Implementando Diferenciação de Serviços em Servidores Web Através de Escalonamento Adaptativo

Carlos Montez
montez@das.ufsc.br
LCMI - Depto de Automação e Sistemas - Univ. Fed. de Santa Catarina
Caixa Postal 476 - 88040-900 - Florianópolis - SC - Brasil

Resumo

Este artigo defende a adequação *modelo relativo proporcional* no contexto da implementação de QoS em servidores web, e propõe o uso do conceito de *valor cumulativo* como métrica adotada na diferenciação de serviços. Objetivando manter a diferenciação de serviços, esse artigo apresenta uma abordagem de escalonamento adaptativo que atua através da heurística *Proportional Cumulative Value Attribution* (PCV). Essa heurística age através de duas políticas que, simultaneamente, ordenam as requisições conforme prioridades atribuídas dinamicamente; e selecionam diferentes versões de páginas web para responder as requisições.

Abstract

In this paper we argue in favor of using the *proportional diferentiation model* in web QoS context, and we propose the *cumulative value* concept as a metric adopted in differentiation of services. With objective of maintaining the differentiation of services, we present an adaptive scheduling approach that implements the *Proportional Cumulative Value Attribution* (PCV) heuristic. That heuristic acts through two policies that, sorts the requests according to dinamically attributed priorities; and while at the same time, selects different versions of web pages to service the requests.

***Palavras-chave**: Web QoS, Qualidade de Serviço, Serviços Internet.*

1. Introdução

A convergência de tecnologias de comunicação e computação tem propiciado o surgimento de novos tipos de aplicações e, conseqüentemente, exigido o desenvolvimento de suportes e novas técnicas que dêem sustentação a essas aplicações. Dois exemplos notáveis desses novos tipos de aplicações são comércio eletrônico e bibliotecas digitais. Ambos adotam a Internet como uma via de informação global, e apresentam novas demandas em questões de segurança, confiabilidade e desempenho, as quais a comunidade científica tem se esforçado em atender. Clientes dessas novas aplicações têm de conviver com clientes de aplicações convencionais da Internet, tais como os de correio eletrônico, cujas necessidades por serviços são bem menos exigentes e que, dessa forma, muitas vezes podem ser simplesmente ignoradas.

Uma característica comum relacionada a essas novas aplicações é que suas demandas não podem aguardar o desenvolvimento de uma "nova" Internet, uma vez que muitas dessas aplicações já estão sendo usadas em nosso dia a dia. Seus projetistas desejam utilizar código legado e soluções já existentes de gerenciadores de banco de dados e servidores web, dentre outros. Entretanto, esbarram no fato que essas soluções são construídas tendo como meta o fornecimento de serviços de forma equânime, desconsiderando as necessidades de cada cliente.

Com o reconhecimento geral da inadequação do paradigma de "mesmo serviço para todos" da Internet atual, faz-se necessário adotar soluções imediatas que forneçam qualidades

de serviço (QoS) diferenciadas para os clientes dessas aplicações. Nesse sentido, organizações padronizadoras tais como a IETF vêm buscando padrões abertos para fornecimento de QoS às aplicações, como os relacionados ao IntServ [2] e DiffServ [3]. Entretanto, pouco adianta o emprego dessas soluções na infraestrutura da rede, caso as aplicações nos pontos finais também não se adeqüem a essa nova realidade. Um exemplo, é o fato notório que as aplicações servidoras existentes (*ex.* servidores web) adotam a política de atendimento de requisições pela suas ordens de chegada (FIFO).

No contexto de infraestrutura de redes de comunicação, entre as diversas propostas atuais de fornecimento de diferenciação de serviços às aplicações, as que adotam o *modelo relativo proporcional* [1] se dispõem a manter valores proporcionais de QoS entre as classes, independentemente da variação na carga de cada agregado.

Este artigo defende a adequação do modelo relativo proporcional, no contexto da implementação de QoS em servidores web, onde condições de sobrecarga, que ocorrem constantemente, tendem a degradar a qualidade do serviço oferecido; e propõe o uso do conceito de *valor cumulativo* [13] como métrica adotada na diferenciação de serviços. Baseado nessa métrica, este artigo introduz uma nova heurística, denominada *Proportional Cumulative Value Attribution* (PCV), que busca manter, dinamicamente, a diferenciação de serviços proporcional em servidores web.

Esse texto é dividido em 7 partes. Na seqüência a essa introdução, nas seções 2 e 3, apresentamos as principais motivações que estão conduzindo os nossos e outros trabalhos à implementação de suporte de QoS na infraestrutura da rede e em servidores web. Em seguida, na seção 4, apresentamos nossa abordagem de implementação de diferenciação de serviços usando políticas de escalonamento adaptativo. Na seção 5 são mostrados e interpretados alguns resultados de simulações da nossa abordagem. Na seqüência, na seção 6 apresentamos uma discussão sobre alguns trabalhos de pesquisas relacionados ao nosso. Finalmente, na seção 7 são apresentadas as conclusões e futuros encaminhamentos dessa pesquisa.

2. Diferenciação de serviços na infraestrutura de rede

O reconhecimento da coexistência de diferentes categorias de usuários na Internet, que apresentam diferentes expectativas na forma que suas requisições serão servidas, tem levado a IETF na direção de buscar soluções que permitam essa convivência de uma forma mais harmoniosa. Nesse sentido, existe um esforço no sentido de implementar arquiteturas baseadas em dois modelos diferentes de fornecimento de QoS às aplicações [1]. O *modelo de garantia absoluta* visa a manutenção de níveis de serviço absolutos. Ou seja, são negociados com as aplicações valores mínimos e máximos de determinados parâmetros, os quais o suporte se esforça em mantê-los. No *modelo de diferenciação relativa*, as aplicações são classificadas e agrupadas em classes de serviço, e o suporte busca a manutenção de uma ordenação relativa na qualidade obtida entre essas classes[1].

Segundo esses modelos, foram propostas pelo IETF duas arquiteturas diferentes e, possivelmente complementares: IntServ [2] e DiffServ [3]. A arquitetura IntServ busca alcançar o modelo de garantia absoluta através de mecanismos de controle de fluxos de pacotes e protocolo de reserva de recursos (RSVP) que lidam com estados individuais de fluxos em roteadores. Essa abordagem exige que os roteadores mantenham controle do estado dos fluxos que passam por ele, apresentando problemas de escalabilidade limitada.

A mais recente arquitetura DiffServ tenta resolver o problema de escalabilidade, implementando o modelo relativo de diferenciação de serviços. Ela lida com agregado de

[1] Obviamente, é considerada alguma métrica para avaliar o desempenho de um serviço. Métricas comumente usadas em suporte de rede são atrasos médios e taxas de perdas de pacotes.

tráfego, ou seja, classifica e agrupa fluxos com requisitos de QoS similares (classes de serviços), e implementa políticas que se esforçam em fornecer QoS para esses agregados, com a premissa central que determinadas classes irão receber melhor serviço que outras.

Subjacente aos diferentes tipos de abordagem está o conceito de *granulosidade de QoS*. Enquanto no modelo absoluto existe um tratamento de QoS individual, por fluxos de pacotes; no modelo relativo, a qualidade de serviço é tratada em agregados de fluxos. Essa última abordagem de diferenciação de serviços sacrifica a QoS individual, em nome da escalabilidade, lidando com um número reduzido e conhecido de classes de serviços.

Mais recentemente, como uma extensão do modelo relativo, foi proposto o *modelo proporcional* (ou *modelo relativo proporcional*) de serviços diferenciados [1]. Nesse modelo, o que se busca é, independente da carga em cada agregado, manter os valores proporcionais de QoS entre as classes. Ou seja, esse modelo pode ser considerado como o modelo de diferenciação relativa que mantém dois atributos importantes[2]: (i) *controlabilidade* – capacidade de operadores de redes ajustarem o espaçamento de níveis de qualidade entre as classes, de acordo com algum critério; e (ii) *previsibilidade* – capacidade de um sistema manter consistente a diferenciação entre as classes, independente de variações na carga.

Mais adiante, na seção 5, iremos mostrar que, devido à instabilidade da Internet, onde não é possível delimitar sua carga, o modelo relativo proporcional é indicado para ser usado em servidores melhor esforço (não tempo real), tais como os servidores web, que executam nesse ambiente. Além disso, é interessante observar que a abordagem DiffServ, que implementa modelos de diferenciação relativa, considera que a complexidade da rede estará em suas bordas: nos roteadores de borda e nas aplicações que se situam nos pontos finais da rede. Entretanto, atualmente os servidores não estão preparados para essa diferenciação de serviços, tratando seus clientes de maneira uniforme, e lidando com suas requisições por ordem de chegada. A próxima seção trata exatamente da questão da diferenciação de serviços nesse tipo de servidor.

3. Diferenciação de serviços em servidores web

Com o crescimento e ubiqüidade da Internet, atualmente estão sendo disponibilizados diversos serviços baseados em web (*ex.* hospedagem de arquivos, assistência remota, submissão de textos, *chat*, e-mail, tradução de textos, conversão de formatos de arquivos). Esses serviços, por sua vez, vêm sendo usados em aplicações mais complexas, tais como, comércio eletrônico, bibliotecas digitais e ensino à distância, que requerem suportes mais rígidos para seus requisitos de qualidade de serviço (QoS).

No contexto específico de servidores web, é possível encontrar diversas justificativas para um suporte adicional de QoS. Por exemplo, no caso de serviços de hospedagem (arquivos, páginas e sítios web) é possível distinguir "classes" de clientes, os quais podem pagar taxas proporcionais aos níveis de serviços oferecidos. Devido ao fato dos recursos (espaço em disco, capacidade de processamento, largura de banda) usados por esse tipo de serviço serem finitos, hoje, clientes já pagam taxas proporcionais ao espaço disponibilizados a eles. Entretanto, deveria haver diferenciação de clientes, também, com relação a outros parâmetros, tais como tempos de respostas e taxas de perdas[3].

---

[2] Esses atributos foram discutidos inicialmente em [14] dentro do contexto de sistemas de tempo real em condições de sobrecarga temporária.

[3] Alguém poderia argumentar que provedores desse tipo de serviço deveriam investir em recursos, de forma que todos os clientes tenham todas suas demandas atendidas. Essa linha de argumento tornaria secundário qualquer suporte de QoS. Entretanto, a experiência mostra que a demanda apresenta tendência crescente, alocando os recursos disponíveis, com o passar do tempo. Portanto, a diferenciação de serviços ainda se torna necessária, apesar de pesquisas em esquemas de *proxies*, servidores web distribuídos e multiprocessados.

Servidores web em sistemas de comércio eletrônico necessitam igualmente de suporte a diferenciação de serviços, devido às diferentes classes de acessos, com importâncias distintas, entre as entidades envolvidas. Como exemplo, é possível identificar acessos (i) entre companhia e consumidores (*ex.* vendas diretas), (ii) entre companhias (*ex.* distribuição de informações, compras); e (iii) internamente em uma companhia (*ex.* comunicações internas, aplicações de colaboração). No mesmo contexto, também poderia haver uma implementação de diferenciação de serviços bem mais simples, baseada apenas na localidade do acesso, ou seja, se este é interno (*intranet*) ou externo.

Ainda considerando sistemas de comércio eletrônico, a diferenciação também poderia ser feita pelo conteúdo de algumas páginas. Por exemplo, páginas que levam a confirmação de fechamento de algum negócio (tal como o preenchimento de dados de cartão de crédito) poderiam ser consideradas mais importantes que outras páginas existentes no sítio web de uma companhia.

Em diversas instituições, além de um grande número de páginas pessoais, os servidores web são usados para armazenar sítios web com finalidades diferentes. Esse compartilhamento muitas vezes foge do controle do administrador da rede, pois alguns usuários podem publicar páginas que acabam se tornando populares, com muitos acessos. Requisições para esses servidores deveriam ser regidas por algum controle acesso que distribua os recursos existentes conforme importâncias atribuídas aos sítios e páginas web.

Um outro argumento para diferenciação de serviços em servidores web é a existência de *proxies* "inteligentes", que fazem requisições especulativas aos servidores, armazenando páginas web em avanço. Esse tipo de acesso apresenta requisitos flexíveis de tempos de resposta, podendo ser tratado como um tipo de acesso com baixa prioridade [8].

Sucintamente, compilando a informação acima, é possível afirmar que a diferenciação pode ser feita baseada (i) nas identidades dos clientes; (ii) nas localizações desses clientes; (iii) na natureza dos conteúdos acessados; ou (iv) nas identidades das pessoas que criaram esses conteúdos. Contudo, conforme enfatizado anteriormente, não obstante a existência de alguns trabalhos recentes [4-9], servidores web ainda não estão preparados para essa diferenciação de serviços. Por conseguinte, visando exatamente preencher essa lacuna, desenvolvemos nosso trabalho que é apresentado na próxima seção.

4. Diferenciação de serviços usando escalonamento adaptativo

Nossa linha de investigação há algum tempo vinha estudando a previsibilidade de sistemas computacionais na presença de sobrecargas temporárias, onde a capacidade computacional disponível se torna provisoriamente insuficiente para atender às demandas. Em um trabalho prévio, investigamos o comportamento de uma classe de aplicações de tempo real, composta por tarefas periódicas que possuem deadlines *firm* [11]. Essas tarefas, exemplificadas por algumas aplicações de controle e de multimídia, em situações de sobrecarga, toleram a perda de deadlines ou descartes de algumas ativações, desde que essas ocorram esparsadamente. Quando essa última condição é violada, considera-se que ocorreu uma *falha temporal*. Nesse sentido, no âmbito de uma política de gerenciamento adaptativo de recursos, desenvolvemos uma heurística de escalonamento, denominada *(p+i,k)-firm* [11-12], que buscava evitar falhas temporais através da conjugação de duas políticas: (i) de atribuição dinâmica de prioridades [10] e, (ii) de seleção de versões precisas/imprecisas na execução de ativações [15].

Nosso interesse agora está direcionado no comportamento de sistemas legado em situações de sobrecarga — mais especificamente, de servidores melhor esforço. Atualmente, os exemplos mais representativos desse tipo de sistema, provavelmente, são os servidores web. Todos os argumentos apresentados na seção anterior direcionaram nossas pesquisas

objetivando a implementação de diferenciação de serviços também nesse tipo de servidor, através do uso de abordagens de *escalonamento adaptativo*.

4.1. Escalonamento adaptativo

A essência do comportamento previsível de muitos sistemas é baseada na execução de políticas de escalonamento, definidas no sentido de determinar a ocupação de recursos compartilhados do sistema. Ou seja, o escalonamento diz respeito à alocação de recursos (processador, largura de banda, dispositivos) necessários para a execução de tarefas, segundo algumas restrições e objetivos. Abordagens de escalonamento adaptativo assumem que as condições do sistema são monitoradas (*ex.* padrões de utilização de recursos, condições de carga), e as decisões do escalonamento são baseadas nessas observações. Essas abordagens estão sendo cada vez mais empregadas pela comunidade de pesquisas em escalonamento tempo real [4-6,10-15] devido ao interesse em executar aplicações tempo real em ambientes dinâmicos, como é o caso da Internet, onde a carga não pode ser previamente determinada.

Diferentemente do modelo periódico, investigado em nossas pesquisas anteriores [11–12], que era formado por um grupo restrito de tarefas, servidores web podem possuir um grande número de clientes, fazendo requisições aperiódicas. Essa característica torna difícil a manutenção da informação do estado de cada cliente, como ocorria na nossa abordagem original. Esse problema de escalabilidade é semelhante ao enfrentado pelo IETF com a arquitetura IntServ. Por conseguinte, considerando uma similaridade com a arquitetura DiffServ, também empregamos a agregação de clientes em classes de serviço previamente definidas. A nossa abordagem de escalonamento adaptativo passa, então, a ser aplicada em cada classe de serviço.

Conforme discutido na seção 2, o objetivo de agregar requisições em classes, é o de implementar o modelo relativo de diferenciação de serviço, objetivando que determinadas classes recebam melhor serviço que outras. Entretanto, aplicações que executam na Internet estão sujeitas a cargas que podem crescer arbitrariamente. Se não houver um controle dinâmico e adaptativo no gerenciamento de recursos, caso haja uma concentração de acessos a classes consideradas "privilegiadas", requisições para essas classes podem receber uma qualidade de serviço inferior às requisições para classes consideradas "inferiores". Isso torna inadequado qualquer esquema estático de provimento de recursos (*ex.* políticas de atribuições estáticas de prioridades às classes).

Com finalidade de buscar uma diferenciação de serviços relativa e proporcional, mantendo suas propriedades de controlabilidade e previsibilidade, implementamos nosso modelo de adaptação baseado na conjugação de duas políticas: *de atribuição de prioridades* e *de seleção de versões*. Antes de descrever nosso modelo de adaptação, é importante discutir os seus objetivos e métricas.

4.2. Objetivos e métricas da adaptação em servidores web

Quando se trata de servidores, a primeira métrica geralmente associada, é a dos “tempos de resposta”. Se existe a possibilidade de um servidor se negar a atender uma requisição de serviço, ou apresentar erros nesse atendimento devido à carga atual estar excedendo sua capacidade, uma segunda métrica usada é a das “taxas de erros”. Entretanto, essa realidade "binária" (serviço atendido ou não atendido) é uma limitação imposta pelo modelo de computação tradicional.

No modelo de computação imprecisa [15] o objetivo passa a ser "executar o serviço que puder, no tempo disponível". 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 talvez seja suficiente para que o cliente tenha a percepção que foi atendido, ainda que, com uma qualidade reduzida. Quando o servidor seleciona a versão que

oferece a melhor qualidade, diz-se que ocorreu uma "execução precisa". Caso contrário, quando é selecionada uma versão parcial, diz-se que ocorreu uma "execução imprecisa". Acessos a servidores implementados segundo esse modelo apresentam a propriedade de "polimorfismo temporal", onde os resultados das requisições dependem das condições de carga do servidor.

Servidores de vídeo sob demanda, por exemplo, em situações de sobrecarga, podem reduzir o tempo de codificação do vídeo, degradando a qualidade do vídeo transmitido para clientes [11]. No caso de servidores web, é possível a implementação de conteúdos de páginas com mais de uma versão. Em situações de sobrecarga, pode-se responder uma requisição de um cliente, com uma versão de página web com qualidade reduzida, porém com tempo de acesso menor.

Medidas efetuadas por [4] mostraram que, quando a carga alcança três vezes a capacidade do servidor, existe um grande atraso no tempo de atendimento das requisições, e mais da metade dos recursos disponíveis (tempo de processador) são gastos em conexões que acabam sendo abortadas pelos clientes, em um fenômeno semelhante a um efeito dominó[4]. Nesse caso, o modelo de computação imprecisa apresenta uma grande contribuição ao modelo tradicional, pois, ao responder algumas requisições na forma imprecisa, reduz a carga e, simultaneamente, oferece um retorno de serviço aos clientes, que, no modelo tradicional, provavelmente receberiam como resposta um erro de "timeout", disparado pelos seus navegadores web.

Toda a discussão acima conduz a um outro tipo de métrica para avaliar esse modelo de adaptação: o *valor cumulativo*. Essa métrica, introduzida por [13], visa mensurar o tempo despendido pelo servidor, na execução das requisições. Por exemplo, se um servidor atender uma requisição através de uma execução precisa, o valor cumulativo é incrementado de 1 (100%); caso haja falha no atendimento da requisição, nada é adicionado ao valor cumulativo (0%); e, finalmente, caso o servidor responda com uma versão imprecisa, um número entre 0 e 1, proporcional ao tempo executado pelo servidor naquela requisição, é adicionado ao valor cumulativo.

4.3. Detecção de condições de sobrecarga em servidores web

Técnicas de escalonamento adaptativo fundamentam-se no monitoramento das condições do ambiente. Quando não existem condições de sobrecarga em um servidor, qualquer política de atendimento às requisições pode ser considerada satisfatória, até mesmo a política de atendimento pela ordem de chegada (FIFO) empregada nos servidores web atuais[5]. Por outro lado, em situações de sobrecarga, muito tempo de serviço pode ser desperdiçado em requisições que acabam sendo canceladas pelos clientes. É nesse momento que uma política de atendimento diferenciado de requisições é importante, pois passa a ser significante a diferença entre a melhor e a pior qualidade no atendimento do serviço requisitado. Dessa forma, é primordial algum tipo de monitoramento no ambiente de execução para detectar essas condições, e acionar algum mecanismo que forneça uma diferenciação de serviços.

Diversas técnicas podem ser empregadas para esse tipo de monitoramento, tal como o uso de mecanismos de monitoramento de carga oferecidos por muitas versões populares de

---

[4] Esse efeito ocorre em algumas políticas de escalonamento tempo real baseadas apenas em restrições temporais, onde a perda de deadline de uma tarefa, leva às tarefas subseqüentes a também perderem seus deadlines.

[5] Novamente, aqui, alguém poderia argumentar que a diferenciação de serviços deveria ser mantida sempre, mesmo em situações de funcionamento favoráveis (carga computacional baixa). Todavia, existe um custo (*overhead*) nos mecanismos de manutenção dessa diferenciação. Acreditamos que, quando o servidor está com carga baixa, as diferenças entre os piores e melhores tempos de resposta de um servidor podem ser imperceptíveis para os clientes, tornando desnecessária a diferenciação de serviços.

sistemas operacionais. Por exemplo, o Linux possui diversas informações sobre o uso de processador dentro do sistema de arquivos virtual “/proc”. Usando essas informações é possível verificar o uso de processador, individualmente, por cada processo. Contudo, essa técnica de monitoramento possui como desvantagem o fato de ser dependente do suporte do sistema operacional.

Por outro lado, é possível a implementação de um monitoramento, no âmbito de *middleware*, em uma camada acima do acima do sistema operacional e da aplicação. Para isso, é necessário alterar as bibliotecas de rotinas TCP (*socket*) usadas pelo servidor web para receber e responder as requisições de clientes. Interceptando requisições e respostas, pode-se, por exemplo, contabilizar o número de requisições que chegam ao servidor, e subtrai-lo do número das que saem. Esse cálculo simples permite verificar, em um dado momento, o número de requisições que um servidor está atendendo. É possível tentar estimar as condições de carga, e julgar a existência de uma condição de sobrecarga, quando o número de requisições atendidas em um servidor ultrapassar a um determinado patamar.

Outra técnica que pode ser empregada em um *middleware* é a do monitoramento direto dos tempos de respostas das requisições. É possível avaliar as condições de carga de um servidor, em um intervalo de tempo, calculando o valor médio dos tempos de atendimento. Nesse caso, é possível considerar o servidor em um estado de sobrecarga, quando os tempos médios ultrapassarem um determinado valor.

Em nosso trabalho, utilizamos uma variação dessa última abordagem. Considerando que os clientes (e seus navegadores web) 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 máximo para ser servida. Esse valor de tempo estipulado pode ser concebido como uma espécie de *deadline* encontrado em tarefas tempo real.

No momento da chegada de uma requisição, um valor de tempo (um *deadline*) é atribuído à requisição. Quando o servidor atende uma requisição, e envia a resposta para o cliente, é verificado se houve uma violação no *deadline* da requisição, ou seja, se ocorreu uma *falha temporal* (ainda que a requisição tenha sido servida até o final).

Em nossa abordagem de escalonamento adaptativo, essa contabilização de falhas tem duas finalidades complementares:

(i) ela é usada para detectar a existência de sobrecarga, e acionar um mecanismo adaptativo no servidor que tenta reduzir a carga, através da política de seleção de requisições na forma imprecisa; e,

(ii) é usada no cálculo do valor cumulativo que irá dirigir a política de atribuição dinâmica de prioridades. Caso o tempo de serviço de uma requisição ultrapasse o valor estipulado, é considerada uma condição de falha temporal (a requisição teve uma qualidade de serviço inadequada), e o valor cumulativo associado à execução é zero.

4.4. Implementação de múltiplas versões em servidores web

Em [4], Abdelzaher faz um excelente levantamento de técnicas para implementação de múltiplas versões de páginas web. De uma forma sucinta, esse trabalho recomenda para implementação de versões imprecisas: (i) redução de tamanho de imagens JPEG através de uma compressão mais "agressiva"; (ii) redução de objetos embarcados em páginas web; (iii) redução de *links* locais existentes em cada página; (iv) eliminação de recursos visuais extras, tais como *gifs* animados e animações *flash*. Cada uma dessas estratégias procura reduzir, ora a taxa de acessos (*hits*) ao servidor, ora o uso de largura de banda, ora os tempos de processamento no servidor.

Páginas web relacionadas ao comércio eletrônico costumam ter muitos efeitos visuais,

visando atrair clientes. 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. Experimentos efetuados mostraram que, usando essa técnica, até 90% das páginas visitadas apresentaram uma melhoria de desempenho de pelo menos 400% [4].

4.5. Mecanismo de escolha de versão

O modelo de computação imprecisa, usado por nossa abordagem de adaptação, implementa duas versões de cada página web hospedada no servidor. Segundo o protocolo HTTP, cada mensagem de requisição contém, em formato ASCII, o sítio web (*host*), nome da página e o diretório em que essa página está armazenada (Figura 1).

```
GET /pesquisadores/principal.html HTTP/1.1
Host: www.das.ufsc.br
(outros dados de cabeçalho)
```

Figura 1. Formato de uma mensagem de requisição no protocolo HTTP 1.1

Esse formato possibilita a implementação do mecanismo de escolha de versão de uma forma razoavelmente simples. Todas as páginas e objetos referentes à versão imprecisa, podem ser armazenados em um diretório "/ver.impr", situado abaixo do diretório onde estão as páginas e objetos da versão original (precisa). Caso a versão escolhida para executar seja a versão precisa, nada precisa ser feito. Essa mensagem é enviada para o servidor sem qualquer alteração. Entretanto, no caso da escolha da versão imprecisa, a mensagem é alterada, e o subdiretório "/ver.impr" é concatenado imediatamente antes do nome da página. Por exemplo, considerando a Figura 1, a primeira linha da mensagem passaria a ser:

```
GET /pesquisadores/ver.impr/principal.html HTTP/1.1
```

Os mecanismos de seleção de versões e de atribuição de *deadlines* no momento da chegada da requisição, e de testes de falha temporal no momento que o servidor responde a uma requisição, podem ser implementados no nível de *middleware* (Figura 2).

4.6. Modelo de adaptação em servidores web

O nosso modelo de adaptação considera a existência de um número pré-definido de classes de serviço. No modelo, apresentado na Figura 2, as classes são representadas por tipos de metais, considerando o nível de serviço oferecido pelo servidor web, proporcional ao valor intrínseco do metal. Cada classe de serviço possui uma fila FIFO associada, que armazena requisições encaminhadas para essa classe.

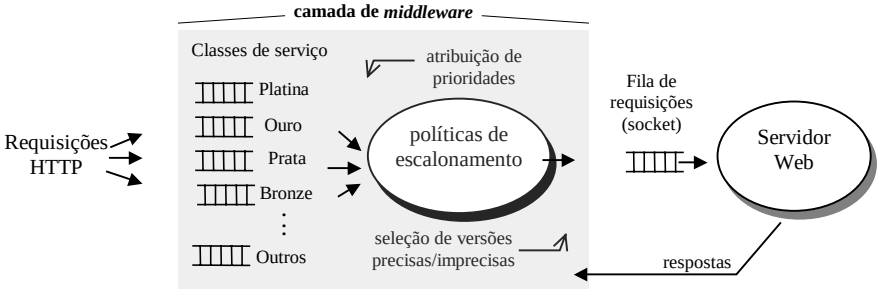

Figura 2. Modelo de adaptação.

A maneira como é feita a classificação de cada requisição está fora do escopo desse texto. Entretanto, sem perda de generalidade, poder-se-ia imaginar que foi aplicado algum esquema baseado no conteúdo da página ou sítio web requisitado, conforme discutido anteriormente na seção 3.

Cada classe de serviço possui uma prioridade associada, que é atribuída dinamicamente. Em cada fila, todas as requisições enfileiradas possuem a mesma prioridade da classe. Essas prioridades indicarão a ordem de execução das requisições, por parte do servidor web, ou seja, a ordem que as requisições serão recebidas pelas primitivas de leitura *socket*, chamadas pelo servidor web.

A *política de atribuição de prioridades* do modelo de adaptação efetua uma atribuição dinâmica de prioridades às classes. Antes de ser executada pelo servidor, a *política de seleção de versões* determina se a execução da requisição será na forma precisa ou imprecisa. A duas políticas trabalham de forma integrada na nossa heurística de escalonamento denominada PCV (*Proportional Cumulative Value Attribution*), e são fundamentadas em históricos de execuções precisas, imprecisas e de perdas de deadline, existentes para cada classe

4.7. Históricos de execução e valor cumulativo de cada classe

Um *histórico de execução* de uma classe, ou simplesmente *histórico*, é uma k-tupla que armazena o estado das execuções das últimas *k* requisições para uma classe. Considere a seguinte representação para cada estado de invocação: "P" para execução precisa, "I" para execução imprecisa, e "X" para perdas de deadline. Para cada novo estado produzido, o histórico é deslocado (da direita para esquerda) e esse estado adicionado na posição mais à direita. A Figura 3 representa a formação de um histórico de uma classe, que perdeu o deadline ("X") em sua última ativação.

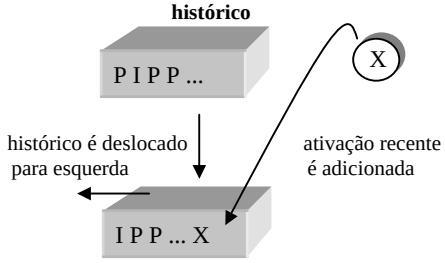

Figura 3. Formação de um histórico de execução.

A manutenção de um histórico atualizado permite o cálculo do valor cumulativo absoluto (*VCabs*), considerando as últimas *k* execuções de requisições de cada classe. Basta efetuar o seguinte cálculo: $\forall classe\ i \quad VCabs_i := VCabs_i - Vativ\_antiga_i + Vativ\_recente_i$ onde, $Vativ\_antiga_i$ e $Vativ\_recente_i$ representam, respectivamente, o valor obtido pela requisição mais antiga e o valor obtido pela requisição mais recente no histórico de uma classe *i*. Ambos podem assumir um valor *Q*, tal que:

$Q = 0$ *se requisição perdeu deadline,*

$Q = 1$ *se requisição executou na forma precisa*

$Q = C$ *se requisição executou na forma imprecisa,* $\mid C \in]0,1[$

Quando uma requisição executa na forma imprecisa, o cálculo do valor cumulativo, considera que essa requisição obtém um valor proporcional ao tempo gasto na sua execução (*C*). Ou seja, caso uma página web tenha sido implementada na forma imprecisa, de tal forma que seu tempo de acesso represente 30% do tempo da versão precisa, o valor cumulativo de uma requisição a essa página é 0.3 (se não houve perda de deadline).

Entretanto, o valor cumulativo calculado acima é absoluto, não possui relação com os valores cumulativos das outras classes. É necessário o cálculo do valor cumulativo proporcional (*VC*) aos de outras classes. Esse valor é obtido através do somatório dos valores cumulativos absolutos de todas as *N* classes (*TotalVC*), e de uma regra de três:

$$TotalVC = \sum_{i=1}^{N} VCabs_i \qquad \text{e} \qquad \forall classe\ i \quad VC_i \leftarrow (VCabs_i * 100) / TotalVC$$

A partir desse ponto de texto assume-se que sempre que não for explicitado o contrário, o termo "valor cumulativo" irá se referir ao valor cumulativo proporcional.

4.8. Políticas de atribuição de prioridades e de seleção de versões

Para 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 pelo operador, e que representa a qualidade almejada para aquela classe, proporcionalmente às outras classes. A soma dos valores de qualidades de todas as classes precisa totalizar 100%:

$$\sum_{i=1}^{N} VQ_i = 100\%$$

As duas políticas da heurística PCV agem de forma integrada, visando a manutenção dos valores cumulativos, o mais próximo possível dos valores de qualidade estipulados. Ou seja, as políticas agem buscando a situação ideal: $\forall classe\ i \quad VC_i = VQ_i$

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. O valor da diferença VC_i - VQ_i em classes nesse estado representa a *distância para falha*. Entretanto, quando o valor cumulativo de uma classe for menor que seu valor de qualidade, esta se encontra em um estado de falha. Nesse último caso, quanto maior a diferença VQ_i - VC_i, maior sua *distância para estabilidade*.

Por conseguinte, a política de atribuição de prioridades atribui as prioridades conforme esses valores de distância, onde, quanto maior a distância para estabilidade de uma classe, maior a sua prioridade. Por outro lado, classes no estado estável recebem a menor prioridade do sistema.

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 versões escolhe apenas classes que estejam em um estado estável. Nesse caso, essa política escolhe sempre a classe com maior distância para falha.

Na medida em que uma classe possui execuções imprecisas consecutivas de suas requisições, sua distância para falha é diminuída, 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, descrita a seguir.

4.9. Heurística PCV – Proportional Cumulative Value Attribution

A heurística de escalonamento é descrita através de três fluxos de execução independentes (Figura 4), responsáveis por: (i) receber e enfileirar requisições (linhas 8-9); (ii) pegar requisições nas cabeças das filas de chegada de cada classe e aplicar as políticas de escalonamento enfileirando as requisições na fila de *socket* (linhas 10-14); e (iii) nos términos de execuções de cada requisição, verificar se esta perdeu seu deadline, atualizar seu histórico e valor cumulativo, e, se for o caso, selecionar alguma classe para executar na forma imprecisa nas suas próximas ativações (linhas 15-25).

Devido ao fato do modelo de escalonamento adotado não ser *overload-aware* [14] (*i.e.*, não pode prever futuras sobrecargas e perdas de deadline), 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 (reduzindo sua qualidade), tomando como base a sua distância para falha. Na heurística, a variável ($prox\_exec_j$) armazena a informação se a próxima execução de uma classe *j* será precisa ou imprecisa.

No inicio da aplicação da heurística (linhas 1-6), todas as classes começam com a execução precisa, partem de um estado estável (*VC = VQ*), e recebem as mesmas prioridades (no caso, é atribuída a menor prioridade do sistema).

```
01:  inicialização:
02:       ∀ classe j faça
03:            inicializa histórico h_j com k valores 'P'
04:            VC_j ← VQ_j
05             prio_j ← {menor prioridade}
06:            prox_exec_j ← "precisa"

07:  *[
08:       □ se chegou requisição para uma classe j →
09:            enfileira requisição no final da fila de chegada da classe

10:       □ se ∃ requisição na cabeça da fila de requisições para uma classe j →
11:            retira requisição da cabeça de sua fila de chegada
12:            se prox_exec_j = "imprecisa" ∧ VC_j ≤ VQ_j
13:                 então prox_exec_j ← "precisa"
14:            enfileira requisição na fila de socket segundo sua prioridade prio_j com versão prox_exec_j

15:       □ se terminou execução da requisição x para uma classe j →
16:            se tempo atual > deadline_x
17:                 então h_j ← shift_left(h_j) + 'X'
18:                 senão h_j ← shift_left(h_j) + versão executada ('P' ou 'I')
19:            calcula VC_j
20:            se VC_j < VQ_j
21                  então prio_j ← VQ_j - VC_j
22                  senão prio_j ← {menor prioridade}
23:            se tempo atual > deadline_x então
24:                 selecione classe j tal que prox_exec_j = "precisa" ∧ max(VC - VQ)
25:                      e faça prox_exec_j ← "imprecisa"
26:  ]
```

Figura 4. Heurística de escalonamento PCV.

À medida que uma classe *j* vai recebendo requisições, sua prioridade vai sendo dinamicamente alterada (linhas 21-22) conforme o seu valor de distância para estabilidade (VQ_i - VC_i). Considera-se, nesse texto, que quanto maior o valor numérico da variável $prio_j$, maior a prioridade atribuída à classe *j*.

Ao término de atendimento de uma requisição, quando é detectada uma sobrecarga (linha 23), a política de seleção de versões escolhe para executar na forma imprecisa, a classe que possua o maior valor de distância para falha (linha 24). Essa classe é então selecionada para executar imprecisamente na próxima requisição e nas seguintes (linha 25), até que alcance sua situação crítica, indicada pela distância para falha (linha 12). Quando esse estado é alcançado, a classe é assinalada novamente para executar sua versão precisa (linha 13).

Acreditamos que o *overhead* adicional, proveniente da implementação da heurística, seja minimizado, e possa até mesmo ser desconsiderado. Latências nos tempos de atendimento nos servidores web são da ordem de milisegundos. Esses tempos são de ordem de grandeza bem maior que os tempos de execução dos mecanismos da heurística PCV. Esse mesmo conceito já é utilizado nos servidores web atuais, que implementam outras políticas no momento de chegada de requisições, tais como as implementadas por módulos de controles de acesso e módulos de redirecionamentos de requisições.

5. Avaliação da abordagem de escalonamento

A abordagem de escalonamento foi avaliada através de simulações. Com esse objetivo, foi desenvolvido um simulador, em linguagem C, com código-fonte disponível na Internet [16].

Esse simulador permite a configuração de uma série de parâmetros tal como o número de classes do sistema e seus valores de qualidade.

Cada classe no simulador recebe requisições, cujos tempos entre chegadas e de serviços são exponencialmente distribuídos, segundo Poisson. Em tempo de configuração são fornecidos ao simulador: uma taxa média de freqüência das requisições, tempos médios de atendimentos de requisições nas versões precisas e imprecisas, e um valor de deadline relativo, usado no cálculo do deadline absoluto no momento de chegada de cada requisição.

Uma série de experimentos foi efetuada para avaliar a abordagem de escalonamento PCV, comparando-a com a FIFO – convencional em servidores web. A cada experimento, diversos dados foram contabilizados, tais como valores cumulativos médios, totais de requisições por classes, de perdas de deadlines e de execuções precisas e imprecisas.

Em todos os experimentos a seguir, considerou-se os tempos médios de atendimento às requisições usando as versões imprecisas (execuções imprecisas) como sendo 20% dos tempos da versão precisa. Nos cálculos dos valores cumulativos, portanto, cada requisição que perde seu deadline nada acrescenta, enquanto que, ao ter seu deadline satisfeito acrescenta 0.2 ou 1, dependendo se executou sua versão imprecisa ou precisa, respectivamente.

5.1. Comparação de desempenho com FIFO

Para esses experimentos, foram definidas três classes, Ouro, Prata e Bronze com valores de qualidade: $VQ_{ouro} = 0,6$; $VQ_{prata} = 0,3$; e $VQ_{bronze} = 0,1$. Considerou-se que essas classes recebem requisições, cujas taxas médias, representadas nos eixos horizontais da Figura 5, são distribuídas igualmente entre as três. Foi especificado um valor de deadline relativo igual ao triplo do tempo médio de uma execução precisa.

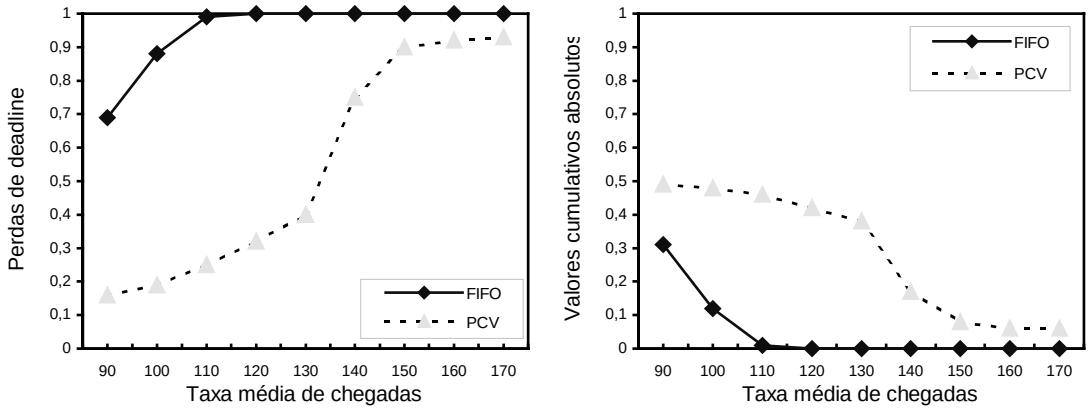

Figura 5. PCV *vs* FIFO: perdas de deadline e valores cumulativos.

A Figura 5 apresenta o levantamento das percentagens de perdas de deadline e do somatório dos valores cumulativos absolutos das classes, considerando várias taxas de chegadas. Os resultados desses experimentos mostram que a abordagem PCV consegue uma redução substancial na quantidade de perdas de deadline com relação à FIFO, até a carga alcançar um determinado valor. O motivo principal dessa redução é o fato que, em situações de sobrecarga, a abordagem PCV pode atender (executar) transitoriamente algumas requisições usando suas versões imprecisas, reduzindo a carga efetiva e, conseqüentemente, a quantidade de perdas de deadlines. Ou seja, a possibilidade de execuções imprecisas é um grau de flexibilidade a mais que o PCV apresenta, e que é determinante para o seu melhor desempenho, segundo essa métrica, em relação à FIFO.

Entretanto, a partir de um valor alto de carga (200%, nesses experimentos), a capacidade de redução de perdas de deadline da heurística PCV se aproxima (da mesma forma que a FIFO) a praticamente zero. Isso se deve ao fato de se ter alcançado o limite de execuções imprecisas da abordagem PCV. Esses resultados apontam em uma direção que

pretendemos investigar em breve: – a adição de uma *política de controle de admissão*, que atuaria junto com as outras duas políticas existentes, de forma a rejeitar imediatamente algumas requisições, reduzindo a carga, porém procurando manter sempre a proporção nos valores cumulativos das classes.

Com relação à comparação dos valores cumulativos, a abordagem PCV consegue manter o somatório dos valores cumulativos absolutos acima da abordagem FIFO, em uma boa faixa da distribuição de carga. Contudo, quando as taxas ultrapassam um determinado patamar, também devido à limitação imposta na política de seleção de versões, a abordagem PCV apresenta uma redução nos valores cumulativos absolutos, aproximando-se dos valores obtidos pela abordagem FIFO.

Apesar de obter uma redução nas taxas de perdas de deadline e, conseqüentemente, manter uma boa média de valores cumulativos absolutos, o objetivo principal da abordagem PCV é a busca na manutenção da diferenciação de serviços proporcionais. A seguir são mostrados alguns resultados que visam avaliar a capacidade da abordagem PCV manter essa diferenciação.

5.2. Diferenciação de serviços na abordagem PCV

Na Figura 6 são apresentados três gráficos mostrando a capacidade de diferenciação de serviços da abordagem PCV com, respectivamente, duas, três e cinco classes definidas. Nos experimentos envolvidos para o levantamento do gráfico (a) da figura, foram definidas duas classes: uma privilegiada, "Ouro" ($VQ_{ouro} = 0,75$), e outra com requisições consideradas menos importantes, "Níquel" ($VQ_{níquel} = 0,25$). De forma análoga, para o gráfico (b) foram definidas três classes, "Ouro", "Prata" e "Bronze" com valores de qualidade, $VQ_{ouro} = 0,6$; $VQ_{prata} = 0,3$; e $VQ_{bronze} = 0,1$. Finalmente, para o gráfico (c) foram definidas cinco classes, "Platina", "Ouro", "Prata", "Bronze" e "Ferro", com valores de qualidade: $VQ_{platina} = 0,5$; $VQ_{ouro} = 0,26$; $VQ_{prata} = 0,14$; $VQ_{bronze} = 0,07$ e $VQ_{ferro} = 0,03$.

Em todos os experimentos, as cargas oferecidas ao sistema foram divididas igualmente entre as classes. Ou seja, no gráfico (a) cada classe recebeu metade da carga; no gráfico (b) cada classe recebeu um terço da carga; e no gráfico (c) da Figura 6, cada uma das 5 classes recebeu 20% da carga.

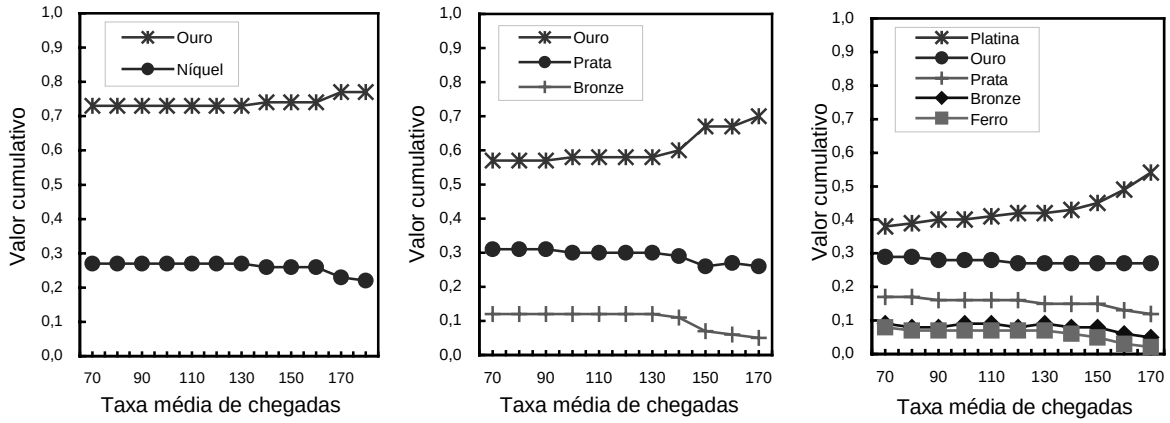

Figura 6. Diferenciação de serviços na heurística PCV: 2, 3 e 5 classes.

O que se observa pelos dados obtidos é que a abordagem PCV consegue manter os valores cumulativos próximos dos valores de qualidade especificados, em quase toda a faixa de distribuição de carga. Entretanto, a partir de uma determinada taxa, conforme foi discutido

anteriormente, a abordagem alcança seu limite de execuções de versões imprecisas. A abordagem, dessa forma, passa a contar apenas com a política de atribuição de prioridades, e encontra dificuldades para manter os valores cumulativos próximos dos especificados. A conseqüência desse efeito na qualidade dos serviços oferecidos, como é possível ver nos três gráficos da Figura 6, é que à medida que a taxa média de chegadas se aproxima de 200%, acaba existindo um distanciamento maior, além do desejado, entre as classes de serviço.

Em outros experimentos efetuados, foram injetadas cargas diferentes em cada classe, e o comportamento de diferenciação de serviços foi observado. Na Figura 7 são mostrados os gráficos dos valores cumulativos das abordagens PCV e FIFO, considerando a definição de três classes, com os seguintes valores de qualidade: $VQ_{ouro} = 0,6$; $VQ_{prata} = 0,3$; e $VQ_{bronze} = 0,1$. A porcentagem de carga média injetada em cada classe obedeceu a seguinte proporção: Ouro: 50%, Prata e Bronze : 25%.

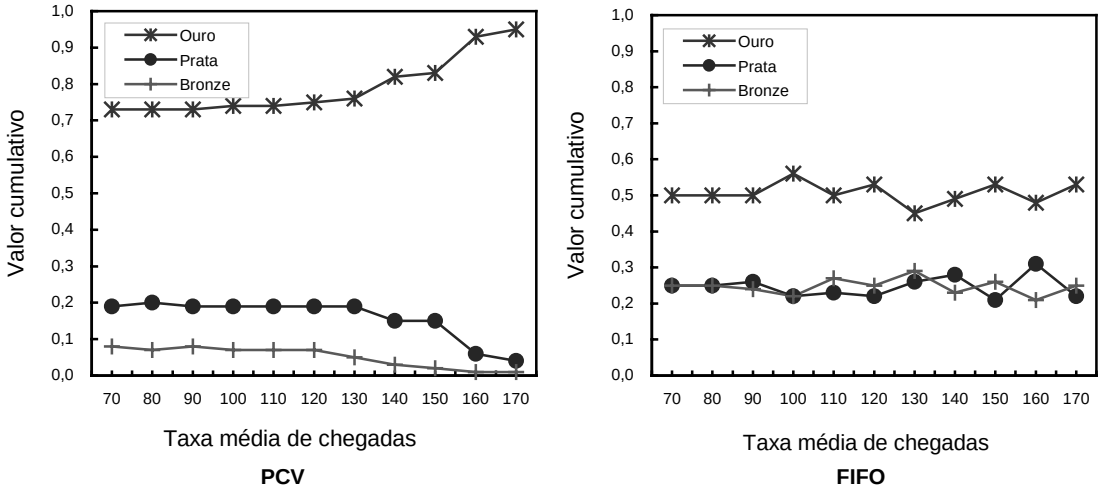

Figura 7. Diferenciação de serviços com cargas diferentes: PCV *vs* FIFO.

É possível observar que, enquanto a abordagem PCV buscou manter os valores cumulativos próximos aos valores de qualidade especificados, a abordagem FIFO, manteve os valores cumulativos na mesma proporção da carga oferecida para cada classe.

5.3. Considerações sobre os resultados da simulação

Os resultados das simulações mostraram que, de uma forma geral, a abordagem PCV consegue, simultaneamente, (i) reduzir a taxa de perdas de deadlines (ou seja, reduzir os tempos médios de serviços) e (ii) manter a diferenciação de serviços proporcional. Contudo, através das simulações, conseguimos detectar uma limitação dessa abordagem, em manter as duas propriedades acima, em situações de cargas altas. Acreditamos que nesses casos, o acréscimo de uma nova política de controle de admissão seja recomendado. Essa política passaria atuar no sentido de reduzir a carga, mas sempre mantendo a proporção na qualidade (valores cumulativos) obtida.

No entanto, é bom observar que, apesar da limitação citada, em todas as situações testadas de carga, em nenhum momento, uma classe com menor valor de qualidade que outra, obteve melhor qualidade de serviço (maior valor cumulativo) que esta última.

6. Trabalhos relacionados

Recentemente, alguns trabalhos vêm reconhecendo a importância de integrar suporte de QoS em servidores melhor esforço. Os trabalhos de Abdelzaher [4] e Lu [5] investigaram formas de implementar a teoria de controle realimentado em servidores web, visando a manutenção da carga oferecida a estes abaixo de valores aceitáveis. Nesse sentido, ambos os trabalhos

implementam uma abordagem adaptativa, realimentada pela observação da carga de trabalho no servidor. Enquanto o trabalho de Lu [4] rejeita conexões (requisições) de clientes como forma de controlar a carga, Abdelzaher [4] usa implementação de várias versões de páginas web, semelhante à nossa abordagem.

Striegel et al. [6] trabalharam com servidores multimídia, agregando requisições em classes de serviço. Eles propuseram o uso de uma heurística de atribuição dinâmica de prioridades [10] como forma de distribuir as perdas de deadline nesses servidores, visando a redução de falhas dinâmicas. Diferentemente de nossa proposta, uma falha dinâmica é assumida quando um cliente possui perdas consecutivas de deadline, acima de valores pré-estabelecidos.

Alguns trabalhos [7-9] investigaram o desempenho do servidor web Apache [17] e seus possíveis "gargalos", e propuseram mecanismos para implementar diferenciação de serviços. Por exemplo, *Eggert* e *Heidemann* [8] consideraram a existência de apenas duas classes de serviço, e propuseram modificação dos códigos-fonte desse servidor, buscando reduzir a qualidade da classe "inferior", (i) reduzindo o tamanho de sua fila de requisições; (ii) reduzindo sua prioridade; e (iii) limitando a largura de banda que essa classe podia usar. O trabalho de *Almeida et al.* [9] propunha mecanismos que lidavam com prioridades das classes, em dois níveis: na aplicação (servidor Apache) e no sistema operacional (Linux). A investigação de *Pandey et al.* [7] era centrada na confecção de uma linguagem de especificação qualidade de serviços para implementação de QoS em servidores web distribuídos.

7. Considerações finais

Esse trabalho mostrou que é possível aplicar técnicas de escalonamento adaptativo, usualmente empregadas em escalonamento tempo real, para manter a diferenciação de serviços em servidores melhor esforço. Classes de serviços foram definidas em servidores web, e foi utilizado o modelo de computação imprecisa [15], onde algumas páginas web são confeccionadas com duas versões diferentes. Em vez de lidar com valores médios de atrasos e taxas de perdas – métricas usuais na infraestrutura de rede – este trabalho empregou valores de deadlines como uma forma de sinalizar atrasos nos atendimentos às requisições, e adotou o valor cumulativo como métrica de avaliação que, em última instância, reflete a qualidade média oferecida para cada classe.

A heurística de escalonamento adaptativo PCV (*Proportional Cumulative Value Attribution*) foi introduzida. Durante ocorrências de sobrecarga, as duas políticas dessa heurística agem procurando: (i) reduzir a carga (selecionando algumas execuções na forma imprecisa) e, simultaneamente, (ii) distribuir as perdas de deadline (priorizando determinadas classes) de forma a manter os valores cumulativos próximos de valores pré-estabelecidos.

Através de simulações, levantamos o comportamento da abordagem PCV, e verificamos a necessidade de adotar uma política de controle de admissão, que atue integrada com as duas políticas existentes. Nossos trabalhos atuais apontam na direção da confecção dessa política, e também na implementação e validação dessa abordagem, em um ambiente real, usando o servidor web Apache [17].

Agradecimentos: O trabalho de pesquisa apresentado aqui, só foi possível graças ao apoio da iniciativa "Kit Enxoval Recém-Doutor" do CNPq – processo institucional 68.0080/01-5. Agradecemos também ao suporte dado por Joni Fraga e Jean-Marie Farines no decorrer de nossa pesquisa.

Referências bibliográficas

[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] R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: an Overview", *IETF RFC 1633*, Jun. 1994.

[3] S. Blake et al., "An Architecture for Differentiated Services", *IETF RFC 2475*, Dec. 1998.

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

[5] C. Lu, "Feedback Control Real-Time Scheduling", PhD Dissertation, University of Virginia, May 2001.

[6] A. Striegel, G. Manimaran, "Dynamic Class-Based Queue Management for Scalable Media Servers", *Sixth IEEE RTAS*, Washington DC, USA, May 2000, pp.228-236.

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

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

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

[10] M. Hamdaoui, P. Ramanathan, "A Dynamic Priority Assignment Technique for Streams with (m,k)-Firm Deadlines", 1994

[11] C. Montez, J. Fraga, R. S. Oliveira, J-M. Farines, "An Adaptive Scheduling Approach in Real-Time CORBA", *In Proc. of the 2nd IEEE ISORC*, Saint-Malo, France, May 1999.

[12] C. Montez, J. Fraga, R. S. Oliveira, "Lidando com Sobrecarga no Escalonamento de Tarefas com Restrições Temporais : A Abordagem (p+i,k)-firm", SCTF'99, Campinas, SP, Brasil, Jul. 1999, pp.72-83.

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

[14] M. Marucheck, J. Strosnider, "Some Insight into the Fault Recovery Properties of Priority-Driven Schedulers", *Real-Time Systems Journal*, Oct. 1995.

[15] J. -Y. Chung, J. W. S. Liu, K. -J. Lin, "Scheduling Periodic Jobs that Allow Imprecise Results", *IEEE Transactions on Computer*, 39(9), 1990, pp. 1156-1174.

[16] Código-fonte do simulador PCV, web site http://www.das.ufsc.br/~montez/PCV, Jan. 2002.

[17] G. Holden, N. Wells, M. Keller, "Apache Server Commentary", *CoriolisOpen Press*, 1999.