente marcado pelo cliente ou por um roteador de borda [10].

Neste trabalho o interesse se concentra no modelo de diferenciação de serviços proporcional (relativo proporcional) [1]. Nesse modelo o que se busca é, independente da carga em cada agregado, manter os valores proporcionais de QoS entre as classes. Aplicações que executam na Internet estão sujeitas a cargas que podem crescer arbitrariamente. Essa particularidade torna inadequado qualquer esquema estático de gerenciamento de recursos (*ex.* políticas de atribuição estática de prioridades).

O modelo de diferenciação proporcional considera dois importantes atributos: (i) *controlabilidade* – capacidade de operadores de redes ajustarem o espaçamento de níveis de qualidade entre as classes; e (ii) *previsibilidade* – capacidade de um sistema manter consistente a diferenciação entre as classes, independente de variações na carga.

2.1 Computação Imprecisa

Neste trabalho, para manter a diferenciação de serviços proporcional entre as classes, adotou-se duas técnicas complementares: atribuição dinâmica de prioridades e computação imprecisa em servidores web. A computação imprecisa – técnica proposta na área de escalonamento tempo real [2] – é implementada neste trabalho com a abordagem de múltiplas versões: no caso, cada página web possui duas versões. Considerando o protocolo HTTP 1.1, todos os objetos de uma página web, requisitados em uma mesma conexão, serão atendidos em uma das duas versões: precisa ou imprecisa.

Para implementação de versões imprecisas em páginas web, pode-se empregar [2]: redução de tamanho de imagens JPEG através de compressão "agressiva"; redução de objetos embarcados em páginas web; e a eliminação de recursos visuais extras, tais como animações *gif* e *flash*.

Páginas web relacionadas ao comércio eletrônico costumam ter muitos objetos visuais. Um levantamento efetuado recentemente na Internet encontrou diversas páginas web com até 50 imagens em formatos gif e png. Essa característica permite uma grande melhoria de desempenho, se considerado o caso extremo de confecção de versões imprecisas dessas páginas com apenas textos em HTML.

Buscando avaliar as vantagens dessa abordagem, experimentos foram efetuados, escolhendo-se uma página web de um sítio de comércio eletrônico popular e implementando-se uma outra versão, imprecisa, com apenas textos. A Figura 1 mostra os tempos de resposta fim a fim (medidas no próprio cliente) obtidos para as requisições para esses dois tipos de páginas. Para esses testes, as requisições foram geradas através do software gerador de carga sintética *httperf*[1] em um servidor web Apache[2].

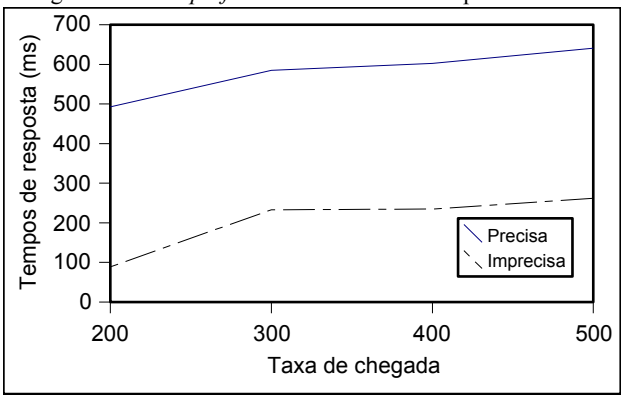

Figura 1. Tempos de resposta fim a fim de versões de uma página web de comércio eletrônico.

Apesar dos tempos absolutos mostrados na Figura 1 não serem relevantes, pois dependem da implementação e configuração do servidor web, da carga da rede, da velocidade do processador, e de outros parâmetros, é possível observar que os tempos de resposta da versão imprecisa podem alcançar menos de 30% dos tempos da versão precisa.

3. MODELO DE ADAPTAÇÃO

O modelo de adaptação considera a existência de diferentes classes de serviço. Sem perda de generalidade, na Figura 2 são representadas apenas três classes – ouro, prata e bronze – caracterizando um serviço olímpico [1].

Cada classe de serviço possui uma fila FIFO (por ordem de chegada) associada, que armazena suas requisições. Em cada fila, todas as requisições enfileiradas possuem a mesma prioridade da classe, a qual é atribuída dinamicamente. Essas prioridades são usadas para ordenar as requisições ao servidor web.

[1] http://www.hpl.hp.com/research/linux/httperf/

[2] http://www.apache.org/

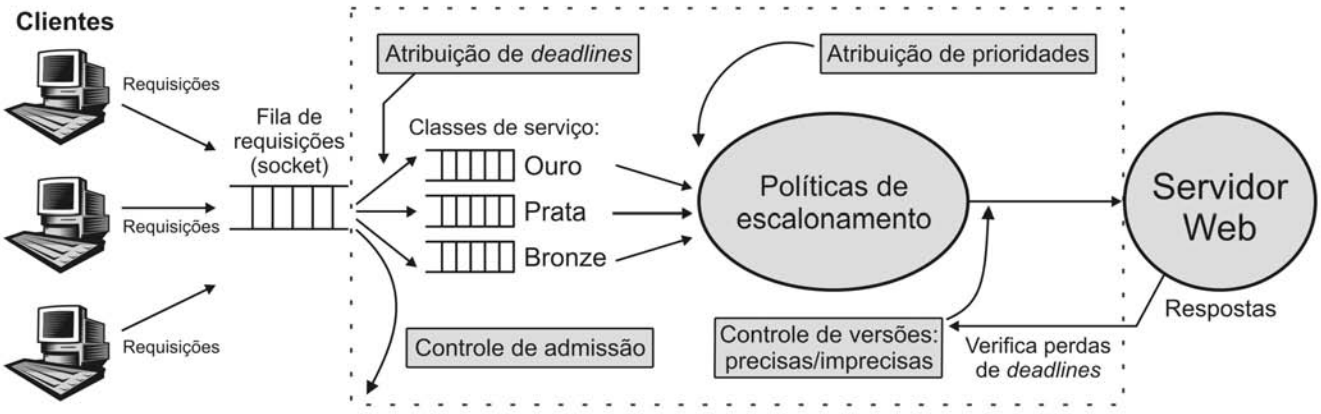

Figura 2. Modelo de adaptação

É possível avaliar as condições de carga de um servidor de diversas formas (a seção 4 traz um estudo sobre o assunto). Uma forma simples de fazê-lo é simplesmente através do monitoramento dos tempos de resposta das requisições. Em nosso modelo é utilizada uma variação dessa última abordagem. Considerando que os clientes não irão esperar as respostas de suas requisições indefinidamente, é possível assumir uma condição de falha quando uma requisição ultrapassar um determinado valor de tempo. Esse valor pode ser concebido como um *deadline*.

No momento da chegada de uma requisição ao servidor, um *deadline* absoluto é atribuído à requisição. Após seu atendimento, é verificada a ocorrência de perda de *deadline*[3]. Essa contabilização é usada, complementarmente, para: (i) detectar a existência de sobrecarga e acionar um mecanismo adaptativo no servidor que tenta reduzir a carga; e (ii) dirigir a política de atribuição dinâmica de prioridades visando manter a diferenciação de serviços proporcional.

A política de atribuição de prioridade do modelo de adaptação efetua uma atribuição dinâmica de prioridades às classes. Antes de ser executada pelo servidor, a política de seleção de versão determina se a execução da requisição será na forma precisa ou imprecisa. A duas políticas trabalham de forma integrada na heurística de escalonamento denominada PCV (*Proportional Cumulative Value Attribution*) [8], e são fundamentadas em um histórico de execução que armazena as *k* últimas execuções (precisa, imprecisa ou perda de *deadline*) de cada classe.

O *controle de admissão* atua na fila de requisições descartando, quando necessário, a requisição recém-chegada. Esse descarte é contabilizado no histórico como uma perda de *deadline*.

3.1 Valor cumulativo de cada classe

O valor cumulativo é uma métrica introduzida por [12] que visa mensurar o tempo despendido pelo servidor na execução das requisições. O valor obtido na última requisição pode assumir: 0 se requisição perdeu *deadline*; 1 se requisição executou na forma precisa; ou *C* se requisição executou na forma imprecisa, tal que $0 < C < 1$. Quando uma requisição executa na forma imprecisa considera-se que ela oferece ao sistema um valor proporcional ao tempo gasto na sua execução (*C*).

A manutenção de um histórico atualizado permite o cálculo do valor cumulativo proporcional (*VC*) levando em conta as últimas *k* execuções de requisições de cada classe e fazendo uma normalização, tal que, para as N classes de serviço: $\sum_{i=1}^{N} VC_i = 1$.

3.2 Políticas de atribuição de prioridade e de seleção de versão

Na implementação do modelo de adaptação considera-se que cada classe de serviço possui um *Valor de Qualidade* (*VQ*) associado. Esse valor é uma percentagem atribuída estaticamente e que representa a qualidade almejada para aquela classe, proporcionalmente às outras classes.

As duas políticas da heurística PCV agem de forma integrada visando manter os valores cumulativos o mais próximo possível dos valores de qualidade estipulados. Ou seja, as políticas agem buscando a situação ideal: Quando o valor cumulativo de uma classe, em um determinado momento, é maior ou igual ao seu valor de qualidade ($VC_i \geq VQ_i$), a classe de requisições encontra-se em um estado estável e, presumidamente, a qualidade média dos serviços oferecidos para as requisições àquela classe está acima ou dentro do esperado. Entretanto, quando o valor cumulativo de uma classe for menor que seu valor de qualidade, esta encontra-se em um estado de falha. Por conseguinte, a política de atribuição de prioridade distribui as prioridades proporcionalmente ao valor: $VQ_i - VC_i$ (classes estáveis possuem valores negativos e, portanto, recebem as menores prioridades do sistema).

A política de seleção de versão é dirigida pelas ocorrências de sobrecarga. A cada indicação de uma perda de *deadline* uma classe é selecionada para executar na sua versão imprecisa. Classes que executam versões imprecisas têm seus valores cumulativos reduzidos. Dessa forma, quando precisa escolher alguma classe para executar na forma imprecisa, a política de seleção de versão escolhe apenas classes que estejam em um estado estável. Nesse caso, essa política escolhe sempre a classe com maior valor: $VC_i - VQ_i$.

[3] O *deadline* é atribuído no momento do estabelecimento da conexão TCP referente à requisição web, e o teste de perda de deadline no fechamento da conexão.

Na medida em que uma classe possui execuções imprecisas consecutivas de suas requisições, seus valores cumulativos são reduzidos até o momento crítico, em que essa classe não pode mais ter requisições respondidas na forma imprecisa (nem perdas de *deadline*) sem que atravesse para um estado de falha. Essa busca por um equilíbrio dinâmico é efetuada pela integração das duas políticas na heurística de escalonamento PCV.

3.3 Política de controle de admissão

Testes efetuados em [8] mostraram que, a partir de uma carga demasiadamente alta, o PCV perde a capacidade de fazer a diferenciação proporcional de serviços.

Por conseguinte, o modelo de adaptação deste trabalho incorpora um controle de admissão (Figura 2) no qual, quando necessário, requisições recém-chegadas são descartadas e contabilizadas como perdas de *deadline*.

3.4 Visão geral do algoritmo

A Figura 3 apresenta uma visão superficial do algoritmo empregado. Com exceção do controle de admissão, apresentado na seção 4.2, o algoritmo é melhor descrito em [8].

```
inicialização:
 ∀ classe j faça
    VCj ← VQj
    priorj ← {menor prioridade}
    prox_execj ← "precisa"
*[
  □ se chegou requisição para uma classe j →
        se carga muito alta
            então descarta requisição segundo controle de admissão
            senão enfileira requisição na fila de chegada da classe
  □ se ∃ requisição na cabeça da fila de requisições para uma classe j →
        retira requisição de sua fila de chegada
        se prox_execj = "imprecisa" ∧ VCj ≤ VQj
            então prox_execj ← "precisa"
        executa requisição com prioridade priorj e versão prox_execj
  □ se terminou execução da requisição x para uma classe j →
        testa se perdeu deadline e calcula VCj
        atribui prioridades:  priorj ← VQj - VCj
        { Considera-se que quanto maior o valor, maior a prioridade.
          Valores negativos têm a menor prioridade do sistema }
        se perdeu deadline então
            selecione classe j tal que
                prox_execj = "precisa" ∧ max(VC – VQ)
            faça prox_execj ← "imprecisa"
]
```

Figura 3. Heurística de escalonamento PCV.

4. IMPLEMENTAÇÃO DO MODELO NO SERVIDOR WEB APACHE

O Apache é o servidor web mais utilizado no mundo[4], por esse motivo foi utilizado para avaliar a implementação do modelo proposto. Nesta seção, inicialmente é fornecida uma breve descrição do Apache. Na seqüência é apresentado um estudo sobre formas de monitorar a carga em um servidor web. No final são apresentados alguns resultados sobre o comportamento do Apache em situação de sobrecarga frente à atribuição de prioridades às requisições de clientes.

4.1 Modelo de Adaptação no Apache 2.2

O servidor web Apache é projetado para trabalhar com uma ampla variedade de plataformas e ambientes. Isto é possível devido à sua implementação modular. A versão 2 do Apache introduziu os modelos de processos MPM (*Multi-Processing-Modules*), responsáveis por gerenciar as portas de comunicação, aceitar conexões e alocar processos ou *threads* para atendimento das requisições.

O MPM utilizado na implementação desse trabalho foi o MPM Worker (Figura 4). Este MPM implementa um servidor multi-processo *multi-thread*. Alterações feitas neste módulo apache, juntamente com a implementação de um módulo DSO (*Dynamic Shared Object*), possibilitaram a implementação do modelo proposto. Neste trabalho adotou-se a última versão estável do Apache, versão 2.2, lançada em dezembro de 2005. Essa nova versão oferece algumas funcionalidades interessantes como a possibilidade de se utilizar a API padrão oferecida pelo Apache.

A classificação dos pedidos de conexão pode ser implementada baseada no endereço IP do cliente, e identificada pelo servidor através da extração dessa informação contida na mensagem HTTP enviada pelo cliente. Uma *thread* ouvinte (*Listener*) aguarda novos pedidos de conexão vindos do serviço de comunicação TCP/IP (**A** na Figura 4) e os insere na fila de requisições (**B** na Figura 4), marca cada requisição com sua respectiva classe de serviço (ouro, prata ou bronze) e atribui à requisição um valor de *deadline.*

O controle de admissão também é implementado nessa fase com base no tamanho da fila de requisições. Alguns pedidos de conexão são descartados, evitando assim que o servidor fique ainda mais sobrecarregado.

As *threads* de atendimento de requisições (**C** na Figura 4) retiram as requisições da fila baseadas nas classes das requisições e nas prioridades de cada classe. Estas requisições então são processadas pelos módulos manipuladores (*handlers*).

A política de seleção de versões precisas/imprecisas é executada na fase de processamento das requisições por um módulo dinâmico DSO – `meu_handler` (Figura 5) – carregado em tempo de execução através da diretiva *LoadModule* do arquivo de configuração *httpd.conf* do Apache. O teste se houve perda de deadline no atendimento da requisição também é efetuado na fase de log (eixo de processamento, Figura 5).

As trocas de informações entre os módulos MPM e o módulo *handler* são feitas utilizando áreas de memória disponibilizadas pelo Apache, conhecidadas como *pools* e através do uso de memória compartilhada. Para a compilação e execução do módulo DSO desenvolvido utilizou-se a ferramenta de construção de módulos dinâmicos oferecida pelo Apache 2.2, `apxs` (*Apache eXtenSion tool*).

[4] http://www.netcraft.com (jun/2005)

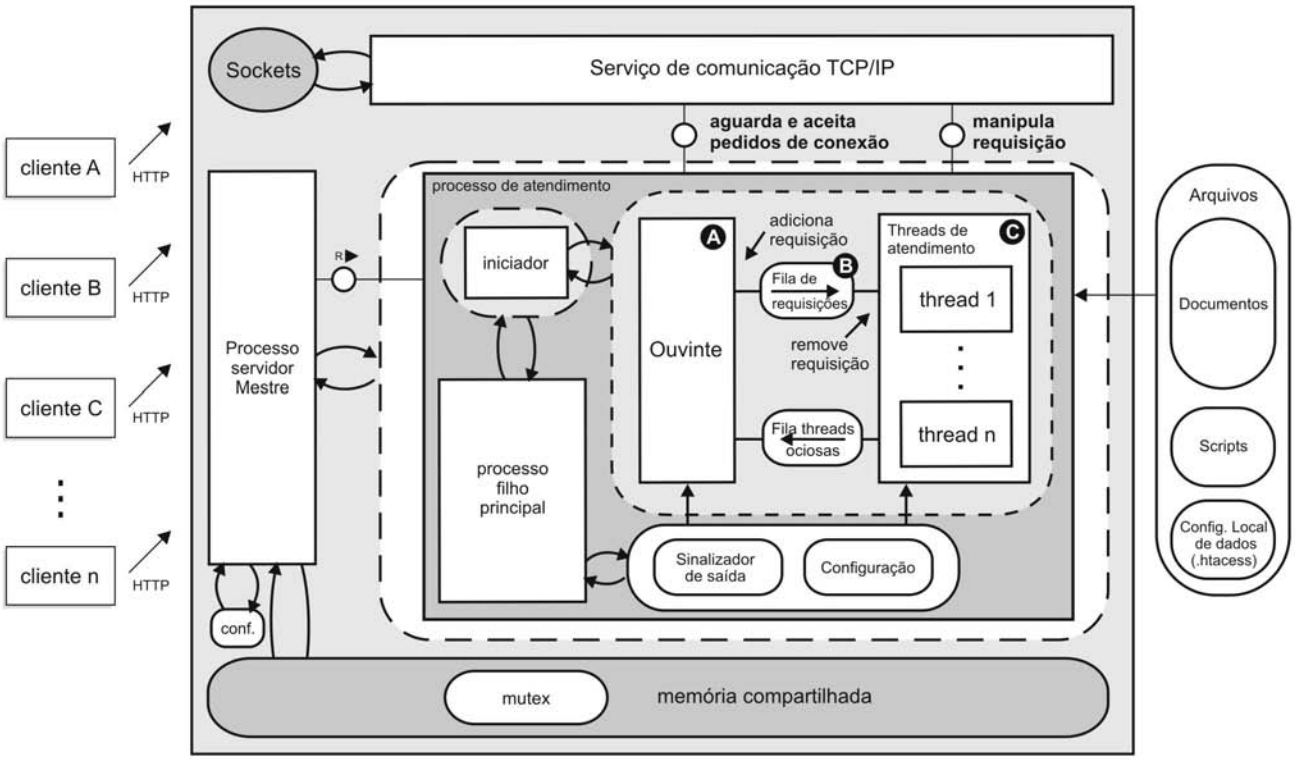

Figura 4. Configuração MPM Worker do Apache (baseado em [11])

É factível a implementação de filas de requisições separadas, para cada classe de serviço, conforme o modelo de adaptação descrito na seção 3. Contudo, buscando simplicidade, neste trabalho optou-se por implementar apenas uma fila de prioridades. As requisições, após serem classificadas e terem suas prioridades atribuídas, são inseridas na fila conforme suas prioridades.

4.2. Avaliação de carga no Apache

Existem diversas abordagens para detectar sobrecargas em servidores web [2]. Buscando avaliar algumas das abordagens, vários experimentos foram efetuados mensurando: (i) ocupação da fila de sockets; (ii) ocupação da fila (B da Figura 4) de requisições do Apache; e (iii) tempo de resposta das requisições.

A abordagem (i) foi descartada neste trabalho, pois o monitoramento da fila de sockets não pode ser efetuada em tempo de execução alterando-se apenas o código-fonte do Apache, pois implica em instrumentalizar o sistema operacional.

As outras duas abordagens foram implementadas e a Figura 6 apresenta alguns resultados obtidos. A carga crescente submetida ao sistema foi gerada pelo *httperf* usando três computadores clientes. O eixo x representa a passagem do tempo, o eixo y da esquerda se refere às médias dos tempos de resposta das requisições e o do lado direito à ocupação da fila.

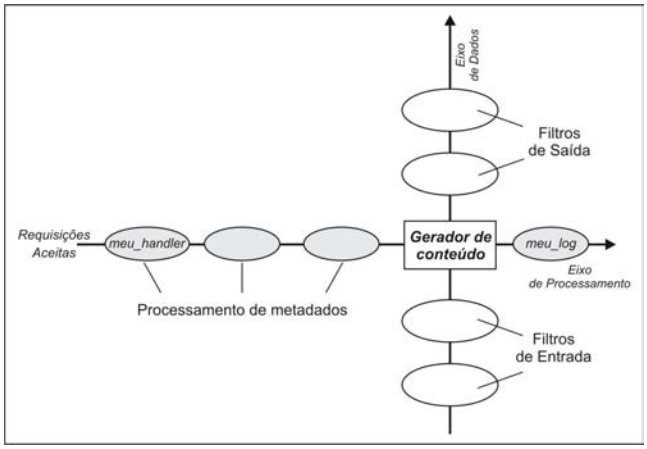

Figura 5. Fases de processamento de requisições

Ao contrapor em um mesmo gráfico as duas medidas, mesmo que de grandezas diferentes, é possível observar as curvas correspondentes crescem de maneira proporcional (possuindo valores de tangente próximos). Desta forma, considerou-se a possibilidade de adotar qualquer uma dessas abordagens para mensurar o crescimento da carga submetida ao servidor. Como alguns trabalhos sugerem o uso dos tempos de resposta [2], este trabalho prosseguiu na análise deste parâmetro.

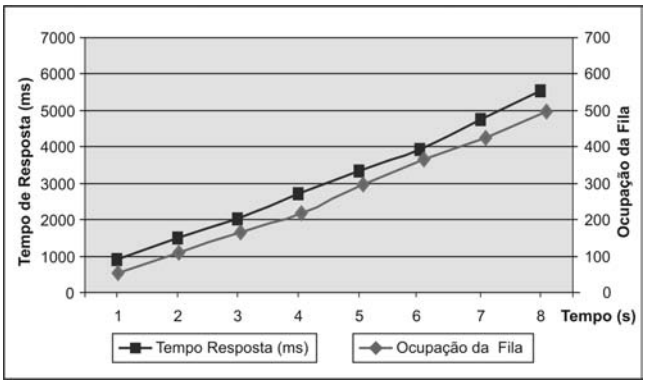

Figura 6. Tempos de resposta vs ocupação da fila em servidores web.

A Figura 7 apresenta três curvas referentes a diferentes formas de avaliar os tempos de resposta: instantâneo, média aritmética em uma janela de tempo (no caso 1 segundo) e média exponencial ponderada. Como os valores instantâneos de tempo de resposta (última amostra obtida) tende a variar bruscamente, dependendo da carga, uma média obtida em uma janela de tempo também apresenta variações, ainda que menores. A adoção de uma abordagem de média móvel ponderada, como a adotada para estimação de valores de *round trip time* em conexões TCP [13] tende a suavizar essa variação. No caso foi adotada a função:

`média = α*média + (1-α)*amostra` tal que `α=0,9`

Essa função pode ser considerada como um filtro passa baixo onde o valor de α especifica a constante de tempo do filtro.

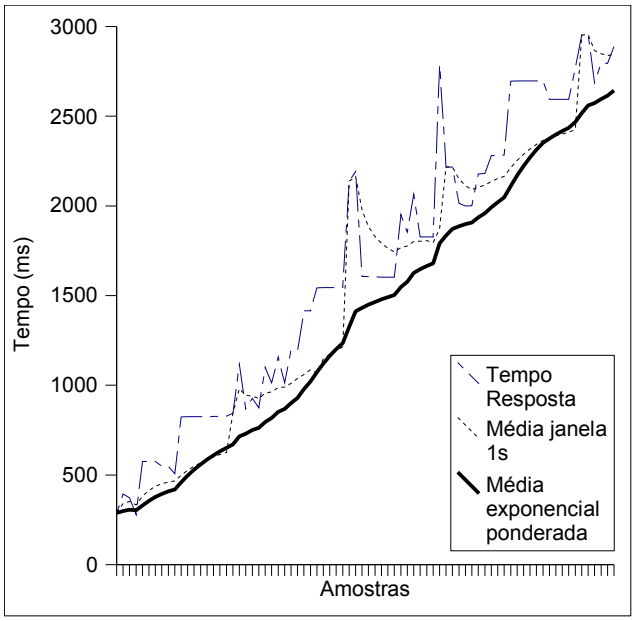

Figura 7. Tempos de resposta instantâneos vs média móvel ponderada.

4.3 Experimentos com prioridades estáticas

Neste trabalho existe o interesse em avaliar o comportamento do Apache frente a requisições com diferentes prioridades (principalmente, considerando que o sistema operacional possui pontos de execução não preemptivos propensos ao problema de inversão de prioridades). No Apache 2.2 é possível criar filas de prioridades para atendimento de requisições. Experimentos foram efetuados com três computadores clientes A, B e C, cujas requisições foram classificadas (pelos seus endereços IP) com ordem decrescente de prioridades.

A Figura 8 apresenta resultados de experimentos, com o eixo x representando a passagem do tempo. Enquanto o servidor se encontrava com carga baixa, entre os tempos 1 a 3, este conseguia executar requisições dos três clientes. Após, esse período de tempo, o servidor em sobrecarga se dedicou a executar quase que exclusivamente as requisições mais prioritárias (Cliente A). Após o instante 24, aproximadamente, a demanda do Cliente A cessou de existir, e o servidor passou a executar requisições do Cliente B. Somente quando a demanda deste último cliente terminou que o servidor passou a atender requisições do Cliente C, cujas requisições são classificadas como menos prioritárias.

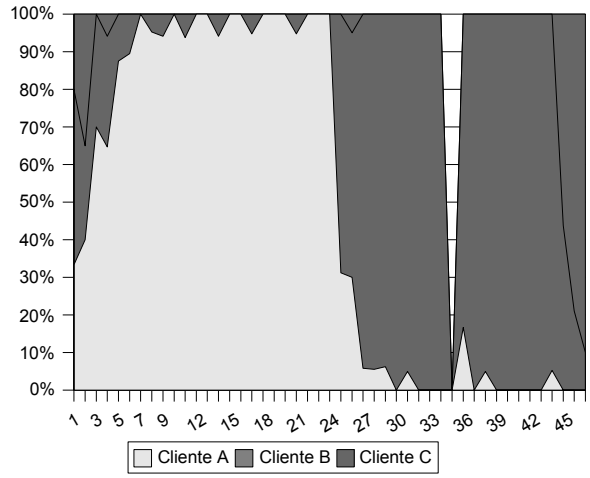

Figura 8. Experimentos com prioridades estáticas.

A Tabela 1 apresenta os tempos de resposta médios obtidos nos clientes, usando *httperf*. Os tempos de resposta do Cliente C foram bem maiores, devido aos tempos de espera na fila, aguardando o atendimento de requisições mais prioritárias.

Tabela 1. Tempos de resposta com prioridades estáticas.

	Cliente A	Cliente B	Cliente C
Prioridade	0 (maior)	1	2
Tempos de Resposta fim a fim (ms)	2378	21487	43610

Os resultados obtidos indicam que a adoção de prioridades estáticas (prioridades fixas) não é indicada pois pode levar às requisições com prioridades menores à inanição (*starvation*).

5. AVALIAÇÃO DO MODELO E DA ABORDAGEM

O modelo proposto foi avaliado em diversos experimentos efetuados em um simulador e em um protótipo construído sobre um servidor web Apache. Esta seção apresenta essas experiências.

5.1 Experimentos com Simulador

A abordagem foi avaliada através de um simulador desenvolvido em Java em nosso laboratório. Cada classe de serviço no simulador recebe requisições, cujo tempo entre chegadas, em um primeiro momento, foi modelado utilizando-se uma distribuição Pareto. Contudo, no sentido de tentar usar uma carga de trabalho mais próxima de um ambiente real, passou-se a gerar carga sintética através do *httperf* e, armazenar os tempos de chegadas medidos na interface de rede do servidor web através do software *Ethereal*[5].

Essa medição se tornou adequada porque o *log* do Apache armazena tempos com uma granularidade muito grossa (intervalos de 1 segundo). Pelo mesmo motivo descartou-se utilizar alguns *logs* conhecidos e utilizados pela literatura, tal como o efetuado na Copa do Mundo de 1998 e que possui mais de 4 milhões de acessos registrados[6].

Nos experimentos, considerou-se os tempos médios de atendimento às requisições usando as versões imprecisas como sendo 30% dos tempos da versão precisa (acrescenta 0,3 no cálculo do valor cumulativo).

Foram definidas três classes, Ouro, Prata e Bronze com valores de qualidade: *0,6*; *0,3*; e *0,1*, respectivamente. Considerou-se que essas classes recebem requisições, cujas taxas de chegadas (retiradas do arquivo de *log* de chegadas) são distribuídas igualmente entre as três classes. Foi especificado um valor de *deadline* relativo fixo, com valor de 3000 ms.

A título de comparação, a abordagem FIFO, utilizada em alguns servidores web, também foi simulada e alguns resultados são apresentados na Figura 9. Além das curvas de valores cumulativo proporcionais de cada classe, é traçada no mesmo gráfico uma curva do valor cumulativo absoluto médio (em linha tracejada). Com uma carga menor que 100% a abordagem mantém uma diferenciação de serviços próxima à proporção da taxa de chegadas (no caso, cerca de 33% para cada classe, já que essas são iguais, apesar da possibilidade haver rajadas de requisições). Após o crescimento da carga, principalmente em situações de sobrecargas severas (não mostradas no gráfico) a diferenciação de serviços passa a assumir um comportamento mais irregular.

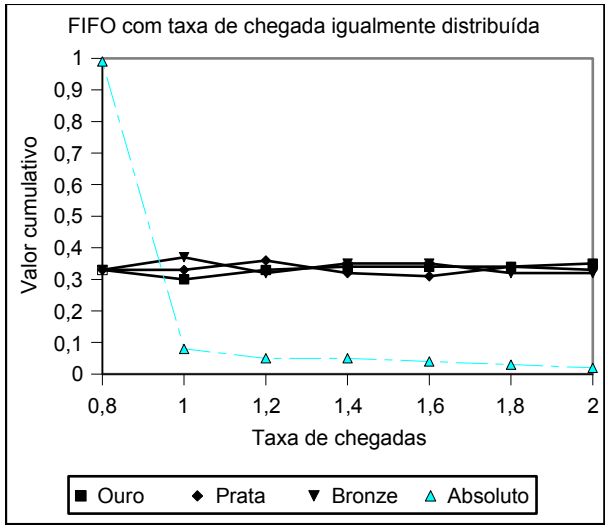

Figura 9. Diferenciação de serviços: FIFO.

Importante observar na Figura 9 que a média do valor cumulativo absoluto (média das três classes) cai próxima de zero quando a demanda ao servidor se aproxima da capacidade (taxa de chegadas de 100%). Esse comportamento se reflete no crescimento elevado nos tempos de resposta das requisições, conforme pode ser observado na Figura 10 (os tempos de resposta estão representados em escala logaritmica).

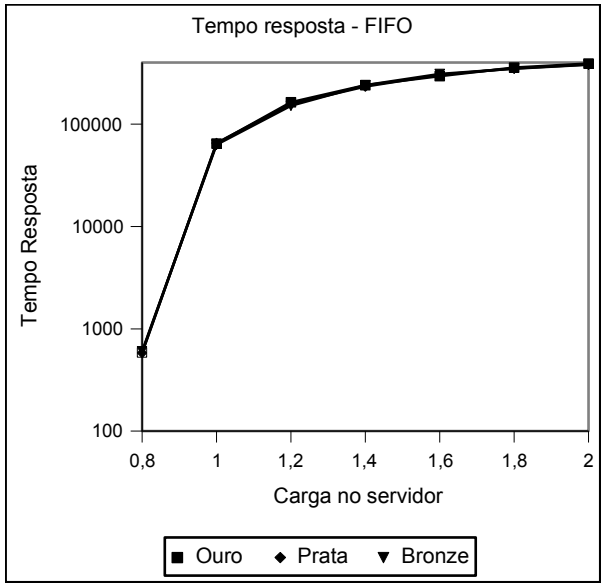

Figura 10. Tempos de resposta: FIFO.

Esse crescimento nos tempos de resposta justifica a existência de diversas abordagens que propõem controles de admissão em servidores web [2,3,9,14].

[5] http://www.ethereal.com/

[6] http://www.hpl.hp.com/techreports/1999/HPL-1999-35R1.html

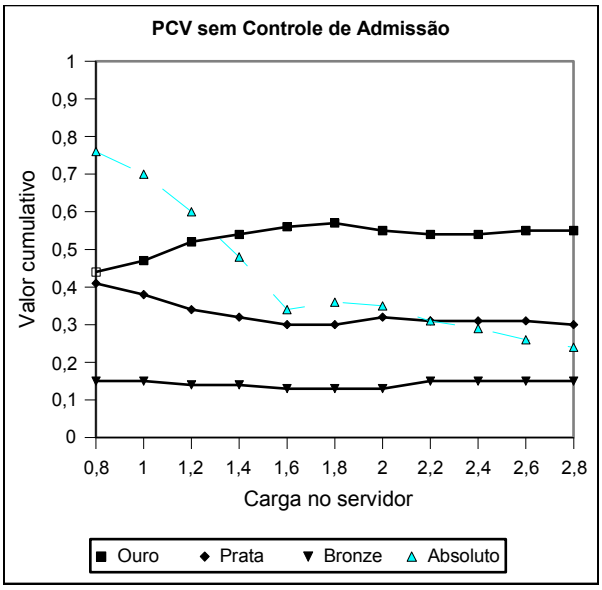

Figura 11. Diferenciação de serviços: PCV.

O PCV, conforme proposto, exibe um comportamento de buscar a diferenciação de serviços proporcional (Figura 11). Pode-se observar que as políticas de diferenciação de serviços (de atribuição de prioridades e de controle de versão) do PCV só começam a agir de forma efetiva a partir de uma situação de sobrecarga (carga maior que 100%). Isso não se constitui em um problema grave, considerando o fato que em situação de carga baixa as requisições têm suas demandas atendidas com baixo tempo de resposta.

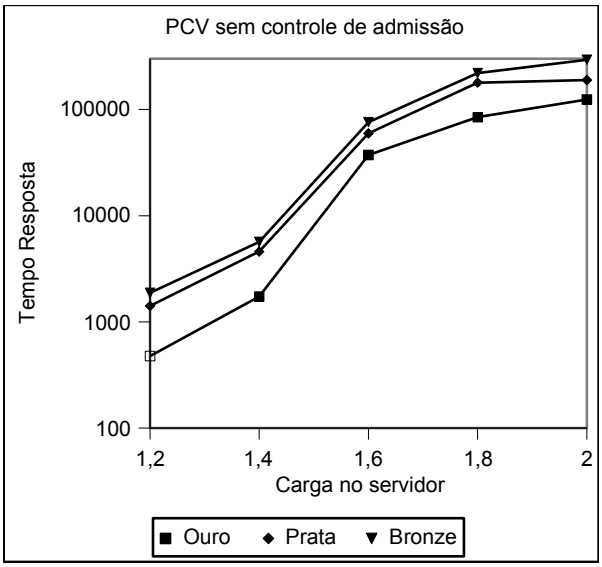

Figura 12. Tempos de resposta do PCV sem controle de admissão.

A métrica adotada para avaliar abordagens que podem descartar, ou executar de forma imprecisa é a do valor cumulativo. Apesar disso, os tempos de resposta do PCV medidos podem ser observados na Figura 12 (fornecidos em escala logaritmica). Percebe-se que houve uma diferenciação proporcional também nos tempos de resposta. Apesar das políticas de controle de versões (precisa ou imprecisa) do PCV atuar no sentido de reduzir a carga em situações de sobrecarga e, apesar do PCV conseguir manter a diferenciação de serviço, os tempos de resposta continuam a crescer rapidamente. Portanto, foi implementada um controle de admissão conforme descrito na seção 4.2.

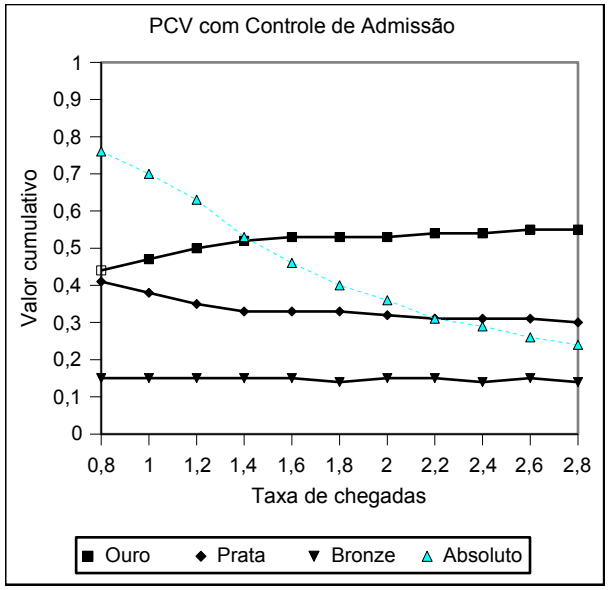

Figura 13. PCV com controle de admissão.

Os resultados obtidos da diferenciação de serviços e dos tempos de resposta são mostrados na Figura 13 e Figura 14.

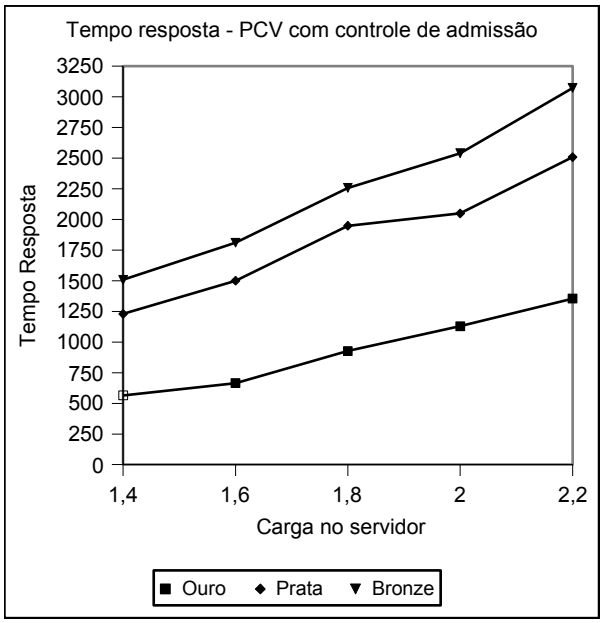

Figura 14. Tempos de resposta do PCV com controle de admissão.

5.2 Experimentos com Apache 2.2

Experimentos foram efetuados utilizando-se um protótipo construído sobre um servidor web Apache visando observar até que ponto a implementação reproduzia o comportamento obtido nas simulações. Principalmente, considerando que diversos *overheads* acabam sendo ignorados em ambientes simulados.

No servidor web utilizou-se um PIV 3.0 GHz, 1024 MB memória RAM e sistema operacional Linux 2.6. Objetivando facilitar à sobrecarga do servidor, algumas configurações do Apache não foram otimizadas. As informações contidas no arquivo de configuração *httpd.conf* podem ser verificadas na Figura 15.

```
<IfModule worker.c>
    ServerLimit          1
    StartServers         1
    MaxClients           50
    MinSpareThreads      25
    MaxSpareThreads      50
    ThreadsPerChild      50
    MaxRequests PerChild 0
</IfModule>
```

Figura 15. Configuração do MPM worker

Na geração da carga dos clientes utilizou-se o gerador de carga sintética *httperf* com número de conexões 1200, taxa de chegada 60 requisições por segundo e timeout 60 segundos. As versões precisas e imprecisas das páginas web foram as mesmas da seção 2.1, cujos tempos de resposta foram apresentados na Figura 1.

As primeiras medições efetuadas objetivaram verificar o comportamento do valor cumulativo (VC), e se estas se mantinham próximo aos valores de qualidade (VQ) especificados. Ou seja, se as propriedades de controlabiliidade e previsibilidade eram respeitadas. Para esses primeiros experimentos, o controle de admissão não foi ativado. A Figura 16 apresenta alguns dos resultados obtidos (o eixo x representa amostras consecutivas obtidas em intervalos de 1 segundo).

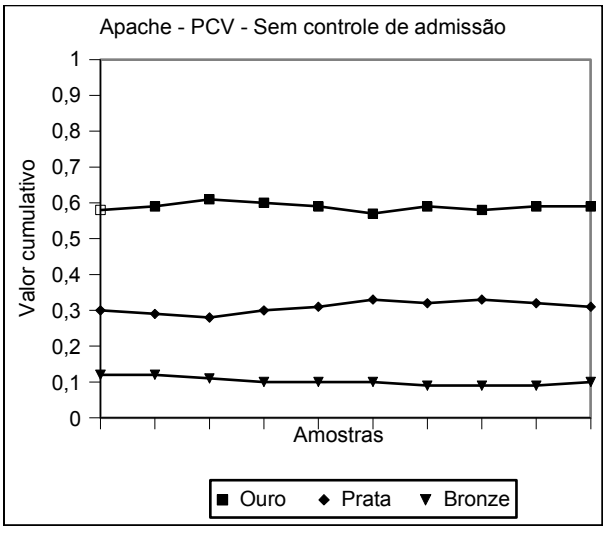

Figura 16. Valores cumulativos do PCV no Apache.

Apesar que, como já foi ressaltado, devido a descartes e execuções imprecisas os valores de tempo de resposta não poderem ser consideradas como a única métrica relevante, nesses mesmos experimentos, os valores foram obtidos e estão apresentados na Figura 17.

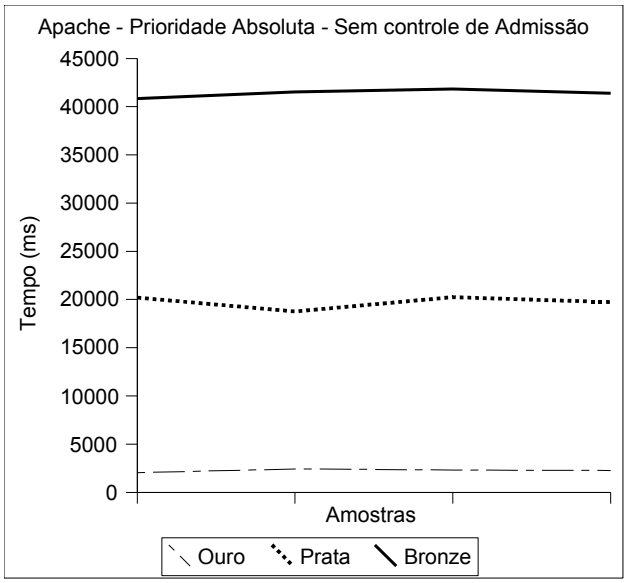

Figura 17. Tempos de resposta fim a fim.

É possível observar na Figura 17 que os valores de tempo de resposta foram condizentes com os valores de qualidade almejados.

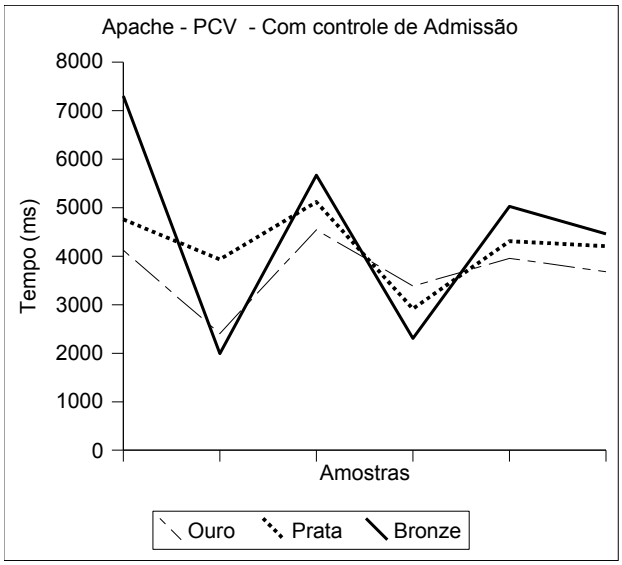

Figura 18. Tempos de resposta fim a fim do PCV no Apache.

Outros experimentos foram efetuados, desta vez com o controle de admissão ativado. O controle de admissão, neste caso, é importante em casos de sobrecarga severa, onde as execuções

imprecisas não conseguem reduzir suficientemente a carga. Os valores cumulativos se mantiveram dentro do esperado, com comportamento muito parecido com o apresentado na Figura 16, por isso, não são apresentados novamente aqui.

No entanto, os tempos de resposta fim a fim obtidos apresentaram flutuação, e são apresentados na Figura 18. Essa flutuação deveu-se, principalmente, pelas inversões de prioridades advindas da implementação de apenas uma fila de requisições (seção 4.1), em vez da implementação de filas separadas para cada classe de serviços.

Interessante observar que os tempos de resposta para a mesma carga, apresentados na Figura 18 (com controle de admissão), comparados com os da Figura 17, são muito menores, conforme já havia sido previsto nos experimentos com o simulador.

6. CONSIDERAÇÕES FINAIS

Este artigo descreveu um modelo de adaptação para servidores web baseado em computação imprecisa e escalonamento por prioridades dinâmicas, com objetivo conseguir uma diferenciação proporcional de serviços.

O uso de versões imprecisas permite responder às requisições de usuários, mesmo com qualidade degradada, e evitar que estes tenham a sensação de negação de serviço. Em situações severas de sobrecarga [14], as execuções imprecisas não conseguem mais reduzir a carga do sistema, sendo melhor descartar ativações. Por esse motivo, um controle de admissão foi adicionado.

Diferentemente de outros trabalhos que usam tempos de resposta como métrica [2,6,7], neste trabalho foi adotado o valor cumulativo, porque este último consegue mensurar a diferença de qualidade obtida por execuções precisas e imprecisas e descartes de requisições. Apesar disso, neste artigo também procurou-se medir individualmente os tempos de resposta.

O modelo foi avaliado tanto em um simulador Java como em um servidor web Apache versão 2.2. Os experimentos efetuados no servidor web mostraram alguns dados interessantes, difíceis de se capturar em um simulador de eventos discretos (seção 5.1). Houve, por exemplo, a possibilidade de medir tempos de resposta reais, fim a fim, e avaliar alguns *overheads* que ocorrem no servidor Apache.

Apesar de obtermos resultados satisfatório com um controle de admissão empregado, no momento estamos investigando um controle de admissão dinâmico, em malha fechada, ligado diretamente com o controle de escolha de versões (precisas/imprecisas). Outra questão que estamos investigando é a de alterar o modelo proposto, considerarando um descarte seletivo onde, em momentos de sobrecarga, a tarefa que pertencer a classe que tiver o maior valor `VC-VQ` será a descartada, independentemente se foi a que acabou de chegar.

AGRADECIMENTOS

Agradecemos à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro.

REFERÊNCIAS

[1] C. Dovrolis, P. Ramanathan, A Case for Relative Differentiated Services and the Proportional Differentiation Model, *IEEE Network*, Sep-Oct. 1999, pp.2-10.

[2] T. Abdelzaher, QoS-Adaptation in Real-Time Systems, PhD Thesis, University of Michigan, Ann Arbor, Aug. 1999.

[3] C. Lu, T. Abdelzaher, J. Stankovic, and S. Son, A A feedback control approach for guaranteeing relative delays in web servers. In: *Proceeding of the 7th IEEE Real-Time technology and Applications Symposium*, 2001, pp. 51-62.

[4] R. Pandey, *et al.*, Supporting Quality of Service in HTTP Servers, Proc. of the 17th Annual SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Puerto Callarta, Mexico, Jun. 1998.

[5] L. Eggert, J. Heidemann, Application-Level Differentiated Services for Web Servers, *World Wide Web Journa*l, Aug. 1999, pp. 133-142.

[6] J. Almeida *et al.*, Providing Differentiated Levels of Service in Web Content Hosting, *First Workshop on Internet Server Performance*, Madison, Wisconsin, Jun. 1998.

[7] M. A. M. Teixeira, M. J. Santana, and R. H. C. Santana. Servidor Web com diferenciação de serviços: Fornecendo QoS para serviços da Internet. In *XXIII SBRC,* vol. 15, pp. 745-770. June 2005.

[8] C. Montez, J. Fraga, Implementing Quality of Service in Web Servers. IEEE SRDS 2002: 32-40, Osaka, Japan.

[9] A. Robertson, B. Wittenmark, M. Kihl, M. Anderson, Design and evaluation of load control in web server systems. In: *Proceeding of the 2004 American Control Conference*, pp 1980-1985, Boston, MA, June 2004.

[10] Blake, S., Black D., Carlson M., Davies E., Wang Z. and W. Weiss, An Architecture for Differentiated Services, RFC2475, December 1998.

[11] B. Grone, A. Knopfel, R. Kugel, O. Schmidt, The Apache Modeling Project. H.P.I – Hasso Plattner Institute for Software Systems Engineering, Potsdam, Germany, jul 2004.

[12] S. Baruah *et al.*, On the Competitiveness of On-Line Real-Time Task Scheduling, Proc. of the 12th IEEE RTSS, pp. 106-115, 1991.

[13] Jacobson, Congestion Avoidance and Control, Proc. of ACM SIGCOMM'88, pp. 314-329.

[14] B. Schoroeder, M. Harchol-Balter, Web servers under overload: How scheduling can help, ACM Transactions on Internet Technology (TOIT), Volume 6, Issue 1 (February 2006), pp. 20 – 52.

[15] C. Estrella, *et al.* Mecanismos de Negociação em um Modelo de Servidor Web com Diferenciação de Serviços. Webmedia 2005, Poços de Caldas, MG, 2005, pp. 100-106.