Leonardo de Araújo Silva
Orientador: Wagner Meira Jr.

Reatividade e Qualidade de Serviço em Aplicações Web

Dissertação apresentada ao Curso de Pós-graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte
3 de Maio de 2006

Agradecimentos

Mostrar gratidão significa, para mim, não somente dizer algumas palavras, mas principalmente, realizar gestos singelos a cada dia. Espero, assim, poder demonstrar meu agradecimento não somente através das palavras abaixo, a todos aqueles que, de alguma forma, contribuíram com essa caminhada que chega ao seu objetivo.

Em primeiro lugar o meu obrigado a meu pai Jarbas, presente nas lembranças que falam muito mais do que palavras. Meu pai foi um grande exemplo de vida e que soube, com simplicidade e dedicação, construir uma família unida e feliz. Sempre presente, cuidou de minha educação e me proporcionou as condições para estudar em Belo Horizonte e alçar os vôos necessários. Nunca imaginei que sua partida seria tão dura, mas sua saudosa memória permanece em cada instante.

À minha singela mãe Maria da Glória pelo seu cuidado, amor e presença constante nas mais diversas circunstâncias. Gostaria de agradecer, e especial, por três coisas: o apoio incondicional pelas escolhas de vida que fiz nos últimos anos, pela compreensão em relação a minha ausência em alguns momentos que demandaram um maior envolvimento com o trabalho e o mestrado, e pelo carinho maternal.

Aos meus irmãos Fernando, Eduardo e Patrícia, verdadeiros amigos. Obrigado por serem meus fiéis companheiros para todos os momentos. Apesar da distância, foram fundamentais, através do apoio e incentivos demonstrados de diversas maneiras. Não posso deixar de agradecer também aos meus demais familiares, meus tios e primos. Em especial agradeço a minha estimada madrinha Alice e os tios Márcio e Paulo.

Ao Meira, pela presença constante e acompanhamento ativo do trabalho. Pelas conversas, sugestões e orientação indispensáveis para o sucesso que agora alcanço. Pelo seu exemplo de competência e capacidade de produção de resultados. Obrigado por todas as oportunidades, desafios e orientações.

Aos demais membros da banca, professoras Rosa Maria e Jussara e professor Virgílio, agradeço pelos conhecimentos transmitidos e importantes contribuições ao trabalho.

Ao Adriano Pereira, por ter sido o companheiro de trabalho ao longo deste mestrado,

que com sua extrema dedicação e disciplina, foi fundamental para que os objetivos do projeto fossem alcançados. É inevitável agradecer pelo seu empenho e contribuições, mas agradeço também por sua amizade e pelos inúmeros conselhos e lições diárias de companheirismo, disciplina, responsabilidade e determinação.

Agradeço ao Walter Santos e Gustavo Franco que deram contribuições importantes ao trabalho. O primeiro com sua participação no SMG, em especial, na elaboração do simulador de QoS. O segundo, por ter lançado as bases do trabalho, estabelecendo junto com a equipe o modelo *USAR* que se tornaria indispensável para a continuidade do trabalho.

À todos da Eteg Tecnologia, essse desafiador empreendimento que cativou meus sonhos e da qual hoje sinto parte importante. Obrigado aos nobres Rafael Paiva, Rodrigo Moreira e Walter Santos, pelo apoio durante esses anos, e confiança no meu trabalho. Pela ousadia, determinação e dedicação ao ideal de uma empresa pautada por competência técnica, empreendedorismo, bom relacionamento e ética. As condições que me foram proporcionadas foram fundamentais e sem elas este mestrado não teria sido possível. Obrigado mesmo.

À Ana Luiza pelo apoio, carinho e presença em momentos importantes dessa caminhada. Agradeço a compreensão pelos momentos de ausência bem como pelas revisões e idéias importantes compartilhadas. Me recordo também do apoio e presença de seus familiares, José, Regina e João André, pelos quais tenho muito carinho e foram presença marcante.

O meu obrigado a todos os amigos que torceram por mim, me apoiaram e estiveram do meu lado nesses anos, em especial ao Márcio, o pessoal da DVD-Session e demais amigos de Barbacena.

Aos amigos do Laboratório e-Speed bem como dos demais laboratórios do DCC, pela convivência fraterna e ambiente propício para a realização de todas as atividades desse trabalho. Infelizmente não é possível citar os nomes de todos que tiveram, até mesmo, pequenas contribuições através de conversas e discussões diversas.

Agradeço à Hewllet-Packard do Brasil (HP do Brasil) que apoiou o projeto CAMPS, patrocinando a pesquisa realizada. Através do apoio financeiro, bem como das atividades de acompanhamento, o trabalho pôde ser desenvolvido plenamente, de modo que os objetivos alcançados são fruto do investimento realizado.

Por fim, agradeço imensamente ao Departamento de Ciência da Computação (DCC), bem como à Univeridade Federal de Minas Gerais (UFMG), pela oportunidade que me foi dada de realizar minha pós-graduação nesse grande centro de excelência em Ciência da Computação.

Resumo

O grande sucesso da Internet trouxe novos desafios em termos das aplicações e da satisfação dos usuários. Os serviços Internet passaram a demandar novos requisitos como, por exemplo, desempenho e escalabilidade, a fim de garantir um bom nível de Qualidade de Serviço (QoS) aos seus usuários. Devido a estes requisitos, o tema da QoS se tornou um tópico relevante para a comunidade técnico-científica. Diversos mecanismos para provê-la foram propostos, mas eles geralmente falham em considerar aspectos relacionados à reatividade, ou seja, o modo como os usuários reagem a um tempo de resposta variável.

Este trabalho avalia o uso da reatividade para propor novas estratégias de QoS. Primeiramente, demonstramos como a reatividade pode ser modelada e replicada utilizando uma metodologia para correlacionar o tempo de resposta e o IAT (tempo entre requisições consecutivas) das ações dos usuários. Baseado neste modelo, implementamos uma nova versão do gerador de cargas *httperf*, capaz de reproduzir a reação dos usuários. O impacto de uma carga de trabalho reativa baseada no *benchmark* TPC-W sobre um servidor Web real é avaliado, demonstrando que cargas reativas são diferentes em comparação às não-reativas, em relação à carga do servidor assim como a sua taxa de serviço e tempo de resposta.

Com a finalidade de avaliar o impacto da reatividade e das novas estratégias de QoS, projetamos um simulador de aplicações Internet denominado *USAR-QoS*, que foi preparado para implementar políticas de controle de admissão e escalonamento. Utilizando o simulador avaliamos o comportamento de cada classe de usuário definida pelo modelo de reatividade.

Baseado na reatividade, propomos novas estratégias de controle de admissão capazes de rejeitar requisições e sessões de acordo com um critério baseado nas reações dos usuários. Através de simulação foi possível verificar que são efetivas em manter um baixo tempo de resposta mas provocam aumentos nas taxas de perda de requisições. Apresentamos também as abordagens de escalonamento PFIN (de *Patient-First Impatient-Next*, ou seja, Paciente-Primeiro Impaciente-depois) e IFPN (de *Impatient-First Patient-Next*, ou seja, Impaciente-Primeiro Paciente-Depois) que demonstraram ser efetivas em reduzir as taxas

de perda de requisições mas podem provocar aumentos no tempo de resposta. No intuito de otimizar os benefícios de cada política reativa de QoS propomos uma abordagem híbrida multi-nível que combina controle de admissão e escalonamento.

As novas estratégias são avaliadas utilizando o simulador e comparadas a um cenário base executando a política de escalonamento de melhor-esforço FIFO e nenhum controle de admissão. Os resultados demonstram a efetividade das novas políticas e o mecanismo híbrido multi-nível apresentou-se como o mais eficiente mecanismo para garantir o QoS considerando a reatividade.

Abstract

The huge success of the Internet has posed new challenges in terms of applications and user satisfaction. In order to meet the Quality of Service (QoS) expected by users, there are new requirements, such as performance and scalability, that must be fulfilled by the applications. Due to these requirements, QoS has become a relevant topic of interest. Several mechanisms to provide QoS have been proposed, but they usually do not consider aspects related to reactivity, e.g., how users react to variable server response time.

This work addresses the use of reactivity to design novel QoS strategies. First, we demonstrate how the reactivity may be modeled and replicated using a methodology to correlate the measures of the response time and IAT (inter-arrival time) of user requests. Based on the model, we implement a new version of the *httperf* workload generator that is capable of reproducing the reaction of users. We evaluate the impact of a reactive workload based on the TPC-W benchmark to the performance of an actual Web server, and verified that reactive workloads are different from the non-reactive ones, affecting the server's load as well as its throughput and response times.

In order to evaluate the impact of the reactivity and the new QoS strategies, we design a simulator of Internet applications named *USAR-QoS*. We prepare it to implement admission control and scheduling policies. Using the simulator, we evaluate the behavior of each user class as defined by the reactivity model.

We then propose new admission control strategies capable of rejecting requests and sessions according to a reaction-based criteria. By simulating the new strategies we verify that they are effective to maintain the response time but may cause an increase in the request loss rates. We also present the PFIN (Patient-First Impatient-Next) and the IFPN (Impatient-First Patient-Next) scheduling strategies. We verify that they are effective for reducing the user request loss rates but may increase the response time. In order to optimize the benefits of each reactive approach we propose a hybrid multi-level approach that combines admission control and scheduling.

The new QoS strategies are evaluated through simulation and compared to a scenario

running the simple Best Effort FIFO scheduling approach and no admission control. The results demonstrate the effectiveness of the new policies and the hybrid multi-level approach presented to be the most effective mechanism to guarantee QoS considering the reactivity.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A rápida evolução e a grande diversidade de serviços disponibilizados despertaram o interesse dos usuários e, em pouco tempo, a Internet alcançou público em todo o mundo. No início, os objetivos principais de construir a rede estavam ligados a questões militares e governamentais. Com o passar do tempo, esses objetivos se diversificaram, alcançando um âmbito muito mais amplo que passou a atingir não somente aquele público inicial, mas toda a sociedade.

A grande variedade dos serviços oferecidos na Internet e a sua freqüente utilização por milhões de pessoas fizeram com que os usuários se tornassem cada vez mais exigentes, requerendo serviços que, além de fáceis de usar e com conteúdo significativo, respondessem de forma rápida e precisa às suas solicitações. Assim, diversos problemas surgiram devido às limitações técnicas e ao uso intenso, expondo as contingências da Internet.

Para o atendimento da demanda crescente, alguns requisitos se tornaram fundamentais e exigiram um grande esforço da comunidade técnico-científica para aperfeiçoar a Internet. Requisitos como escalabilidade, segurança, desempenho, extensibilidade, entre outros, se tornaram indispensáveis para o sucesso da grande rede e a satisfação dos seus usuários. Prover um serviço com qualidade para os usuários não somente com relação ao conteúdo disponibilizado, mas também com relação a esses requisitos se tornou fundamental. O atendimento a eles constitui uma importante linha de pesquisa em redes de computadores

e sistemas distribuídos, denominada Qualidade de Serviço (QoS).

Ao longo dos anos, diversos mecanismos para prover QoS em redes de computadores foram propostos. Para aplicações Internet foram desenvolvidas abordagens como controle de admissão, escalonamento, balanceamento de carga, alocação dinâmica de recursos, entre outros, com a finalidade de atender de maneira mais eficiente e precisa os usuários. Esses métodos permitem que um servidor *Web* ou um conjunto de servidores em contenção atendam às requisições dos usuários otimizando o uso dos recursos. Entretanto, tais métodos desenvolvidos até então são eficientes e provêem bons resultados, mas uma solução definitiva de QoS ainda não existe.

Neste contexto, um tópico não explorado completamente pelas soluções de QoS existentes é o uso de conhecimento sobre as características dos usuários, em especial o conhecimento sobre o seu comportamento em circunstâncias em que o serviço oferece melhor ou pior qualidade. Os mecanismos tradicionais tipicamente se baseiam somente no conhecimento sobre a intensidade de carga do servidor e o tipo do serviço provido, podendo falhar em atender a usuários que possuam diferentes características em termos da tolerância ao QoS provido. Deste modo, o estudo de técnicas que considerem as diferenças de comportamento dos usuários pode ser muito útil, permitindo que um maior número deles atinja um nível mais alto de satisfação à QoS provida por um servidor.

O comportamento de um usuário, ao interagir com uma aplicação, pode ser identificado e mensurado através de diversos critérios, como por exemplo as páginas requisitadas, o tempo de atividade, os serviços (funções) da aplicação requisitados, a duração das sessões e o tempo entre requisições consecutivas (IAT). Do ponto de vista de um servidor, o comportamento corresponde a sequências de *ações* em que o usuário clica em um *link* ou requisita uma página em um programa navegador.

Em especial, uma dimensão importante do comportamento do usuário pode ser expressa pela *Reatividade*. Este conceito parte do princípio de que as ações de um usuário dependem das respostas servidas pela aplicação anteriormente, ou seja, depende do QoS percebido pelo usuário, que reage de acordo com diferentes padrões de comportamento ao serviço provido. As ações dos usuários, por sua vez, impactam a carga submetida ao servidor,

influenciando o seu desempenho.

Este trabalho tem a finalidade de estudar o uso de conhecimento sobre o comportamento reativo dos usuários para avaliar o seu impacto sobre uma aplicação Internet e propor melhorias que permitam um aumento da satisfação dos usuários ao QoS provido. Em primeiro lugar, deseja-se identificar a reatividade e propor um modelo de representação desse comportamento. Utilizando o modelo proposto, objetiva-se realizar a geração de cargas de trabalho que permitam avaliar o impacto de uma carga com características reativas sobre o desempenho de uma aplicação e, a partir disto, propor estratégias que permitam ao serviço atender de modo mais adequado ao seu público.

1.1 Objetivo

Este trabalho tem como objetivo estudar o comportamento reativo dos usuários de aplicações Internet em relação ao desempenho provido pelo serviço e propor modelos de Qualidade de Serviço mais adequados que considerem o comportamento dos clientes. Esta tarefa representa um grande desafio, pois envolve um estudo aprofundado dos mecanismos utilizados por servidores Internet para proverem seus serviços, bem como o conhecimento de técnicas para caracterizar e reproduzir padrões de comportamento.

Em primeiro lugar, o trabalho pretende verificar se é possível que o comportamento reativo dos clientes seja identificado, caracterizado e replicado. Deseja-se identificar a reatividade em aplicações reais, caracterizando as informações identificadas com a finalidade de estabelecer um modelo.

A partir da obtenção e validação de um modelo, o trabalho objetiva avaliar as implicações da reatividade para um serviço Web, verificando quais os impactos que uma carga de trabalho com características reativas possa trazer em comparação a uma carga de trabalho baseada nos modelos atuais de geração de carga sintética. Caso haja diferenças significativas, será possível concluir que as cargas sintéticas tradicionais estão incompletas e os mecanismos para prover QoS utilizados demandam aperfeiçoamento.

Por fim, uma vez que o impacto de cargas com características de comportamento reativo

esteja entendido e quantificado, deseja-se investigar e elaborar estratégias que permitam a um serviço prover uma Qualidade de Serviço mais adequada a cargas dessa natureza. O trabalho objetiva propor mecanismos de QoS mais adequados, apresentando uma evolução nos modelos atuais de QoS. Deseja-se apresentar mecanismos que sejam mais eficientes em atender às demandas específicas de cada usuário, de acordo com seu perfil, em especial, mecanismos de controle de admissão e escalonamento de requisições.

1.2 Contribuições

Aspectos reativos relacionados ao comportamento de usuários tendem a causar impacto direto nas características das cargas de trabalho submetidas ao servidor. À partir do momento em que consideramos a dinâmica reativa, torna-se necessário entender que mudanças ocorrem no desempenho e escalabilidade dos servidores de aplicações Internet, identificando que aspectos deverão ser considerados para a construção de modelos de Qualidade de Serviço mais próximos da realidade. Os modelos atuais de caracterização de cargas de trabalho e de desempenho de servidores não utilizam informações referentes à dinâmica reativa de seus usuários, assumindo simplesmente distribuições estatísticas em suas modelagens.

Sendo assim, a contribuição principal deste trabalho é o estudo do impacto do comportamento reativo sobre servidores *Web* e a proposição e implementação de estratégias de Qualidade de Serviço mais adequadas às características de seus usuários.

Durante o trabalho ficou claro que um estudo sobre a reatividade aplicada a QoS, como o desenvolvido, não havia sido realizado. Alguns trabalhos tratam do comportamento do usuário e, até mesmo, comprovam sua importância. Porém, estes trabalhos falham em aprofundar o assunto.

O trabalho apresentado nesta dissertação faz parte de um estudo mais amplo desenvolvido pelo Centro de Análise de Performance de Sistemas (CAMPS) do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais. Fizeram parte desse estudo outros pesquisadores que trouxeram importantes contribuições para o trabalho. Es-

tas contribuições estão publicadas nos artigos [Pereira et al., 2004a], [Pereira et al., 2004b], [Franco and Meira, Jr., 2004], [Pereira et al., 2005] e [Pereira et al., 2006c], no trabalho de dissertação de mestrado [Franco, 2004] e na defesa de proposta de tese de doutorado [Pereira, 2005].

Este trabalho é uma continuidade dos estudos anteriores e suas principais contribuições foram publicadas em [Pereira et al., 2006a], [Pereira et al., 2006b], [Silva et al., 2006] e [Silva et al., 2007]. São elas:

- Estudo do comportamento reativo e seu impacto em clientes e servidores a fim de entender teoricamente os possíveis efeitos da reatividade.
- Implementação de um simulador para avaliar o cenário típico de uma aplicação Internet (servidor e clientes), que permite a execução de políticas de controle de admissão e escalonamento para provimento de QoS.
- Avaliação das políticas de controle de admissão baseadas em reatividade inicialmente propostas em [Pereira, 2005] que utilizam conhecimento sobre o comportamento do usuário para admitir e rejeitar requisições.
- Proposição de novas políticas de escalonamento baseadas em reatividade que realizam o escalonamento de requisições de acordo com os perfis de comportamento dos usuários.
- Proposição de uma política de QoS híbrida que combina estratégias de controle de admissão e escalonamento baseadas em reatividade.
- Realização de avaliação experimental das novas políticas de QoS citadas nos itens anteriores através de simulação.

1.3 Organização

A dissertação está organizada de acordo com os capítulos listados abaixo.

O Capítulo 2 introduz os principais conceitos relacionados a aplicações Web, apresentando a arquitetura típica e as principais entidades envolvidas. Discute também o tema da reatividade e como ela pode ser identificada em um cenário real. Por fim, apresenta o conceito de Qualidade de Serviço nessas aplicações, abordando as principais estratégias utilizadas para melhorá-lo.

O Capítulo 3 discorre sobre os trabalhos relacionados aos principais temas tratados pelo trabalho. Primeiramente discute os trabalhos que tratam da identificação e modelagem do comportamento do usuário. A seguir, trata dos trabalhos de geração de cargas de trabalho. Finalmente, aborda o tema da Qualidade de Serviço para aplicações Web, discutindo como esse tópico tem sido tratado por diversos autores.

O Capítulo 4 trata da metodologia de caracterização de cargas de trabalho apresentada em [Franco, 2004] e o Modelo de Reatividade que foram avaliados em um estudo de caso.

O Capítulo 5 avalia a importância da reatividade, verificando a relavância do seu estudo no contexto de aplicações Internet. Para isto, primeiramente, avalia os requisitos para tornar possível a geração de cargas de trabalho que repliquem o comportamento reativo. Baseado nisto, discute as implementações realizadas sobre o gerador de cargas *httperf* para atender a esses requisitos. A nova versão do gerador é utilizada em um experimento real com a finalidade de avaliar o impacto de uma carga reativa sobre um servidor real. Além disto, o capítulo realiza uma análise teórica sobre o comportamento reativo e apresenta o simulador *USAR-QoS* utilizado para avaliar esse comportamento e a Qualidade de Serviço em uma aplicação Internet.

O Capítulo 6 apresenta as novas políticas para melhoria da QoS de um serviço baseadas em reatividade. Propõe novas abordagens de controle de admissão baseadas na rejeição de *bursts* ou sessões de um usuário. Propõe também novas políticas de escalonamento denominadas de PFIN (*Patient-First Impatient-Next*) e IFPN (*Impatient-First Patient-Next*). Discute também a junção das estratégias de controle de admissão e escalonamento em um mecanismo híbrido, seguindo parâmetros que otimizem o uso de cada uma das abordagens, de modo a reduzir os efeitos negativos e a reforçar os positivos de cada uma. Cada uma dessas políticas é avaliada utilizando o simulador *USAR-QoS*.

O Capítulo 7 conclui o trabalho, sumarizando os principais resultados e apresentando as propostas de evolução da linha de pequisa.

Capítulo 2

Aplicações Web

Este capítulo tem a finalidade de apresentar os principais conceitos envolvidos no trabalho, principalmente no que se refere à arquitetura de serviços tratada, ao conceito de reatividade, e às estratégias para melhorar a QoS nessa arquitetura. A seção 2.1 apresenta a arquitetura típica de um serviço Web, explicando quais os seus componentes e como eles interagem. A seção 2.2 descreve como é a organização dos elementos dessa arquitetura e quais os tipos de servidores que tipicamente a compõem. A seção 2.3 discute as principais medidas de interação utilizadas para avaliar a eficiência de um serviço. A seção 2.4 apresenta o conceito de *Qualidade de Serviço* em aplicações Web, bem como algumas das estratégias típicas existentes para provê-lo. E, por fim, a seção 2.5 introduz o conceito de *Reatividade* e discute como ela pode ser identificada e modelada.

2.1 Arquitetura

Os principais componentes envolvidos em um serviço Web são os clientes e os servidores. Um servidor Web é um programa apto a responder a uma requisição WWW, ou seja, uma mensagem codificada em certo protocolo (freqüentemente HTTP), requisitando uma ação que, geralmente, corresponde ao processamento e envio de páginas, isto é, documentos textuais que podem ser estáticos ou dinâmicos. As páginas estáticas são construídas a

partir de arquivos armazenados no servidor. As páginas dinâmicas são geradas em tempo real em função de parâmetros enviados na requisição. Estas páginas são compostas, em sua maioria, de resultados de consultas a bancos de dados, de resultados de processamento, ou de atualizações feitas no servidor. Os servidores Web processam requisições em paralelo, ou seja, várias requisições são atendidas simultaneamente, sendo que diversas estratégias são utilizadas para implementar o paralelismo.

As requisições submetidas aos servidores Web são enviadas por clientes. Um cliente Web é um software que permite a obtenção dos recursos disponíveis na WWW. Os recursos podem ser as páginas Web que são requisitadas e visualizadas através de um software chamado navegador. Uma página é normalmente composta por um conjunto de arquivos de vários formatos como texto, imagens, animação, áudio, vídeo e código executável, entre outros. Cada arquivo que compõe uma página é requisitado separadamente, sendo feita uma requisição HTTP para cada arquivo. Portanto, a requisição de uma página pode gerar vários pedidos ao servidor, um para cada arquivo que compõe a página pedida [Meira, Jr. et al., 2000].

Tanto o termo *cliente* como o termo *servidor* são geralmente utilizados para se referir ao computador no qual o programa cliente ou servidor é executado, incluindo o hardware e o sistema operacional. Ao se falar em um cliente pode-se referir também a agentes de software construídos para executar requisições WWW, tais como robôs. Já o termo servidor pode se referir também a um conjunto de máquinas, também chamadas individualmente de servidores que, juntas, provêm o serviço, sendo utilizadas em conjunto por motivos de especialização do serviço prestado por cada uma (por exemplo, podemos ter servidores provendo especificamente serviços de banco de dados, aplicações, ou atendimento a requisições Web), ou para distribuir as requisições de modo a realizar um melhor atendimento em termos da rapidez de resposta.

2.2 Organização

A arquitetura básica de um serviço Web consiste de um ou mais servidores que, juntos, provêm o serviço implementado. Geralmente pode-se distinguir as seguintes especializações de servidores presentes em um serviço:

- **Servidor Web**: servidor responsável pelo recebimento, organização, processamento (ou seja, execução e envio de requisições aos servidores de aplicação e banco de dados, quando necessário) e resposta às requisições dos clientes.
- **Servidor de Aplicação**: implementa a lógica associada à aplicação disponibilizada pelo serviço tendo um papel intermediário, já que recebe as solicitações de serviço advindas do servidor *Web* e, ao processá-las, encaminha as requisições necessárias ao servidor de banco de dados. Após receber as respostas advindas do servidor de banco de dados e, após finalizar os processamentos devidos, retorna a resposta ao servidor Web. Entre as suas funções estão a manutenção de estado, controle de acesso, persistência e segurança da aplicação.
- **Servidor de Banco de Dados**: sendo responsável pelo armazenamento dos dados, o servidor de banco de dados recebe e processa as solicitações de acesso advindas do servidor de aplicação.

A figura 2.1 apresenta um exemplo de organização destes servidores, onde é importante notar que o número de cada um deles pode ser variado, conforme a estrutura escolhida para o serviço, de acordo com a sua demanda. Por exemplo, pode haver somente um servidor Web, um servidor de aplicação e outro de banco de dados, assim como podem existir vários de cada um deles, sendo que, neste caso, um mecanismo de distribuição das requisições é feito por um outro tipo de servidor (chamado de *balanceador de carga*). Outro aspecto importante diz respeito ao fato de que diferentes tipos de servidores podem ser executados em uma mesma máquina. Por exemplo, é comum que o servidor Web e o servidor de aplicação estejam em uma mesma máquina.

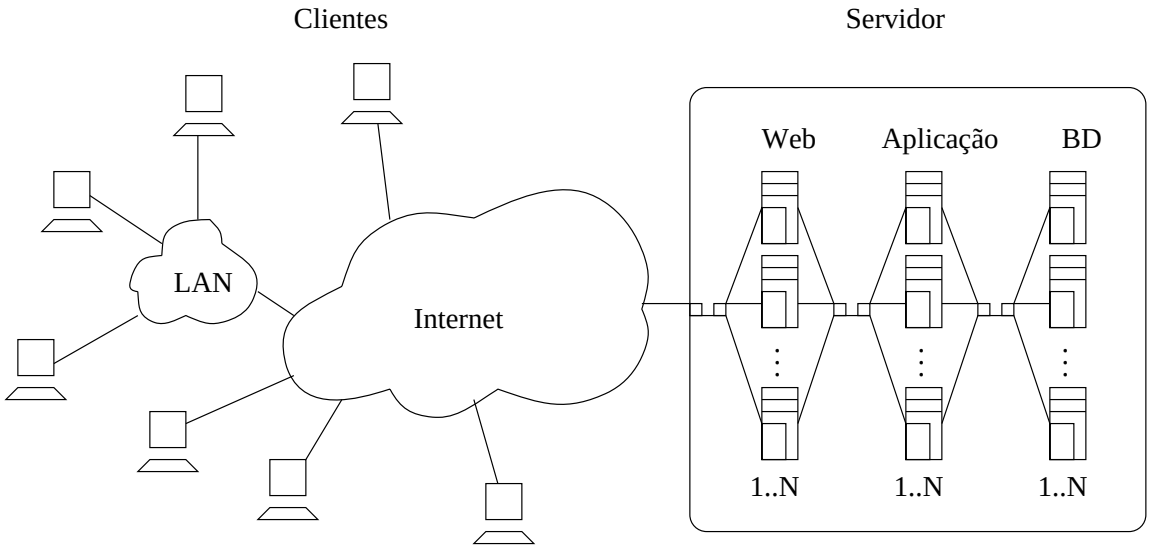


Figura 2.1: Arquitetura de um serviço Web

2.3 Medidas de Interação

É importante definir algumas medidas muito importantes que nos ajudam a entender os aspectos de eficiência de um serviço.

Na figura 2.2 vemos a representação simplificada de um servidor e um cliente, onde o cliente submete requisições e o servidor as responde. As requisições e respostas enviadas trafegam pela Internet, e geralmente são realizadas utilizando o protocolo HTTP. Quando o cliente deseja um dado serviço (uma página, por exemplo) ele envia uma requisição. Esta trafega pela rede até chegar ao servidor que realiza o seu processamento e responde após certo tempo, enviando a resposta. Esta resposta trafega pela rede, passando pela Internet, até chegar ao cliente, que antes ou após recebê-la, pode realizar mais requisições, num processo que se repete, até que o cliente termine sua interação com o serviço. Assim, podemos observar algumas medidas de tempo que são importantes para analisar o cenário da figura:

- **Tempo de Rede**: corresponde ao tempo que a requisição ou a resposta leva para percorrer a rede, considerando a Internet e as demais redes percorridas desde o cliente até o servidor, ou vice-versa.

- **Tempo de Serviço**: corresponde ao tempo que o servidor demora para atender a uma requisição, considerando desde o instante em que a requisição chega ao mesmo até o momento em que ele completa o processamento e envia a resposta.

- **Tempo de Resposta** ou *Latência*: essa medida se refere ao tempo de atendimento de uma requisição de um cliente, desde o instante de envio da submissão até o recebimento da resposta correspondente. Essa medida deve englobar, assim, tanto o *tempo de serviço* como o *tempo de rede*.

- **Think-time (Tempo de Processamento do Cliente)**: corresponde ao tempo que o cliente despende entre o recebimento da resposta de uma requisição e a realização de uma outra requisição seguinte. Este valor pode ser negativo quando um cliente não espera o recebimento da resposta da requisição anterior para enviar uma outra requisição.

- **IAT (Inter-arrival time)**: corresponde ao tempo decorrido entre a chegada de requisições consecutivas de uma mesma sessão de interação de um usuário em um servidor.

- **IRT (Inter-request time)**: corresponde ao tempo decorrido entre o envio de duas requisições consecutivas em uma sessão de um usuário.

O *tempo de resposta* é um fator crítico para usuários de sistemas interativos e é evidente que a satisfação deles aumenta à medida que essa medida diminui, sendo que variações modestas do seu valor médio são aceitáveis, mas grandes variações podem afetar o comportamento do cliente. Trabalhos anteriores [Menascé et al., 2004] descrevem que um tempo de resposta de até 1 segundo está dentro de uma margem ideal em que o provimento de serviço não sofre interrupções. Já um valor próximo de 10 segundos se torna crítico e o usuário pode perder a atenção e terminar sua interação com o sistema. Assim, adotaremos em nossas análises o valor de 10 segundos como o limite máximo para o tempo de resposta percebido por um cliente como aceitável.

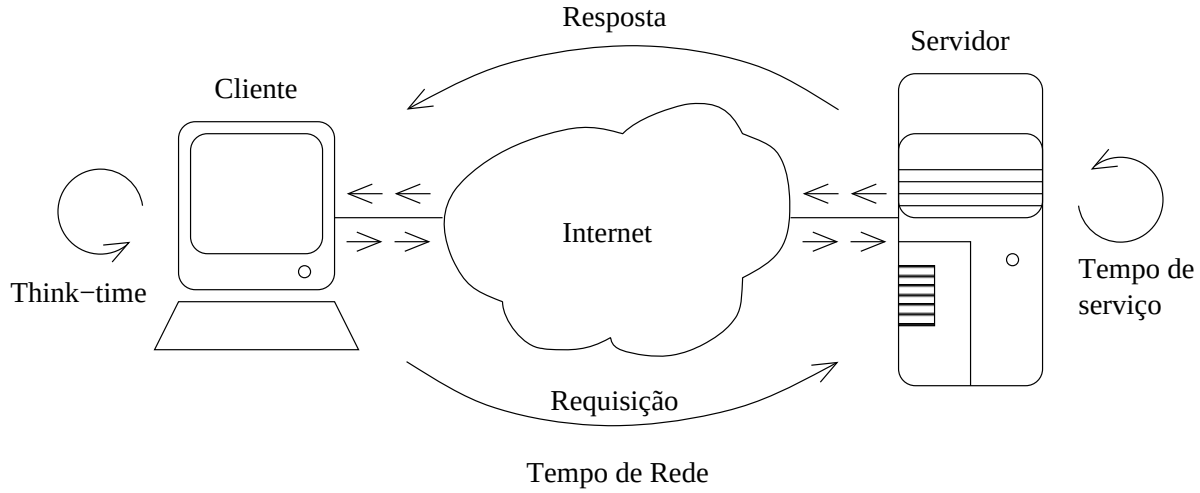


Figura 2.2: Interação entre um cliente e um servidor Web

2.4 Qualidade de Serviço

A resposta de um servidor, demandada para o atendimento de uma dada requisição, depende de diversos fatores relacionados à organização, arquitetura, e condições de carga presentes em cada instante. Quanto mais rápido for o atendimento às requisições submetidas, melhor será a qualidade da resposta percebida pelo cliente. Assim, é importante citar um conceito muito utilizado para se referir à eficiência de atendimento provida por um servidor:

- **Qualidade de serviço, ou QoS (***Quality of Service***)**: este conceito diz respeito à percepção que se tem sobre a garantia, segurança e eficiência com que um servidor atende a seus usuários. Pode ser medido (quantificado) através de métricas como o *tempo de resposta* e o *throughput* (ou *taxa de serviço*, isto é, a taxa de requisições respondidas a cada instante de tempo).

Existem basicamente duas maneiras de prover garantias de Qualidade de Serviço em aplicações *Web*. A primeira consiste, simplesmente, em prover recursos suficientes para atender às condições de pico em termos da expectativa do uso do serviço, mais uma margem de segurança. Este método é simples, mas muitos acreditam que é muito caro na prática,

além do fato de que não é capaz de se adaptar a condições onde a demanda cresce muito rapidamente, acima do previsto, já que disponibilizar mais recursos exige um certo tempo.

A segunda maneira consiste em utilizar técnicas de controle de QoS de modo a garantir que as métricas estejam dentro de limites específicos, de acordo com a política adotada. Diversas estratégias podem ser adotadas e, entre elas, as mais importantes são as seguintes:

- **Controle de Admissão**: consiste em implementar mecanismos que selecionam as requisições que serão atendidas, de acordo com políticas diversas, rejeitando algumas delas, caso seja necessário.

- **Balanceamento de Carga**: consiste de mecanismos que visam a distribuição das requisições entre um conjunto de servidores para otimizar o desempenho provido pelo serviço como um todo.

- **Escalonamento**: através de técnicas de escalonamento pode-se estabelecer prioridades de processamento, de acordo com políticas diversas. As requisições são colocadas em uma fila, e, através das políticas, se escolhe quais requisições serão atendidas primeiramente pelo servidor.

- **Alocação Dinâmica de Recursos**: semelhante ao balanceamento de carga, estas técnicas visam a distribuição de execução dos serviços entre os recursos físicos disponíveis. Aqui, dinamicamente se alocam os recursos, em função das necessidades do provedor do serviço.

Estas estratégias são utilizadas, entre outras finalidades, para otimizar o acesso aos recursos e o atendimento das requisições, de modo a prover um serviço com maior qualidade, para um maior número de clientes, de acordo com a política de atendimento de requisições definida.

2.5 Reatividade

A *Reatividade* representa o modo como um usuário reage de acordo com o QoS percebido por ele. À medida que o servidor muda seu comportamento em termos das respostas enviadas, o usuário reage de um modo diferente de acordo com o seu perfil. Além disto, conforme o comportamento dos usuários, a carga submetida ao servidor poderá variar, impactando as respostas às requisições submetidas.

Uma das principais medidas que permitem avaliar a qualidade oferecida por um serviço é o *tempo de resposta*. Este pode variar de acordo com diversos aspectos, como por exemplo, a carga total que o servidor deve servir e sua capacidade para atendê-la.

Já o comportamento do usuário, reagindo à QoS, pode ser captado através do IRT entre duas requisições. Esta medida representa o tempo em que o usuário leva para submeter uma nova requisição. O IRT pode variar de acordo com o seu perfil ou de acordo com o *tempo de resposta* observado pelo usuário, que pode ser maior ou menor de acordo com a capacidade do usuário de processar o conteúdo servido ou esperar pelo recebimento da requisição submetida. Em alguns casos, o usuário pode submeter uma nova requisição antes ou depois do recebimento da resposta anterior. Este comportamento se repete até que o usuário decida terminar sua interação.

Como será discutido no Capítulo 3, diversos trabalhos desenvolveram metodologias para caracterizar cargas de trabalho considerando métricas relacionados ao usuário ou ao servidor, mas ignorando a correlação entre elas, isto é, a reatividade. Com a finalidade de complementar estes estudos, o trabalho aqui descrito apresenta uma metodologia para caracterizar a reatividade utilizando informações associadas ao comportamento do servidor (*tempo de resposta*) e do usuário (IRT).

Capítulo 3

Revisão Bibliográfica

Nesta seção apresentamos um conjunto de trabalhos relacionados ao trabalho aqui proposto. A seção 3.1 apresenta diversos estudos que abordam a identificação, análise e modelagem de aspectos de comportamento dos usuários. A seção 3.2 discute o problema da geração de cargas de trabalho. A seção 3.3 apresenta os trabalhos que tratam do problema do desempenho e Qualidade de Serviço, seja através da análise e caracterização, seja através da proposição e implementação de mecanismos e estratégias. E, por fim, a seção 3.4 apresenta uma conclusão final sobre os trabalhos relacionados examinados.

3.1 Comportamento do Usuário

No intuito de aperfeiçoar as técnicas de QoS, nosso trabalho propõe o uso de informação sobre a reatividade do usuário, isto é, o comportamento do usuário reagindo ao desempenho observado de um serviço, para propor estratégias e políticas mais adequadas a cenários reais. O comportamento do usuário tem sido observado e analisado através de diversos critérios, como a natureza das requisições, padrões de navegação, tipos de serviços ou funções acessadas [Menascé et al., 2003a], IAT entre as requisições, além de outras informações.

[Henderson, 2001] utiliza a latência para estudar o comportamento do usuário em uma

aplicação de jogos, detectando que o atraso da rede tem efeito sobre o comportamento dos jogadores, provocando, inclusive, que os usuários terminem suas interações e abandonem o serviço.

[Chatterjee et al., 1998] modela o fluxo de cliques dos usuários no contexto de anúncios de propagandas na Web e [Costa et al., 2004] analisa a correlação entre requisições em aplicações *streaming media*, tentando determinar tendências no processo de interação do usuário.

[Balachandran et al., 2002] caracteriza o comportamento do usuário em uma rede pública sem fio, considerando a distribuição de usuários, duração de sessões, popularidade da aplicação e mobilidade. O autor pretende com essa caracterização otimizar o acesso dos usuários às estações que provêm os diversos serviços *wireless*.

[Hlavacs and Kotsis, 1999] propôs um *framework* para modelagem do comportamento do usuário, estruturado e construído de maneira *top-down*, que consiste de várias camadas, sendo baseado em modelos matemáticos. Esse trabalho se preocupa, principalmente, em modelar o tráfego HTTP que é utilizado para simulação do tráfego na rede. Ele também foi usado como base para a construção de um gerador de carga orientado ao usuário [Hlavacs et al., 2000].

De fato, nenhum destes estudos modela correlações entre o comportamento do servidor e o comportamento do usuário. Eles falham em capturar aspectos relacionados ao modo como os usuários reagem à QoS provida pelo serviço. Outros trabalhos, entretanto, tentam entender a influência da QoS no comportamento dos clientes.

Para que um serviço esteja disponível, não é necessário simplesmente que ele responda às requisições: o tempo de resposta também é importante. [Selvridge et al., 2000] comprova que longos atrasos aumentam a frustração do usuário e diminuem a eficiência e a possibilidade de a tarefa ser realizada com sucesso.

[Bhatti et al., 2000] apresenta um experimento onde os usuários podem classificar a Qualidade de Serviço provida por uma aplicação de comércio eletrônico que apresenta tempo de resposta variável. Os autores perceberam que os usuários mudam seu comportamento, por exemplo, terminando suas sessões em alguns casos. Entre outras coisas, dois

aspectos foram observados:

1. os usuários toleram diferentes níveis de atraso para diferentes tarefas, ou seja, *seu comportamento é variado;*

2. *usuários que experimentam longos atrasos têm uma maior possibilidade de terminar suas sessões ativas prematuramente.*

Estas conclusões sugerem que o comportamento dos usuários de aplicações Web é altamente influenciado pelo desempenho percebido por eles daquele serviço.

Os estudos acima, embora apontem a importância do QoS no comportamento dos usuários, falham em modelar este comportamento reativo. Deste modo, um trabalho que vise entendê-lo com profundidade, utilizando os resultados para desenvolver técnicas destinadas a prover serviços com melhor QoS, pode ser de grande utilidade.

3.2 Geração de Cargas de Trabalho

Geradores de cargas de trabalho são ferramentas destinadas a gerar *logs* sintéticos, compostos por requisições que simulam requisições reais de usuários, capazes de exercitar os servidores através do envio repetitivo dessas requisições. Assim como a caracterização, a geração de cargas de trabalho é um instrumento essencial para avaliar sistemas Web. Deste modo, geradores de cargas de trabalho Web têm sido estudados extensivamente.

SPECWeb99 [Spe, 1999], WebBench [Web, 2002] e TPC-W [García and García, 2003] são *benchmarks* destinados a prover referências sobre a composição de cargas de trabalho que possam ser utilizadas na correta avaliação de um servidor Web. Eles provêm padrões representativos que permitem a avaliação da capacidade de um servidor atender o público esperado.

SURGE [Barford and Crovella, 1998] e *httperf* [Mosberger and Jin, 1998] constituem geradores de carga, desenvolvidos com a finalidade de exercitar servidores Web através da submissão de seqüências de requisições que simulam usuários reais.

Além desses geradores de carga, outros foram desenvolvidos com a finalidade de serem utilizados para avaliar aplicações e cenários específicos. Gismo [Jin and Bestavros, 2001] e MediSyn [Tang et al., 2003] foram desenvolvidos para avaliar aplicações de *streaming media*. ProWGen [Busari and Williamson, 2002], por sua vez, é destinado à avaliação de servidores *proxy-cache*.

Estas ferramentas de geração de carga são poderosas, mas falham em simular cenários com reatividade. Eles replicam um conjunto de padrões de comportamento de usuários sem considerar que estes padrões podem variar de acordo com as respostas dadas por um serviço. Esses geradores adotam um processo de envio de requisições que é independente do desempenho, gerando a mesma carga em cenários distintos. A estratégia de geração não considera nem o desempenho do servidor nem as características específicas dos usuários.

Sendo assim, o estudo de técnicas capazes de preencher esse espaço é importante para melhorar os modelos existentes de geração de carga, tornando possível que avaliações mais precisas de servidores Web sejam realizadas. Logo, a geração de cargas mais realistas tem a finalidade de melhorar os modelos de desempenho de servidores, bem como atuar em aspectos cruciais de QoS como sua escalabilidade e disponibilidade.

3.3 Qualidade de Serviço em Aplicações Web

O objetivo principal do trabalho refere-se à elaboração de estratégias mais adequadas para prover um serviço com melhores condições de Qualidade de Serviço. Sendo assim, é fundamental analisar os trabalhos já realizados que tratam principalmente de estratégias para prover QoS, mas também de análises e estudos sobre o desempenho de servidores Web. Assim, nesta seção procuramos abordar os trabalhos correlatos para os principais tópicos relacionados.

3.3.1 Avaliação de Desempenho

Avaliações de desempenho consistem em um importante tema destinado a verificar a Qualidade de Serviço provida por um site Web quando diferentes cargas de trabalho são aplicadas sobre ele. Muitos trabalhos tratam deste problema, estudando diversos aspectos relacionados.

[Krishnamurthy et al., 2003] apresenta o desenho, implementação e avaliação de um servidor baseado em um mecanismo proposto em [Krishnamurthy and Wills, 2002] que permite a um servidor Web detectar informações referentes à conectividade dos clientes com a finalidade de tomar ações referentes à cada requisição. Os resultados obtidos mostram a possibilidade de melhorar o tempo de resposta através de ações baseadas em algum tipo de classificação dos clientes.

[Nahum et al., 2001] avalia como o ambiente onde o servidor Web está localizado influencia no seu desempenho. Eles emulam o ambiente de uma rede WAN introduzindo elementos como o atraso e perda de pacotes, e realizam testes para analisar o impacto do novo cenário no *throughput* e no tempo de resposta do servidor. Os autores apontam que os *benchmarks* existentes não consideram estes componentes em seus modelos.

[Avritzer et al., 2002] apresenta uma abordagem de avaliações de desempenho e realiza a sua aplicação em um estudo de caso. Primeiramente, dados são coletados e analisados no ponto de vista das suas características. Em seguida, o impacto de desempenho em termos do consumo de CPU é avaliado para cada tipo de requisição. Finalmente, os pontos de contenção são identificados e soluções são propostas.

[Feitelson, 2002] mostra que o desempenho de um sistema depende das características da carga que ele deve servir. Assim, avaliações de desempenho requerem que o uso de cargas de trabalho representativas sejam utilizadas para produzir resultados representativos.

[Olshefski et al., 2002] apresenta um mecanismo para servidores Web capaz de mensurar o tempo de resposta percebido pelos clientes. Para isto, é utilizado um modelo baseado no protocolo TCP que quantifica o efeito que as quedas de conexão têm sobre o tempo de resposta dos clientes, utilizando três medidas simples do servidor: a razão de quedas de

conexão, a razão da aceitação de conexões e a razão das conexões completadas.

[Arlitt and Williamson, 1997] apresenta um estudo sobre a caracterização de cargas de seis diferentes conjuntos de dados com diferentes características, discutindo questões de desempenho e sugerindo melhorias nos servidores Web. As proposições discutidas são focadas em estratégias baseadas em caches.

[Cherkasova et al., 2003] propõe um monitor localizado no lado cliente (*back-end*) que permite avaliar o desempenho de um site Web através de uma coleta passiva de pacotes. Ela apresenta uma implementação inicial do monitor e analisa o desempenho em três diferentes sites Web.

Deste modo, a melhoria da QoS tem sido uma tarefa desafiadora tratada por muitos trabalhos em diversas áreas. No contexto de serviços Internet, diversos estudos tratam de técnicas e estratégias para melhorar a QoS da Internet, como controle de admissão, balanceamento de carga, alocação de recursos e escalonamento. Abaixo tratamos destes temas.

3.3.2 Modelos de QoS

Em termos da arquitetura de redes de computadores, atualmente existem dois modelos para implementar QoS na Internet: serviços Integrados (IntServ) [Braden et al., 1994] e serviços diferenciados (DiffServ) [Blake et al., 1998]. IntServ é um modelo baseado em reserva de recursos, enquanto serviços diferenciados é uma proposta onde os pacotes são marcados de acordo com classes de serviços pré-determinadas ([Xiao and Ni, 1999], [dos Santos, 1999]).

O modelo de serviços integrados é caracterizado pela reserva de recursos. Antes de iniciar uma comunicação, o emissor solicita ao receptor a alocação de recursos necessários para se definir uma boa qualidade na transmissão dos dados. O protocolo RSVP (Resource Reservation Protocol) é utilizado, nesse modelo, para troca de mensagens de controle de alocação dos recursos. Esta diz respeito à largura de banda e ao tempo em que será mantida a conexão. Neste período de tempo, o emissor daquele serviço tem uma faixa da largura de banda disponível para transmitir seus dados.

O IntServ é caracterizado pela alocação de recursos para dois novos tipos de serviços que são os garantidos para aplicações que necessitam de um atraso constante, e os de carga controlada para aplicações que requerem segurança e destacam o serviço de melhor esforço.

O modelo de serviços diferenciados implementa QoS com base na definição de tipos de serviços. No cabeçalho de um pacote IP existe um campo chamado TOS (*Type of Service*) que pode representar esse tipo. No entanto, os serviços diferenciados ampliam a representação e o tratamento que pode ser dado para encaminhar um pacote, definindo um novo layout para o TOS, passando a chamá-lo de DS Field (Differentiated Service Field). No DS Field, são codificadas as classes para os serviços diferenciados. O campo TOS já existia na definição do pacote IP, mas só recentemente se definiu uma utilização para o mesmo.

A arquitetura DiffServ parte do princípio que domínios adjacentes tenham um acordo sobre os serviços que serão disponibilizados entre os mesmos. Este acordo denomina-se SLA (*Service Level Agreement*), que determina as classes de serviços suportadas e a quantidade de tráfego na banda entre os domínios. Os domínios podem definir um SLA estático ou dinâmico, sendo que, neste último caso, um protocolo de sinalização e controle será necessário para o gerenciamento da banda.

3.3.3 Controle de Admissão

Para a disponibilização de uma aplicação Internet, mecanismos de controle de admissão são uma das principais técnicas para prover Qualidade de Serviço. [Iyengar et al., 1997] analisa um servidor Web e, a partir da sua carga de trabalho, conclui que é fundamental o uso de algum mecanismo de controle de admissão para que o servidor atenda seus usuários e consiga um bom nível de desempenho.

[Chen et al., 2001] propõe um algoritmo de controle de admissão chamado *PACERS* para prover diferentes níveis de serviço baseado nas características da carga de trabalho do servidor. A qualidade do serviço é garantida através da alocação periódica de recursos de sistema baseado na estimativa da taxa de requisições e nos requisitos de sistema das

tarefas que tem maior prioridade. Os autores apresentam a análise teórica e experimental do algoritmo, bem como os ganhos obtidos em termos da taxa de requisições atendidas e do tempo de resposta para as requisições com maior prioridade, em diversos cenários em termos da carga de trabalho.

[Bhatti and Friedrich, 1999] discute diversas questões referentes a QoS para servidores Web e investiga um mecanismo simples de controle de admissão, sugerindo que as requisições sejam classificadas em três níveis de prioridade utilizando informação sobre o endereço IP e os Web sites requisitados.

[Cherkasova and Phaal, 1999] apresenta um mecanismo de controle de admissão baseado em sessões chamado *SBAC*, que previne situações de sobrecarga em um servidor e garante que sessões mais longas sejam completadas. Para isso, o mecanismo permite que novas sessões sejam aceitas somente quando o servidor tem capacidade de processar todas as requisições futuras daquela sessão. Assim, para garantir estabilidade, o mecanismo proposto abre mão da capacidade de atender novas sessões. Os mesmos autores em [Cherkasova and Phaal, 2000] complementam o trabalho através de um mecanismo que prevê o número de novas sessões que um servidor consegue atender, e ainda garante o processamento de um número estimado de requisições futuras. O trabalho pode ser encontrado também em [Cherkasova and Phaal, 2002]. A abordagem por sessão é utilizada em nosso trabalho.

Por fim, [Carter and Cherkasova, 2000] desenvolve um método para remover requisições pertencentes a sessões inativas, obtendo melhorias no desempenho do servidor.

3.3.4 Escalonamento

Mecanismos de escalonamento também são uma abordagem importante em aplicações Internet, pois constituem um mecanismo bastante flexível e efetivo para implementar diferenciação do serviço em um servidor.

[Chen and Mohapatra, 2003] explora e modela as dependências entre as requisições das sessões e, utilizando as dependências encontradas, modela algumas funções de conformação

do tráfego. A partir daí, propõe e implementa um algoritmo de escalonamento dinâmico para controlar sobrecargas em servidores Web, que utiliza as funções para estimar a probabilidade de término de sessões. Baseado nisso, o escalonamento é realizado e as requisições são atendidas. Esse é mais um trabalho sobre mecanismos baseados nas divisões das requisições em sessões.

[Ferrari, 2000] investiga o efeito no desempenho de um serviço utilizando os algoritmos de escalonamento do tipo *Priority Queuing* (PQ) e *Weighted Fair Queuing*(WFQ).

[Ye et al., 2005] utiliza somente uma fila para todas as requisições submetidas ao servidor utilizando, entretanto, uma regra de escalonamento avançada para diferenciar o serviço. As regras utilizadas baseadas em planos de produção do domínio de empresas manufatureiras são a *Weighted Shortest Processing Time* (WSPT), *Apparent Tardiness Cost* (ATC) e *Earliest Due Date* (EDD). O desempenho dessas regras são comparados com o escalonamento típico FIFO e as regras WSPT e ATC apresentam um melhor desempenho quando o serviço excede a capacidade do servidor.

3.3.5 Outras Estratégias

Alguns trabalhos fazem a tentativa do uso de mais de uma técnica para resolver o problema da sobrecarga em servidores. Em [Urgaonkar and Shenoy, 2005] os autores tratam do problema da sobrecarga propondo um mecanismo que permite a aplicação, em um único sistema, das técnicas de controle de admissão, provisionamento dinâmico de recursos e degradação adaptativa da QoS. Eles implementam um protótipo e analisam os seus benefícios.

[Lee et al., 2002] apresenta dois algoritmos para realizar o controle de admissão que maximizam o lucro potencial ou o número de clientes admitidos no serviço. Além disto, apresenta dois algoritmos adaptativos para reduzir o custo do controle de admissão, um baseado no servidor e outro distribuído.

Já [Elnikety et al., 2004] apresenta um mecanismo baseado em controle de admissão e no escalonamento de requisições para obter tanto um funcionamento estável em condições

de sobrecarga, como tempos de resposta otimizados. Baseado no custo de execução dos diversos tipos de requisições submetidas, o mecanismo realiza a proteção contra sobrecarga e um escalonamento preferencial. A principal contribuição deste trabalho está no fato de que para a sua implementação, não são necessárias modificações muito extensas na aplicação servidora, já que as modificações puderam ser feitas numa entidade externa ao servidor (um *proxy-cache*).

[Brataas and Hughes, 2004] apresenta uma abordagem hierárquica para prover uma arquitetura escalável através da aplicação de modelos estáticos e dinâmicos, verificando alguns requisitos importantes para avaliar a escalabilidade da arquitetura de um sistema.

[Abdelzaher and Bhatti, 1999a, Abdelzaher and Bhatti, 1999b] propõe um mecanismo para adaptar o conteúdo provido em vez de rejeitar requisições para diminuir a carga do servidor.

[Chandra and Shenoy, 2003] trata da implementação de um protótipo de um centro de dados, que é utilizado para o estudo da eficiência da alocação dinâmica de recursos para tratar do problema de sobrecargas repentinas.

Alguns trabalhos propõem arquiteturas orientadas por serviço. [Menascé et al., 2004] propõe uma onde cada componente provê um conjunto de serviços para os demais de modo que eles sejam *QoS-aware*, sendo capazes de fazer negociações em termos de QoS com os demais. Esse trabalho especifica a arquitetura dos componentes, o protocolo para a negociação e admissão entre eles, os implementa e realiza uma validação experimental. Sua abordagem diverge da que adotamos no nosso trabalho.

Já [Aleksandrov and Voinov, 1998] trata do problema da implementação e elaboração de sistemas de controle de variáveis associadas à Qualidade de Serviço de um serviço Web. Ele investiga várias políticas de gerenciamento e apresenta um mecanismo para monitoração e controle de várias propriedades, como o tempo de resposta médio, taxa de resposta e qualidade das imagens retornadas. Mecanismos de monitoração poderiam ser utilizados para auxiliar a implementação, em um servidor Web real, das estratégias propostas neste trabalho.

3.4 Sumário

Proporcionar melhorias na QoS tem sido uma tarefa desafiadora tratada por muitos trabalhos em diversas áreas, inclusive em aplicações Internet. Nesse contexto, as estratégias evoluíram significativamente e diversas técnicas e estratégias foram propostas, como os mecanismos de controle de admissão, escalonamento, balanceamento de carga e alocação de recursos físicos.

Entretanto, até onde pudemos avaliar, esses trabalhos falham em considerar aspectos relacionados ao comportamento dos usuários em relação ao desempenho provido pelo serviço, falhando assim em compreender os aspectos dinâmicos que afetam tanto o desempenho do servidor como o comportamento dos usuários. Entretanto, como mencionado na seção 1.2, alguns trabalhos discutem modelos de reatividade como, por exemplo, [Pereira et al., 2004b] e [Franco, 2004]. Eles permitem verificar a efetividade de se considerar a reatividade, propondo novas abordagens que são a base para esta dissertação.

Capítulo 4

Modelagem da Reatividade

Este capítulo detalha como a reatividade é identificada e caracterizada em aplicações Internet, tendo por base a metodologia apresentada em [Franco, 2004]. A Seção 4.1 apresenta uma metodologia de caracterização de aplicações Internet que permite a identificação do comportamento reativo. A Seção 4.2 apresenta o modelo para caracterizar a reatividade que é utilizado pela metodologia da seção anterior. A Seção 4.3 apresenta um estudo de caso que demonstra a aplicabilidade da metodologia de caracterização e do modelo de reatividade. Todos os níveis da metodologia são discutidos brevemente, em especial o nível de usuário, onde o modelo é aplicado. A Seção 4.4 apresenta uma validação da caracterização realizada no estudo de caso, através da replicação de padrões de comportamento reativo. Por fim, a Seção 4.5 apresenta um sumário do capítulo discutindo as principais conclusões e inferências que podem ser obtidas.

4.1 Metodologia de Caracterização do Comportamento Reativo

Com a finalidade de obter um modelo que permita representar o comportamento reativo, foi desenvolvida uma metodologia de caracterização que estende as metodologias anteriores e permite entender melhor as interações entre cliente e servidor.

Como visto no Capítulo 3, pesquisas anteriores se esforçaram por criar metodologias, algumas delas hierárquicas, para realizar a caracterização de cargas de trabalho considerando métricas tanto do lado do cliente quanto do servidor, mas ignorando a correlação entre esses dois lados e aspectos de comportamento dos usuários. Para preencher esse espaço foi elaborada uma metodologia denominada *USAR* [Pereira et al., 2004a] que propõe uma nova estratégia de caracterização de cargas de trabalho Web que estende [Menascé et al., 2000, Menascé et al., 2003b] e tem o foco na análise e modelagem do comportamento do usuário.

Baseada em um modelo hierárquico, a metodologia de caracterização *USAR* define quatro níveis de acordo com as diferentes formas de abstrair a interação do usuário com o *site* em sessões. A Figura 4.1 apresenta os quatro níveis de caracterização propostos pela metodologia. A hierarquia de níveis tem a finalidade de guiar a análise de uma carga de trabalho de acordo com diferentes perspectivas e níveis de abstração. Essa separação facilita o processo de caracterização, uma vez que ela pode ser feita de acordo com diferentes visões associadas a cada nível. Assim, esse processo se torna mais claro e produz uma caracterização mais detalhada. Abaixo descrevemos cada um dos níveis:

Figura 4.1: Níveis de caracterização da metodologia *USAR*

Requisição (R): Este nível identifica e modela aspectos relacionados aos objetos requisitados pelos usuários em suas requisições. Neste nível são caracterizados o processo de chegada de requisições e as distribuições de probabilidade associadas.

Ação (A): Este nível modela aspectos relacionados às ações submetidas ao servidor por um usuário. Uma ação acontece quando o usuário clica em um *link* ou requisita uma página durante uma sessão. Cada ação representa o comportamento de um navegador Internet típico, onde um clique exige do navegador que primeiro requisite o objeto selecionado e então os seus objetos embutidos. Neste trabalho adotaremos o termo *b*urst como sinônimo para ação. Assim, para cada ação ou burst de um usuário, uma ou mais requisições poderão ser requisitadas ao servidor. Neste nível é caracterizada a distribuição de probabilidade dos tipos de ações realizadas e as correlações entre tempo de resposta e IRT.

Sessão (S): Neste nível são analisadas as sessões dos usuários, que se constituem por conjuntos de ações realizadas sequencialmente. As sessões podem ser identificadas em um *log* através dos períodos de atividade que estão separados por períodos de inatividade maiores que um determinado limite τ. Neste nível são caracterizadas o tamanho da sessão, a duração da sessão, a composição em termos de ações, *bytes transferidos* e o tempo entre sessões.

Usuário (U): Este nível modela a reatividade em termos do comportamento do usuário em relação à qualidade de serviço provida. A caracterização no nível de usuário tem a finalidade de identificar e modelar o comportamento reativo, mais especificamente as suas reações a diferentes condições de QoS. A metodologia descreve os passos descritos abaixo para a caracterização neste nível. É importante notar que esses passos dizem quais as etapas que devem seguidas, não detalhando aspectos específicos sobre como eles devem ser realizados.

1. Realizar a preparação do *log*, gerando um *log* temporário Lu a partir do original L, identificando as sessões e ações presentes;
2. Analisar as sessões e ações de acordo com as seguintes perspectivas: IRTs e tempo de resposta associados a cada ação, razão entre os valores de IRT e tempo de resposta, e diferença entre os valores de IRT e tempo de resposta,

obtendo as distribuições de probabilidade destas variáveis;

3. Discretizar as medidas de IRT e tempo de resposta utilizando uma função para correlacioná-los. A metodologia propõe uma função que permite agrupar as ações com comportamento similar em termos dessas medidas que será descrita na próxima seção. Estes comportamentos similares obtidos através da discretização são denominados *classes de ação*;
4. Transformar as sessões em sequências de classes de ação identificadas através do método de discretização mencionado;
5. Identificar as sequências de ações mais comuns utilizando algum algoritmo de mineração de sequências, como [Zaki, 2001];
6. Avaliar as sequências possíveis com a finalidade de agrupá-las e classificá-las de acordo com algum critério de similaridade;
7. Processar o log Lu aplicando uma função $f(Lu)$, que mapeia as sequências de classes nos grupos definidos na etapa anterior;
8. Aplicar alguma técnica de agrupamento, como por exemplo o algoritmo K-Means [Garner, 1995], para determinar os agrupamentos que contêm sessões similares;
9. Analisar os agrupamentos e classificá-los de acordo com a probabilidade associada a cada ação, concluindo a análise do comportamento do usuário.

A metodologia de caracterização *USAR* permite, assim, que uma carga de trabalho seja entendida sobre diferentes perspectivas e analisada do ponto de vista do comportamento do usuário. Através dos níveis da metodologia é possível realizar a caracterização partindo do nível de requisição até o nível de usuário. Após o entendimento da carga e obtenção dos modelos que a descrevem é possível realizar a geração de uma carga sintética e a reprodução de padrões de comportamento do usuário, partindo do nível de usuário até o nível de requisição, utilizando a metodologia *USAR*.

4.2 Modelo de Reatividade

A metodologia de caracterização *USAR* estabelece um conjunto de passos para a caracterização no nível de usuário como visto na seção anterior. Ela estabelece que um modelo de discretização deve ser utilizado para permitir que a reatividade seja identificada, modelada e replicada, utilizando funções que relacionam o tempo de resposta (R) e o IRT de cada ação do usuário. O modelo de discretização da metodologia *USAR* utiliza as seguintes funções:

$$RAT(k) = \begin{cases} \text{I(k,k+1)/R(k), DIF(k)} > 0 \\ \text{R(k)/I(k,k+1), DIF(k)} < 0 \ e \\ \text{1, DIF(k)} = 0 \end{cases}$$

$$DIF(k) = I(k, k+1) - R(k),$$

$\forall k \in$ *carga de trabalho*, onde k corresponde a cada ação do usuário, $I(k, k+1)$ representa o IRT entre as ações k e $k+1$, e $R(k)$ corresponde ao tempo de resposta associado à ação k.

As funções RAT e DIF são usadas no modelo de discretização representado na Figura 4.2. O eixo x está associado com a função DIF, que representa a diferença entre o IRT e tempo de resposta, e o eixo y com a função RAT, que representa a razão entre as mesmas variáveis. O modelo define sete *classes de ação* do usuário (A a G), utilizando dois valores limites para cada eixo. Os valores k_1 e k_2 dividem o lado positivo do lado negativo da função DIF, definindo um setor próximo de zero, onde os valores do IRT e do tempo de resposta têm valores muito próximos um do outro. Os valores k_3 e k_4 dividem o setor onde a função DIF é negativa, ou seja, o tempo de resposta é maior que o IRT, em três diferentes setores, de acordo com a função RAT que permite quantificar a relação entre o IRT e o tempo de resposta. As classes A, B e C representam comportamentos onde os usuários não esperam a resposta à sua ação para requisitar a próxima ação. Os valores k_5 e k_6 dividem o setor onde o IRT é maior que o tempo de resposta, ou seja, onde o usuário leva um tempo maior para realizar sua próxima ação, após receber sua resposta anterior.

As classes E, F e G representam este comportamento. A classe D representa situações onde o usuário efetua a próxima ação quase que ao mesmo tempo em que recebe a sua resposta anterior.

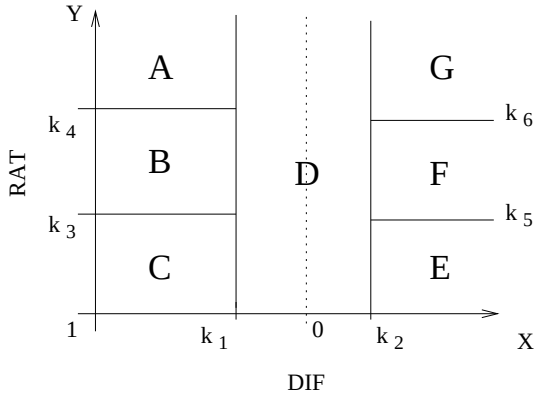


Figura 4.2: Modelo de Discretização

A Figura 4.3 apresenta uma escala denominada *escala de paciência* obtida através das classes de ação de usuário do modelo de discretização. As classes presentes no lado esquerdo da escala representam classes de ação que têm perfil *impaciente*, onde o usuário não espera o recebimento da sua resposta para efetuar sua próxima ação. O lado direito representa classes de ação onde o usuário é *paciente*, esperando sua resposta antes de realizar a ação seguinte.

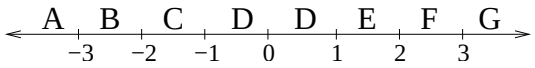


Figura 4.3: Escala de Paciência

É importante salientar que este modelo se baseia em um número determinado de classes de ação, igual a sete (classes de A a G), que através da análise do estudo de caso realizado descrito nas seções 4.3 e 4.4, se mostrou um número sufiente para representar os comportamentos existentes (número de agrupamentos encontrados na caracterização). Entretanto, o modelo apresentado é flexível e em outros cenários um número diferente de classes podem ser utilizadas, bastanto, para isto, acrescentar ou reduzir o número e valores das constantes k_x presentes no modelo.

4.3 Estudo de Caso

Os servidores *Web* podem ser configurados para armazenar informação sobre todas as ações de seus clientes. Os arquivos que contêm essas informações são chamados de arquivos de *log*, ou simplesmente *log*. Para se caracterizar a carga de um servidor, utiliza-se o *log* de acesso do mesmo, composto de informações sobre as requisições processadas pelo servidor. Cada linha desse arquivo registra uma única requisição a um documento.

A caracterização apresentada se refere a um arquivo de *log* do servidor *proxy-cache Squid* da Universidade Federal de Minas Gerais (UFMG) durante 4 semanas consecutivas. As informações contidas neste *log* se referem aos acessos dos usuários da universidade, contendo um número considerável de requisições diárias. O arquivo contém uma entrada para cada requisição requisitada por um usuário, composta das seguintes informações: *timestamp* (momento em que o *socket* do cliente foi fechado), tempo transcorrido, endereço do cliente, status do objeto, código HTTP da resposta, tamanho, método da requisição, URL, identificação do usuário, dados da hierarquia e tipo de conteúdo. Um exemplo de uma linha do *log* de acesso do *Squid* é:

```
1074011245.011   4896 150.164.10.196 TCP_MISS/200 17225 GET
http://www.ufmg.br/index.html - DIRECT/- text/html
```

4.3.1 Caracterização do Nível de Requisição

No nível de requisição é realizada a caracterização da carga abstraindo conhecimento sobre ações e sessões de usuários.

Aplicando a metodologia, verifica-se que o *log* contém cerca de 9 milhões de requisições vindas de cerca de 500 diferentes endereços IP estaticamente associados, gerando um tráfego de quase 90 Gigabytes. Assim, como esperado, 98% das requisições acessam objetos HTTP e o restante corresponde a objetos FTP ou HTTPS. A Tabela 4.1 apresenta informações gerais sobre o *log*. Podemos observar que em cada uma das semanas houve mais de 2 milhões de requisições e um número significativo de objetos únicos, IPs únicos e

sessões de usuários.

Métrica	Semana			
	1	**2**	**3**	**4**
Nº de Requisições (10^6)	2,44	2,10	2,15	2,08
MegaBytes (10^3)	34,4	19,9	16,8	20,7
Nº de IPs únicos	488	485	497	507
Nº de Sessões	6.952	6.979	7.369	7.756
Nº de Objetos únicos (10^5)	4,82	4,13	4,29	3,97

Tabela 4.1: Informações gerais sobre a carga de trabalho

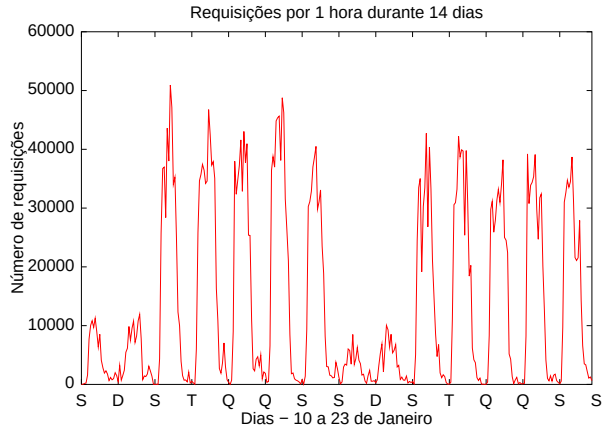
A figura 4.4 apresenta três gráficos descrevendo a chegada de requisições presentes no *log* em três diferentes escalas de tempo distintas. A inspeção visual das curvas nos permite verificar que o processo de chegada apresenta uma característica de auto-similaridade.

A figura 4.5 apresenta dois gráficos que mostram a distribuição de probabilidade da popularidade do tamanho dos objetos. Em ambos os casos se verifica que ambas as distribuições apresentam um comportamento semelhante a outras caracterizações de tráfego Web anteriores.

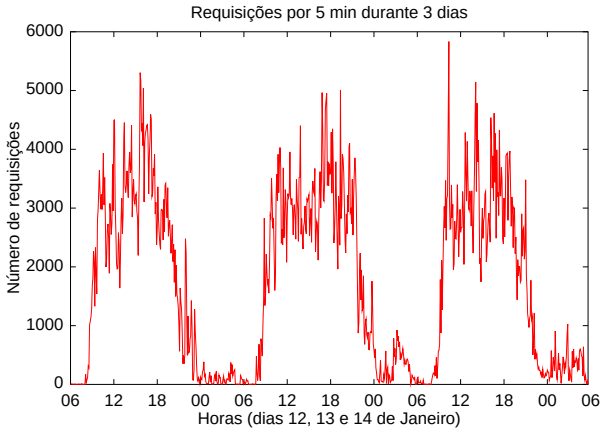
4.3.2 Caracterização do Nível de Ação

A caracterização do nível de ação se inicia com a geração de um arquivo de *log* temporário contendo as ações realizadas, à partir do conjunto total de requisições. A caracterização continua através da identificação dos tipos de ação, o que está diretamente relacionado às funções providas pela aplicação e à uma análise em multi-escala das ações, além de outras análises relevantes.

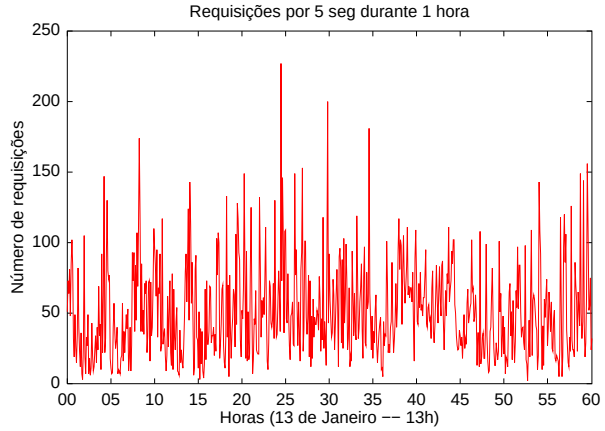
Uma vez que nossa análise é baseada no arquivo de *log* de um *proxy*, não é possível determinar o tipo de ação simplesmente analisando a URL e outros dados que compõem o *log*, uma vez que nós não temos informação do contexto do serviço sendo provido. Entretanto, para o nosso estudo de caso, os tipos de ação não são tão relevantes em comparação com outros domínios de aplicação, como, por exemplo, serviços de E-business.



(a) Chegadas por hora durante duas semanas



(b) Chegadas a cada 300s durante três dias



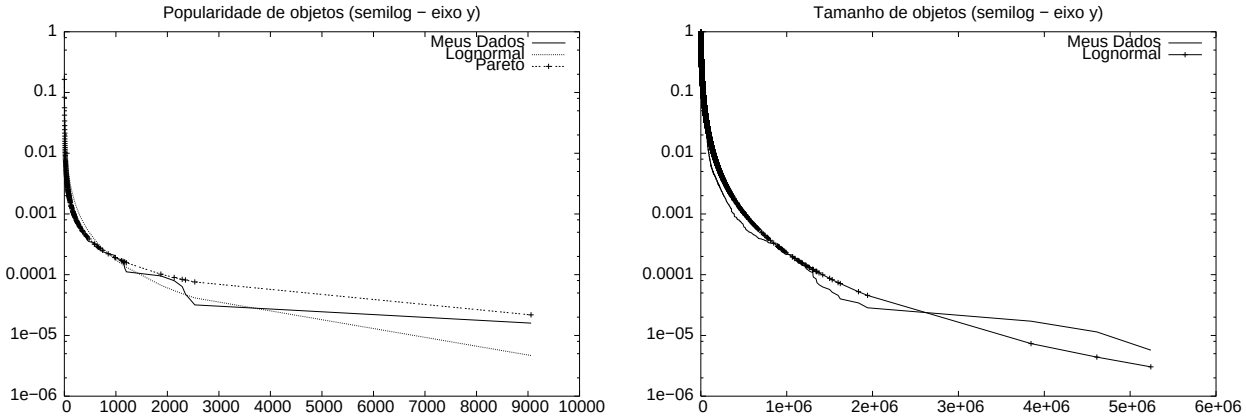
(c) Chegadas a cada 5s durante uma hora

Figura 4.4: Número de requisições presentes no *log* vistas em escalas de tempo diferentes

Considerando somente as requisições a objetos HTML, o número total de ações de usuário é de 160585. Modelamos a distribuição de probabilidade dos tempos de resposta das ações e verificamos que ela coincide com uma distribuição lognormal, como mostrado no gráfico da figura 4.6. Isto demonstra a alta variação dos tempos de resposta observados, o que abre espaço para a investigação da correlação entre o tempo de resposta do servidor e a reação dos usuários.

4.3.3 Caracterização do Nível de Sessão

Considerando somente as requisições aos 62770 objetos HTML únicos e utilizando o limite de 1800 segundos [Menascé et al., 2004] para o tamanho das sessões, identificamos 14352



(a) Popularidade dos objetos (b) Tamanho dos objetos

Figura 4.5: Popularidade e tamanho dos objetos

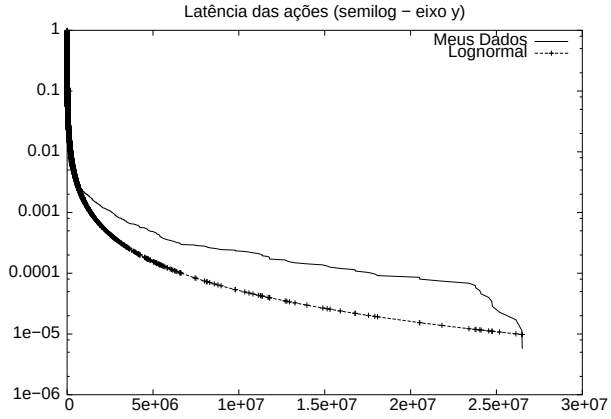


Figura 4.6: Tempo de resposta das ações

sessões associadas a 518 endereços IP únicos. Para realizarmos a caracterização no nível de sessão, iniciamos através da geração de um log Ls para cada sessão s.

A figura 4.7 apresenta o gráfico do tamanho das sessões, em função da quantidade de ações. Comparando-se a distribuição observada com distribuições tradicionais de cauda pesada, como *lognormal* e Pareto, nota-se que a curva está entre ambas aproximando-se mais de *lognormal*. Esse comportamento de cauda pesada existe, provavelmente, pela utilização de vários usuários em um mesmo endereço IP, ocasionando sessões muito grandes. Entretanto, podemos perceber que mais do que 90% das sessões são compostas de, no máximo, 26 ações, sendo que o valor médio e máximo é de 15 e 1108 ações, respectivamente.

Assim, é importante ressaltar que a significativa variação na distribuição das ações pelas sessões e o tamanho médio de sessão observado mostram a pertinência da utilização do *log* do servidor *proxy-cache* para esse trabalho, uma vez que sessões pequenas não provêem informação suficiente para se modelar o comportamento dos usuários.

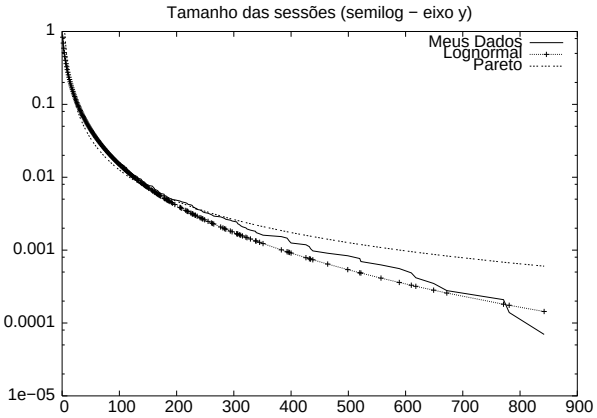


Figura 4.7: Tamanho da sessão

4.3.4 Caracterização do Nível de Usuário

O primeiro passo para realizar a caracterização neste nível consiste em obter um *log* temporário das sessões dos usuários. Então analisamos os dados de acordo com as seguintes perspectivas: IRT entre ações consecutivas, tempo de resposta de cada ação e razão e diferença entre o IRT e a tempo de resposta.

Foram geradas as funções de distribuição cumulativa e de probabilidade (CDF e PDF) para a razão e a diferença entre *IRT* e tempo de resposta. De acordo com a metodologia apresentada na seção anterior, essas funções podem ser usadas para modelar e discretizar essas variáveis, mas nesse estudo de caso esta técnica não mostrou um bom resultado, não permitindo a distinção de classes bem definidas. Então, as medidas de *IRT* e tempo de resposta foram discretizadas usando funções que as correlacionam, mais especificamente as métricas de razão (RAT) e diferença (DIF) descritas na seção anterior.

O log *Lu* foi analisado utilizando a estratégia de discretização baseada no modelo de reatividade. A transformação realizada consiste de um mapeamento direto um-para-um de

cada ação presente no *log* para uma função que correlaciona as métricas RAT e DIF. Como resultado, obtém-se um par $u(DIF(k), RAT(k))$, onde k é a ação corrente na sessão. Esse par corresponde a um ponto no gráfico do modelo de discretização. O número total de ações de usuário é 160.585.

Para discretizar os valores foi necessário definir quais valores são ideais para cada constante k_i presente no modelo de reatividade. Foi esboçado um histograma de pontos e, por inspeção visual, foram escolhidos os valores que permitiram uma boa distribuição das ações pelas diversas classes. Com isso tem-se $k_1 = -0,1$ e $k_2 = 1$, $k_3 = 2$ e $k_4 = 4$ para a discretização das classes A, B e C e, $k_5 = 4$ e $k_6 = 8$ para se delimitar as classes E, F e G. A tabela 4.2 apresenta a frequência das classes de usuário.

Classes de ação de usuário (%)						
A	**B**	**C**	**D**	**E**	**F**	**G**
1,56	1,47	2,45	3,80	8,40	8,97	73,35

Tabela 4.2: Distribuição de classes de ação

A caracterização neste nível pode envolver outros aspectos associados ao comportamento dos usuários, como por exemplo, a análise dos padrões de acesso dos usuários e a identificação de tendências de comportamento. Uma caracterização mais detalhada baseada na metodologia *USAR* e no modelo de reatividade pode ser vista no trabalho [Franco, 2004].

4.4 Validação

O objetivo da validação realizada é mostrar a aplicabilidade da metodologia de caracterização para se gerar um *log* sintético que represente as ações dos usuários em resposta à qualidade do serviço provido. Utilizando um método eficiente para a geração de variáveis aleatórias discretas com distribuições gerais [Walker, 1977], foi simulada de forma minuciosa a distribuição das sessões entre os comportamentos de usuário, utilizando o simulador

de comportamento apresentado em [Franco, 2004].

Mesmo conseguindo realizar uma simulação precisa do comportamento dos usuários, foi difícil reproduzir, no *log* gerado, exatamente as mesmas sequências de ação que identificam uma tendência no perfil do usuário. Isso se deve ao fato de que a distribuição de tamanho das sessões foi calculada para o *log* como um todo, independente da identificação dos agrupamentos ou de uma análise mais detalhada dos perfis. No entanto, uma análise mais fina mostra um resultado muito bom relativo à frequência de ocorrência de cada classe de ação de usuário no *log* sintético, quando comparado ao real. A Tabela 4.3 apresenta os resultados para esse estudo de caso.

	Classes de Ação de Usuário (%)						
Log	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Real	1,56	1,47	2,45	3,8	8,4	8,97	73,35
Sintético	1,54	1,44	2,4	3,76	8,38	8,95	73,51

Tabela 4.3: Distribuição das classes de ações de usuário

Os valores da simulação para todas as classes de ação de usuário são muito próximos aos obtidos com a caracterização do *log* real e demonstra a viabilidade da adoção de metodologias bem definidas, baseadas no comportamento do usuário, para a caracterização de cargas de trabalho *Web* e a simulação desse comportamento, considerando-se a reação do usuário a tempos de resposta variados. Maiores detalhes sobre essa geração e validação podem ser encontrados em [Franco, 2004] e [Pereira et al., 2004b].

4.5 Sumário

Este capítulo apresentou a metodologia de caracterização *USAR* e o modelo de reatividade que permitem a caracterização e geração de cargas de trabalho de aplicações Internet considerando aspectos de comportamento do usuário em termos da sua reação à qualidade de serviço provida pelo serviço, identificada através das medidas do tempo de resposta.

A metodologia *USAR* de caracterização apresentada é hierárquica, sendo composta dos níveis de requisição, ação, sessão e usuário. A hierarquia de níveis tem a finalidade de organizar a tarefa de caracterização, estabelecendo níveis de abstração das informações. O modelo de reatividade orienta a caracterização do nível de ação para modelar o comportamento do usuário baseado nas informações do tempo de resposta e do IRT presente no *log*. O modelo compreende também uma estratégia de validação que é o primeiro passo para a geração de cargas de trabalho mais realistas.

A metodologia foi validada em um estudo de caso, sendo aplicada a uma carga de trabalho real de um servidor de *proxy-cache Squid* da UFMG. Esse estudo de caso permitiu comprovar que a reatividade pode ser identificada e modelada em cargas de trabalho e que é possível replicá-la com um alto nível de precisão.

Sendo assim, o assunto tratado neste capítulo é a base para o trabalho como um todo, já que para se gerar cargas reativas e propor novas estratégias de QoS é necessário, primeiramente, identificar e modelar o comportamento reativo.

O modelo de reatividade, embora não permita uma identificação precisa, já que a discretização implica em uma certa abstração, representando um grupo de ações com características semelhantes, permite que a reatividade seja identificada e utilizada para a geração de padrões de comportamento. É importante salientar que a metodologia não define um número fixo de k_s mas permite que o seu número seja definido de acordo com os interesses e objetivos da caracterização a ser realizada.

Capítulo 5

Avaliação do Comportamento Reativo

Este capítulo analisa o comportamento reativo e seu impacto em um aplicação Internet. O objetivo é entender como o comportamento descrito por cada uma das *classes de ação* definidas pelo modelo de reatividade apresentado no capítulo 4 pode afetar um servidor, colaborando para o aumento ou diminuição da sua carga. De fato, cada uma das classes corresponde a uma diferente relação entre o tempo de resposta e o IAT, e assim, a uma relação de comportamento diferente.

Primeiramente, na seção 5.1, discutimos como o modelo de reatividade do capítulo 4 pode ser utilizado para realizarmos a geração de cargas de trabalho mais realistas. Baseado nisto, implementamos um gerador de carga que é utilizado em um experimento visando avaliar o impacto de cargas reativas sobre um servidor Web real.

A seção 5.2 discute o comportamento reativo em função das relações entre IRT e tempo de resposta e o possível impacto sobre o servidor.

A seção 5.3 apresenta o simulador *USAR-QoS* implementado para permitir avaliações de aplicações Internet considerando uma arquitetura Web típica, composta de clientes e um servidor. Essa seção apresenta também a metodologia e resultados de experimentos realizados com diferentes configurações de carga, analisando diversas métricas.

5.1 Geração de Cargas de Trabalho Reativas

Esta seção discute o problema da geração de cargas de trabalho com características reativas, onde o comportamento dos usuários se adapta dinamicamente ao QoS provido. Discutimos como uma carga de trabalho com reatividade pode ser gerada utilizando o gerador de cargas *httperf* e avaliamos qual o impacto dessas cargas sobre um servidor Web real.

A seção tem a seguinte organização. A seção 5.1.1 discute os requisitos necessários a um gerador de cargas para reproduzir o comportamento reativo de acordo com o modelo de reatividade apresentado neste trabalho. A seção 5.1.2 detalha as modificações necessárias ao gerador de cargas *httperf*. A seção 5.1.3 discute a preparação de arquivos de *trace* baseados no *benchmark* TPC-W para que possam ser executados pela nova versão do gerador *httperf*. A seção 5.1.4 apresenta a metodologia para avaliar o impacto de uma carga de trabalho reativa sobre um servidor Internet real. A seção 5.1.5 apresenta os experimentos realizados e os resultados obtidos que nos permitem verificar as diferenças entre uma carga real e uma com reatividade. A seção 5.1.6 discute os resultados e apresenta as conclusões.

5.1.1 Geração de Cargas

O modelo de reatividade da metodologia *USAR* introduz novos conceitos que permitem modelar e reproduzir como os usuários reagem a variações do tempo de resposta provido por um servidor. Para que seja possível gerar cargas com esse comportamento, algumas características importantes são necessárias a um gerador:

- Capacidade de iniciar uma nova requisição enquanto a última requisição não tiver sido finalizada, isto é, quando não tiver sido recebida a resposta anterior. Este aspecto é muito importante, principalmente, quando o tempo de resposta aumenta, visto que alguns usuários podem ter comportamento do tipo impaciente, não esperando pela resposta para requisitar a próxima requisição.

- O IAT dos usuários não deve ser estático, podendo variar dinamicamente de acordo

com o tempo de resposta percebido e a classe de ação do modelo de reatividade, que associa a cada burst um valor para correlacionar o tempo de resposta com o IAT.

O Capítulo 3 descreve diversos geradores de cargas de trabalho utilizados em diferentes aplicações. Como observado, nenhum deles permite que o comportamento reativo seja replicado, ou seja, a carga gerada por eles é basicamente a mesma, não importando qual desempenho o servidor esteja apresentando. Esses geradores assumem que, para submeter uma nova requisição, o usuário deve esperar pela resposta à requisição anterior. Esta abordagem não permite representar a situação de impaciência ao QoS provido. O comportamento impaciente corresponde às classes de ação do lado impaciente da escala de paciência vista no Capítulo 4. Essa nova situação demanda que o gerador de carga execute sessões *não blocantes*, ou seja, cujos *bursts* podem ser iniciados e requisitados antes que o anterior tenha sido completado.

Duas opções seriam possíveis para atingir esses objetivos: implementar um novo gerador de carga ou adaptar um dos existentes para executar uma carga reativa. Optamos por esta segunda opção pois ela é mais simples e aproveita modelos já consolidados de geração de carga.

O gerador escolhido foi o *httperf* [Mosberger and Jin, 1998] pois foi verificado que ele poderia ser adaptado para os propósitos do trabalho. Ele consiste de um *framework* altamente flexível baseado em eventos que, de um modo efetivo, permite gerar cargas HTTP e conduzir avaliações de desempenho de um servidor. Verificamos que, quando uma requisição demora muito tempo para ser completada, esse gerador permanece esperando indefinidamente. O único parâmetro que permitiria um funcionamento próximo ao desejado seria o valor do tempo de *time-out*, porém esta é uma solução muito radical, diferente da desejada, já que quando uma requisição esgota o tempo de *time-out*, o *httperf* finaliza toda a sessão sendo executada. Por esta razão foi necessária a implementação de uma nova versão do gerador compatível com a metodologia *USAR* e seu modelo de reatividade.

A figura 5.1 ilustra o mecanismo tradicional de geração de cargas presente no *httperf* e em outros geradores de carga, e a nova abordagem que implementa o comportamento

impaciente. A figura apresenta a execução de uma sequência de *bursts* em uma sessão. O lado esquerdo representa o cliente (usuário) e o lado direito o servidor. Vários eventos presentes no gerador associados à execução de sessões, bursts e requisições estão descritos. As requisições são representadas por linhas indo do lado cliente para o lado servidor, e vice versa. De cima para baixo, na vertical, está representado o tempo. A figura ilustra o tempo de duração da sessão, o *think-time* (tempo de pensamento do usuário), e o IAT. A requisição principal de cada burst é representada por linhas em negrito e as requisições embutidas são as linhas simples.

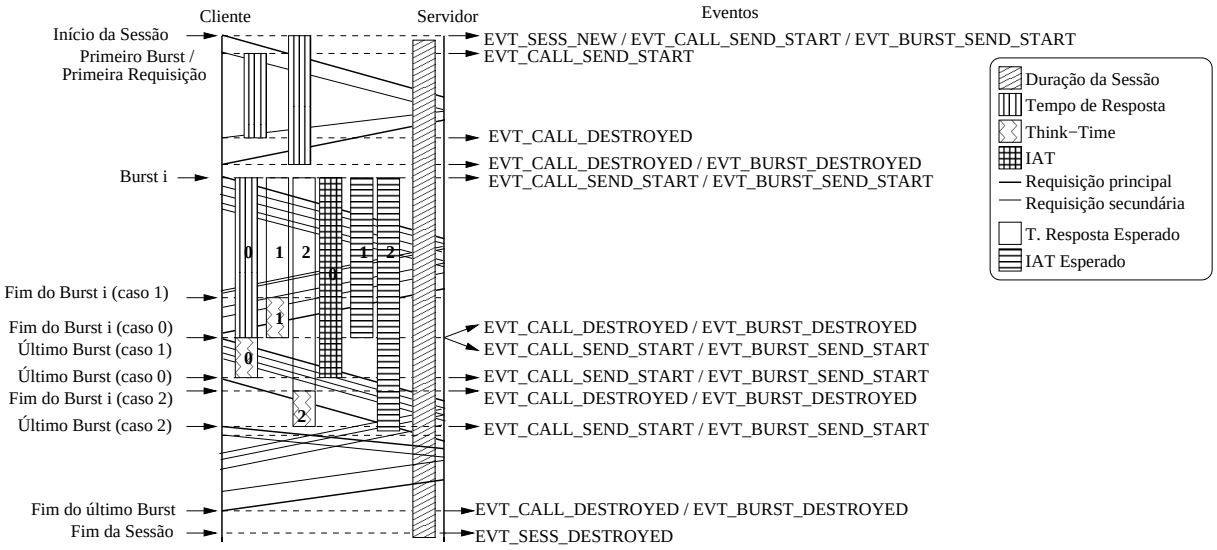


Figura 5.1: Interação entre cliente e servidor

A figura apresenta o mecanismo tradicional de execução realizado pelo *httperf* representado pelos elementos numerados com *0* (zero) e o mecanismo reativo que foi implementado através dos elementos numerados com *1* (um) e *2* (dois). A figura também introduz os conceitos de tempo-de-resposta-esperado (ER) e IAT-esperado (EIAT), que são medidas definidas dinamicamente de acordo com o modelo de reatividade e apresentadas na próxima seção.

5.1.2 Implementação da Reatividade no Gerador *httperf*

O gerador de cargas de trabalho *httperf* possui um módulo chamado *wsesslog* que realiza a submissão de requisições baseadas em um arquivo que contém as sessões que deverão ser executadas. Esse módulo implementa diversos aspectos e controles associados à execução de sessões, como o número e a sequência dos bursts, o método HTTP (GET ou POST), o *think-time* e o tamanho do burst. Com a finalidade de adaptar o gerador ao modelo reativo, foram adicionadas informações sobre a classe de ação de usuário à estrutura de sessões.

Para reproduzir o comportamento reativo ao executar a sessão de um usuário, o gerador de cargas deve obter, a cada instante, o tempo de resposta apresentado pelo servidor, para que, à partir da classe de ação do usuário oriundo do modelo de reatividade, seja possível determinar qual comportamento será apresentando pelo cliente em termos do IAT. Assim, durante a execução de uma sessão é necessário obter o tempo de resposta observado pelo cliente (no caso do gerador, o tempo observado pelo próprio *httperf*).

Uma vez que o gerador *httperf* é baseado em uma estrutura de eventos, o tempo de resposta pode ser obtido facilmente, já que toda vez que uma sessão é criada ou destruída, ou uma requisição é iniciada ou finalizada, um novo evento é criado. Esses eventos podem ser capturados através de rotinas de *call-back* (*call-back handles*), definidas utilizando as funções de API do *httperf*. Assim, o tempo de resposta associado a cada requisição ou burst pode ser obtido utilizando dois eventos, um engatilhado no início de uma requisição e outro quando ela se completa.

A partir da obtenção do tempo de resposta, alteramos o modo como o *httperf* escalona os bursts de cada sessão, com a finalidade de reproduzir o comportamento impaciente. A implementação original espera que o último burst submetido termine para iniciar um evento contador de tempo para engatilhar o seguinte. Para isso, o módulo *wsesslog* foi adaptado de modo a iniciar um contador de tempo assim que o primeiro burst tenha sido submetido. Mas por quanto tempo este contador deve esperar antes de iniciar um novo burst? Este valor, que denominamos EIAT (de *expected-IAT*, ou seja, o IAT esperado),

pode ser calculado utilizando a classe de ação presente na estrutura *wsesslog* e o tempo de resposta da requisição anterior, denominada ER (de *expected-response-time*, ou seja, o tempo de resposta esperado). Assim, os seguintes passos são executados para a reprodução do comportamento reativo:

1. Para cada sessão que esteja sendo executada, registra-se o último tempo de resposta observado, ou seja, quanto tempo levou para que o último burst tenha sido respondido pelo servidor. Registra-se também o tamanho do burst em bytes. Essas duas métricas são importantes para se estimar o tempo de resposta esperado (ER) para o burst seguinte, ou seja, o tempo que o usuário deverá esperar para receber a próxima resposta.

2. Cada burst a ser executado tem associado a ele uma *classe de ação*. Utilizando o valor do tempo de resposta esperado ER, calcula-se o valor do IAT-esperado (EIAT) utilizando a relação estabelecida pela classe de ação correspondente. Relembrando, a classe de ação refere-se às classes do modelo de reatividade e representa uma relação entre os valores da diferença e razão entre o tempo de resposta e o IAT e um conjunto de constantes k_n.

3. Utiliza-se o valor do EIAT para determinar a quantidade de tempo máxima que o cliente simulado deve esperar pela resposta. Caso contrário, o próximo burst é requisitado ao servidor.

A nova versão do *httperf* foi instrumentada para registrar alguns eventos importantes para cada burst e requisição, que são os seguintes: identificador da sessão (SESSID), instante de envio (SNDREQ, SNDBUR) e recebimento (RCVREQ, RCVBUR), tempo de resposta (na perspectiva do cliente), bytes recebidos, informações indicadoras da ocorrência de *time-out* e se a requisição não foi completada (error). Para cada sessão também foi implementado o registro do número de sessões ativas (SESCNT).

Ao final, uma nova versão do gerador foi produzida que será denominada de *httperf-reativo* ou *httperf-não-blocante*.

5.1.3 Preparação do arquivo de *trace*

Como observado na seção anterior, o *httperf* utiliza um arquivo contendo as informações de cada sessão a ser executada para realizar a geração de carga. Esse arquivo chamado *arquivo de trace* contém os endereços das páginas a serem acessadas, os endereços dos respectivos arquivos embutidos, informações sobre o tamanho dos arquivos e o *think-time* a ser utilizado durante a simulação. Para a execução da nova versão do gerador *httperf* o formato desse aquivo foi alterado, incluindo informação sobre a classe de ação dos respectivos bursts.

Com a finalidade de preparar um arquivo de *trace* para ser utilizado nos experimentos, foi necessário obter um conjunto de dados relevante que permitissem a simulação de um experimento real. Para isso, será utilizado o *benchmark TPC-W* que fornece um modelo de referência baseado em uma aplicação Web real.

5.1.3.1 O *benchmark TPC-W*

O *benchmark TPC-W* [García and García, 2003, Menascé, 2001] é um modelo de referência de uma aplicação de comércio eletrônico, elaborado pelo TPC (*Tansaction Processing Council*). O modelo utiliza um ambiente que simula uma loja de comércio eletrônico na Internet e fornece guias para a geração de cargas de trabalho sintéticas que serão executadas no ambiente simulado de uma loja online.

O *TPC-W* foi criado, principalmente, para permitir a avaliação do número de interações processadas por segundo, onde as requisições simulam a atividade de uma loja de varejo. Os relatórios produzidos podem ser utilizados para avaliar soluções completas de hardware e software, de acordo com o número de transações por segundo e preço por transação. Com relação à simulação da navegação de um usuário, o *benchmark TPC-W* determina que, após um certo período de tempo, uma nova requisição deve ser escolhida através de uma seleção randômica dentro de uma das possíveis navegações legais a partir da página corrente [García and García, 2003].

É importante observar que cada requisição possui um *think-time*, isto é, o tempo que o usuário levará para submeter uma nova requisição após receber a resposta à requisição

corrente, que varia de 7 a 70 segundos, e que o gerador de cargas deve utilizar para simular o comportamento da carga.

5.1.3.2 Adicionando as *classes de ação* do modelo de reatividade

Baseado no modelo de referência *TPC-W* foi produzido um arquivo de *trace* no formato de execução do gerador *httperf*. Entretanto, como esse modelo não engloba informações sobre o modelo de reatividade, foi necessário acrescentar algumas informações. Para isto, adotou-se um experimento de referência e um conjunto de etapas para a obtenção das informações das classes de usuário que descrevemos abaixo.

Foi preparado um ambiente experimental que permitiria a simulação de um aplicação Internet tradicional. Este ambiente é composto de um servidor HTTP (Apache), um servidor de aplicação (Apache Tomcat), um servidor de banco de dados (MySQL), e um cliente (*httperf*), executados em diferentes máquinas. Cada máquina possui o sistema operacional Linux com a versão 2.4.25 do seu *kernel*, executando sobre um processador Intel Pentium 4 1.80GHz de CPU e 1GB de memória RAM. Para otimizar o desempenho, todos os serviços desnecessários foram desabilitados e o sistema operacional foi configurado para suportar um número de descritores de arquivo (*file descriptors*) suficiente para os experimentos (65.000). Esse ambiente foi preparado para executar uma aplicação Java que implementa o *benchmark TPC-W*. Então foram seguidos os seguintes passos:

1. Foi criada uma carga de trabalho seguindo as recomendações do modelo *TPC-W* e o seu *CBMG* (*Customer Behavior Model Graph* [Menascé et al., 2004]). A carga de trabalho gerada denominada *wl-tpcw* é composta de 5.000 sessões de usuário com um tamanho médio de sessão de 124 bursts.

2. A carga *wl-tpcw* foi convertida em outra, denominada *wl-httperf*, compatível com o formato utilizado pelo módulo *wsesslog* do gerador *httperf*.

3. A carga *wl-httperf* foi utilizada em um experimento utilizando o ambiente descrito acima e a versão original do cliente *httperf* com a finalidade de gerar um registro dos

tempos de resposta fornecidos pelo servidor.

4. A partir dos tempos de resposta registrados e da carga *wl-httperf*, foi aplicado o modelo de reatividade, obtendo assim uma distribuição de classes de usuário para cada burst executado.

5. À carga de trabalho *wl-httperf* foram adicionadas as informações obtidas na etapa anterior, obtendo uma carga que denominamos *wl-httperf-reactive* para ser usada pela nova versão do gerador *httperf*.

5.1.4 Metodologia

Utilizando o mesmo ambiente experimental descrito na seção anterior, foram realizados experimentos com a finalidade de verificar o impacto de uma carga de trabalho com características reativas utilizando a nova versão do gerador de carga *httperf* descrita neste capítulo. Para isto, são utilizadas diferentes configurações em termos do número de sessões e da sua taxa de criação. Apresentamos na próxima seção os resultados de três cenários, onde o gerador *httperf* foi configurado para executar 100, 1.000, e 5.000 sessões de usuário, com uma taxa de criação de 100 sessões por segundo. Essas configurações foram escolhidas para avaliar o impacto em condições de intensidade de carga leve, média e pesada. Para cada configuração de carga foram avaliados tanto o contexto reativo como o não-reativo.

Os experimentos avaliam um conjunto de métricas para cada cenário:

- Throughput ou taxa de serviço: avaliamos tanto a taxa de chegada quanto de saída do ponto de vista do cliente. A primeira representa a taxa de respostas enviadas pelo servidor e recebidas pelo cliente em cada unidade de tempo. Já o último corresponde à taxa de requisições submetidas por unidade de tempo ao servidor.

- Throughput cumulativo: avaliamos o throughput cumulativo tanto de chegada como de saída.

- Tempo de resposta: refere-se ao tempo de resposta percebido pelo cliente, consistindo do intervalo de tempo entre a submissão da requisição e o tempo que o cliente termina de receber a resposta.

- Bursts ativos: representa o número de bursts requisitados ao servidor mas que não foram respondidos até um determinado instante de tempo.

- Sessões ativas: representa o número de sessões iniciadas mas não terminadas, ou seja, que tem bursts não submetidos ainda ou que foram submetidos mas que não tiveram uma resposta recebida.

A análise exibida abaixo está focada no período de carga mais intensa, que corresponde aos primeiros dez minutos de experimento.

5.1.5 Avaliação do Impacto de Cargas de Trabalho Reativas

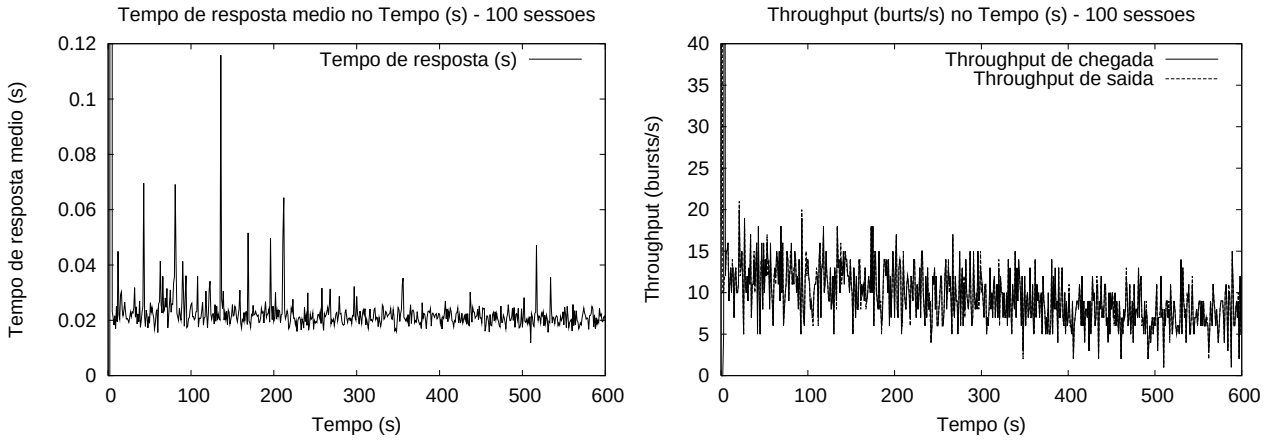
Os principais resultados dos experimentos realizados estão sumarizados na tabela 5.1. Nela estão listadas algumas medidas úteis para analisarmos o impacto de cargas de trabalho reativas no desempenho de um servidor, que são as seguintes: número total de bursts, taxa de bursts por segundo (Bursts/s), número total de requisições, taxa de requisições por segundo (Requisições/s), tempo de resposta médio, percentagem de sessões finalizadas. Apresentamos tanto o cenário não-reativo (NR) como o reativo (R).

	100 sessões		**1000 sessões**		**5000 sessões**	
Medida	**NR**	**R**	**NR**	**R**	**NR**	**R**
Nº de Bursts (10^3)	6	10	57	78	80	80
Bursts/s	9,2	16,1	92,2	114,4	123	133
Nº de Requisições (10^3)	50	90	500	650	580	690
Requisições/s	100	190	800	1200	1180	1280
Tempo de Resposta (s)	0,027	0,039	0,1	0,35	40,7	13,7
Nº de Sessões (%)	45	85	45	90	20	25

Tabela 5.1: Sumário dos experimentos

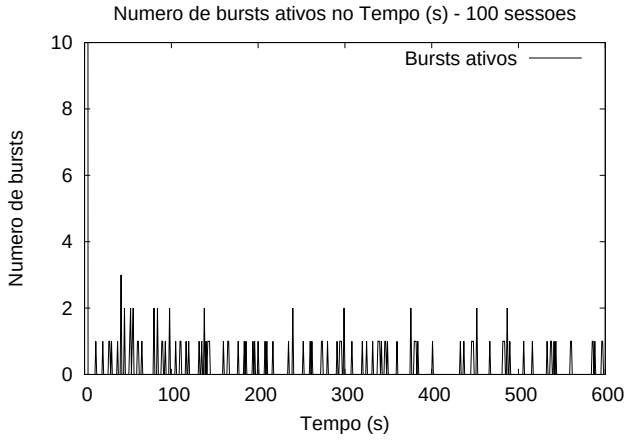
5.1.5.1 Execução de 100 sessões

As figuras 5.2 e 5.3 apresentam, respectivamente, os cenários não-reativo e reativo, para o experimento contendo 100 sessões. Em (a) vemos o tempo de resposta médio, em (b) o throughput, em (c) o número de bursts ativos e em (d) o número de sessões ativas.

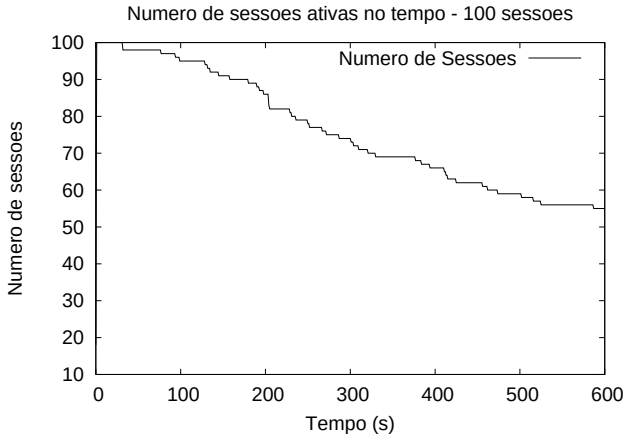


(a) Tempo de resposta médio

(b) Throughput de bursts



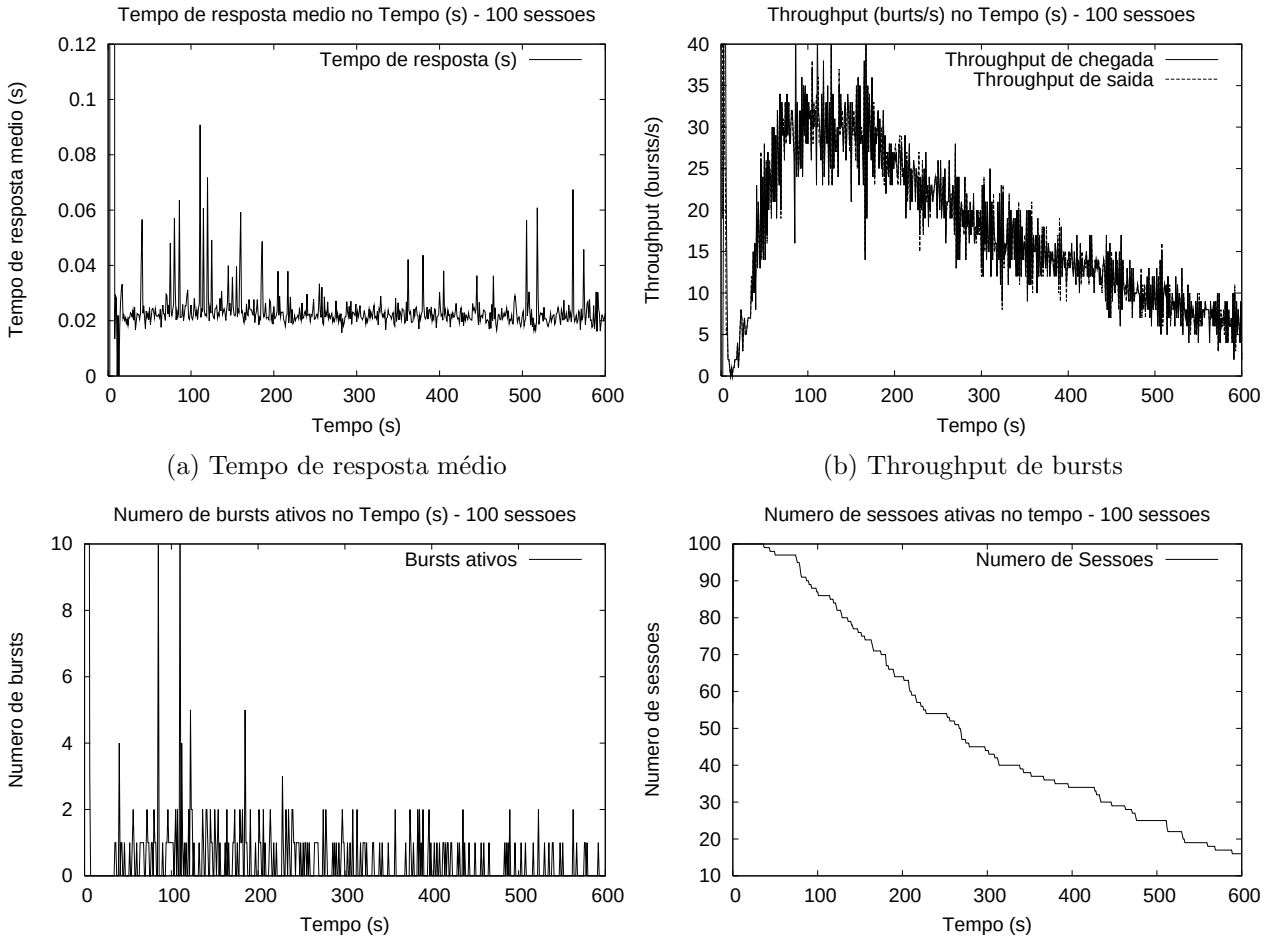
(c) Bursts ativos



(d) Sessões ativas

Figura 5.2: Experimento não-reativo com 100 sessões

O experimento não-reativo apresenta um total de 6.000 bursts e 50.000 requisições, com um throughput médio de 9,2 bursts por segundo e 100 requisições por segundo. O tempo de resposta médio é de cerca de 0,027 segundo, um valor muito pequeno, próximo de zero, que confirma o estado de não-sobrecarga.



(a) Tempo de resposta médio

(b) Throughput de bursts

(c) Bursts ativos

(d) Sessões ativas

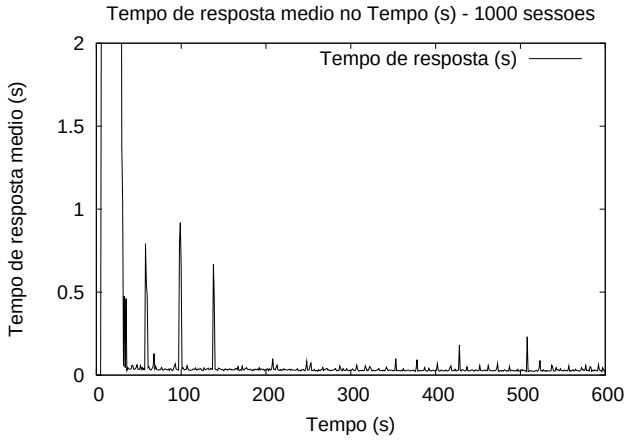
Figura 5.3: Experimento reativo com 100 sessões

O experimento reativo atinge mais de 10.000 bursts e 90.000 requisições, com um throughput médio de 16,1 bursts por segundo e 190 requisições por segundo. O tempo de resposta médio é de cerca de 0,039 segundos. Pode-se verificar a partir desta análise que o throughput aumenta, sem mudanças significativas no tempo de resposta, indicando o estado de não-sobrecarga do servidor.

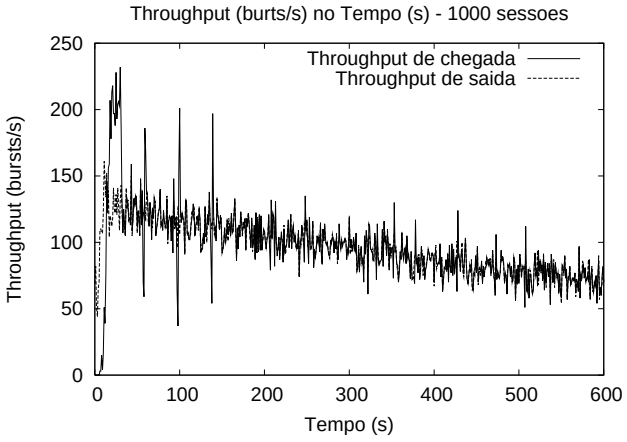
O número de bursts ativos a cada instante no experimento reativo é sempre muito baixo, uma vez que não há problemas de desempenho. 85% das sessões haviam terminado ao final do tempo de amostragem, mostrando que a reatividade permite que usuários diminuam o tempo estimado de sessão, uma vez que o tempo de resposta para os seus bursts é pequeno.

5.1.5.2 Execução de 1.000 sessões

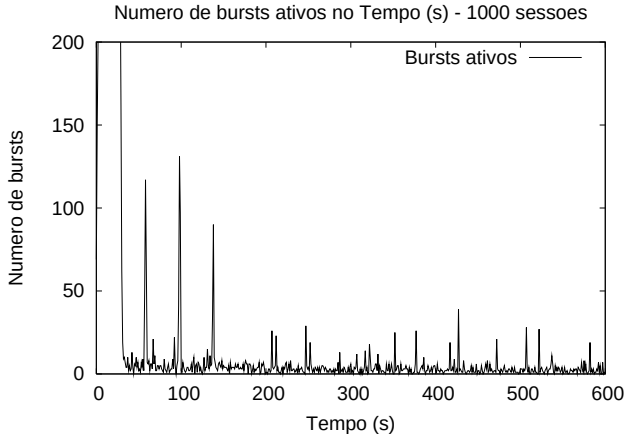
As figuras 5.4 e 5.5 apresentam os experimentos reativo e não-reativo onde são executadas 1.000 sessões a uma taxa de 100 sessões iniciadas por segundo.



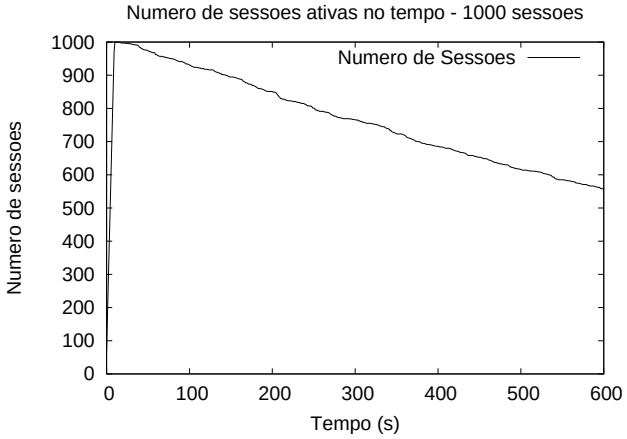
(a) Tempo de resposta médio



(b) Throughput de bursts



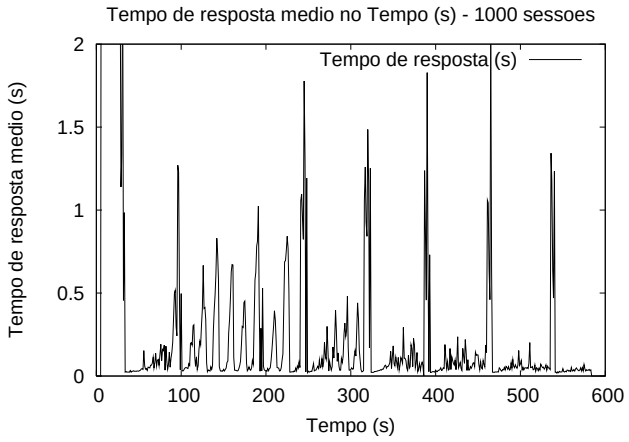
(c) Bursts ativos



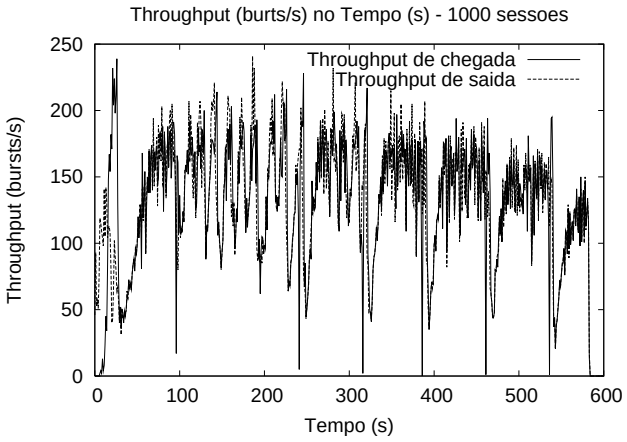
(d) Sessões ativas

Figura 5.4: Experimento não-reativo com 1.000 sessões

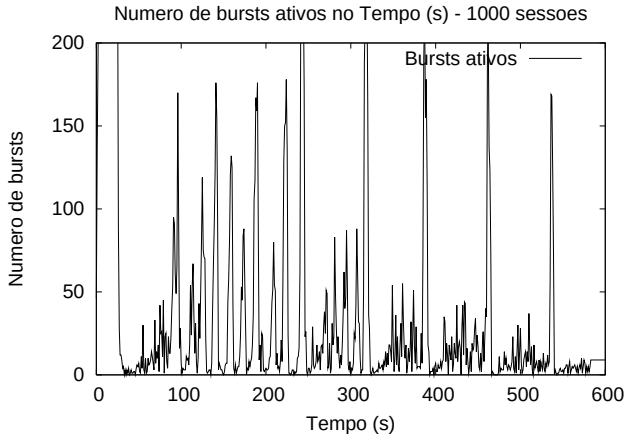
O experimento não-reativo apresenta um throughput médio de 92,2 bursts por segundo e um tempo de resposta médio de cerca de 0,24 segundos. Em termos de requisições, apresenta um throughput médio de 800 requisições por segundo. O tempo de resposta ainda é muito pequeno, havendo picos de até 1 segundo. Esses valores indicam um situação de não-saturação do servidor.



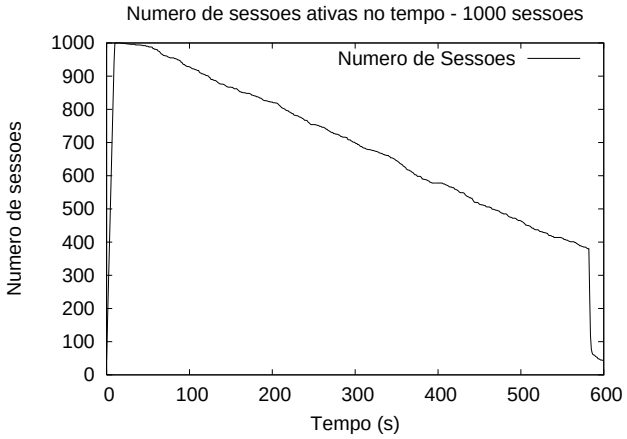
(a) Tempo de resposta médio



(b) Throughput de bursts



(c) Bursts ativos



(d) Sessões ativas

Figura 5.5: Experimento reativo com 1.000 sessões

O número de bursts ativos durante o experimento não-reativo apresenta um comportamento estável, uma vez que não há problemas com desempenho. Há poucos picos, que podem ser explicados através de uma comparação com a curva de tempo de resposta. Esses picos ocorrem exatamente no momento em que o tempo de resposta apresenta atrasos. Durante o experimento, 45% das sessões terminaram, o mesmo percentual apresentado pelo experimento não-reativo com 100 sessões.

O experimento reativo apresenta um total de 78.000 bursts e 65.0000 requisições. O throughput chegou a 114,4 bursts por segundo e 1200 requisições por segundo. O tempo de resposta médio é de 0,35 segundos, mas existem picos de até 2 segundos no tempo de

resposta em situações isoladas, que não prejudicam o desempenho do servidor. Assim, alguns usuários podem observar pequenos atrasos na resposta do servidor.

O número de bursts ativos durante o experimento reativo apresenta variações, o que pode explicado quando comparado ao comportamento do tempo de resposta do experimento. Esses picos ocorrem exatamente quando o tempo de resposta apresenta atrasos, como esperado. Neste experimento, 90% das sessões terminaram.

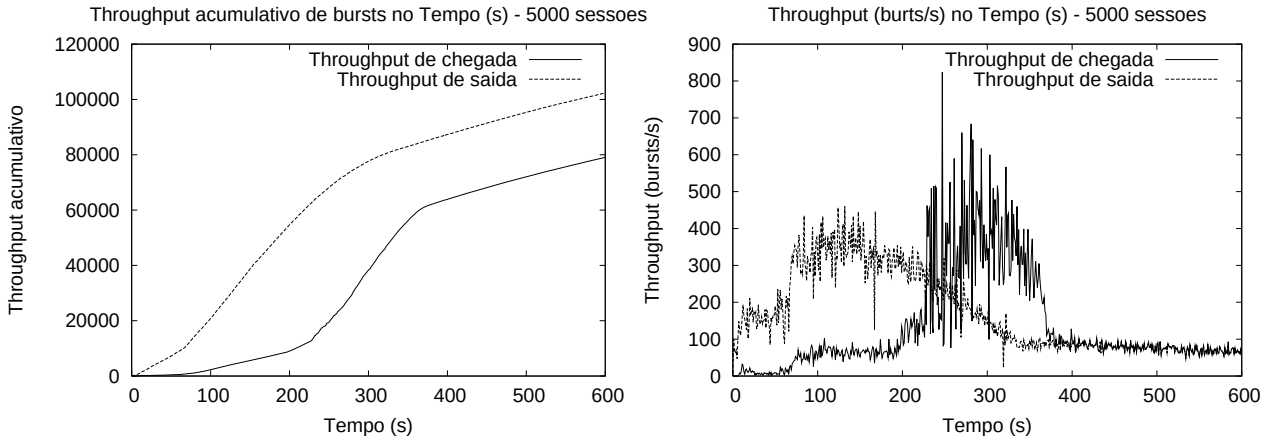
É interessante observar que o throughput do experimento reativo com 1.000 sessões diminui exatamente quando o tempo de resposta aumenta, mas neste caso, a mudança na reação dos usuários ocasiona que a taxa de throughput aumente novamente após certo tempo de experimento. Observando os gráficos, podemos verificar que o servidor mantém uma taxa muito boa do tempo de resposta, sem apresentar sobrecarga.

5.1.5.3 Execução de 5.000 sessões

As figuras 5.6, 5.7, 5.8 e 5.9 apresentam os experimentos que utilizam a carga de trabalho contendo 5.000 sessões. Elas apresentam o throughput, tempo de resposta médio, sessões ativas e bursts ativos.

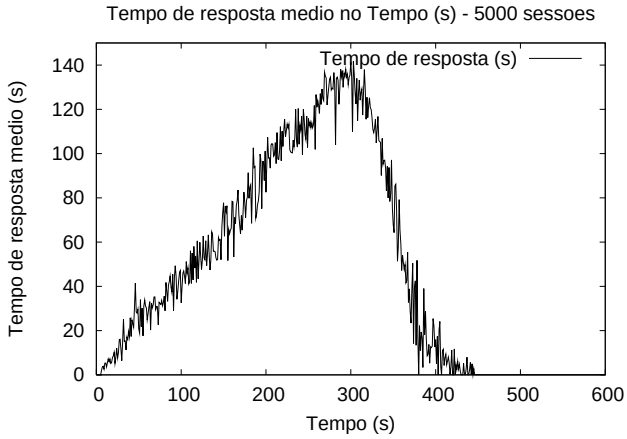
O experimento não-reativo executa um total de 80000 bursts e 580000 requisições, apresentando um throughput variando de 100 a 400 bursts por segundo e de 200 a 1600 requisições por segundo. O tempo de resposta aumenta de poucos segundos para mais do que 120 segundos, com um valor médio de 40,7 segundos. Deste modo, pode-se dizer que o servidor atinge uma situação de sobrecarga, pois o tempo de resposta, já após os 30 segundos iniciais, apresenta um tempo de resposta maior que 10 segundos.

É importante analisar o que ocorre próximo ao instante 360 segundos. Os seguintes aspectos são registrados: o tempo de resposta começa a diminuir, o throughput diminui, o número de bursts ativos se estabiliza, e o número de sessões ativas diminui rapidamente. Uma investigação detalhada permite detectar a causa dessa anomalia. Algumas conexões TCP/IP expiraram (*timed-out*), retornando erros número *110*. Esse problema causou o seguinte comportamento anormal nas medidas:

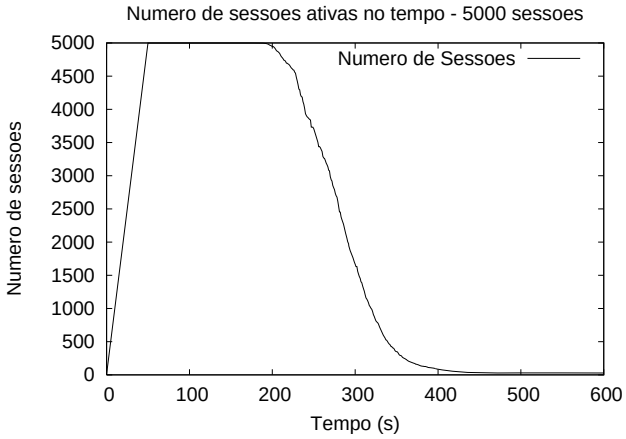


(a) Throughput cumulativo de bursts

(b) Throughput de bursts



(c) Tempo de resposta médio



(d) Sessões ativas

Figura 5.6: Experimento não-reativo com 5.000 sessões

- Throughput cumulativo: a diferença entre as taxas de envio e recebimento é de mais de 20000 bursts. Isto ocorre pois a execução desses bursts não foi finalizada, já que as conexões foram fechadas antes de ocorrer a expiração prevista pelo protocolo TCP/IP.

- Throughput: a taxa de serviço diminui e se estabiliza perto de 100 bursts por segundo, conseqüência do pequeno número de sessões que permanecem ativas após o problema.

- Tempo de resposta: o tempo de resposta médio diminui rapidamente quando o problema com as conexões do TCP/IP ocorre. Somente após 450 segundos ele atinge valores aceitáveis para os usuários.

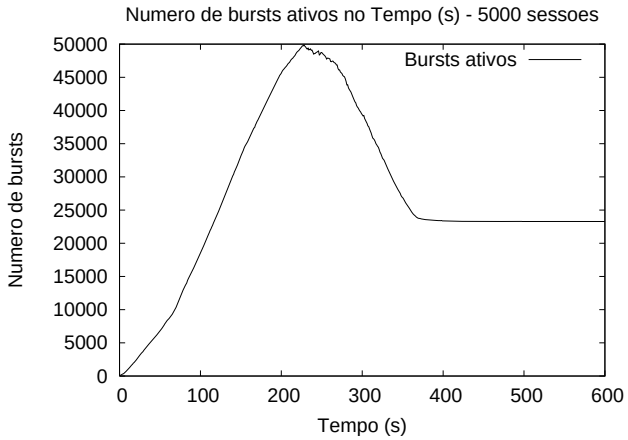


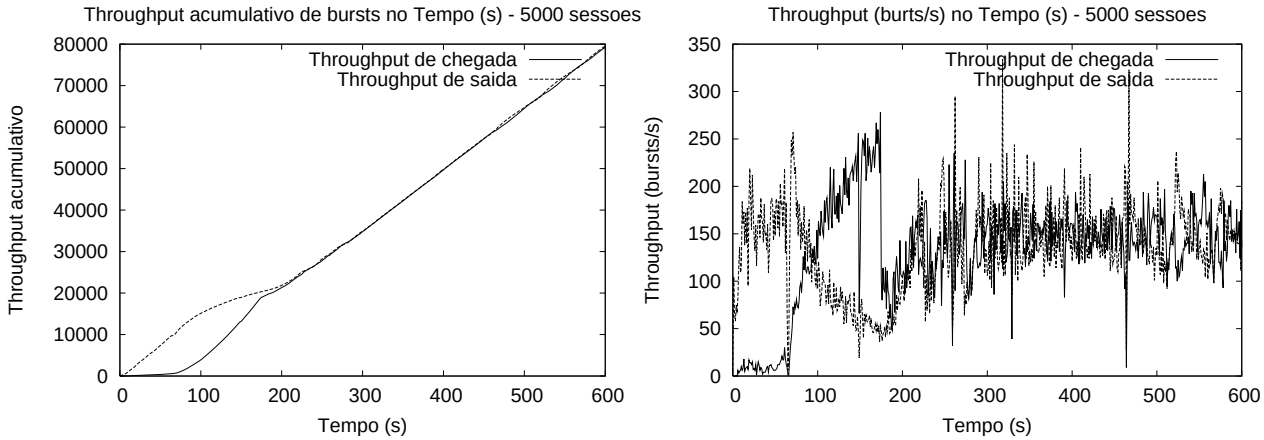
Figura 5.7: Experimento não-reativo com 5.000 sessões - Bursts ativos

- Bursts ativos: o número de bursts ativos continua alto, já que muitos bursts não são finalizados em consequência da expiração do TCP/IP. Após 400 segundos, o valor se torna estável, com pequenas variações.

- Sessões ativas: o número de sessões diminui rapidamente, o que demonstra que muitas sessões falham em conseqüência do erro ocorrido. Quando o gerador de cargas tenta abrir ou enviar requisições e o protocolo TPC retorna erros, a sessão corrente falha e fecha após não haverem mais conexões disponíveis. Somente um conjunto de 100 sessões estão ativas após o tempo 400 segundos, representando os usuários que geram carga ao servidor deste ponto até o fim do experimento.

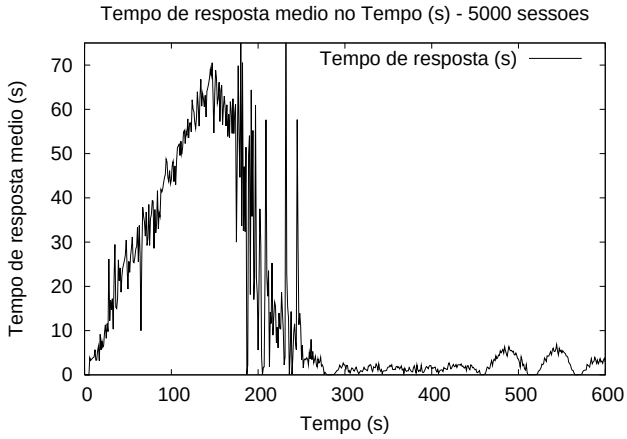
Neste experimento não-reativo nós podemos identificar uma situação de alta sobrecarga no servidor, o que causa um desempenho muito ruim. O tempo de resposta observado se torna inaceitável. Ainda, a indisponibilidade do servidor representa um problema muito grande, um dos mais sérios que uma sobrecarga pode causar, já que 80% dos usuários não recebem a resposta do servidor.

O experimento reativo executa 80000 bursts com um throughput médio de 133 burst por segundo, variando de 25 a 250 bursts por segundo após os segundos iniciais. Em termos de requisições, atinge 690000 requisições, com um throughput variando de 50 a 1800 requisições por segundo. Essa quantidade de requisições é somente 6% maior do que

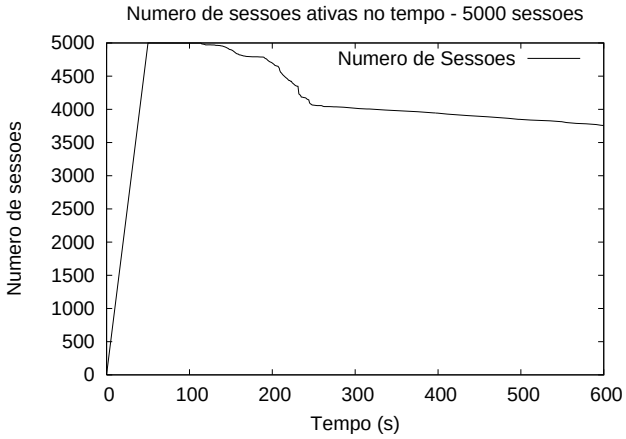


(a) Throughput cumulativo de bursts

(b) Throughput de bursts



(c) Tempo de resposta médio



(d) Sessões ativas

Figura 5.8: Experimento reativo com 5.000 sessões

a quantidade observada no experimento com 1000 sessões. Observando os gráficos pode-se perceber que o servidor atinge sobrecarga, já que após 20 segundos do início do experimento o tempo de resposta já atingiu mais do que 10 segundos.

Observando a curva do throughput observamos que a taxa de recebimento aumenta e a de envio diminui entre o período de 100 a 200 segundos, sendo possível observar uma correlação entre o tempo de resposta e o número de bursts ativos, como esperado. A mudança do modo como os usuários reagem quando o servidor aumenta o seu tempo de resposta provoca um atraso na duração de sessões, e um valor próximo a 75% permanecem ativas após o tempo de experimento.

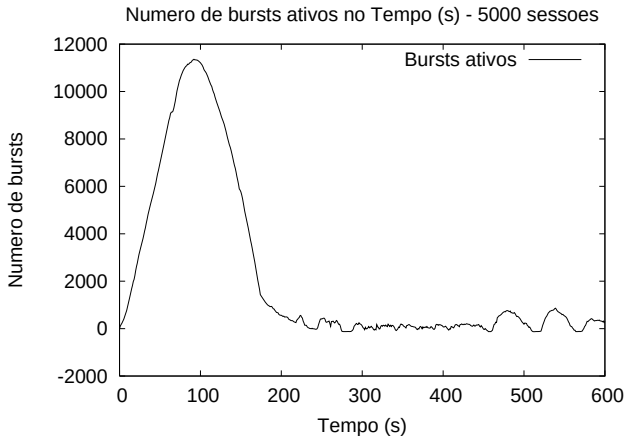


Figura 5.9: Experimento reativo com 5.000 sessões - Bursts ativos

5.1.6 Sumário

Os resultados desta seção nos permitem verificar que cargas de trabalho reativas apresentam um comportamento diferente em comparação com cargas não-reativas, permitindo comprovar que diferentes tempos de resposta influenciam o comportamento dos clientes, mudando a taxa de requisições submetidas e, conseqüentemente, alterando a carga do servidor. Assim, demonstra-se que aspectos relacionados no lado do cliente afetam o servidor e vice versa, demonstrando a importância da reatividade.

Os experimentos com 100 e 1.000 sessões mostram que o servidor atinge um bom nível de desempenho, garantindo que o usuário perceba uma resposta instantânea para os seus bursts. Uma boa taxa de tempo de resposta permitiu que os usuários do experimento reativo requisitassem mais rapidamente os seus próximos bursts. O aumento na taxa de throughput sem aumento do tempo de resposta mostra que o servidor não estava sobrecarregado. A diminuição no IAT entre bursts no experimento reativo permite que um maior número de sessões terminem em comparação com o experimento não-reativo.

Já os experimentos não-reativo e reativo com 5.000 sessões apresentam diferentes cenários, sendo que o primeiro causou uma sobrecarga intensa no servidor, o mantendo indisponível para a maior parte dos usuários. O experimento reativo apresenta também sobrecarga, mas a reação dos usuários a valores de tempo de resposta inaceitáveis muda o seu com-

portamento global, permitindo que o servidor economize recursos e retorne a um tempo de resposta aceitável após isto.

Assim, vemos que ocorrem mudanças de comportamento do servidor Web quando uma carga com reatividade é aplicada em comparação com cargas não-reativas. Os gráficos mostram alterações nos valores e na função da curva de cada variável no tempo. Tanto o throughput como o tempo de resposta apresentam variações significativas, principalmente no experimento com 5.000 sessões.

A principal conseqüência do fato de que a reatividade impacta o desempenho está no fato de que os modelo atuais de desempenho da Web estão incompletos e poderiam ser melhorados, uma vez que não consideram esses aspectos. Isto motiva a investigação de novas estratégias de QoS que possam, até mesmo, mitigar os efeitos negativos da reatividade e reforçar os aspectos positivos.

Além disto, os resultados demonstram que é importante considerar a correlação entre o lado do cliente e do servidor, uma vez que isto pode diminuir a distância entre cenários realistas e modelos. O novo gerador de cargas e o modelo apresentado apresentam uma metodologia que permite entender e avaliar a interação entre os lados, demonstrando a importância de considerar a reatividade.

5.2 Comportamento Reativo

Nesta seção procuramos entender o comportamento reativo, avaliando a relação entre as métricas de IRT e tempo de resposta envolvidas no modelo de reatividade e o possível impacto sobre o servidor.

O modelo de reatividade estabelece sete *classes de ação* apresentadas no capítulo 4. Cada classe representa um comportamento diferente que pode ser observado analisando a relação entre os valores do IAT e do tempo de resposta (R) estabelecida através das funções RAT e DIF. Através delas é possível inferir as relações típicas de comportamento existente em cada classe, apresentadas na tabela 5.2.

A classe A tem o maior valor RAT dentre as classes impacientes (A, B e C). Sendo

assim, o tempo de resposta será maior que o IAT. Essa relação é representada pelo símbolo $<<$. Para as classes B e C, o IAT ainda é menor que R, mas o valor da função RAT é menor do que o seu valor para a classe A. Assim, representamos a relação entre o IAT e R para as classes B e C com um símbolo $<$. O mesmo acontece para as classes E, F e G só que inversamente, já que essas são classes de comportamento paciente. A classe G tem o maior valor para a função RAT comparado a E e F. Por essa razão, a relação entre o IAT e R para a classe G é representada com o símbolo $>>$ e com o símbolo $>$ para as classes E e F. No caso da classe de ação D, o valor do IAT é similar ao valor de R. O símbolo $\approx$ é utilizado para a classe D.

Classes	**Função DIF**	**Função RAT**	**Relação**
A	IAT - R $< k_1$	R / IAT $> k_4$	IAT $<<$ R
B	IAT - R $< k_1$	$k_3 <$ R / IAT $< k_4$	IAT $<$ R
C	IAT - R $< k_1$	R / IAT $< k_3$	IAT $<$ R
D	$k_1 <$ IAT - R $< k_2$	-	IAT $\approx$ R
E	IAT - R $> k_2$	IAT / R $< k_5$	IAT $>$ R
F	IAT - R $> k_2$	$k_5 <$ IAT / R $< k_6$	IAT $>$ R
G	IAT - R $> k_2$	IAT / R $> k_6$	IAT $>>$ R

Tabela 5.2: Relação entre o IAT e o tempo de resposta (R) para cada classe de ação

Com a finalidade de facilitar o entendimento do comportamento de cada classe de ação, a Figura 5.10 apresenta as representações de cenários típicos de requisição-resposta entre um usuário cliente e o servidor. Para cada situação está representado o cliente que requisita um *burst* ao servidor que responde após um certo tempo (tempo de resposta) de acordo com a sua carga e sua capacidade. A figura representa cenários típicos onde o servidor não está sobrecarregado e responde com um tempo de resposta baixo, abaixo do limite de 10 segundos a cada requisição, e também um cenário de sobrecarga, onde o tempo de resposta cresce, atingindo um valor acima dos 10 segundos.

Classes de ação associadas ao comportamento impaciente apresentam um comportamento de acordo com as classes A, B e C. Como pode ser observado na figura, o IAT é menor que o tempo de resposta. Em um cenário de não-sobrecarga do servidor, a diferença

entre o IAT e o tempo de resposta não é tão significativa quanto em sobrecarga. Quando sobrecarregado, o servidor leva mais tempo para responder aos *bursts* e a impaciência do cliente irá provocar a submissão de mais requisições antes de receber a resposta para a requisição anterior. Na perspectiva do servidor, um usuário impaciente poderá submeter um número maior de requisições antes de receber uma resposta, provocando aumentos na carga do servidor e, em última instância, colaborando para que a situação de sobrecarga se agrave.

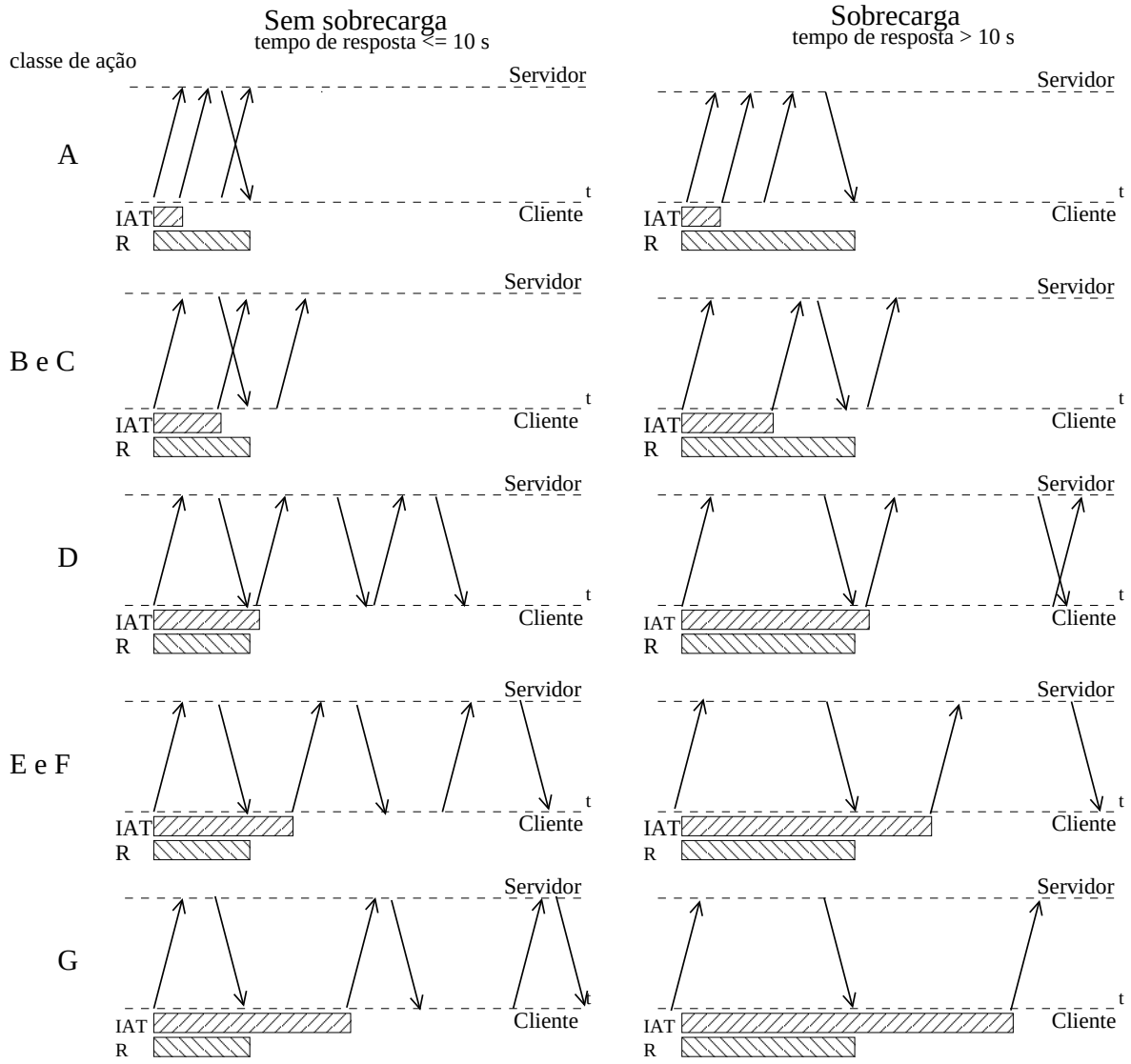


Figura 5.10: Comportamento Reativo por classe de ação

Classes de usuário pacientes apresentam comportamento descrito pelas classes E, F e G. O comportamento típico, como representado na figura, tem o IAT maior que o tempo de resposta, significando que, para cada requisição submetida ao servidor, o usuário tende a esperar a resposta antes de requisitar o próximo burst. Em situações de sobrecarga, usuários com comportamento paciente tendem a esperar a resposta e ainda demorar um tempo para processar a resposta, correspondente ao *think-time*. Isto é muito importante, visto que a sobrecarga do servidor pode diminuir devido ao comportamento de paciência desses clientes. A classe G apresenta o comportamento mais paciente visto que o IAT tende a ser maior em relação às classes E e F.

Do ponto de vista do cliente, diferentes tempos de resposta fazem com que o cliente se comporte de um modo diferente, reagindo à qualidade de serviço percebida. Isto pode ser observado comparando ambos os lados da figura para cada classe de ação.

Do ponto de vista do servidor, as reações dos usuários provocam diferentes mudanças em termos de carga, uma vez que variações em termos do tempo de resposta afetam a razão das requisições submetidas. De fato, o comportamento impaciente tende a causar o aumento na carga do servidor, uma vez que usuários que se comportam de acordo com as classes A, B e C usualmente requisitam requisições com taxas mais altas. Já classes com comportamento paciente tendem a provocar uma diminuição na carga do servidor devido ao comportamento dos usuários das classes E, F e G.

Em um cenário real, o número de usuários se comportando de acordo com cada classe de ação é variável e entender o impacto disto no desempenho de um servidor não é óbvio, devido à complexidade de tal cenário. Este trabalho explora este assunto através de experimentos utilizando um simulador que permite avaliar cenários reais de aplicações Internet.

5.3 Avaliação e Simulação do Comportamento Reativo

Nesta seção simulamos o comportamento reativo e avaliamos o impacto de cada uma das *classes de ação* definidas pelo modelo de reatividade, apresentado no capítulo 4.

A seção 5.3.1 apresenta o simulador *USAR-QoS* implementado para permitir avaliações de aplicações Internet considerando uma arquitetura Web típica, composta de clientes e um servidor. A seção 5.3.2 apresenta a metodologia e resultados dos experimentos realizados com diferentes configurações de carga, analisando diversas métricas. Por fim, a seção 5.3.3 discute os resultados.

5.3.1 USAR-QoS: Simulador de Qualidade de Serviço para Aplicações Internet

No intuito de avaliar o impacto da reatividade sobre uma aplicação Internet, bem como as novas políticas de QoS propostas neste trabalho, foi projetado o simulador *USAR-QoS*. A arquitetura do simulador se baseia em uma aplicação Internet real composta de um servidor provendo um certo serviço e um conjunto de clientes. O servidor contém uma ou mais filas e uma unidade de processamento com uma capacidade limitada de atendimento de requisições. O comportamento do cliente é baseado na versão reativa do gerador de cargas *httperf*, apresentado no Capítulo 5.1.

O *USAR-QoS* foi implementado utilizando a ferramenta *Simpack Toolkit* que consiste de um ambiente de simulação desenvolvido em C++ [Fishwick, 1992]. Sua arquitetura é orientada a eventos e construída respeitando modularidade, de modo a permitir sua extensão com novas políticas e recursos de QoS, como estratégias de controle de admissão e escalonamento.

A Figura 5.11 apresenta um diagrama de estados que representa o simulador *USAR-QoS*. Os eventos descritos são os seguintes:

- *EvntSource()*: inicia a execução do *USAR-QoS*, carregando os parâmetros da carga

de trabalho e as classes de políticas de QoS a serem utilizadas na execução. Ao final da sua execução o evento *Sess_Creation* é escalonado para cada sessão a ser criada e executada.

- *Sess_Create()*: representa a criação de uma nova sessão. Para cada execução escalona a seguir o evento *Call_Send_Start*.

- *Call_Send_Start()*: representa a submissão de um burst ao servidor. Para cada burst, este evento escalona a execução dos eventos *Sess_Admission_Control* e *Call_TimeOut*.

- *Sess_Admission_Control()*: verifica se a sessão corrente tem a permissão para ser atendida pelo servidor, de acordo com a política de controle de admissão de sessões sendo executada. Caso a sessão seja aceita, é escalonado o evento *Burst_Admission_Control*. Caso contrário a sessão deve ser finalizada escalonando o evento *Sess_Destroy*.

- *Burst_Admission_Control()*: verifica se o burst corrente tem permissão para ser executado pelo servidor de acordo com a política de controle de admissão de bursts. Se for aceito, o evento *Server* deve ser escalonado. Caso contrário, nada precisa ser feito já que o evento *Call_TimeOut* será executado posteriormente para aquele burst.

- *Server()*: modela o servidor e suas filas de prioridade, onde a política de escalonamento é executada. Para cada burst é associada uma fila, de acordo com a política de escalonamento. Ao final da sua execução o evento *Call_Send_Stop* é escalonado.

- *Call_Send_Stop()*: finaliza a execução de um burst, obtendo informações sobre sua execução e escalona o evento *Call_Send_Finalize*.

- *Call_Send_Finalize()*: este evento é executado uma única vez para cada burst, sendo escalonado pelo evento *Call_Send_Stop* ou pelo evento *Call_Timeout*, de acordo com o comportamento do cliente. Esse mecanismo garante que cada burst é corretamente finalizado. Caso o burst executado seja o último da sessão e todos os outros bursts já tenham sido executados, este evento também é responsável por escalonar o evento

Sess_Destroy. Caso ainda hajam bursts para serem executados na sessão, ao finalizar sua execução, escalona *Call_Send_Start* para que o próximo burst seja executado.

- *Call_Timeout()*: controla o tempo limite de espera pela chegada da resposta ao burst requisitado, de acordo com o comportamento reativo do cliente e o modelo de reatividade. Caso a resposta do burst correspondente não tenha sido recebida quando da sua execução, o evento *Call_Send_Finalize* é executado com a finalidade de finalizar o burst corrente. Durante períodos em que o desempenho do servidor piora, aumentando o tempo de resposta, este mecanismo simula o comportamento impaciente, engatilhando o evento *Call_Send_Finalize* antes que o *Call_Send_Stop* o faça.

- *Sess_Destroy()*: executado quando uma sessão deve ser destruída. Finaliza a execução do simulador quando todas as sessões já foram executadas.

O simulador *USAR-QoS* foi preparado para registrar as seguintes informações sobre cada execução:

- Throughput de *bursts* respondidos: a taxa de *bursts* respondidos pelo servidor em cada instante de tempo.

- Throughput de *bursts* requisitados: a taxa em que os *bursts* são requisitados ao servidor pelos usuários em cada período de tempo.

- Throughput de *bursts* expirados: a taxa de *bursts* em que o usuário requisita o próximo antes de receber a resposta referente ao corrente, devido ao comportamento impaciente e a altos tempos de resposta. Isto acontece quando o evento *Call_Timeout* engatilha a execução do evento *Call_Send_Finalize*, de acordo com o descrito acima.

- Throughput de *bursts* rejeitados: a taxa de *bursts* rejeitados pelo mecanismo de controle de admissão de *bursts* a cada instante de tempo.

- Sessões ativas: o número de sessões iniciadas e que não foram finalizadas, isto é, que tem *bursts* ativos ou que não tenham sidos submetidos a cada instante de tempo.

- Taxa de sessões rejeitadas: a taxa com que o mecanismo de controle de admissão de sessões rejeita sessões a cada instante de tempo.
- Tempo de resposta: o tempo de resposta percebido pelo usuário, compreendendo o intervalo de tempo entre a requisição do *burst* e o recebimento de sua resposta.
- Tamanho da fila: o número de *bursts* esperando serviço em uma das filas do servidor a cada instante de tempo.
- Utilização do servidor: proporção de tempo em que o servidor está ocupado.

Os dados de throughput e tempo de resposta foram registrados, ainda, aplicando uma função de suavização chamada *smooth bezier* que reduz os picos e permite verificar melhor o comportamento das curvas.

Para realizar uma simulação utilizando o *USAR-QoS*, os seguintes parâmetros devem ser informados: taxa média com que as sessões serão criadas, número total de sessões a serem executadas, política de controle de admissão de sessões, política de controle de admissão de *bursts*, política de escalonamento de requisições, tipo de execução (reativa ou não-reativa), e o arquivo de *trace* contendo as sessões a serem executadas. É importante salientar que o servidor foi configurado com uma capacidade máxima de 50 bursts por segundo.

5.3.2 Avaliação Experimental

Nesta seção apresentamos os resultados experimentais utilizados para avaliar o comportamento reativo e o simulador *USAR-QoS*. Primeiramente discutimos metodologia utilizada e então apresentamos os resultados experimentais.

5.3.2.1 Metodologia

No intuito de avaliar o comportamento reativo foram realizados experimentos no simulador *USAR-QoS* utilizando diversos cenários de carga. Apresentamos logo abaixo as simulações

envolvendo o mesmo grupo de 5.000 sessões utilizadas no experimento real apresentado no capítulo 5.1 baseadas no *benchmark TPC-W*. Para cada cenário a carga de trabalho foi configurada com diferentes distribuições de classes de ação de usuário. Foram avaliadas configurações com distribuições exclusivas, contendo 100% de cada uma das classes de usuário, e com uma distribuição mista (A 5%, B 10%, C 10%, D 15%, E 15%, F 15%, and G 30%).

Para cada um dos experimentos, apresentamos uma tabela contendo um resumo dos principais dados obtidos. São os seguintes:

- Carga: identificação da carga de trabalho utilizada no experimento.
- Duração (s): total de tempo simulado no experimento.
- Máximo R (s): máximo valor de tempo de resposta atingido no experimento.
- Médio R (s): tempo de resposta médio durante o período de maior saturação. Para esta última medida, consideramos o intervalo de pico em termos do número de sessões requisitando respostas ao sistema. Para isto, obtemos o máximo valor do número de sessões ativas e consideramos o intervalo em que este valor tenha variado em até 10% do seu valor máximo. Então, calculamos o valor médio do tempo de resposta para este intervalo.
- Bursts requisitados, respondidos e expirados: respectivamente, o número de bursts que foram requisitados ao servidor, o número daqueles cuja resposta foi enviada, e o número daqueles onde o cliente apresentou comportamento impaciente, requisitando outra requisição antes da chegada da anterior.
- Taxa de perda: percentual de burts expirados em função do número total de bursts requisitados.

5.3.2.2 Resultados

As figuras 5.12 apresentam os gráficos obtidos para o experimento que utiliza o simulador *USAR-QoS* com uma configuração mista da carga de trabalho em termos das classes de usuário. Um total de 5.000 sessões são criadas com uma taxa de 10 sessões por segundo. Assim, em cerca de 500 segundos todas as sessões já terão sido criadas, o que pode ser observado no gráfico (a). O tempo de resposta médio visto no gráfico (b) varia indo de valores mais baixos a picos de até 30 segundos. Este efeito se deve à intensidade de carga e ao comportamento dos clientes. Podemos observar que o tempo de resposta, entre os instantes 10 e 4.000, aproximadamente, atinge valores maiores do que 10 segundos em média, demonstrando o estado de saturação do servidor cujo throughput chega ao seu limite, como pode ser visto nos gráficos (e) e (f), havendo um grande número de bursts que expiram durante a execução.

A tabela 5.3 apresenta o sumário do experimento. Como pode ser visto, o tempo simulado foi de 13.253 segundos. O tempo de resposta máximo atingido por um burst foi de 31,18 segundos e o valor médio para o intervalo de saturação foi de 12,23 segundos. Um total de 621.342 bursts foram executados sendo que 37,11% deles expiraram devido alto tempo de resposta atingido e ao comportamento impaciente presente em algumas sessões.

Carga	Duração (s)	Máximo R (s)	Médio R (s)	Bursts requisitados	Bursts respondidos	Bursts expirados	Taxa de Perda
Carga mista	13.253	31,18	12,23	621.342	621.342	230.606	37,11%

Tabela 5.3: Resumo do experimento com a carga de composição mista

As figuras de 5.13 a 5.19 e a tabela 5.4 apresentam os experimentos com as cargas contendo distribuição exclusiva das classes de A a G. As figuras mostram as curvas do tempo de resposta médio e throughput cumulativo. Pode ser observado claramente que cada configuração de carga traz um impacto diferente no desempenho do servidor.

A carga composta de classes A, apresentada na figura 5.13, apresenta uma grande carga para o servidor, que responde com um tempo de resposta máximo que chega a

680,50 segundos e médio no período de saturação de cerca de 503,13 segundos. As cargas B e C apresentadas nos gráficos de 5.14 e 5.15 também apresentam um nível significativo de carga, que obriga o servidor a responder com tempos de resposta médios no período de saturação de cerca de 299,42 e 162,89 segundos, respectivamente. Nos experimentos com as cargas A, B e C observa-se uma alta taxa de expiração de cerca 100%, confirmando a tendência de *insatisfação* dos usuários.

O experimento com a carga que contém classes D, apresentado na figura 5.16, apresenta um tempo de resposta máximo de 96,07 segundos e médio no intervalo de saturação de cerca de 91,29 segundos, que começa a decrescer gradualmente. Neste caso, para 68,77% dos bursts há expiração.

Já os experimentos contendo as classes E, F, e G apresentam diferenças representativas se comparadas com os grupos anteriores. O tempo de resposta médio de E, apresentado na figura 5.17, chega a picos de 51,69 segundos no período de maior carga, mas apresenta uma média no intervalo de saturação de 30,87 segundos. Os experimentos com classes F e G, nos gráficos de 5.18 e 5.19, resultam em tempos de resposta que chegam a picos de 31,39 e 18,47 segundos, respectivamente, mas com valores médios bem menores que chegam a 11,39 e 4,60 segundos. Para essas cargas há um alto grau de satisfação em termos da taxa de perda de requisições que se aproxima de 0%.

Os experimentos realizados permitem verficar, enfim, qual impacto o comportamento reativo tem sobre um servidor com contenção de recursos. Podemos verificar que diferentes configurações de carga resultam em diferentes comportamentos, tanto do ponto de vista do cliente como do servidor, confirmando as inferências descritas na seção 5.2.

Classe	Duração (s)	Máximo R (s)	Médio R (s)	Bursts requisitados	Bursts respondidos	Bursts expirados	Taxa de Perda
A	12.432	680,50	503,13	621.342	621.342	621.316	100,00%
B	12.431	332,69	299,42	621.342	621.342	621.317	100,00%
C	12.430	172,53	162,89	621.342	621.342	621.220	99,98%
D	12.430	96,07	91,29	621.342	621.342	427.318	68,77%
E	12.431	51,69	30,87	621.342	621.342	890	0,14%
F	12.466	31,39	11,39	621.342	621.342	49	0,01%
G	12.578	18,47	4,60	621.342	621.342	89	0,01%

Tabela 5.4: Sumário dos experimentos com cargas compostas de uma única classe de usuário

5.3.3 Sumário

Este capítulo apresenta o simulador *USAR-QoS* construído para permitir uma avaliação mais precisa de políticas de escalonamento e controle de admissão para prover Qualidade de Serviço. A arquitetura simulada compõe-se de um conjunto de clientes requisitando serviços a um servidor Web com uma capacidade limitada de atendimento de requisições.

O capítulo analisa e avalia o comportamento reativo, detalhando o mecanismo de ação e reação correspondente a cada classe de ação do modelo de reatividade. As conclusões dessa avaliação indicaram a necessidade de uma avaliação mais detalhada quando uma carga é composta de um conjunto misto de usuários, em termos das suas classes de ação.

O conjunto de experimentos realizados utilizando uma carga de composição mista e cargas compostas de uma única classe de ação para avaliar o comportamento reativo, permitem comprovar que a reatividade causa um impacto significante no desempenho de um servidor, contribuindo para variações dos tempos de resposta. As classes de comportamento impaciente, tipicamente, colaboram para o aumento da carga submetida ao servidor e as de comportamento paciente, tipicamente, apresentam uma taxa requisição menos intensa, contribuindo para que a carga do servidor seja menor.

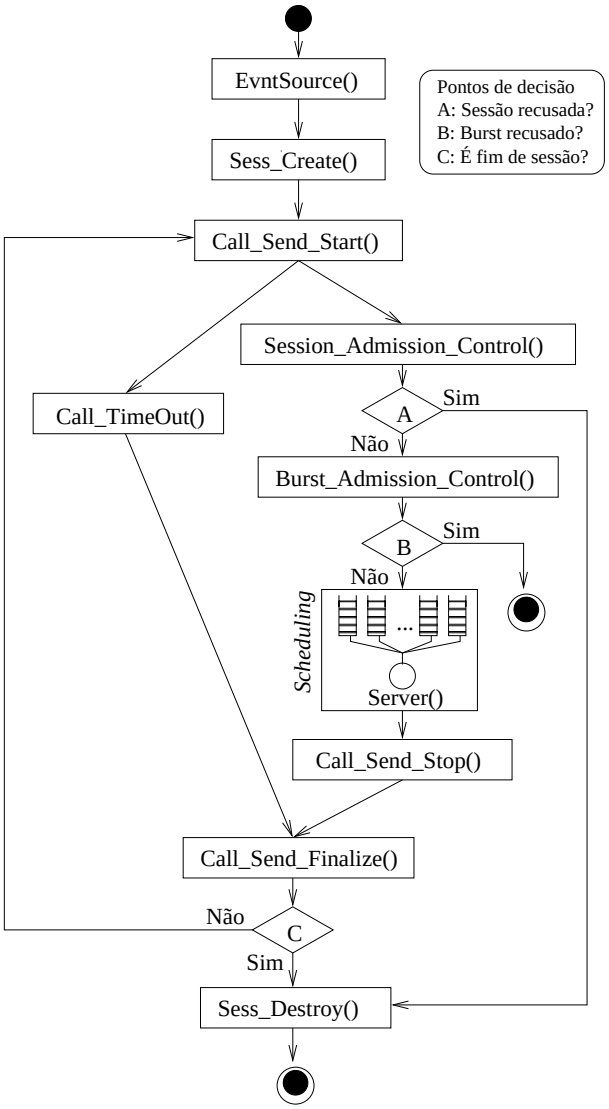
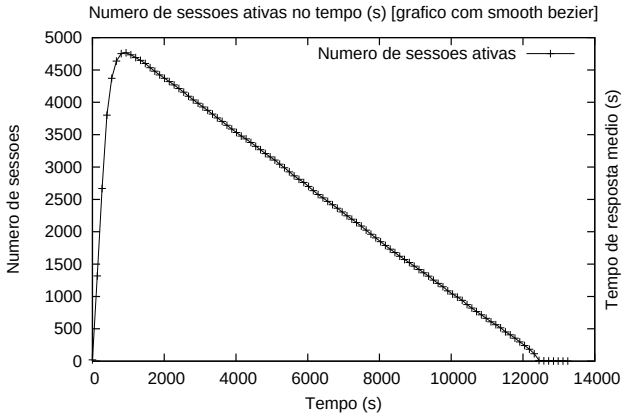
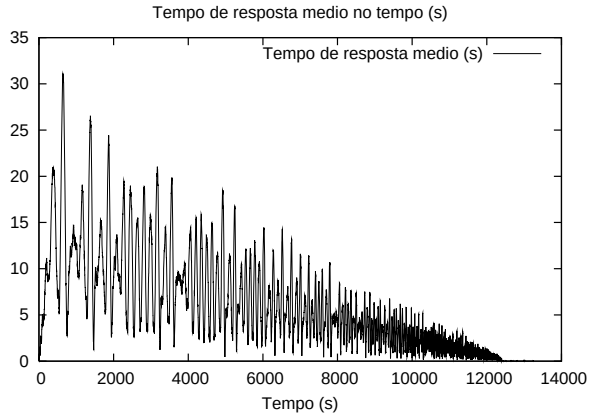


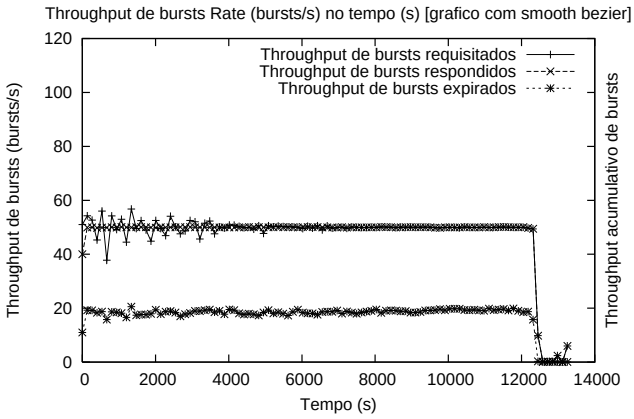
Figura 5.11: Diagrama de estados representando os eventos do simulador *USAR-QoS*



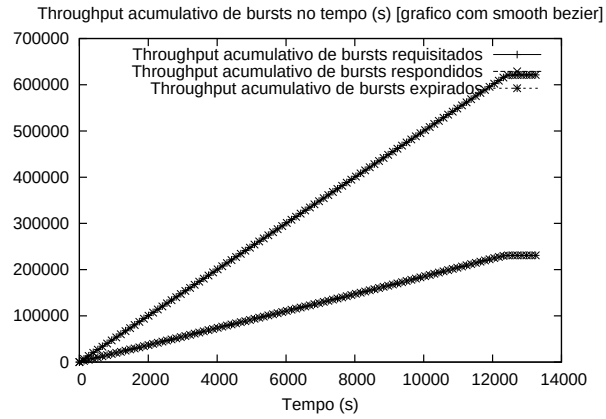
(a) Número de sessões



(b) Tempo de resposta médio

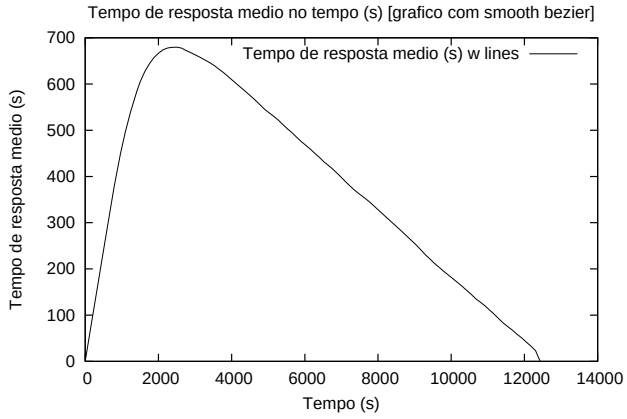


(c) Throughput

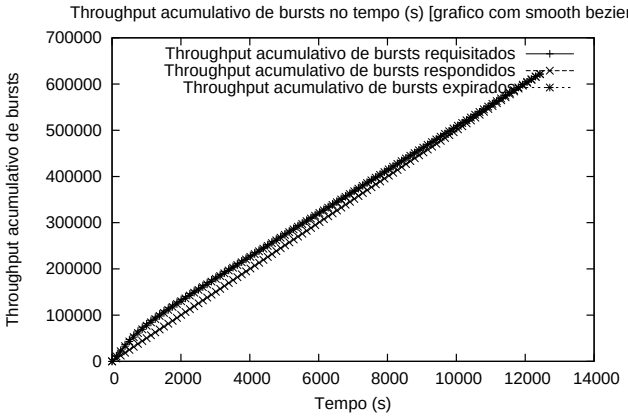


(d) Throughput cumulativo

Figura 5.12: Simulação utilizando carga com usuários com comportamento misto

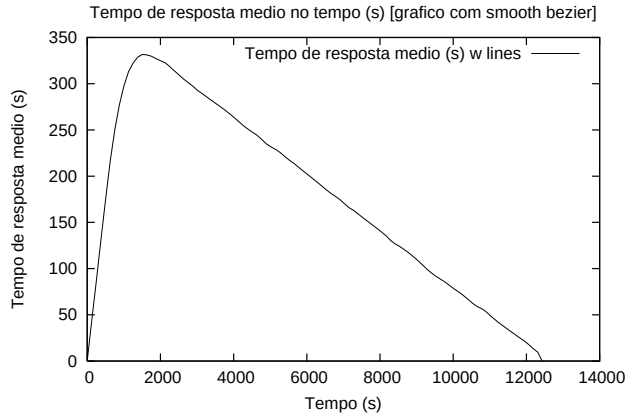


(a) Tempo de resposta médio

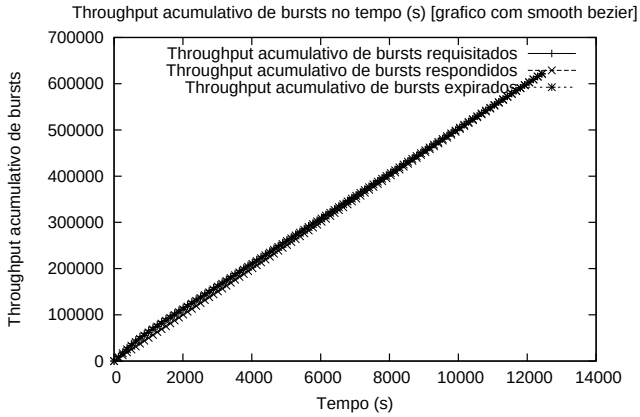


(b) Throughput cumulativo

Figura 5.13: Carga da classe A

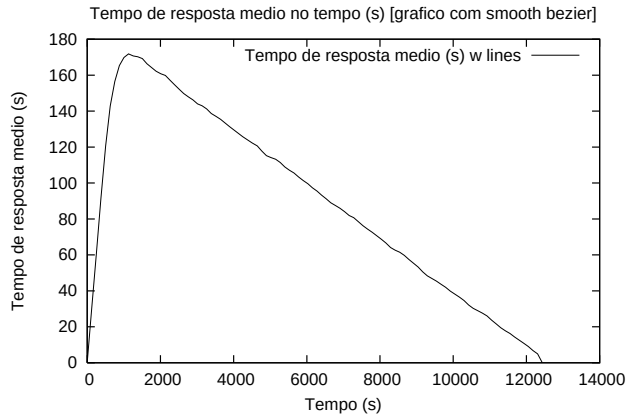


(a) Tempo de resposta médio

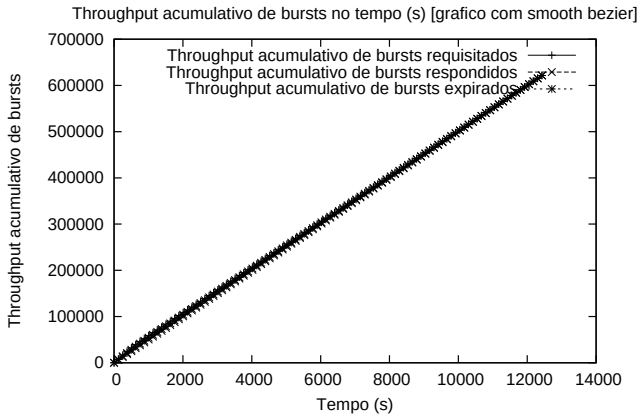


(b) Throughput cumulativo

Figura 5.14: Carga da classe B

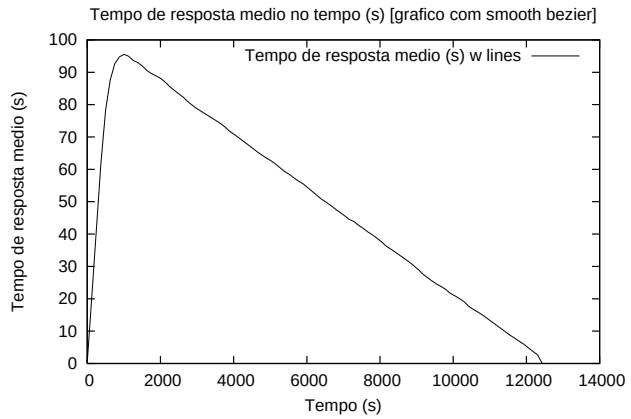


(a) Tempo de resposta médio

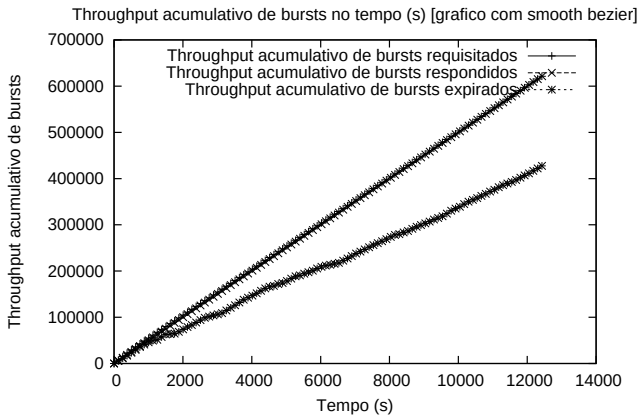


(b) Throughput cumulativo

Figura 5.15: Carga da classe C

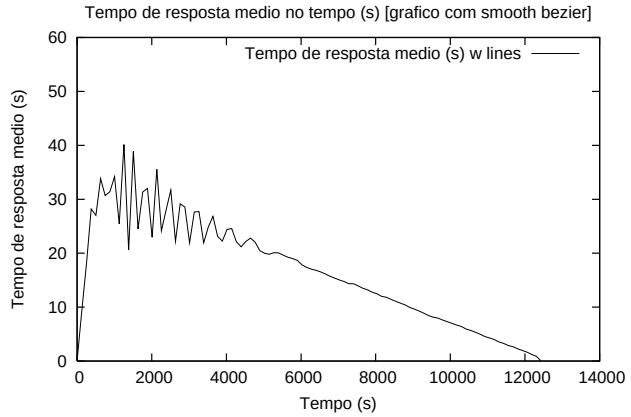


(a) Tempo de resposta médio

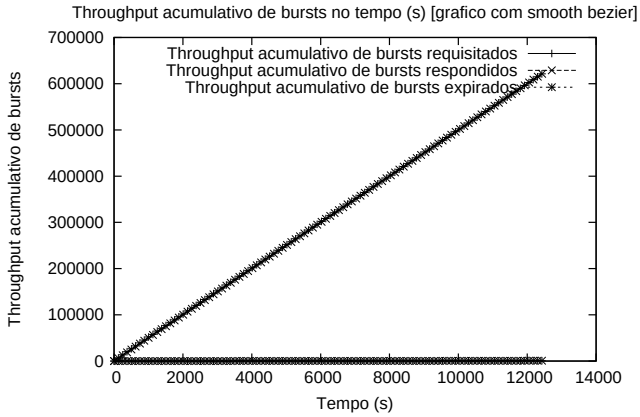


(b) Throughput cumulativo

Figura 5.16: Carga da classe D

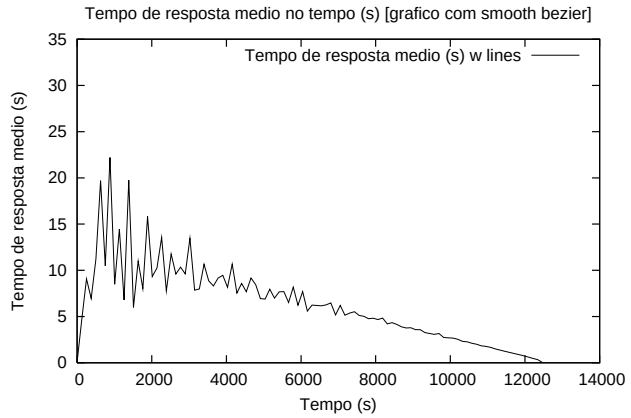


(a) Tempo de resposta médio

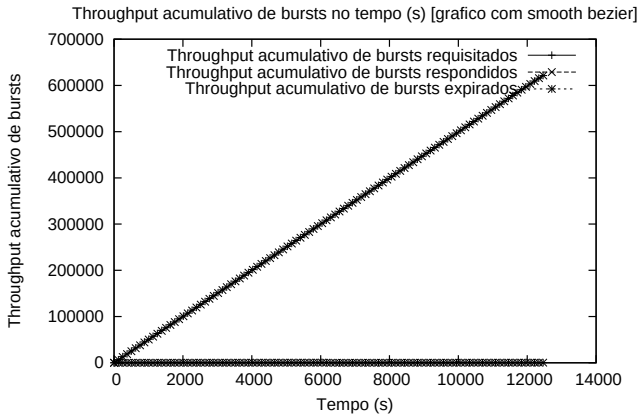


(b) Throughput cumulativo

Figura 5.17: Carga da classe E

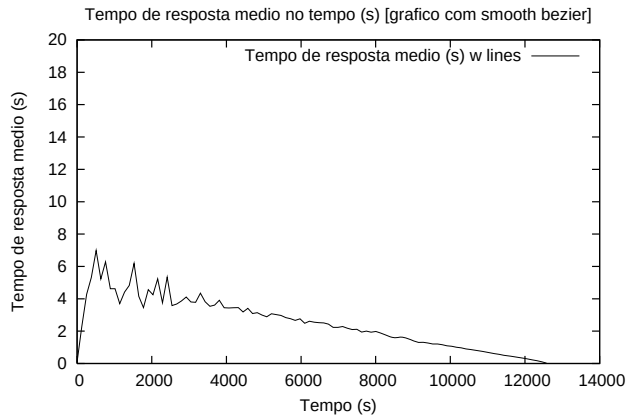


(a) Tempo de resposta médio

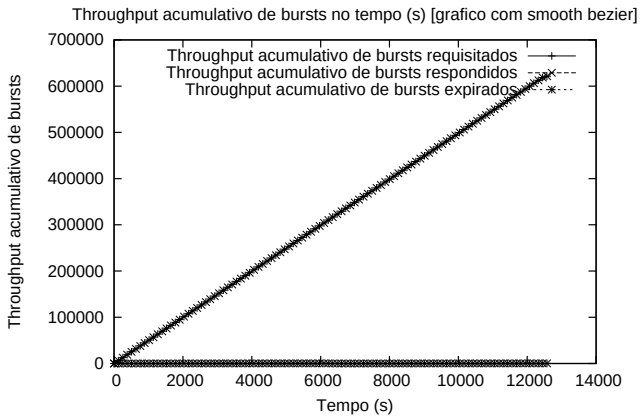


(b) Throughput cumulativo

Figura 5.18: Carga da classe F



(a) Tempo de resposta médio



(b) Throughput cumulativo

Figura 5.19: Carga da classe G

Capítulo 6

Políticas Reativas de Qualidade de Serviço

Este capítulo discute as estratégias para provimento de Qualidade de Serviço baseadas em controle de admissão e escalonamento. Apresentamos novas abordagens baseadas no critério de reatividade. As abordagens propostas são avaliadas utilizando o simulador *USAR-QoS*.

A seção 6.1 apresenta e avalia as políticas reativas de controle de admissão. A seção 6.2 apresenta e avalia as políticas reativas de escalonamento. A seção 6.3 discute o uso de controle de admissão e escalonamento única abordagem conjunta de provimento de Qualidade de Serviço.

6.1 Políticas Reativas de Controle de Admissão

Esta seção discute a estratégia para provimento de Qualidade de Serviço baseada no controle de admissão. Apresentamos novas abordagens baseadas na admissão de ações e sessões baseadas no critério de reatividade. As abordagens propostas são avaliadas utilizando o simulador *USAR-QoS*.

A seção 6.1.1 introduz o assunto de controle de admissão, explicando o conceito envol-

vido. A seção 6.1.2 apresenta as novas estratégias de controle de admissão baseadas em informações sobre a reatividade dos usuários. A seção 6.1.3 apresenta a metodologia de avaliação das novas políticas e os resultados obtidos. A seção 6.1.4 discute os resultados obtidos e infere as principais conclusões a respeito do assunto tratado na seção.

6.1.1 Controle de Admissão

Mecanismos de controle de admissão têm a finalidade de controlar a intensidade de carga que o servidor deve atender através da rejeição de requisições de acordo com um critério previamente definido. Quando a taxa de chegada está muito alta, acima do limite suportado pelo servidor, por exemplo, é necessário manter uma intensidade de carga aceitável no sistema. Sem mecanismos de controle de admissão, o tempo de resposta pode aumentar quando o sistema entra em estado de saturação.

Tipicamente, a medida da taxa de utilização do servidor ou o tamanho da fila são critérios utilizados em mecanismos de controle de admissão. Entretanto, essas abordagens desconsideram as diferenças de comportamento que os usuários podem apresentar. Sendo assim, este trabalho propõe novas estratégias que utilizam informações sobre a reatividade dos usuários para implementar um mecanismo de controle de admissão adaptável às características da carga de trabalho. Propomos duas abordagens: a primeira se baseia na rejeição de bursts e a segunda na rejeição de sessões.

6.1.2 Novas políticas de controle de admissão

6.1.2.1 Abordagem Reativa Baseada em Ações

Baseado no modelo de reatividade, este trabalho propõe uma nova política que considera como os usuários tendem a reagir de acordo com o desempenho apresentado pelo serviço Internet. Essa nova abordagem rejeita as ações quando é identificado que uma certa regra foi atendida. As políticas tradicionais utilizam, tipicamente, somente um limite de tempo de resposta, rejeitando as ações assim que o limite for atingido. A abordagem proposta

define a política em função do tempo de resposta do serviço para cada ação do usuário (R) de acordo com as seguintes regras:

- $\alpha \leq R < \beta$: rejeitar as ações classificados como sendo das classes de ação A, B e C.
- $\beta \leq R < \theta$: rejeitar as ações classificados como sendo das classes de ação A, B, C e D.
- $R \geq \theta$: rejeitar as ações associados a todas as classes de ação.

Nessas regras, os valores de α, β e θ são determinados de acordo com resultados empíricos, de modo que o tempo de resposta esteja abaixo do limite de 10 segundos conforme discussão apresentada na seção 2.3.

A idéia desta política é que os usuários que têm perfis associados às classes de ação mais impacientes tendem a reagir mais rapidamente do que os outros usuários quando o servidor apresenta altos tempos de resposta, o que pode provocar, como visto no capítulo 5, um aumento da carga submetida ao servidor. O mecanismo proposto tem uma regra multi-critério no intuito de minimizar o impacto da rejeição, uma vez que menos usuários terão ações recusados pelo mecanismo de controle de admissão. Em suma, essa política se baseia na idéia de que, em cenários de sobrecarga, é preferível priorizar usuários que terão mais chance de esperar por respostas a suas requisições.

6.1.2.2 Abordagem Reativa Baseada em Sessões

Políticas de controle de admissão baseadas em sessões realizam a rejeição de ações de sessões determinadas a partir do momento em que ocorre a violação de certo critério, com a finalidade de atingir menos usuários, uma vez que somente ações de sessões específicas serão rejeitadas. Na abordagem por ações, todas as sessões poderão ser afetadas. Deste modo, o aspecto mais importante dessa política está na definição do mecanismo para identificar quais sessões devem ser rejeitadas.

As políticas tradicionais utilizam um único limite do tempo de resposta para iniciar a rejeição das sessões. Para a elaboração da política reativa baseada em sessões que propo-

mos, utilizamos informações sobre o tempo de resposta médio (R) e a classe de usuário característica de cada sessão, USC (de *user session class*) para definir as seguintes regras:

- $\alpha \leq R < \beta$: rejeitar as ações de todas as sessões com $USC < 4$, ou seja, sessões associadas às classes de ação A, B e C.
- $\beta \leq R < \theta$: rejeitar as ações de sessões com $USC < 5$, ou seja, sessões associadas a classes de ação A, B, C, e D.
- $R \geq \theta$: rejeitar as ações de todas as sessões.

Para obter a classe de ação característica de cada sessão, ou seja, o valor de USC, pode-se utilizar a média dos valores das classes de ação de cada ação já atendida para a sessão.

Essa abordagem reativa para realizar o controle de admissão de sessões se baseia na mesma idéia utilizada para a elaboração da abordagem baseada em ações.

6.1.2.3 Abordagem Reativa de Dois Níveis

Baseado nas abordagens reativas de controle de admissão de ações e sessões apresentadas nas seções anteriores, propomos um mecanismo híbrido que permite a utilização de ambos os critérios, de modo a agregar os benefícios de cada abordagem individual em uma única estratégia equilibrada.

Na abordagem por sessão, a rejeição de ações de sessões inteiras se inicia assim que o tempo de resposta aumenta. Na abordagem de dois níveis que propomos, primeiramente, a rejeição de ações é iniciada, antes que a rejeição de sessões se inicie. Mesclando as estratégias pode-se evitar a rejeição de sessões, através de uma etapa anterior. Uma vez que a rejeição não esteja sendo efetiva para diminuir o tempo de resposta, a rejeição de sessões pode ser iniciada. Assim, os critérios das abordagens são redefinidos de acordo com as seguintes regras:

- $\alpha_1 \leq R < \beta_1$: rejeitar ações associados às classes de ação A, B e C.

- $\beta_1 \leq R < \theta_1$: rejeitar ações associados às classes de ação A, B, C e D.
- $R \geq \theta_1$: rejeitar ações associados a todas as classes de ação.
- $\alpha_2 \leq R < \beta_2$: rejeitar todos as ações de sessões classificadas como de classes de ação impacientes, $USC < 4$, ou seja, sessões associadas às classes de ação A, B e C.
- $\beta_2 \leq R < \theta_2$: rejeitar todos as ações de sessões classificadas com $USC < 5$, ou seja, associadas às classe de ação A, B C, e D.
- $R \geq \theta_2$: rejeitar todas as sessões.

Nessa política, enfim, a rejeição baseada em ações e sessões é utilizada, mas baseada em valores limites diferentes, de um modo balanceado.

6.1.3 Avaliação Experimental

Nesta seção apresentamos a metodologia utilizada e os resultados obtidos para os experimentos que implementam as políticas de controle de admissão.

6.1.3.1 Metodologia

Com a finalidade de avaliar a efetividade das políticas de controle de admissão, foram realizados experimentos utilizando o simulador *USAR-QoS*. Para isto, foi necessário, primeiramente, implementar as políticas reativas propostas, escolhendo cuidadosamente os devidos valores para as variáveis α, β e θ de cada uma. O método para escolha dos valores dessas variáveis foi a análise dos resultados do experimento apresentado na seção 5.2, que descreve um cenário simulado utilizando uma carga de composição mista, em termos das classes de ação, que não utilizou nenhuma política de controle de admissão e o mecanismo de escalonamento de melhor esforço FIFO. Observando os valores do tempo de resposta obtidos e levando em consideração seu valor limite aceitável de 10 segundos, como discutido na seção 2.3, escolhemos valores para as variáveis que permitissem a diminuição dos efeitos da sobrecarga.

Foram escolhidos os seguintes valores para esses parâmetros:

- Política baseada em ações: $\alpha = 3,0$, $\beta = 5,0$, e $\theta = 7,0$.
- Política baseada em sessões: $\alpha = 5,0$, $\beta = 7,0$, e $\theta = 9,0$.
- Política de dois níveis: $\alpha_1 = 3,0$, $\alpha_2 = 5,0$, $\beta_1 = 5,0$, $\beta_2 = 7,0$, $\theta_1 = 7,0$ e $\theta_2 = 9.0$.

Para observarmos como o comportamento do servidor seria afetado, além da carga mista utilizada no experimento base da seção 5.3.2, foram simulados outros cenários utilizando cargas com diferentes distribuições de classes de ação. Apresentamos, entretanto, somente o cenário contendo a carga do experimento base, já que os resultados obtidos são suficientes para efetuarmos uma análise conclusiva sobre os cenários avaliados. Relembrando, a carga era composta de 5000 sessões que seriam criadas a uma taxa média de 10 sessões por segundo.

Além das novas políticas propostas, foram implementados os mecanismos tradicionais não-reativos (ou seja, que não consideram a reatividade) de controle de admissão a fim de efetuarmos uma comparação entre os resultados obtidos. Foram implementadas duas políticas não-reativas, uma baseada em ações e outra em sessões, que iniciam a rejeição independente da classe de ação e a partir de um único limite λ escolhido. Os valores dessa variável foram escolhidos de modo a manter níveis aceitáveis de tempo de resposta, abaixo do limite de 10 segundos. Os valores escolhidos foram $\lambda = 7,0$ e $\lambda = 9,0$ na abordagem por ações e por sessões, respectivamente.

6.1.3.2 Avaliação das Políticas Baseadas em Ações

As figuras 6.1 e 6.2 e a tabela 6.1 apresentam os experimentos que avaliam as políticas de controle de admissão baseadas em ações não-reativa e reativa. Nas figuras observamos os gráficos do tempo de resposta (a), o número cumulativo de sessões criadas (b), e o throughput e throughput cumulativo em (c) e (d), respectivamente.

Abordagem de rejeição	Duração (s)	Max R (s)	Médio R (s)	Ações requisitadas	Ações respondidas	Ações expiradas	Taxa de Perda
Não-reativa	11.394	7,32	7,01	621.342	536.978	284.691	45,82%
Reativa	9.751	7,22	6,34	621.342	465.974	243.799	39,24%

Tabela 6.1: Sumário dos experimentos de controle de admissão por ações

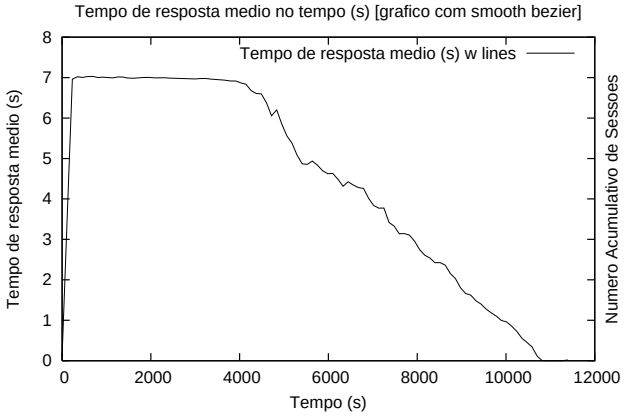
Para o experimento que avalia a política não-reativa, verificamos que o tempo de resposta apresenta um limite máximo de 7,32 segundos, que é próximo ao limite estabelecido para a política, como explicado na seção anterior. Podemos verificar que essa política é efetiva em manter o limite dos valores do tempo de resposta.

Podemos observar que o valor do throughput de ações expiradas apresenta uma alta taxa de perda que chega a 45,82%. Ou seja, essa abordagem é efetiva em manter um baixo tempo de resposta, mas a rejeição de ações, devido ao comportamento dos usuários, provoca uma taxa de instatisfação alta, o que se reflete na taxa de perda.

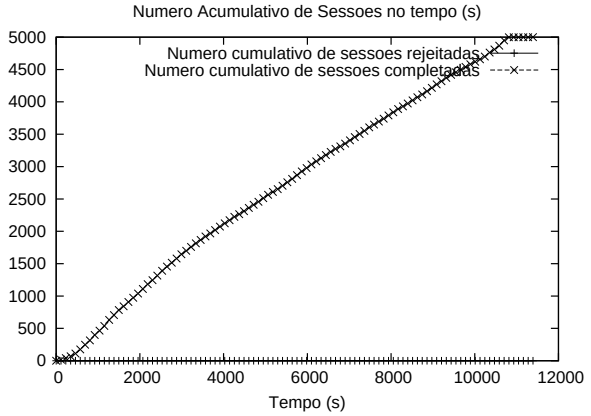
No experimento reativo observamos que o tempo de resposta aumenta, mas não ultrapassa o valor de 7,22 segundos, que é próximo do valor definido para a variável θ definida para a política. Como existem outros patamares de rejeição para o tempo de resposta, definidos pelas outras variáveis α e β, a taxa de rejeição de ações nesse experimento ainda é alta, mas inferior ao valor do experimento não-reativo, chegando a 39,24%.

Se compararmos esses resultados com o experimento que não utiliza controle de admissão apresentado na seção 5.2, o controle de admissão de ações reativo apresenta um valor de tempo de resposta muito inferior e o valor médio é cerca de 50% menor, com uma taxa de perda somente um pouco maior. O experimento não-reativo, também apresenta um valor bastante inferior para o tempo de resposta, mas a taxa de perda, nesse caso, é maior.

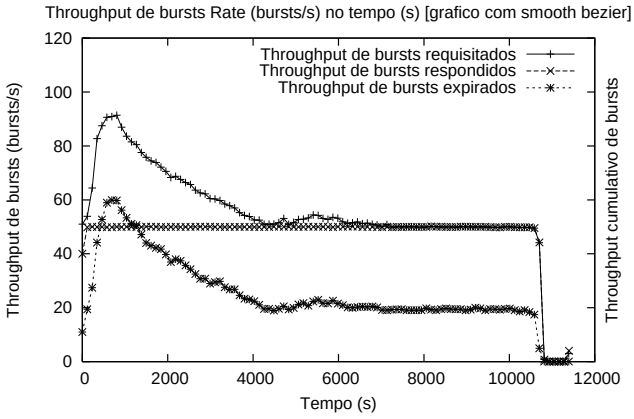
Deste modo podemos observar que a abordagem reativa, em comparação com a não reativa, atinge um resultado mais eficiente, pois ela considera os aspectos comportamentais presentes na carga. Ela é efetiva em manter um tempo de resposta dentro de valores aceitáveis sem apresentar diminuição da taxa de perda.



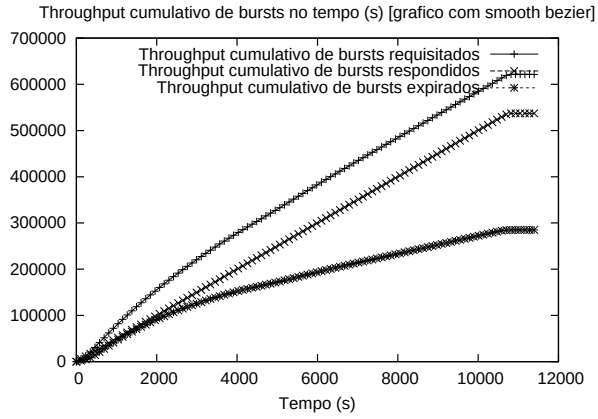
(a) Tempo de resposta médio



(b) Número cumulativo de sessões



(c) Throughput



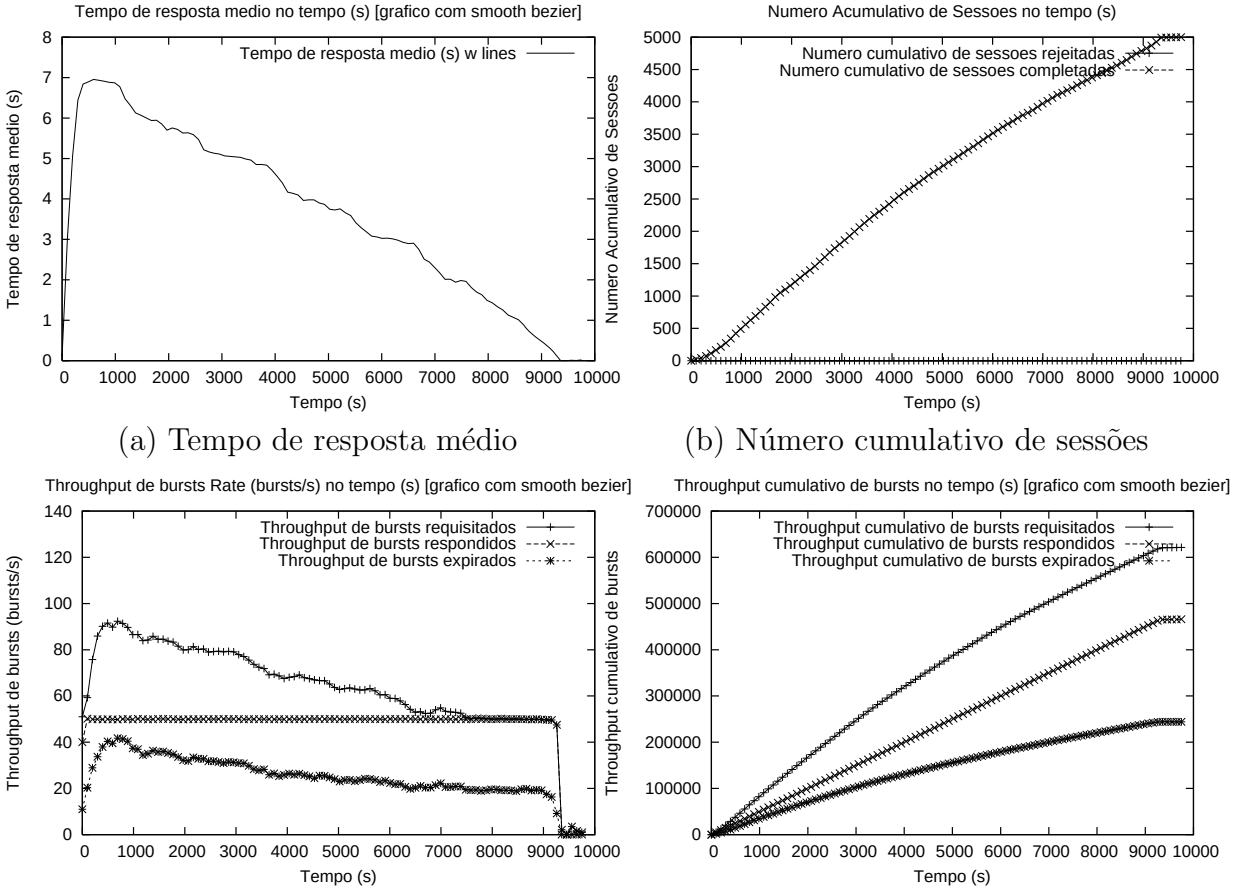
(d) Throughput cumulativo

Figura 6.1: Abordagem não-reativa de controle de admissão baseado em ações

6.1.3.3 Avaliação das Políticas Baseadas em Sessão

A tabela 6.2 e as figuras 6.3 e 6.4 apresentam os resultados obtidos para os experimentos que avaliam as políticas baseadas em sessão, não-reativa e reativa. As figuras apresentam, do mesmo modo como na subseção anterior, informações sobre o tempo de resposta, o número cumulativo de sessões e o throughput.

O experimento não-reativo apresenta um tempo de resposta máximo de 9,33 segundos, que corresponde, aproximadamente, ao valor limite definido para esta política. O valor médio do tempo de resposta no período de saturação foi de 5,96 segundos. A taxa de perda de ações foi de 37,91%. Entretanto, há um número muito grande de sessões rejeitadas que chega a 2544. Ou seja, para garantir um tempo de resposta dentro de limites aceitáveis, foi



(a) Tempo de resposta médio (b) Número cumulativo de sessões

(c) Throughput (d) Throughput cumulativo

Figura 6.2: Abordagem reativa de controle de admissão baseado em ações

necessária a interrupção de um grande número de sessões, o que pode gerar um nível muito grande de instatisfação de muitos clientes. Entretanto, essa política é muito eficiente em manter os tempos de resposta dentro de níveis aceitáveis.

O experimento reativo apresenta um cenário sutilmente melhor, já que o tempo de resposta apresenta um valor máximo de 9,24 segundos e um valor médio de 4,88, ou seja, quase 20% melhor em comparação com o experimento não-reativo. Entretanto, o número de sessões rejeitadas foi cerca de 14% maior em relação ao experimento não reativo, chegando a 2906 sessões, já que a rejeição se inicia previamente.

Comparando esses resultados com a abordagem reativa de rejeição de ações, podemos verificar que o tempo de resposta médio obtido e a taxa de rejeição são cerca de 23% e

Abordagem de rejeição	Duração (s)	Max R (s)	Médio R (s)	Ações reqs.	Ações resps.	Ações exps.	Taxa de Perda	Sessões rejeitadas
Não-reativa	6.592	9,33	5,96	319.193	316.649	121.014	37,91%	2.544
Reativa	6.040	9,24	4,88	276.561	273.655	103.926	37,58%	2.906

Tabela 6.2: Sumário dos experimentos de controle de admissão por sessão

14% menores, respectivamente. Entretanto, apesar de conseguir atender eficientemente às sessões não-rejeitadas, um número muito alto de sessões são rejeitadas, e assim, muitos clientes não são atendidos satisfatoriamente.

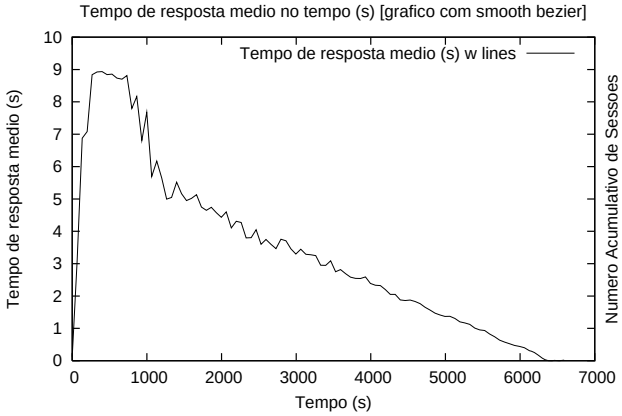
6.1.3.4 Avaliação de Políticas de Dois Níveis

Foi implementado o mecanismo de controle de admissão de dois níveis no simulador de qualidade de serviço *USAR-QoS*, sendo avaliadas as mesmas cargas utilizadas nos outros cenários. Foram avaliados outras configurações de políticas, mas apresentamos somente aquele que considera os resultados das seções **??** e 6.1.3.3, otimizando a abordagem de controle de admissão reativo. A tabela 6.3 e as figuras 6.5 e 6.6 apresentam os resultados obtidos.

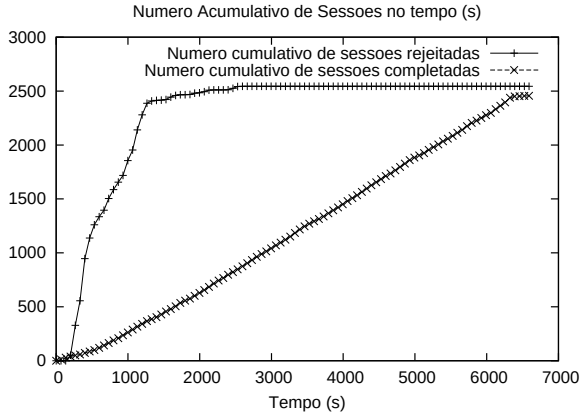
Podemos observar que o tempo de resposta máximo obtido foi de 7,22 segundos e o valor médio no intervalo de saturação de 5,70 segundos, que são equivalentes e até melhores do que os obtidos nos experimentos utilizando individualmente cada mecanismo reativo de controle de admissão de ações ou sessão. Para garantir esses tempos, houve a rejeição de 38,62% das ações e de 717 sessões.

Duração (s)	Max. R (s)	Médio R (s)	Ações requisitadas	Ações respondidas	Ações expiradas	Taxa de Perda	Sessões rejeitadas
9031	7,22	5,70	535.018	418.299	206.622	38,62%	717

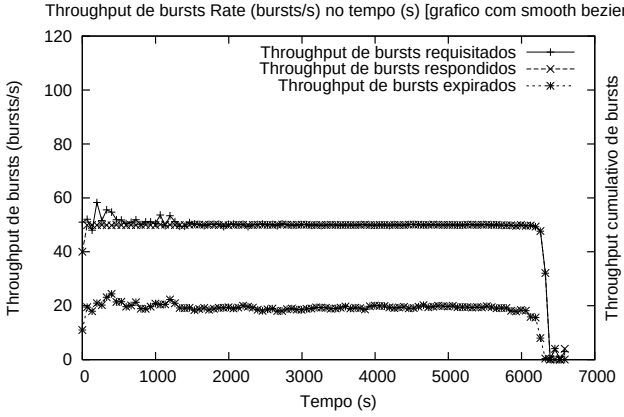
Tabela 6.3: Sumário do experimento para avaliar a abordagem de dois níveis



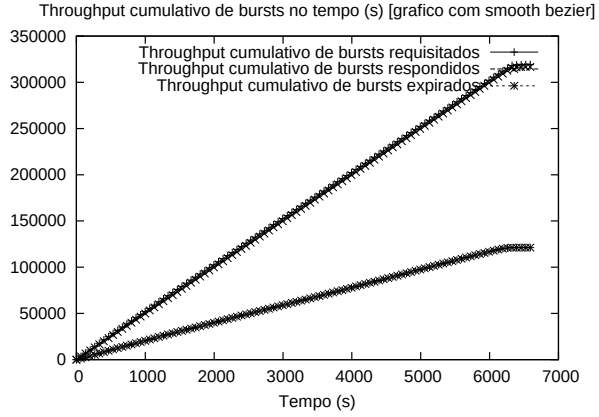
(a) Tempo de resposta médio



(b) Número cumulativo de sessões



(c) Throughput

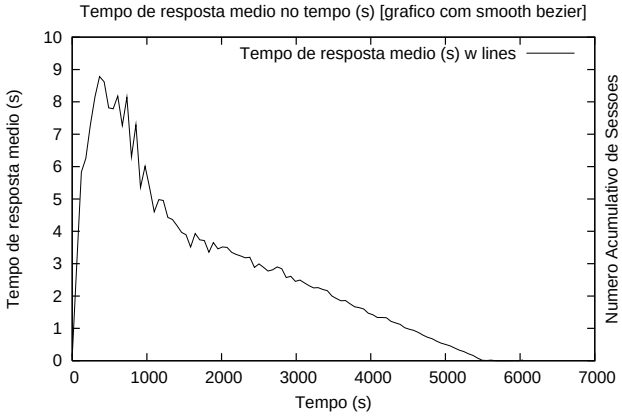


(d) Throughput cumulativo

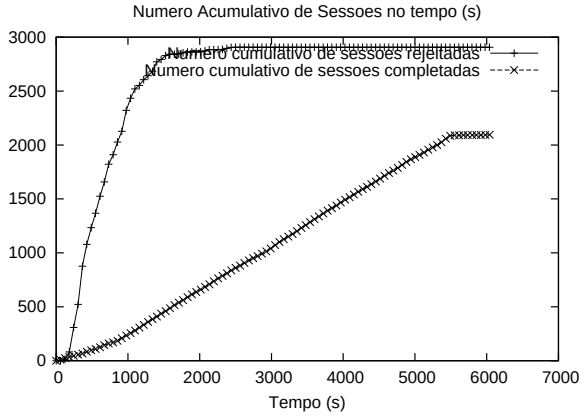
Figura 6.3: Abordagem não-reativa de controle de admissão baseada em sessões

Comparando esses valores com o experimento que utiliza a política de controle de admissão baseada em ações, vemos que o tempo de resposta apresenta um valor melhor e ainda com um número menor de ações rejeitadas. Comparando com o experimento que realiza o controle de admissão de sessões, o tempo de resposta máximo é menor, mas o valor médio no período de saturação é ligeiramente maior. Entretanto, o número de sessões rejeitadas é significativamente mais baixo.

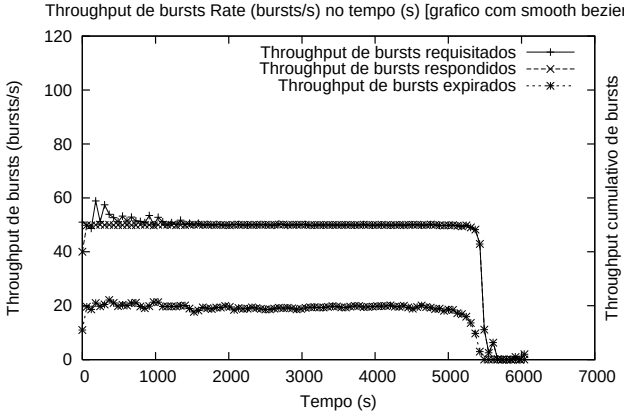
Sendo assim, a abordagem de dois níveis apresenta um cenário melhor, já que o tempo de resposta foi mantido e um número menor de rejeições foi necessário.



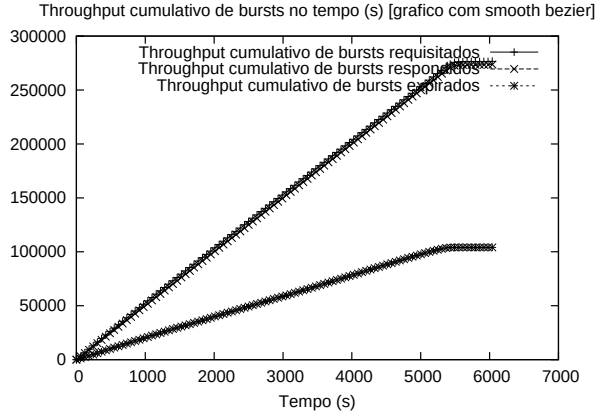
(a) Tempo de resposta médio



(b) Número cumulativo de sessões



(c) Throughput



(d) Throughput cumulativo

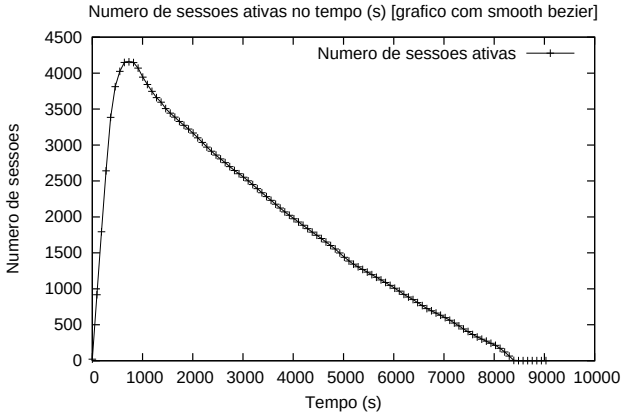
Figura 6.4: Abordagem reativa de controle de admissão baseada em sessões

6.1.4 Sumário

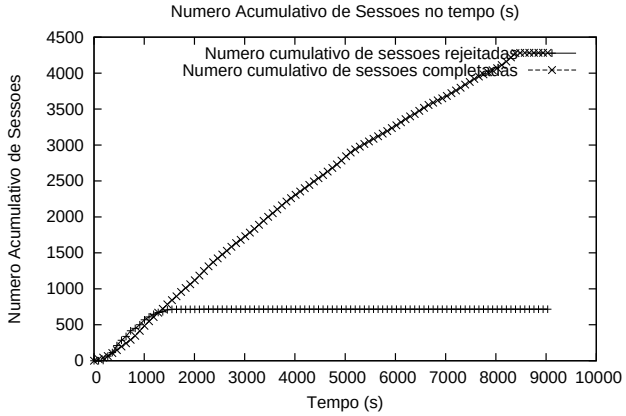
Esta seção apresentou as novas técnicas de controle de admissão baseadas em conhecimento sobre a reatividade dos usuários para prover um mecanismo de garantia de Qualidade de Serviço para serviços Internet. As novas políticas foram implementadas no simulador *USAR-QoS* que permitiu a avaliação das políticas de acordo com diferentes métricas, tornando possível comparar os resultados obtidos.

Os resultados mostram que a abordagem de rejeição de ações reativa é mais eficiente do que a abordagem não-reativa, já que é capaz de manter um valor aceitável de tempo de resposta sem provocar aumento significativo na taxa de perda de requisições.

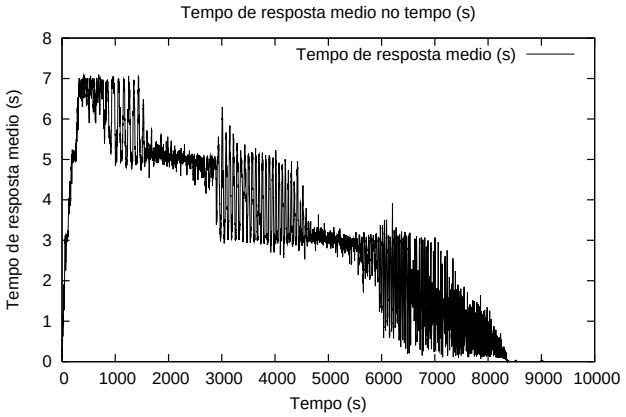
Podemos verificar também que a abordagem de rejeição de sessões também é eficiente em



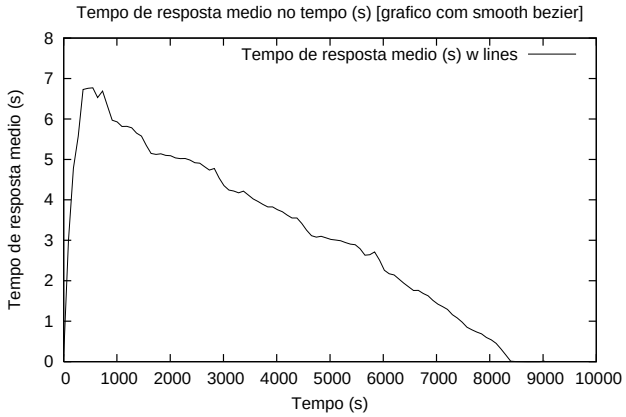
(a) Número de sessões



(b) Número cumulativo de sessões



(c) Tempo de resposta médio

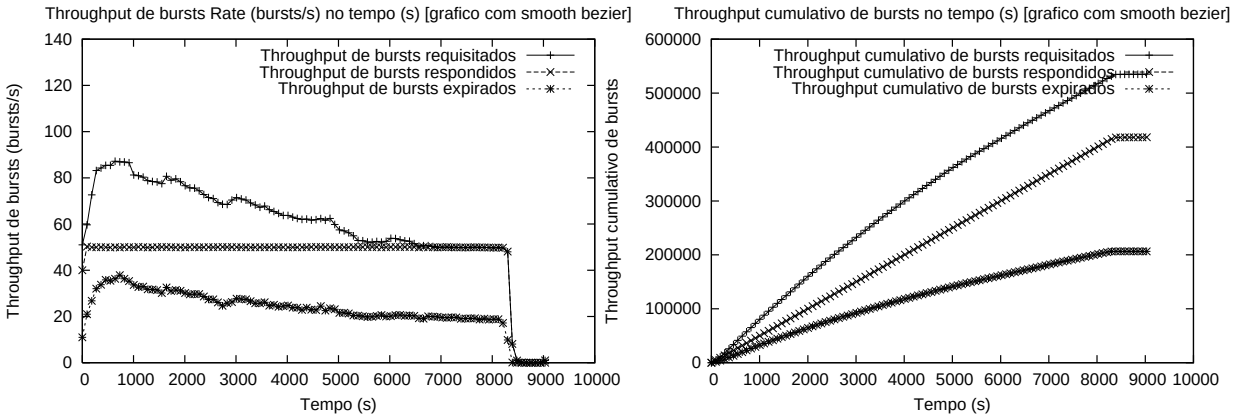


(d) Tempo de resposta médio com [smooth bezier]

Figura 6.5: Abordagem reativa dois níveis de controle de admissão

manter o tempo de resposta, até mesmo de modo mais eficiente do que para a abordagem de rejeição de ações. Entretanto, um grande número de sessões tem que ser rejeitadas, podendo gerar uma alta taxa de insatisfação por parte dos usuários.

Já a política de dois níveis tem a vantagem de ser capaz de mesclar as duas abordagens, equilibrando a rejeição de ações e sessões, e ao mesmo tempo, garantindo um bom tempo de resposta, como pode ser observado nos resultados. Apesar disto, o número de rejeições ainda é muito alto, e, deste modo, o estudo de métodos para reduzí-lo é importante. Com esta finalidade, na próxima seção, discutimos abordagens para prover QoS baseadas em mecanismos de escalonamento.



(a) Throughput (b) Throughput cumulativo

Figura 6.6: Abordagem reativa dois níveis de controle de admissão

6.2 Políticas Reativas de Escalonamento

Esta seção apresenta as novas estratégias de escalonamento baseadas em informações da reatividade dos usuários que propomos neste trabalho. Apresentamos também a avaliação delas utilizando o simulador *USAR-QoS*. A seção 6.2.1 introduz o tema do escalonamento, explicando o que é e como geralmente é implementado. A seção 6.2.2 apresenta as novas abordagens de escalonamento propostas. A seção 6.2.3 apresenta a metodologia e os resultados dos experimentos realizados. Finalmente, a seção 6.2.4 apresenta uma discussão geral sobre a avaliação experimental.

6.2.1 Escalonamento

Em um cenário de sobrecarga, um servidor Internet recebe um número muito alto de ações de serviço, que ele não consegue atender ao mesmo tempo. Tipicamente, ele irá utilizar um *buffer* ou uma fila para armazená-las à medida que chegam, para que aguardem a disponibilização dos recursos que a processarão. As requisições armazenadas na fila são tipicamente armazenadas em ordem de chegada. O servidor irá atender à primeira requisição da fila em um mecanismo de escalonamento conhecido como FIFO (de *First-In First-Out*).

A maioria dos servidores Internet provêm serviços de acordo com o modelo de melhor-

esforço, utilizando o escalonamento FIFO [Ye et al., 2005], mas, como apresentado no capítulo 3, diversos trabalhos propõem diferentes mecanismos para melhorar a abordagem de escalonamento FIFO. Esses trabalhos apresentam ganhos comparados com a abordagem de melhor-esforço, mas acreditamos que o modelo de reatividade pode ser útil para torná-las mais precisas e eficientes em situações de aumento de carga.

6.2.2 Novas políticas de Escalonamento

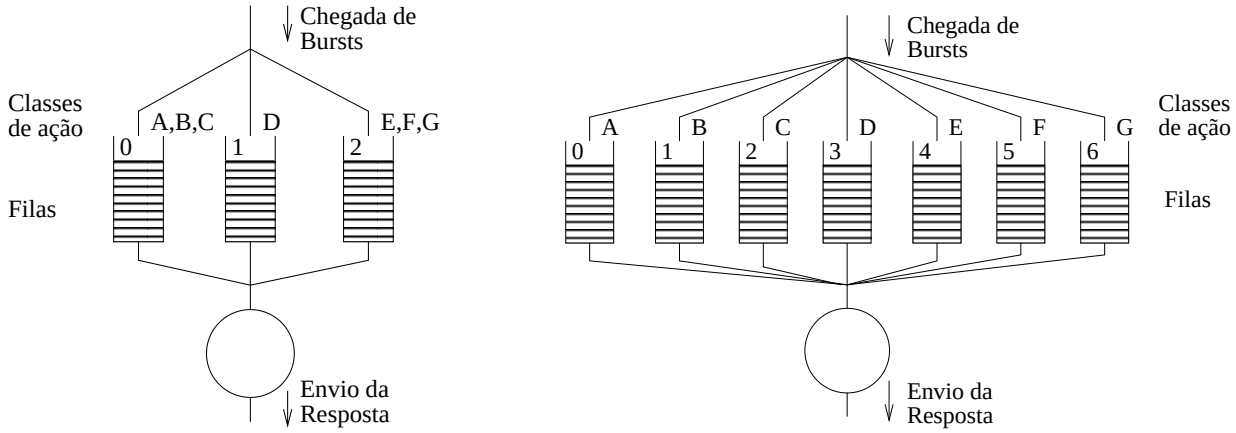
6.2.2.1 Abordagem de Escalonamento PFIN (*Patient-First Impatient-Next*)

Propomos a abordagem PFIN (*Patient-First Impatient-Next*, ou seja, Paciente-Primeiro Impaciente-Depois), onde as ações classificadas como sendo de classes de ação do tipo impaciente, ou seja, A, B ou C, são colocados em filas de baixa prioridade, e aquelas de classes do tipo paciente, para filas de alta prioridade.

A figura 6.7 apresenta duas variações da abordagem PFIN que se diferenciam em termos do número de filas de prioridade usadas para escalonar as ações. Em (a) está representada a abordagem com três filas de prioridade. Todas as ações identificadas com classes impacientes são escalonadas na fila de menor prioridade e aquelas classificadas com classes pacientes são direcionadas para a fila de maior prioridade. Assim, as ações das classes A, B e C vão para a fila q_0. Aquelas da classe D vão para a fila q_1. E, finalmente, aquelas das classes E, F e G vão para a fila q_2.

Em (b) são utilizadas sete filas, uma para cada classe de ação. As ações de classes mais impacientes vão para as filas de prioridade mais baixa. Assim, ações classificadas como A vão para a fila q_1, como B para a q_2, e assim por diante até a classe F que vão para a fila q_7.

Esse mecanismo visa priorizar o comportamento paciente baseado na idéia de que quando a carga do servidor aumenta, os usuários que tem um comportamento mais paciente tendem a reagir de modo menos intenso, demorando mais tempo para fazer novas ações, e, assim, fazendo com que haja uma redução da carga no servidor. Além disso, visa



(a) Três filas de prioridade (b) Sete filas de prioridade

Figura 6.7: Servidor implementando a abordagem de escalonamento PFIN

não prejudicar ou até aumentar os níveis de satisfação desses clientes. Em situações de maior carga, essa política pode não ser efetiva, pois caso muitas requisições de usuários impacientes estiverem na fila, um aumento da carga poderá ocorrer, coibindo os efeitos positivos dessa política. Assim, uma avaliação cuidadosa é necessária para verificar se essa política é adequada em cada cenário de carga.

6.2.2.2 Abordagem de Escalonamento IFPN (*Impatient-First Patient-Next*)

Na abordagem IFPN (*Impatient-First Patient-Next*, ou seja, Impaciente-Primeiro Paciente-Depois), as ações classificadas como como sendo de classes impacientes, ou seja, A, B, ou C vão para as filas de mais alta prioridade, e aquelas com classes pacientes, E, F e G, vão para as filas de menor prioridade. Assim como na abordagem PFIN, duas variações são propostas, com três e sete filas de prioridade.

Uma política do tipo IFPN se baseia na idéia de que quando a carga do servidor está aumentando, é melhor responder primeiramente aos usuários impacientes, para aumentar o seu nível de satisfação. A vantagem de atrasar respostas para os usuários com perfil paciente está no fato de que a satisfação deles tende a demorar um maior tempo para diminuir. Resumindo, essa abordagem tem a premissa de que, em situações de sobrecarga, usuários de comportamento paciente têm maior chance de esperar pelas respostas às suas

ações.

6.2.3 Avaliação Experimental

Na seção 6.2.3.1 apresentamos a metodologia de avaliação das novas políticas de escalonamento. Na seção 6.2.3.2 apresentamos os experimentos simulando a abordagem PFIN, e na seção 6.2.3.3 discutimos as abordagens IFPN.

6.2.3.1 Metodologia

No intuito de avaliar as novas políticas de escalonamento baseadas em informações sobre a reatividade, foram realizados diversos experimentos utilizando o simulador de Qualidade de Serviço *USAR-QoS*, onde as novas políticas foram implementadas.

Além da carga mista utilizada no experimento da seção 5.2, foram simulados outros cenários para observar como o comportamento do servidor seria afetado. Esses cenários utilizam cargas com diferentes distribuições de classes de ação. Apresentamos, entretanto, somente o cenário contendo a carga utilizada no experimento anterior, já que os resultados obtidos são suficientes para uma análise sobre os cenários avaliados. Relembrando, a carga era composta de 5000 sessões que seriam criadas a uma taxa média de 10 sessões por segundo. É importante observar que o experimento apresentado na seção 5.2 utiliza a abordagem típica de escalonamento de melhor-esforço FIFO, utilizando somente uma fila de requisições, que servirá de base para avaliarmos as novas políticas.

6.2.3.2 Avaliação da Abordagem PFIN

As tabelas 6.4, 6.5 e 6.6 e as figuras 6.8 e 6.9 apresentam os experimentos que avaliam as políticas de controle de escalonamento PFIN com três e sete filas. Nas figuras observamos os gráficos do tempo de resposta (a) e (b) e o throughput e throughput cumulativo em (c) e (d), respectivamente.

Abordagem de escalonamento	Duração (s)	Máximo R (s)	Médio R (s)	Ações requisitadas	Ações expiradas	Taxa de Perda
3 filas	13.002	84,07	12,45	621.342	157.537	25,35%
7 filas	13.252	338,62	13,66	621.342	100.083	16,11%

Tabela 6.4: Sumário dos experimentos de escalonamento PFIN

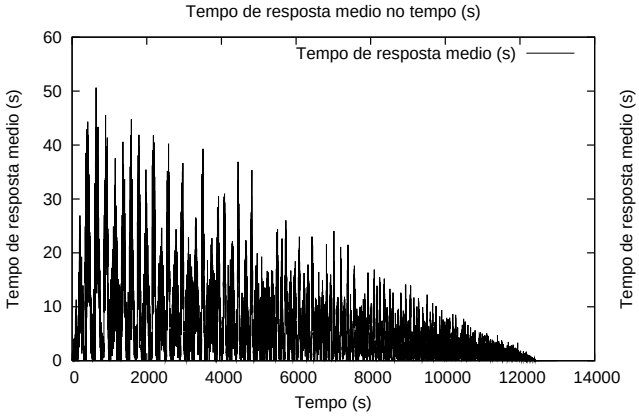
Classes de ação	Máximo R (s)	Médio R (s)	Taxa de Perda
Todas	84,07	12,45	25,35%
A, B e C	84,07	39,21	99,50%
D	20,26	0,74	2,95%
E, F e G	0,79	0,05	0,00%

Tabela 6.5: Dados por classe de ação para escalonamento PFIN com 3 filas de prioridade

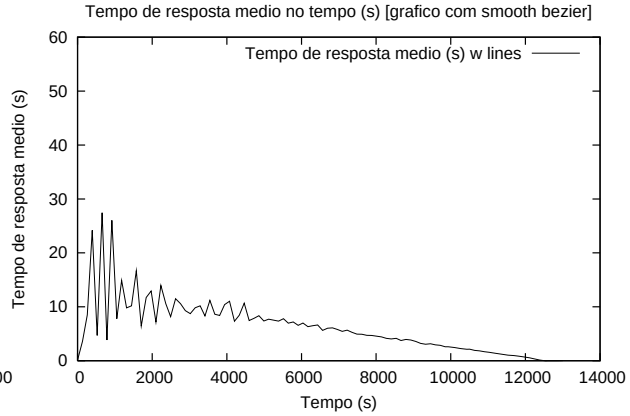
Para o experimento com três filas de prioridade, o tempo de resposta máximo chegou a 84,07 segundos e o valor médio no intervalo de saturação foi de 12,45 segundos. Já a taxa de perda de ações foi de 25,35%, o que representa um resultado melhor para esta métrica em comparação com os experimentos anteriores que utilizam controle de admissão e o experimento básico que utiliza a política de escalonamento FIFO apresentado na seção 5.2. Sendo assim, esse experimento apresenta um bom resultado, já que consegue um bom valor de tempo de resposta médio, com uma taxa de perda de ações baixa.

A tabela 6.5 apresenta os valores do tempo de resposta e da taxa de expiração para cada conjunto de ações classificadas por classe de ação. Como podemos observar, tanto o tempo de resposta como a taxa de perda são maiores para as classes com comportamento mais impaciente, diminuindo para as classes mais pacientes. Isto ocorre devido ao mecanismo de escalonamento que considera três filas e prioriza as classes pacientes.

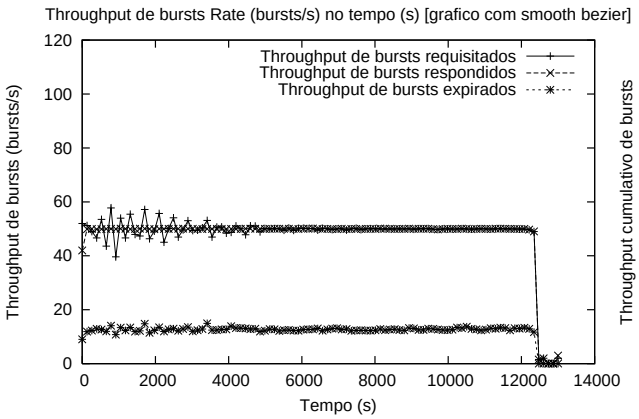
Para o experimento com sete filas de prioridade observamos um valor muito alto para o tempo de resposta máximo atingido, que chegou a 338,62 segundos. Em compensação, o valor médio no intervalo de saturação foi de 13,66 segundos, um valor praticamente idêntico



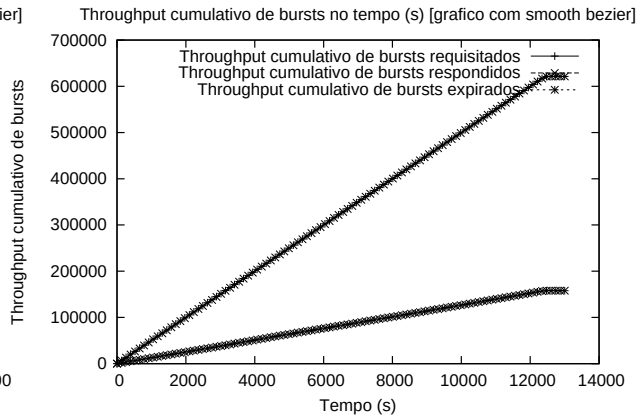
(a) Tempo de resposta médio



(b) Tempo de resposta como função *smooth*



(c) Throughput

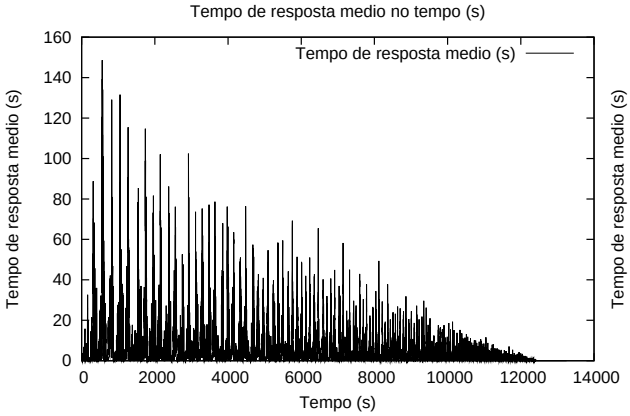


(d) Throughput cumulativo

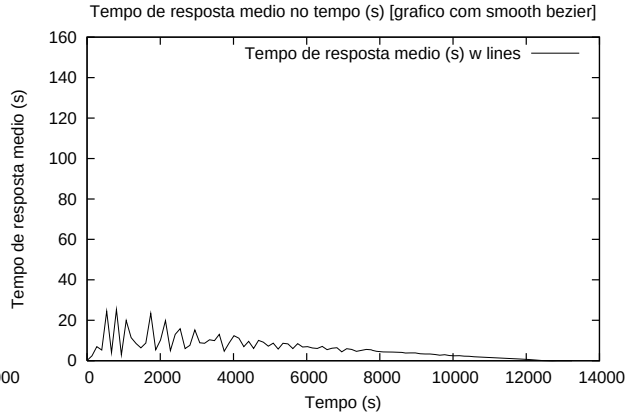
Figura 6.8: Abordagem de escalonamento PFIN com 3 filas de prioridade

ao correspondente no experimento com três filas de prioridade. Já a taxa de perda foi de 16,11%, um valor menor em comparação com as demais políticas mostradas anteriormente. Sendo assim, esse experimento apresenta um bom resultado, já que consegue um bom valor de tempo de resposta médio, com uma taxa de expiração baixa.

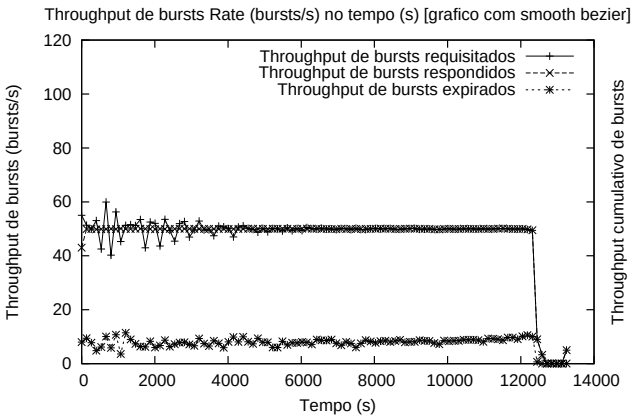
Analisando os motivos para o comportamento observado, apresentamos a tabela 6.6 contendo os valores de tempo de resposta e taxa de perda para as ações classificadas com cada classe de ação. Como pode ser observado, as classes de comportamento mais impaciente apresentam valores maiores de tempo de resposta e taxa de expiração, pois essas ações são direcionadas para as filas de mais baixa prioridade. Já as classes mais pacientes apresentam valores bem inferiores, já que o mecanismo de escalonamento lhes dá



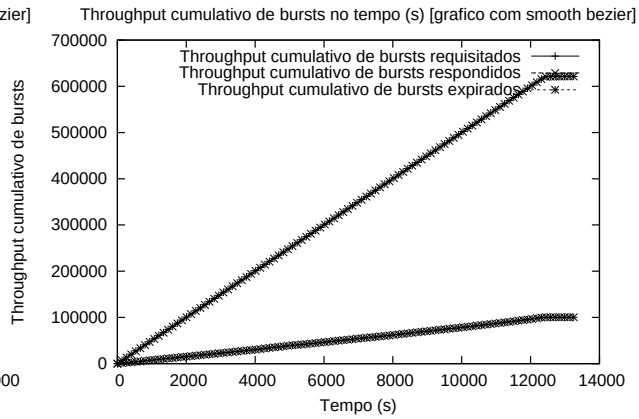
(a) Tempo de resposta médio



(b) Tempo de resposta com função *smooth*



(c) Throughput



(d) Throughput cumulativo

Figura 6.9: Abordagem de escalonamento PFIN com 7 filas de prioridade

prioridade.

6.2.3.3 Avaliação da Abordagem IFPN

Os experimentos que avaliam a abordagem de escalonamento IFPN com três e sete filas estão apresentados nas tabelas 6.7, 6.8 e 6.9 e nas figuras 6.10 e 6.11. Nas figuras observamos, do mesmo modo como explicado na seção anterior, as curvas do tempo de resposta e o throughput.

O experimento que avalia a política IFPN com três filas de prioridade apresenta um tempo de resposta máximo de 44,37 segundos e médio no intervalo de saturação de 11,90

Classes de ação	Máximo R (s)	Médio R (s)	Taxa de Perda
Todas	338,62	13,66	16,11%
A	338,62	42,87	99,63%
B	107,55	14,03	76,21%
C	60,68	3,57	30,55%
D	18,19	0,68	2,75%
E	1,90	0,08	0,04%
F	0,52	0,05	0,00%
G	0,22	0,03	0,00%

Tabela 6.6: Dados por classe de ação para escalonamento PFIN com 7 filas de prioridade

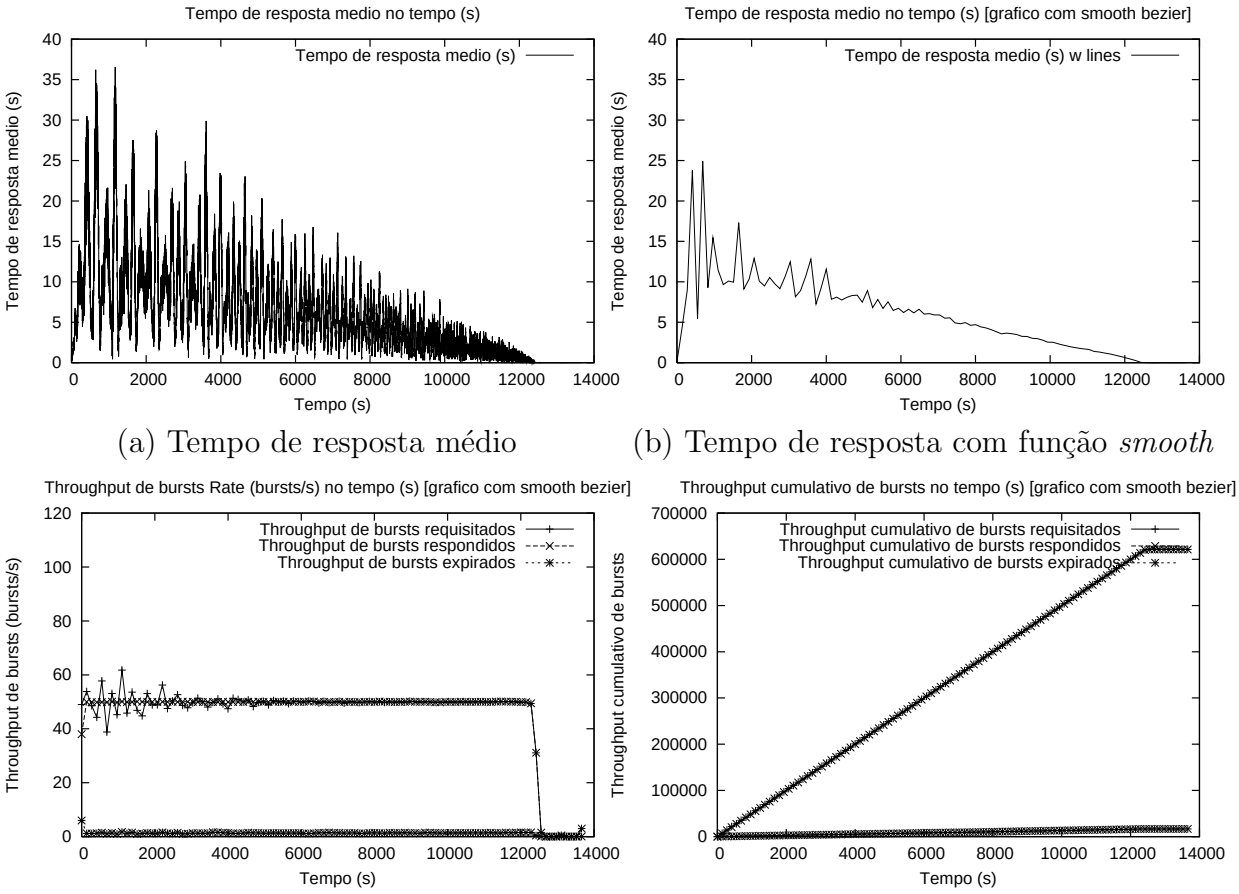
Abordagem de escalonamento	Duração (s)	Máximo R (s)	Médio R (s)	Ações requisitadas	Ações expiradas	Taxa de Perda
3 filas	13.660	44,37	11,90	621.342	16.500	2,66%
7 filas	13.132	73,21	12,67	621.342	3.412	0,55%

Tabela 6.7: Sumário dos experimentos de escalonamento IFPN

segundos. A taxa de perda de ações foi de 2,66%. Estes valores são menores do que aqueles obtidos para os experimentos que avaliam a abordagem de escalonamento PFIN.

A tabela 6.8 apresenta os valores do tempo de resposta e taxa de perda para os grupos de classes utilizados para o escalonamento. Observamos um maior tempo de resposta para as classes mais pacientes, uma vez que elas tem menor prioridade. Com essa política, a taxa de perda para os usuários com comportamento impaciente foi próxima de 0%, visto que o mecanismo de escalonamento foi capaz de atender com muita eficiência a esses usuários, aumentando a sua satisfação.

O experimento com sete filas apresentou um tempo de resposta máximo de 73,21 e médio no intervalo de saturação de 12,67. Entretanto, a taxa de perda apresentou um valor muito pequeno, já que foi de somente 0,55%. Isto quer dizer que o número de ações em que o usuário apresentou um comportamento impaciente foi de somente 0,55% das ações, o que demonstra a efetividade dessa abordagem de escalonamento.



(a) Tempo de resposta médio (b) Tempo de resposta com função *smooth*

(c) Throughput (d) Throughput cumulativo

Figura 6.10: Abordagem de escalonamento IFPN com 3 filas de prioridade

O comportamento de cada classe pode ser visto na tabela 6.9 onde podemos observar que, para as classes de A a F, a taxa de perda foi próxima de 0%. Já o tempo de resposta é maior para as classes pacientes, pois elas têm menor prioridade. Como efeito da reatividade, a taxa de perda não é muito grande, pois esses usuários têm tendência de comportamento paciente, esperando pela resposta do servidor.

6.2.4 Sumário

Este capítulo analisou o uso de informações sobre o comportamento reativo dos usuários para a estruturação de políticas de escalonamento mais eficientes em termos de Qualidade de Serviço. Foram propostas duas abordagens de escalonamento que priorizam o compor-

Classes de ação	Máximo R (s)	Médio R (s)	Taxa de Perda
Todas	44,37	11,90	2,66%
A, B e C	0,19	0,03	0,00%
D	0,51	0,04	0,02%
E, F e G	44,37	19,52	4,43%

Tabela 6.8: Dados por classe de ação para escalonamento IFPN com 3 filas de prioridade

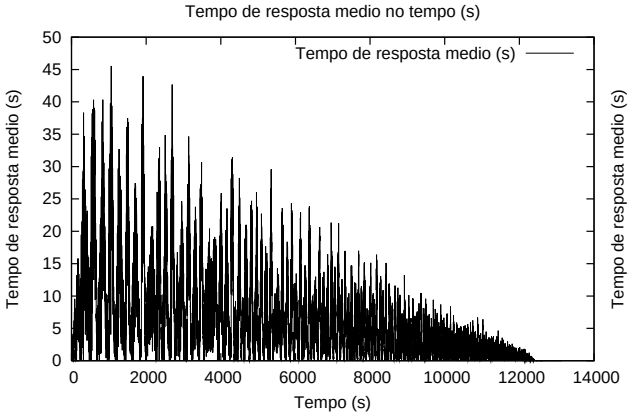
Classes de ação	Máximo R (s)	Médio R (s)	Taxa de Perda
Todas	73,21	12,67	0,55%
A	0,11	0,03	0,00%
B	0,17	0,03	0,00%
C	0,21	0,04	0,00%
D	0,42	0,04	0,01%
E	2,38	0,06	0,02%
F	11,80	0,29	0,02%
G	73,21	36,33	1,80%

Tabela 6.9: Dados por classe de ação para escalonamento IFPN com 7 filas de prioridade

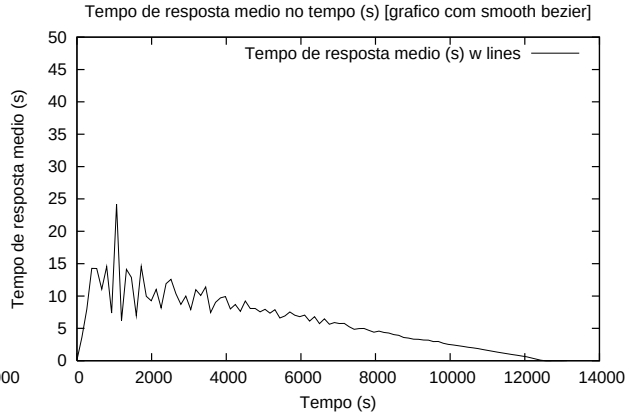
tamento impaciente, como é o caso da abordagem IFPN, e que priorizam o comportamento paciente, como no escalonamento PFIN. Para cada uma das abordagens foi discutido o uso de três ou sete filas de prioridade.

Cada uma das abordagens propostas foram avaliadas através do simulador *USAR-QoS* utilizando a mesma carga TPC-W composta de 5000 sessões, apresentada no capítulo 5. Os resultados foram comparados com os dados obtidos para o experimento básico que utiliza o escalonamento típico FIFO, e com os experimentos que utilizam controle de admissão.

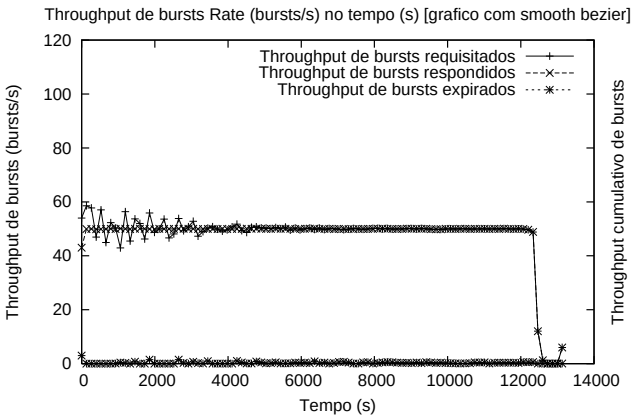
Podemos observar que os experimentos que utilizam as políticas propostas de escalonamento apresentam uma taxa de perda de ações muito baixa, devido ao mecanismo de escalonamento que prioriza certos tipos de comportamento. O tempo de resposta médio apresentou valores baixos, mas maiores do que aqueles obtidos para os experimentos que utilizam controle de admissão. Já os valores do tempo de resposta máximo foram muito al-



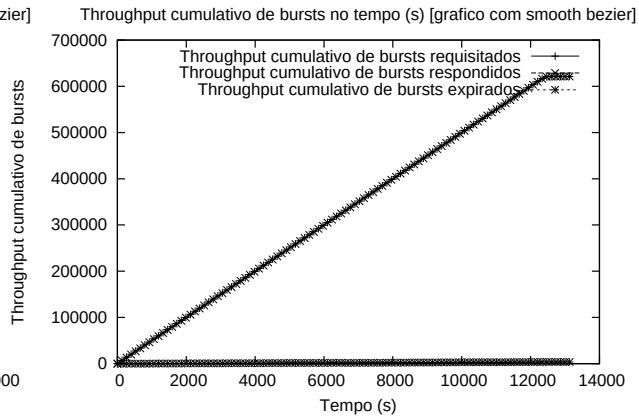
(a) Tempo de resposta médio



(b) Tempo de resposta com função *smooth*



(c) Throughput



(d) Throughput cumulativo

Figura 6.11: Abordagem de escalonamento IFPN com 7 filas de prioridade

tos, havendo inclusive possibilidade de que ocorra *starvation*, ou seja, processos que nunca são atendidos, esperando indefinidamente em suas filas. Este problema não será tratado neste trabalho.

Como os experimentos que utilizam controle de admissão são eficientes para controlar e manter o tempo de resposta em bons níveis e as abordagens que utilizam escalonamento são eficientes para diminuir as taxas de perda de ações, decidimos mesclar essas políticas em uma única abordagem que discutimos na próxima seção.

6.3 Combinação de Estratégias Reativas de Qualidade de Serviço

Esta seção apresenta uma abordagem para prover Qualidade de Serviço que combina as estratégias reativas de controle de admissão e escalonamento, de acordo com um conjunto de critérios que procura reduzir os pontos negativos e reforçar os positivos de cada política.

A seção 6.3.1 apresenta a estratégia proposta para combinar as políticas de controle de admissão e escalonamento em uma única solução. A seção 6.3.2 apresenta a metodologia e os experimentos realizados. A seção 6.3.3 sumariza o capítulo e discute os resultados.

6.3.1 Combinando Estratégias de Controle de Admissão e Escalonamento

Propomos uma abordagem híbrida, constituída de três níveis de tratamento, que combina as duas estratégias de controle de admissão e a abordagem de escalonamento. Como visto na seção 6.1, o mecanismo reativo de controle de admissão é eficiente para controlar o tempo de resposta, mas a taxa de expiração de requisições permanece alta. Já a seção 6.2 conclui que as estratégias de escalonamento são eficientes para reduzir a taxa de expiração de requisições, mas o tempo de resposta pode atingir valores mais altos.

Como visto na seção 6.1, a abordagem de controle de admissão de sessões realiza a rejeição de sessões drasticamente, quando o tempo de resposta aumenta. Na abordagem de dois níveis, vista na mesma seção, a rejeição de ações é iniciada previamente, antes da rejeição por sessões, o que pode reduzir um pouco o efeito negativo da rejeição de sessões. Na abordagem híbrida que apresentamos, o escalonamento é utilizado em paralelo, produzindo uma ordenação das ações, que pode reduzir o número de sessões com usuários insatisfeitos. Assim, definimos as seguintes regras:

- $R > \gamma$: ativar o mecanismo de escalonamento IFPN ou PFIN.
- $\alpha_1 \leq R < \beta_1$: rejeitar ações associadas às classes de ação A, B e C.

- $\beta_1 \leq R < \theta_1$: rejeitar ações associadas às classes de ação A, B, C e D.
- $R \geq \theta_1$: rejeitar ações associadas a todas as classes de ação.
- $\alpha_2 \leq R < \beta_2$: rejeitar todas as ações de sessões classificadas como de classes de ação impacientes, $USC < 4$, ou seja, sessões associadas às classes de ação A, B e C.
- $\beta_2 \leq R < \theta_2$: rejeitar todas as ações de sessões classificadas com $USC < 5$, ou seja, associadas às classe de ação A, B C, e D.
- $R \geq \theta_2$: rejeitar todas as sessões.

Este trabalho sugere que haja a seguinte relação das variáveis acima $\gamma < \alpha_1 < \alpha_2$, o que permitirá que o escalonamento seja iniciado antes da rejeição de ações que por sua vez será iniciado antes da rejeição de sessões. Com isto é possível aumentar o benefício de cada abordagem, reduzindo seus efeitos negativos, e obtendo uma solução mais robusta para prover Qualidade de Serviço.

6.3.2 Avaliação Experimental

6.3.2.1 Metodologia

O simulador *USAR-QoS* foi utilizado para avaliar o mecanismo que combina as estratégias reativas de controle de admissão e escalonamento em abordagem híbrida. As políticas reativas propostas foram implementadas e os valores para as variáveis α, β e θ de cada política foram escolhidos cuidadosamente. O método para escolha dos valores dessas variáveis foi a análise dos resultados do experimento apresentado na seção 5.2, que descreve um cenário simulado, utilizando uma carga de composição mista em termos das classes de ação, que não utilizou nenhuma política de controle de admissão ou escalonamento. Foram escolhidos os seguintes valores para esses parâmetros:

Além da carga mista utilizada no experimento da seção 5.2, foram simulados outros cenários para avaliar o comportamento do servidor. Apresentamos, entretanto, somente o cenário contendo a carga utilizada no experimento anterior, já que os resultados obtidos são

γ	α_1	β_1	θ_1	α_2	β_2	θ_2
0,0	3,0	5,0	7,0	5,0	7,0	9,0

Tabela 6.10: Parâmetros para avaliação da combinação das estratégias reativas

suficientes para efetuarmos uma análise conclusiva sobre os cenários avaliados. Finalmente, é preciso mencionar que não apresentamos os resultados para as abordagens que combinam controle de admissão com a abordagem PFIN, pois como visto na seção 6.2, a abordagem IFPN apresenta melhores resultados.

6.3.2.2 Resultados

A tabela 6.11 e as figuras 6.12 e 6.13 apresentam os experimentos que utilizam a abordagem de controle de QoS que combina as estratégias de controle de admissão e escalonamento IFPN. As figuras apresentam os gráficos do número total e cumulativo de sessões (a) e (b), o tempo de resposta (c) e (d) e o throughput (e) e (f).

O experimento utilizando o escalonamento IFPN com 3 filas de prioridade obteve um tempo de resposta máximo de 7,12 segundos e médio no intervalo de saturação de 5,71 segundos, uma taxa de perda de 23,33% e 640 sessões rejeitadas. Já o experimento que utiliza a política IFPN com 7 filas apresentou um tempo de resposta máximo de 15,13 segundos e médio de 5,75 segundos, com uma taxa de perda de ações de 22,09% e um número de 684 sessões rejeitadas.

Sendo assim, esses dois cenários apresentam resultados mais equilibrados, já que, apesar de o tempo de resposta não ser o melhor em comparação com os experimentos anteriormente apresentados, ele representa um valor aceitável, e ao mesmo tempo, a taxa de perda e o número de sessões rejeitadas apresentam valores baixos mas não ótimos em comparação com os demais experimentos.

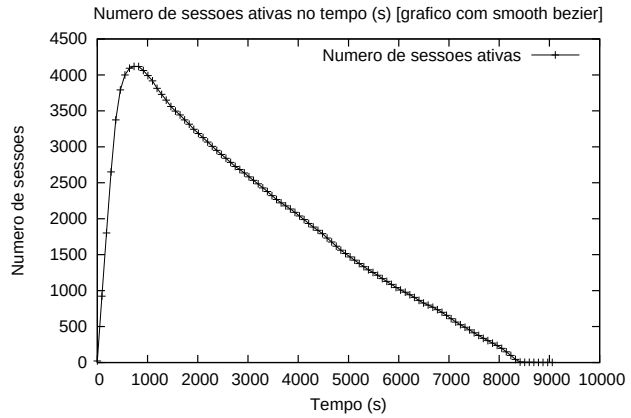
Abordagem	Duração (s)	Max R (s)	Médio R (s)	Bursts reqs.	Bursts resps.	Bursts exps.	Taxa de Perda	Sessões rejeitadas
IFPN 3 filas	8.910	7,12	5,71	544.078	423.679	126.925	23,33%	640
IFPN 7 filas	9.058	15,13	5,75	536.387	418.734	11.8484	22,09%	684

Tabela 6.11: Sumário do experimento para avaliar a combinação das estratégias

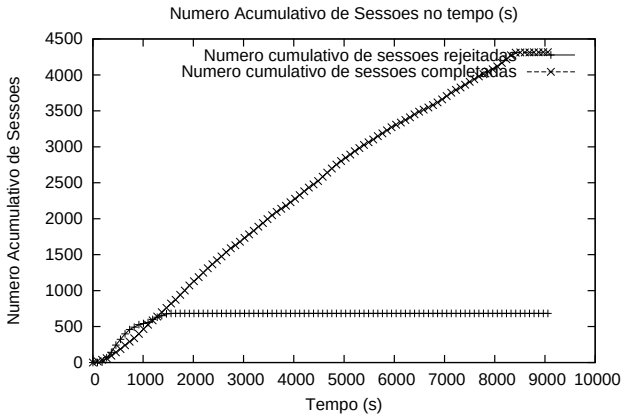
6.3.3 Sumário

A abordagem que combina as estratégias reativas de controle de admissão e escalonamento, apresentada nesta seção, constitui uma solução mais equilibrada para o controle de QoS já que é capaz de apresentar valores baixos para o tempo de resposta, a taxa de perda de *ações* e o número de sessões rejeitadas.

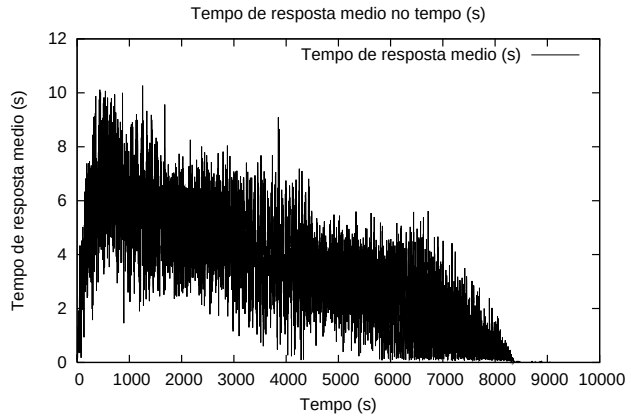
Como visto na seção 6.1, as políticas de controle de admissão que utilizam reatividade apresentam melhorias, mas a taxa de perda de ações atinge valores elevados. Já na seção 6.2, as novas políticas de escalonamento, em especial a abordagem IFPN, apresenta taxas de perda baixíssimas mas o tempo de resposta médio chega a ser mais do que duas vezes maior do que o valor atingido nas políticas de controle de admissão. Assim, a abordagem híbrida apresentada constitui uma solução mais adequada, pois, apesar de não apresentar os melhores valores individualmente, é capaz de prover uma melhor solução em conjunto, com ganhos em todos os aspectos.



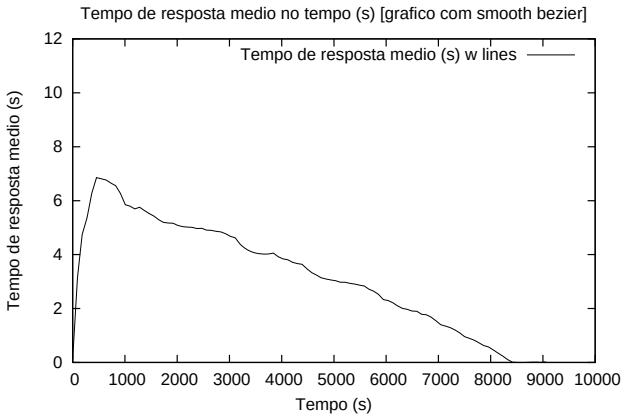
(a) Número de sessões



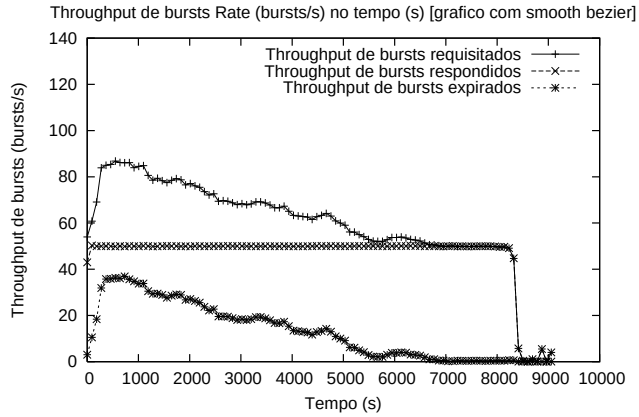
(b) Número cumulativo de sessões



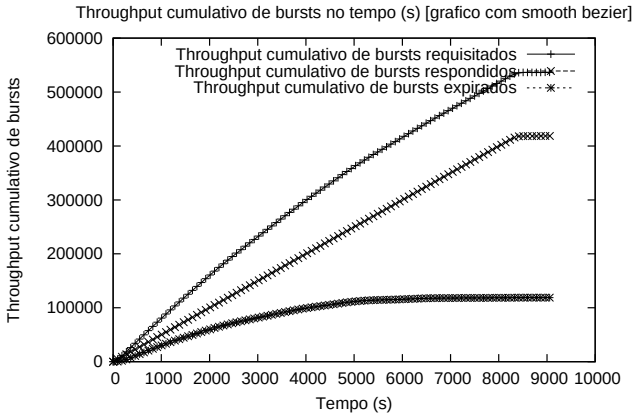
(c) Tempo de resposta médio



(d) Tempo de resposta médio com [smooth bezier]

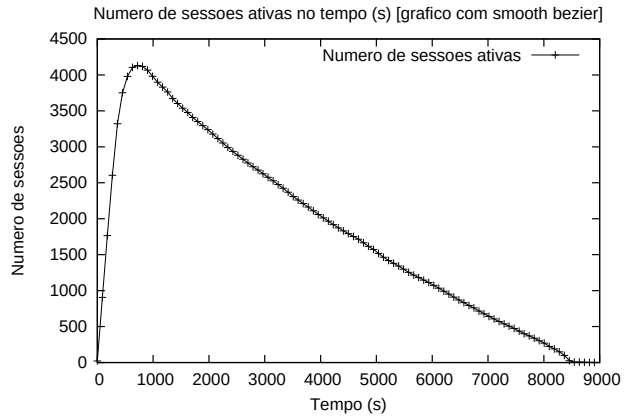


(e) Throughput

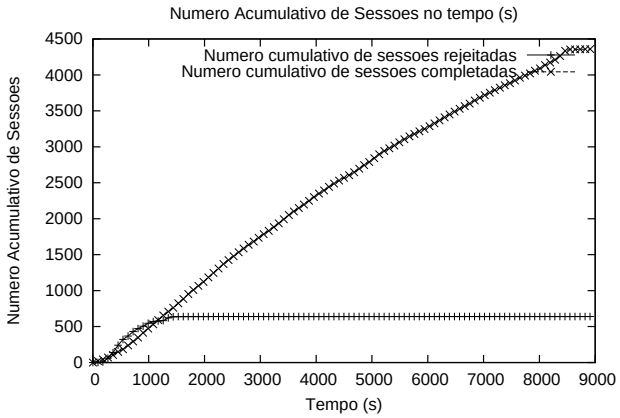


(f) Throughput cumulativo

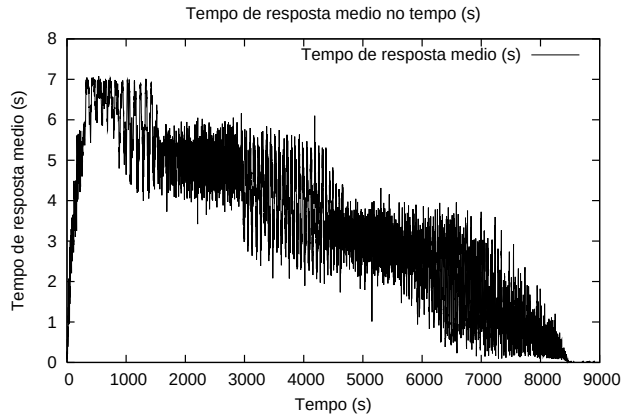
Figura 6.12: Experimento combinando abordagem reativa de controle de admissão e escalonamento IFPN com 7 filas



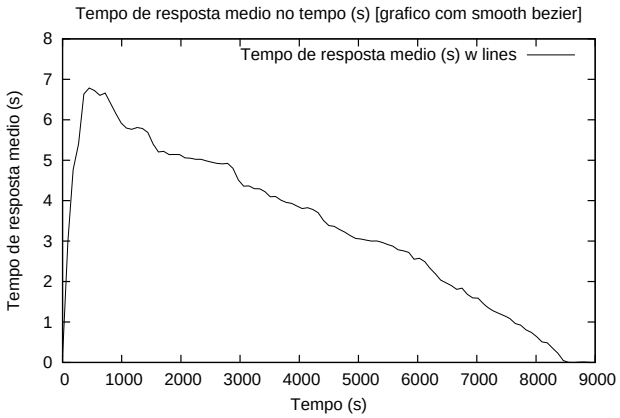
(a) Número de sessões



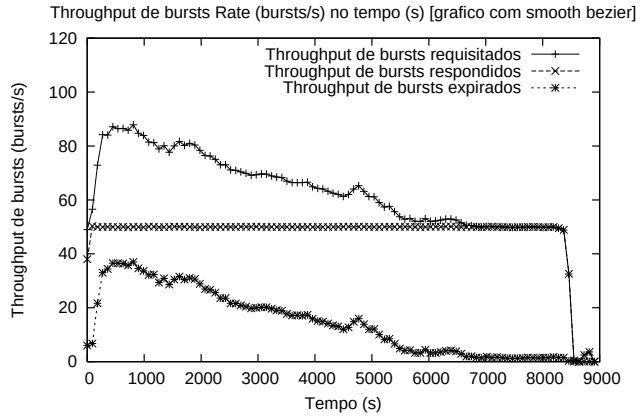
(b) Número cumulativo de sessões



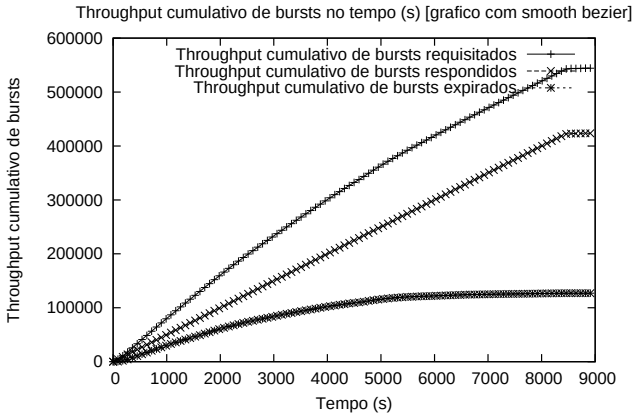
(c) Tempo de resposta médio



(d) Tempo de resposta médio com [smooth bezier]



(e) Throughput



(f) Throughput cumulativo

Figura 6.13: Experimento combinando abordagem reativa de controle de admissão e escalonamento IFPN com 3 filas

Capítulo 7

Conclusão

Devido ao grande aumento do número de usuários e às limitações técnicas da grande rede, requisitos como desempenho e escalabilidade se tornaram fundamentais, despertando o interesse da comunidade científica para o tema da qualidade de serviço (QoS) de aplicações Internet. Ao longo dos anos, diversos trabalhos foram propostos para garantir níveis aceitáveis de QoS, mas uma solução definitiva ainda não existe.

Sendo assim, um tópico ainda não completamente explorado diz respeito ao uso de informações sobre o comportamento dos usuários para melhorar o provimento de QoS. Em especial, o uso da *Reatividade*, ou seja, o mecanismo de ação e reação existente entre os usuários e o servidor, que poderia agregar informações importantes aos modelos atuais de desempenho.

Este trabalho tem a finalidade de apresentar uma visão completa sobre o uso da reatividade para prover Qualidade de Serviço em aplicações Internet. Deseja-se, sumariamente, identificar, modelar, reproduzir e avaliar o impacto da reatividade sobre servidores *Web*, além de propor mecanismos mais eficientes de QoS que a considerem.

Primeiramente, o trabalho discute a modelagem e replicação da reatividade. Para isto, foi apresentada a metodologia de caracterização de cargas *USAR*, mais detalhadamente discutida em [Franco, 2004] e o modelo de reatividade que permite a identificação da reatividade em uma carga de trabalho tradicional utilizando uma função que correlaciona

o tempo de resposta apresentado por um serviço e o IAT (de *inter-arrival time*, ou seja, o tempo entre duas requisições consecutivas de um mesmo usuário que são submetidas a um serviço). A aplicação desse modelo foi realizada em um estudo de caso onde foi caracterizada a carga de trabalho de um servidor *proxy-cache*.

Baseado no modelo de reatividade, o trabalho apresenta uma nova versão do gerador de cargas *httperf* que permite a execução de experimentos onde a dinâmica reativa e o ciclo de ação e reação entre servidor e cliente sejam considerados. A nova versão do gerador foi utilizada em um conjunto de experimentos utilizando um arquivo de *trace* configurado nos padrões do *benchmark TPC-W*. Esses experimentos avaliam diferentes cenários, demonstrando que uma carga com comportamento reativo apresenta uma dinâmica diferente, podendo influenciar a sua intensidade em comparação com uma carga com característica estática em termos da reação ao tempo de resposta percebido.

A partir disto abordamos o desafio de avaliar a viabilidade de utilizar o comportamento reativo para auxiliar a estruturação de mecanismos de controle de QoS mais eficientes. Para conduzir esta avaliação, este trabalho apresenta o simulador *USAR-QoS*, que permite a simulação de uma aplicação Internet composta de um conjunto de clientes acessando um servidor *Web*. Decidiu-se pela simulação devido à facilidade de experimentação e extensão para novas configurações.

Foram propostas novas estratégias de controle de admissão, baseadas na rejeição de bursts (um conjunto de requisições que representa a ação de um usuário, como o clique sobre um *link*) e sessões baseadas na reatividade. Verificamos que a abordagem reativa apresenta melhores resultados do que a abordagem não-reativa. Ambas são eficientes para manter o tempo de resposta dentro de certos limites, mas a abordagem reativa é mais efetiva em reduzir o número de requisições onde, devido à sua impaciência, o usuário realiza uma nova requisição antes de receber a resposta anterior, o que corresponde ao conceito de *taxa de perda*. A abordagem de rejeição de sessões apresentou um baixo tempo de resposta médio, mas, às custas da rejeição de um número muito grande de sessões. Assim, uma solução que permita uma diminuição dos tempos de resposta sem o aumento das taxas de perda e rejeição de sessões seria importante para a definição de uma estratégia

de Qualidade de Serviço mais robusta.

Com esta motivação o trabalho apresentou novas estratégias de escalonamento de requisições baseadas em informações sobre o comportamento reativo dos usuários. Foram propostas as políticas IFPN (de *Patient-First Impatient-Next*), que prioriza o atendimento de requisições de usuários com tendência de comportamento impaciente, e PFIN (de *Impatient-First Patient-Next*), que prioriza o comportamento paciente. Para ambas as políticas foram propostas duas abordagens, uma com três e outra com sete filas de prioridade. Os resultados apresentados demonstram que as melhores taxas de perda foram atingidas pelas abordagens IFPN com sete e três filas de prioridade, pois o atendimento de usuários com tendência impaciente permite o aumento da sua satisfação em termos das taxas de perda de bursts. Entretanto, os valores do tempo de resposta apresentam picos com valores muito altos e médias que, apesar de estarem dentro de limites aceitáveis, são mais do que duas vezes maior do que o valor obtido para o experimento com controle de admissão. Assim, as políticas de escalonamento apresentam um resultado excelente com relação às taxas de perda de requisições, mas o tempo de resposta apresenta valores muito altos.

Uma vez que a abordagem de controle de admissão é capaz de garantir um baixo tempo de resposta, mas produz uma alta taxa de perda, e a abordagem de escalonamento é eficiente para garantir uma baixa taxa de perda, mas com um tempo de resposta não tão eficiente, este trabalho propõe o uso de ambos os mecanismos em uma única solução híbrida, que permita otimizar a solução. Essa nova abordagem utiliza informações sobre a reatividade para balancear a rejeição de bursts e sessões e o escalonamento de bursts. Os resultados apresentados demonstram que os valores do tempo de resposta e das taxas de perda e rejeição apresentam valores mais equilibrados, formando uma melhor solução em conjunto, embora não atinjam os melhores valores individualmente. Assim, a abordagem que combina as estratégias se mostrou mais completa e eficiente para aplicações Internet.

É importante observar que as políticas propostas podem ser *injustas* (*unfair*) para com uma parte dos usuários, uma vez que o comportamento deles é identificado e classificado, e conforme a situação de carga do servidor, suas requisições podem ser rejeitadas, de modo

que esses usuários não serão atendidos. Assim, as políticas propostas atuam privilegiando comportamentos, o que por um lado é ruim, mas por outro pode ser uma boa forma de atenuar um problema de sobrecarga em um servidor. Uma possibilidade para atenuar o caráter *injusto* das políticas propostas é o uso de mecanismos do tipo WFQ, de *Weighted Fair Queuing* [Ferrari, 2000] que consiste um tópico a ser atingido futuramente por este trabalho.

Outro ponto a ser trabalho diz respeito à configuração e parametrização das constantes envolvidas no modelo de reatividade e nas políticas de QoS. Os valores utilizados neste trabalho se basearam em um estudo de caso específico, em inspeção visual e em referências da literatura. Um estudo mais aprofundado deve ser realizado, utilizando técnicas como análise de sensibilidade para refinar os valores.

Um dos problemas não tratados em profundidade pelo trabalho consiste da criação de um *classificador*, ou seja, da rotina que permita a um servidor Internet classificar suas requisições corretamente. Foi sugerido o uso de uma função baseada nas reações anteriores. Entretanto, este assunto não foi avaliado com maiores detalhes pelo trabalho. Além disto, não foi estimado o custo dessa classificação e do processamento das políticas de QoS, embora seja possível avaliar previamente que a quantidade de memória utilizada será da ordem de $O(n)$, onde n é o número de sessões a cada instante. Quanto à complexidade de tempo, para cada requisição haverá o custo de $O(1)$ para realizar a operação de atualização do classificador da respectiva sessão. Deste modo, para o total de requisições haverá um custo $O(m)$, onde m equivale ao número de requisições. Esse custo pode ser considerado muito pequeno, mas uma avaliação experimental não foi realizada.

O trabalho também não aborda a avaliação experimental em um ambiente real das novas políticas e a criação de um modelo analítico. Esses tópicos constituem trabalhos futuros.

Finalmente, pode-se concluir que a reatividade é uma dimensão importante de uma carga de trabalho, que pode ser modelada, replicada, avaliada e utilizada para a obtenção de políticas eficientes de QoS para aplicações Internet.

Bibliografia

[Spe, 1999] (1999). Specweb99 benchmark. `http://www.specbench.org/osg/web99/`.

[Web, 2002] (2002). Webbench 5.0 benchmark. `http://www.veritest.com/benchmarks/webbench/`.

[Abdelzaher and Bhatti, 1999a] Abdelzaher, T. F. and Bhatti, N. (1999a). Web content adaptation to improve server overload behavior. In *WWW '99: Proceeding of the eighth international conference on World Wide Web*, pages 1563–1577, New York, NY, USA. Elsevier North-Holland, Inc.

[Abdelzaher and Bhatti, 1999b] Abdelzaher, T. F. and Bhatti, N. (1999b). Web content adaptation to improve server overload behavior. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1563–1577.

[Aleksandrov and Voinov, 1998] Aleksandrov, M. and Voinov, V. (1998). Designing and implementing qos management of the web. In *CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, page 2. IBM Press.

[Arlitt and Williamson, 1997] Arlitt, M. F. and Williamson, C. L. (1997). Internet web servers: workload characterization and performance implications. *IEEE/ACM Trans. Netw.*, 5(5):631–645.

[Avritzer et al., 2002] Avritzer, A., Kondek, J., Liu, D., and Weyuker, E. J. (2002). Software performance testing based on workload characterization. In *WOSP '02: Proceedings of the third international workshop on Software and performance*, pages 17–24. ACM Press.

[Balachandran et al., 2002] Balachandran, A., Voelker, G., Bahl, P., and Rangan, P. (2002). Characterizing user behavior and network performance in a public wireless lan. In *Proceedings of ACM SIGMETRICS'02*.

[Barford and Crovella, 1998] Barford, P. and Crovella, M. (1998). Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 151–160. ACM Press.

[Bhatti et al., 2000] Bhatti, N., Bouch, A., and Kuchinsky, A. (2000). Integrating user-perceived quality into web server design. In *Proceedings of the 9th International World Wide Web Conference*.

[Bhatti and Friedrich, 1999] Bhatti, N. and Friedrich, R. (1999). Web server support for tiered services. *IEEE Network*, 13(5):64–71.

[Blake et al., 1998] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W. (1998). An architecture for differentiated services. Technical Report RFC 2475, IETF.

[Braden et al., 1994] Braden, R., Clark, D., and Shenker, S. (1994). Integrated services in the Internet architecture: an overview. Technical Report RFC 1633, IETF.

[Brataas and Hughes, 2004] Brataas, G. and Hughes, P. (2004). Exploring architectural scalability. *SIGSOFT Softw. Eng. Notes*, 29(1):125–129.

[Busari and Williamson, 2002] Busari, M. and Williamson, C. (2002). Prowgen: a synthetic workload generation tool for simulation evaluation of web proxy caches. *Comput. Networks*, 38(6):779–794.

[Carter and Cherkasova, 2000] Carter, R. and Cherkasova, L. (2000). Detecting timed-out client requests for avoiding livelock and improving web server performance. In *Proc. of 5th IEEE Symposium on Computers and Communications*, pages pp 2–7, Antibes, FRANCE. IEEE Computer Society.

[Chandra and Shenoy, 2003] Chandra, A. and Shenoy, P. (2003). Effectiveness of dynamic resource allocation for handling internet flash crowds. Technical Report TR03-37, Department of Computer Science, University of Massachusetts Amherst.

[Chatterjee et al., 1998] Chatterjee, P., Hoffman, D., and Novak, T. (1998). Modeling the clickstream: Implications for web-based advertising efforts. In Working Paper. Vanderbilt University, 1998.

[Chen and Mohapatra, 2003] Chen, H. and Mohapatra, P. (2003). Overload control in qos-aware web servers. *Compututer Networks*, 42(1):119–133.

[Chen et al., 2001] Chen, X., Mohapatra, P., and Chen, H. (2001). An admission control scheme for predictable server response time for web accesses. In *WWW '01: Proceedings of the tenth international conference on World Wide Web*, pages 545–554. ACM Press.

[Cherkasova et al., 2003] Cherkasova, L., Fu, Y., Tang, W., and Vahdat, A. (2003). Measuring and characterizing end-to-end internet service performance. *ACM Trans. Inter. Tech.*, 3(4):347–391.

[Cherkasova and Phaal, 1999] Cherkasova, L. and Phaal, P. (1999). Session-based admission control: A mechanism for improving performance of commercial web sites. In *Proc. Seventh Int'l Workshop Quality of Service*.

[Cherkasova and Phaal, 2000] Cherkasova, L. and Phaal, P. (2000). Predictive admission control strategy for overloaded commercial web server. In *MASCOTS '00: Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 500, Washington, DC, USA. IEEE Computer Society.

[Cherkasova and Phaal, 2002] Cherkasova, L. and Phaal, P. (2002). Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Trans. Comput.*, 51(6):669–685.

[Costa et al., 2004] Costa, C., Cunha, I., Borges, A., Ramos, C., Rocha, M., Almeida, J., and Ribeiro-Neto, B. (2004). Analyzing client interactivity in streaming media. In *Proceedings of the 13th World Wide Web Conference*.

[dos Santos, 1999] dos Santos, A. P. S. (1999). Qualidade de serviço na internet. *RNP (Rede Nacional de Ensino e Pesquisa) Boletim bimestral sobre tecnologia de redes*, 3(6).

[Elnikety et al., 2004] Elnikety, S., Nahum, E., Tracey, J., and Zwaenepoel, W. (2004). A method for transparent admission control and request scheduling in e-commerce web

sites. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 276–286. ACM Press.

[Feitelson, 2002] Feitelson, D. (2002). Workload modeling for performance evaluation. Lecture Notes in Computer Science. 2459:114–141.

[Ferrari, 2000] Ferrari, T. (2000). End-to-end performance analysis with traffic aggregation. *Computer Networks*, 34(6):905–914.

[Fishwick, 1992] Fishwick, P. A. (1992). Simpack: Getting started with simulation programming in c and c++. In *Winter Simulation Conference*, pages 154–162.

[Franco, 2004] Franco, G. (2004). Uma metodologia de caracterização de comportamento de usuários de serviços internet. Master's thesis, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, Minas Gerais, Brazil.

[Franco and Meira, Jr., 2004] Franco, G. and Meira, Jr., W. (2004). Usar: a user behavior characterization model. In *Proceedings of the PhD and MSc Workshop of the 2nd Latin American Web Congress and the 10th Brazilian Symposium on Multimedia and the Web (LA-WebMedia 2004)*, Ribeirão Preto, SP, Brazil. IEEE Computer Society.

[García and García, 2003] García, D. F. and García, J. (2003). Tpc-w e-commerce benchmark evaluation. *Computer*, 36(2):42–48.

[Garner, 1995] Garner, S. (1995). Weka: The waikato environment for knowledge analysis. In *Proceedings of the New Zealand Computer Science Research Students Conference*, pages 57–64.

[Henderson, 2001] Henderson, T. (2001). Latency and user behaviour on a multiplayer game server. In *Proceedings of the Third International COST264 Workshop on Networked Group Communication*, pages 1–13. Springer-Verlag.

[Hlavacs et al., 2000] Hlavacs, H., Hotop, E., and Kotsis, G. (2000). Workload generation by modeling user behavior. In *Proceedings of OPNETWORKS 2000*.

[Hlavacs and Kotsis, 1999] Hlavacs, H. and Kotsis, G. (1999). Modeling user behavior: A layered approach. In *MASCOTS*, pages 218–225.

[Iyengar et al., 1997] Iyengar, A., MacNair, E., and Nguyen, T. (1997). An analysis of web server performance. In *In Proceedings of GLOBECOM '97*, Phoenix, Arizona, USA.

[Jin and Bestavros, 2001] Jin, S. and Bestavros, A. (2001). Gismo: a generator of internet streaming media objects and workloads. *SIGMETRICS Perform. Eval. Rev.*, 29(3):2–10.

[Krishnamurthy and Wills, 2002] Krishnamurthy, B. and Wills, C. (2002). Improving web performance by client characterization driven server adaptation. In *WWW '02: Proceedings of the eleventh international conference on World Wide Web*, pages 305–316. ACM Press.

[Krishnamurthy et al., 2003] Krishnamurthy, B., Zhang, Y., Wills, C., and Vishwanath, K. (2003). Design, implementation, and evaluation of a client characterization driven web server. In *WWW'03: Proceedings of the twelfth international conference on World Wide Web*, pages 138–147. ACM Press.

[Lee et al., 2002] Lee, S., Lui, J., and Yau, D. (2002). Admission control and dynamic adaptation for a proportional delay diffserv-enabled web server. In *ACM SIGMETRICS Conference*, Marina del Rey, CA, USA.

[Meira, Jr. et al., 2000] Meira, Jr., W., Murta, C., and Resende, R. (2000). *Comércio eletrônico na World Wide Web*. IME-USP.

[Menascé et al., 2004] Menascé, D., Ruan, H., and Gomaa, H. (2004). A framework for qos-aware software components. In *WOSP '04: Proceedings of the fourth international workshop on Software and performance*, pages 186–196. ACM Press.

[Menascé et al., 2004] Menascé, D., Almeida, V., and Dowdy, L. (2004). *Performance by Design*. Prentice Hall.

[Menascé et al., 2003a] Menascé, D., Almeida, V., Riedi, R., Ribeiro, F., Fonseca, R., and Jr., W. M. (2003a). A hierarchical and multiscale approach to analyze e-business workloads. *Perform. Eval.*, 54(1):33–57.

[Menascé et al., 2000] Menascé, D., Almeida, V., Riedi, R., Ribeiro, F., Fonseca, R., and Meira, Jr., W. (2000). In search of invariants for e-business workloads. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 56–65. ACM Press.

[Menascé, 2001] Menascé, D. A. (2001). Testing e-commerce site scalability with tpc-w. In *Int. CMG Conference*, pages 457–466.

[Menascé et al., 2003b] Menascé, D. A., Almeida, V. A. F., Riedi, R., Ribeiro, F., Fonseca, R., and Meira, Jr., W. (2003b). A hierarchical and multiscale approach to analyze e-business workloads. *Perform. Eval.*, 54(1):33–57.

[Mosberger and Jin, 1998] Mosberger, D. and Jin, T. (1998). httperf–tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37.

[Nahum et al., 2001] Nahum, E. M., Rosu, M.-C., Seshan, S., and Almeida, J. (2001). The effects of wide-area conditions on www server performance. In *SIGMETRICS '01: Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 257–267. ACM Press.

[Olshefski et al., 2002] Olshefski, D., Nieh, J., and Agrawal, D. (2002). Inferring client response time at the web server. In *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 160–171. ACM Press.

[Pereira, 2005] Pereira, A. (2005). Caracterização da reatividade em sistemas distribuídos. Proposta de tese. Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brasil.

[Pereira et al., 2004a] Pereira, A., Franco, G., Silva, L., and Meira, Jr., W. (2004a). A hierarchical characterization of user behavior. In *Proceedings of the 2nd Latin American Web Congress and the 10th Brazilian Symposium on Multimedia and the Web (LA-WebMedia 2004)*, Ribeirão Preto, SP, Brazil. IEEE Computer Society.

[Pereira et al., 2004b] Pereira, A., Franco, G., Silva, L., Meira, Jr., W., and Santos, W. (2004b). The *usar* characterization model. In *Proceedings of the IEEE 7th Annual Workshop on Workload Characterization (WWC-7)*, Austin, Texas, USA. IEEE Computer Society.

[Pereira et al., 2006a] Pereira, A., Silva, L., and Meira, Jr., W. (2006a). Evaluating the impact of reactivity on the performance of web applications. In *25th IEEE International Performance Computing and Communications Conference (IPCCC 2006)*, Phoenix, Arizona, USA. IEEE Computer Society.

[Pereira et al., 2005] Pereira, A., Silva, L., Meira, Jr., W., and Santos, W. (2005). Assessing the impact of user reactivity on the performance of web applications. Technical

Report RT.DCC 007/2005, Department of Computer Science, Federal University of Minas Gerais, Brazil.

[Pereira et al., 2006b] Pereira, A., Silva, L., Meira, Jr., W., and Santos, W. (2006b). Assessing reactive qos strategies for internet services. In *Proceedings of the IEEE/IPSJ Symposium on Applications and the Internet (SAINT2006)*, Phoenix, Arizona, USA. IEEE Computer Society (IEEE-CS) and Information Processing Society of Japan (IPSJ).

[Pereira et al., 2006c] Pereira, A., Silva, L., Meira, Jr., W., and Santos, W. (2006c). Assessing the impact of reactive workloads on the performance of web applications. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2006)*, Austin, Texas, USA. IEEE Computer Society.

[Selvridge et al., 2000] Selvridge, P., Chaparro, B., and Bender, G. (2000). The world wide wait: Effects of delays on user performance. In *Proceedings of the IEA 2000/HFES 2000 Congress*.

[Silva et al., 2006] Silva, L., Pereira, A., and Meira, Jr., W. (2006). Reactivity-based scheduling approaches for internet services. In *Proceedings of the 4th Latin American Web Congress (LA-WebMedia 2004)*, Cholula, Puebla, Mexico. IEEE Computer Society. To appear.

[Silva et al., 2007] Silva, L., Pereira, A., and Meira, Jr., W. (2007). Reactivity-based quality of service strategies for web applications. In *Proceedings of the IEEE/IPSJ Symposium on Applications and the Internet (SAINT2007)*, Hiroshima, Japan. IEEE Computer Society (IEEE-CS) and Information Processing Society of Japan (IPSJ). To appear.

[Tang et al., 2003] Tang, W., Fu, Y., Cherkasova, L., and Vahdat, A. (2003). Medisyn: a synthetic streaming media service workload generator. In *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 12–21. ACM Press.

[Urgaonkar and Shenoy, 2005] Urgaonkar, B. and Shenoy, P. (2005). Cataclysm: Handling extreme overloads in internet applications. In *Proceedings of the Fourteenth International World Wide Web Conference (WWW 2005)*, Chiba, Japan.

[Walker, 1977] Walker, A. J. (1977). An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Softw.*, 3(3):253–256.

[Xiao and Ni, 1999] Xiao, X. and Ni, L. M. (1999). Internet qos: A big picture. *IEEE Network*, 13(2):8–18.

[Ye et al., 2005] Ye, N., Gel, E. S., Li, X., Farley, T., and Lai, Y.-C. (2005). Web server qos models: applying scheduling rules from production planning. *Computers & Operations Research*, 32:1147–1164.

[Zaki, 2001] Zaki, M. J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60.