Estudo de algoritmos de controle de admissão em servidores web distribuídos com QoS

Paulo Sérgio Franco Eustáquio

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura:_____

Estudo de algoritmos de controle de admissão em servidores web distribuídos com QoS

Paulo Sérgio Franco Eustáquio

Orientadora: **Profa. Dra. Sarita Mazzini Bruschi**

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

USP – São Carlos
Julho de 2011

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

E91e

Eustáquio, Paulo Sérgio Franco
Estudo de algoritmos de controle de admissão em servidores web distribuídos com QoS / Paulo Sérgio Franco Eustáquio; orientador Sarita Mazzini Bruschi -- São Carlos, 2011.
85 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2011.

1. Servidor Web. 2. Qualidade de Serviços. 3. Controle de Admissão. 4. Cluster. 5. Apache. I. Bruschi, Sarita Mazzini , orient. II. Título.

“Imagine uma nova história para sua vida e acredite nela.”
(Paulo Coelho)

Agradecimentos

Agradeço a Deus por ter concluído parte dos meus objetivos, aos meus pais José e Marli, que além do apoio psicológico me deram todo o suporte financeiro para a minha estadia nesta cidade.

Aos amigos e colegas de laboratório Ricardo Nogueira, Bruno Tardiole, Júlio Estrella, Bruno Guazzelli, Maycon Leone, Pedro Prado, Thiago Caproni, Douglas Rodrigues, Luis Nakamura, Alessandro Nakamuta, Fabiano Teixeira, Dionísio Leite, Lourenço Alves, Daniel Pigatto, Adriana, Rubens Kenji, Mário Machado, Rene Pinto, Pedro Nobile, Edwin Mamani e Roni Guilhermo que me ajudaram neste trabalho, tanto com idéias, detalhes técnicos e também pelos momentos de descontração.

Aos meus colegas de república Bruno Guazzelli, Gabriel Massote, Bruno Tardiole, Daniel Pigatto e João Paulo pelos ótimos momentos que passamos juntos e pelos fortes laços de amizades que construímos ao longo destes anos.

A Lívia, Lidiane e Nathália do Campus estendido da Rep. Tibilisku pelas noites de pão de queijo, festas e afins.

Agradeço ao meu irmão João Paulo pelos poucos e bons momentos que passamos juntos quando nos encontramos na casa de nossos pais.

Aos meus amigos de Ipuiúna.

O desenvolvimento deste trabalho não teve méritos exclusivos do autor. Por isso, agradeço à minha orientadora Sarita Bruschi, que me guiou ao longo desta jornada.

Aos professores Marcos Santana e Regina Santana pelas dicas fundamentais para a elaboração deste trabalho e principalmente ao Msc. Ricardo Nogueira, pois sem ele seria muito difícil a conclusão deste mestrado no tempo estipulado.

Agradeço à agência de fomento FAPESP pelo financiamento deste projeto.

Enfim, a todos aqueles que de alguma forma contribuíram para a realização deste trabalho.

Resumo

Esta dissertação de mestrado apresenta a prototipação de uma arquitetura denominada Servidor Web com Diferenciação de Serviços (SWDS). O trabalho também apresenta a proposta, implementação e a avaliação de desempenho de dois algoritmos de controle de admissão denominados Algoritmo de Negociação e Algoritmo de Reserva de Conexões. O objetivo principal deste trabalho é a avaliação de desempenho dos algoritmos de controle de admissão no protótipo do SWDS. Como resultados, verificou-se que a Classe de maior prioridade consegue atender mais requisições que a Classe de menor prioridade em todos as configurações de carga. Também pode-se observar que o algoritmo de reserva de conexões consegue se adaptar a todas as variações de cargas.

Abstract

This master's dissertation presents the prototype of an architecture named SWDS. This work presents the proposal, implementation and performance evaluation of two admission control algorithms named negotiation algorithm and connection reserve algorithm. The aim of this work is to evaluate the performance of the control admission algorithms in the SWDS prototype. As a result, it could be seen the high priority Class met a higher number of requests than the low priority Class. Also it could be seen the connection reserve algorithm can adapt to all workloads variation.

Sumário

Resumo **v**

Abstract **vii**

Lista de Siglas **xv**

1 Introdução **1**
1.1 Contextualização 1
1.2 Motivação e Objetivos 4

2 Qualidade de Serviços **7**
2.1 Considerações Iniciais 7
2.2 Qualidade de Serviços 7
2.3 QoS em Nível de Rede 8
2.3.1 Serviços Integrados 9
2.3.2 Serviços Diferenciados 10
2.4 QoS em Nível de Aplicação 11
2.5 Considerações Finais 12

3 Servidor Web com Diferenciação de Serviços (SWDS) **13**
3.1 Considerações Iniciais 13
3.2 Servidor Web 13
3.2.1 Apache 14
3.3 Servidor Web Distribuído 19
3.4 Modelo do Servidor SWDS 20
3.5 Mecanismos de Diferenciação de Serviços 22
3.5.1 Reserva de Recursos 22
3.5.2 Mecanismo de Prioridade Rigoroso 22
3.5.3 Mecanismo de Prioridade Adaptativo 23
3.5.4 Cenários de Utilização 23
3.6 Controle de Admissão 24
3.6.1 Mecanismos de Controle de Admissão 24
3.7 Implementação da Arquitetura SWDS 27
3.8 Considerações Finais 29

4 Algoritmos de Controle de Admissão Propostos **31**
4.1 Considerações iniciais 31
4.2 Algoritmo de Negociação 31
4.3 Algoritmo de Reserva de Conexões 33
4.4 Considerações Finais 34

5 Ambiente de Experimentos **37**
5.1 Considerações Iniciais 37
5.2 Ambiente de Experimentos 37
5.3 Configuração dos Algoritmos 39
5.4 Considerações Finais 47

6 Análise de Resultados **49**
6.1 Considerações Iniciais 49
6.2 Planejamento de Experimentos 49
6.3 Avaliação Comportamental 50
6.3.1 Algoritmo de Negociação 50
6.3.2 Algoritmo de Reserva de Conexões 60
6.3.3 Comparativo Entre os Algoritmos 67
6.4 Estudo da Influência dos Fatores 68
6.4.1 Influência dos Fatores no Tempo Médio de Resposta 69
6.4.2 Influência dos Fatores na Porcentagem de Requisições Aceitas 72
6.5 Considerações Finais 75

7 Conclusões e Trabalhos Futuros **77**
7.1 Conclusões 77
7.2 Contribuições 79
7.3 Trabalhos Futuros 79

Lista de Figuras

2.1 Funcionamento de sinalização do RSVP. 9

3.1 Levantamento dos servidores *Web* mais utilizados (Netcraft, 2011). 14
3.2 Fases de processamento do Apache (Semprebom, 2007). 15
3.3 *MPM Prefork*. 16
3.4 *MPM Worker*. 16
3.5 Módulos do SWDS(Teixeira, 2004). 21
3.6 Arquitetura do SWDS. 27
3.7 Arquitetura do SWDS implementada. 28

4.1 Algoritmo de Negociação. 32
4.2 Fluxograma da Negociação. 32
4.3 Pseudo-código do algoritmo de negociação 33
4.4 Ilustração do exemplo de reserva de conexões 35

5.1 Ambiente de testes. 38
5.2 Tempo médio de resposta do primeiro byte - Negociação com 2 máquinas exclusivas para a Classe 1 40
5.3 Tempo médio de resposta do primeiro byte - Negociação com 3 máquinas exclusivas para a Classe 1 40
5.4 Tempo médio de resposta do primeiro byte - Negociação com 4 máquinas exclusivas para a Classe 1 40
5.5 Tempo médio de resposta total - Negociação com 2 máquinas exclusivas para a Classe 1 41
5.6 Tempo médio de resposta total - Negociação com 3 máquinas exclusivas para a Classe 1 41
5.7 Tempo médio de resposta total - Negociação com 4 máquinas exclusivas para a Classe 1 41
5.8 Porcentagem de requisições aceitas - Negociação com 2 máquinas exclusivas para a Classe 1 42
5.9 Porcentagem de requisições aceitas - Negociação com 3 máquinas exclusivas para a Classe 1 42
5.10 Porcentagem de requisições aceitas - Negociação com 4 máquinas exclusivas para a Classe 1 43
5.11 Tempo médio de resposta do primeiro byte - Reserva de conexões com PR1 fixado em 50% 44

5.12 Tempo médio de resposta do primeiro byte - Reserva de conexões com PR1 fixado em 62,5% 44
5.13 Tempo médio de resposta do primeiro byte - Reserva de conexões com PR1 fixado em 75% 44
5.14 Tempo médio de resposta total - Reserva de conexões com PR1 fixado em 50% . . 45
5.15 Tempo médio de resposta total - Reserva de conexões com PR1 fixado em 62,5% . 45
5.16 Tempo médio de resposta total - Reserva de conexões com PR1 fixado em 75% . . 45
5.17 Porcentagem de requisições aceitas - Reserva de conexões com PR1 fixado em 50% 46
5.18 Porcentagem de requisições aceitas - Reserva de conexões com PR1 fixado em 62,5% 46
5.19 Porcentagem de requisições aceitas - Reserva de conexões com PR1 fixado em 75% 46

6.1 Tempo médio de resposta do primeiro byte - negociação - limite 400. 51
6.2 Tempo médio de resposta total - negociação - limite 400. 51
6.3 Tempo médio de resposta total - negociação - limite 400. 52
6.4 Porcentagem de requisições aceitas - negociação - limite 400. 53
6.5 Tempo médio de resposta total - negociação - limite 600. 54
6.6 Porcentagem de requisições aceitas - negociação - limite 600. 55
6.7 Tempo médio de resposta total - negociação - limite 800. 56
6.8 Porcentagem de requisições aceitas - negociação - limite 800. 57
6.9 Tempo médio de resposta total - negociação - limite 1000. 57
6.10 Porcentagem de requisições aceitas - negociação - limite 1000. 58
6.11 Tempo médio de resposta total - negociação - limite 1200. 59
6.12 Porcentagem de requisições aceitas - negociação - limite 1200. 60
6.13 Tempo médio de resposta do primeiro *byte* - reserva de conexões - limite 400. . . . 61
6.14 Tempo médio de resposta total - reserva de conexões - limite 400. 61
6.15 Tempo médio de resposta total - reserva de conexões - limite 400. 62
6.16 Porcentagem de requisições aceitas - reserva de conexões - limite 400. 62
6.17 Tempo médio de resposta total - reserva de conexões - limite 600. 63
6.18 Porcentagem de requisições aceitas - reserva de conexões - limite 600. 63
6.19 Tempo médio de resposta total - reserva de conexões - limite 800. 64
6.20 Porcentagem de requisições aceitas - reserva de conexões - limite 800. 65
6.21 Tempo médio de resposta total - reserva de conexões - limite 1000. 65
6.22 Porcentagem de requisições aceitas - reserva de conexões - limite 1000. 66
6.23 Tempo médio de resposta total - reserva de conexões - limite 1200. 66
6.24 Porcentagem de requisições aceitas - reserva de conexões - limite 1200. 67
6.25 Influência dos fatores no tempo médio de resposta da Classe 1. 70
6.26 Influência dos fatores no tempo médio de resposta da Classe 1. 70
6.27 Influência dos fatores no tempo médio de resposta da Classe 1. 71
6.28 Influência dos fatores no tempo médio de resposta da Classe 2. 71
6.29 Influência dos fatores no tempo médio de resposta da Classe 2. 72
6.30 Influência dos fatores no tempo médio de resposta da Classe 2. 72
6.31 Influência dos fatores na porcentagem de requisições aceitas da Classe 1. 73
6.32 Influência dos fatores na porcentagem de requisições aceitas da Classe 1. 73
6.33 Influência dos fatores na porcentagem de requisições aceitas da Classe 1. 73
6.34 Influência dos fatores na porcentagem de requisições aceitas da Classe 2. 74
6.35 Influência dos fatores na porcentagem de requisições aceitas da Classe 2. 74
6.36 Influência dos fatores na porcentagem de requisições aceitas da Classe 2. 75

Lista de Tabelas

5.1 Configuração de Software e Hardware 38

6.1 Fatores e níveis do planejamento de experimentos 50
6.2 Comparação entre os algoritmos - Carga 25% para a Classe 1 e 75% para a Classe 2 68
6.3 Comparação entre os algoritmos - Carga 50% para a Classe 1 e 50% para a Classe 2 68
6.4 Comparação entre os algoritmos - Carga 75% para a Classe 1 e 25% para a Classe 2 69

Lista de Siglas

APSP - Adaptive Priority Scheduling Policy

CAP - Client-Aware Policy

CERN - Conseil Européen pour la Recherche Nucléaire

CPU - Central Processing Unit

DNS - Domain Name Server

DS - Differentiated Services

FCFS - First Come First Serve

HTML - HyperText Markup Language

HTTP - Hypertext Transfer Protocol

IETF - Internet Engineering Task Force

IP - Internet Protocol

IPv4 - Internet Protocol version 4

IPv6 - Internet Protocol version 6

ISO - International Organization for Standardization

ISS - Internet Information Services

HTTP - HyperText Transfer Protocol

DS - Internet Information ServiceDS (Differentiated Services)

MMA - Maximum Admission Algorithm

MPA	-	Maximum Profit Algorithm
MPM	-	Multi-Processing Module
NAT	-	Network Address Translation
NCSA	-	National Strencht and Conditioning Associate
QoS	-	Quality of Service
POP3	-	Post Office Protocol
QoS	-	Quality of Service
RFAC	-	Request Admission Feedback Control
RSVP	-	Resource Reservation Protocol
SLA	-	Service Level Agreement
SWDS	-	Servidor Web com Diferenciação de Serviço
TCP	-	Transmission Control Protocol
ToS	-	Type of service
URL	-	Universal Resource Locator
WEKA	-	Waikato Environment for Knowledge Analysis
WRR	-	Weighted Round Robin

CAPÍTULO

1

Introdução

1.1 Contextualização

Atualmente, o meio mais eficiente de proporcionar interação em tempo real entre as pessoas é a *Internet*. A popularização da Internet fez com que a *Web* se tornasse um meio de amplo uso para troca de informações entre usuários.

Recentemente notou-se o surgimento de um novo conceito na *Internet*: o paradigma orientado a negócios. Empresas estão sendo motivadas cada vez mais a migrarem seus serviços para a Web (Shan et al., 2005) e assim, elas podem ter uma plataforma que seja acessível de qualquer lugar com apenas um *browser*.

Neste cenário pode-se notar que a quantidade de informações que trafega pela *Internet* é tão grande que causa uma nítida degradação em diversos aspectos no uso da rede mundial. Uma proposta inicial para solucionar este problema foi o aumento da capacidade operacional da rede. Isto seria uma solução parcial, pois com o crescente número de usuários na *Internet*, a capacidade da rede seria degradada em pouco tempo. Para solucionar tais problemas, foi necessária então a criação de protocolos capazes de reduzir o congestionamento da rede.

Foi criado um padrão denominado *best effort* (melhor esforço), o qual tenta entregar a resposta ao usuário da melhor forma possível, não tendo nenhuma regra de diferenciação. No entanto, nem todas as requisições têm a mesma prioridade, mas são tratadas do mesmo modo (Dovrolis e Stiliadis, 1999). Há uma quantidade expressiva de pesquisas com o objetivo de oferecer Qualidade de Serviço *QoS (Quality of Service)* em nível de rede; um exemplo de organização com este objetivo é a *IETF (Internet Engineering Task Force)* que contribui com especificações visando esta finalidade, onde são enfatizadas estruturas tais como os Serviços Integrados (Braden et al., 1994) e os

Serviços Diferenciados (Blake et al., 1998). Porém, do que adianta oferecer qualidade de serviço em nível de rede, se em nível de aplicação as requisições são tratadas de forma igualitária?

A fim de prover diferenciação de serviços entre os diversos tipos de usuários em nível de aplicação, vários trabalhos vêm sendo desenvolvidos. Dois deles serão citados a seguir:

- O trabalho desenvolvido por (Andreolini et al., 2004) onde é implementado um servidor *Web* distribuído, cuja finalidade é prover *QoS* em nível de aplicação. Para isto foi estipulado um contrato *SLA (Service Level Agreement)* para múltiplas classes de usuários e serviços, e foi implementado um *Web switch* que transforma um *cluster* de servidores *Web* baseado no modelo *best effort* em um sistema que suporta *QoS*. Para que o trabalho fosse realizado, foi necessária a utilização do servidor Apache, o qual tem a capacidade de trabalhar como um *proxy* no *Web switch* e trabalhar como um servidor convencional no *backend*[1]. Os primeiros testes foram feitos utilizando os conceitos básicos de *QoS* definidos para roteadores de rede e protocolos que estão na camada do servidor *Web*. Também foi observado que só as políticas que satisfazem todos os requisitos de *QoS* são capazes de manter um contrato *SLA* mesmo sob condições de estresse do servidor.

- Outro trabalho relevante na área de *QoS* em nível de aplicação é o de (Li et al., 2008), o qual propõe um modelo de controle de *QoS* baseado em realocação dinâmica de recursos no servidor *Web*. Este modelo pode monitorar o estado real dos recursos do servidor, escalonar e realocar estes recursos para satisfazer, na prática, diferentes demandas de clientes. Para que o trabalho fosse desenvolvido, foi criada uma camada denominada Controle Dinâmico (*Dynamic Crontrol Layer*), a qual intercepta requisições *HTTP*, classifica-as e as encaminha para o servidor *Web*. Enquanto isso, o comportamento dos clientes é monitorado pela camada de controle dinâmico e informações como tempo de resposta do sistema e *trhoughput* são coletadas e processadas. Baseado neste monitoramento, a camada de controle dinâmica realiza a classificação de requisições e, utilizando uma política de admissão com realocação de recursos dinâmicos, garante que o servidor *Web* trabalhe com cargas normais e mantendo tempos de respostas para que o servidor consiga garantir *QoS* a diferentes classes de prioridade.

  Para a realização dos experimentos foi utilizado o servidor *Web IIS* e o gerador de carga *JMeter*. Resultados comprovam que, quando utilizado o módulo de realocação dinâmica de recursos, as requisições de alta prioridade não têm um melhoramento muito significante no tempo de resposta. Porém as requisições de média e baixa prioridade têm seus tempos de resposta reduzidos consideravelmente. Analisando do lado do servidor, pode-se concluir que com a aplicação do módulo de realocação de recursos, o número de requisições aceitas pelo sistema é consideravelmente maior do que quando não se aplicava o módulo.

[1] Servidores que processam requisições

Com a intenção de satisfazer as requisições de modo eficiente e respeitando as prioridades de cada usuário, foi proposta uma arquitetura capaz de satisfazer essas condições, denominada Servidor *Web* com Diferenciação de Serviços (SWDS). Esta arquitetura foi detalhada por Teixeira em seu trabalho de doutorado (Teixeira, 2004).

A arquitetura proposta é composta pelos módulos: Classificador, Controle de Admissão e um *cluster* de processos ou servidores *Web*, os quais serão detalhados no Capítulo 3.

O Classificador é o elemento responsável por receber as requisições que chegam ao sistema e subdividi-las em classes de serviços, segundo critérios preestabelecidos. Após essa fase, a nova requisição entra no sistema em uma determinada categoria e recebe um tratamento condizente com a classe a qual pertence.

O Controle de Admissão recebe as requisições já classificadas e gerencia a sua aceitação pelo servidor, levando em conta as políticas de atendimento vigentes e as informações da carga de trabalho do sistema. Caso o servidor esteja sobrecarregado, uma requisição poderá ser rejeitada (descarte) ou ter suas exigências de qualidade de serviço relaxadas (negociação), a fim de que possa ser aceita em uma classe de prioridade inferior.

Após ser admitida no sistema, a requisição é atribuída a um dos nós do *cluster* de servidores *Web*, sendo atendida conforme o algoritmo de escalonamento ou diferenciação de serviços vigente. Após a conclusão do processamento, os resultados são retornados ao cliente que originou a solicitação.

A partir desse projeto de doutorado proposto por (Teixeira, 2004), diversos outros trabalhos foram desenvolvidos no Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC-USP/São Carlos), em vários níveis, incluindo iniciações científicas ((Barbato et al., 2004); (Traldi et al., 2006)), mestrados ((Barbato, 2008); (Estrella, 2006); (Messias et al., 2007); (Mourão et al., 2007); (Silva, 2006); (Traldi et al., 2006)) e um doutorado (Abraão, 2008).

Para este projeto de mestrado, todos os trabalhos anteriores são importantes, destacando-se:

- O trabalho de (Messias et al., 2007), onde foi implementado um protótipo de servidor *Web* distribuído, considerando-se um *cluster* formado a partir de nós de uma rede de computadores (foi utilizada a rede de computadores do Laboratório de Redes e Sistemas Distribuídos do ICMC-USP). Alguns algoritmos foram implementados e testados, visando exercitar o protótipo e verificar o seu funcionamento. Neste projeto, o trabalho de Messias foi estudado e revisado com o intuito de se obter a arquitetura mais adequada para a implementação de um novo protótipo, agora baseado em um *cluster* dedicado, cujos nós serão interconectados por uma *switch*, também dedicada.

- Os trabalhos de (Barbato et al., 2006), (Barbato et al., 2004), (Barbato, 2008) e (Estrella, 2006) onde foram discutidos e estudados algoritmos visando ao controle de admissão de requisições submetidas a um servidor *Web* com diferenciação de serviços, com e sem o conceito de sessão.

É tendo como base o projeto de doutorado de (Teixeira, 2004) que este trabalho de mestrado foi desenvolvido.

1.2 Motivação e Objetivos

Uma das principais motivações para o desenvolvimento deste projeto é que servidores *Web* convencionais como, por exemplo *Internet Information Service (IIS)*, Apache, etc., adotam o modelo de melhor esforço, o qual emprega um modelo de filas. Isto pode não ser uma boa alternativa, pois não garante *QoS* pelo fato de que o tempo de resposta de uma requisição vai depender do quão ocupado está o *buffer* do servidor. Com isso não se pode garantir que em situações de sobrecarga todas as requisições importantes sejam aceitas ou atendidas em um tempo adequado, resultando na insatisfação de clientes mais prioritários como também de usuários menos prioritários (Li et al., 2008).

O objetivo principal deste projeto foi o estudo, implementação e teste de algoritmos de controle de admissão de requisições, visando o emprego em servidores *Web* distribuído com Qualidade de Serviço.

Em uma primeira etapa, foi estudada e aprimorada a implementação de um servidor *Web* distribuído, baseado em um *cluster* dedicado que serve de bancada de testes para este projeto e diversos outros que estão em andamento no Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC) do Departamento de Sistemas de Computação, ICMC-USP. Esta etapa foi desenvolvida em conjunto com outro projeto, também de mestrado e que já foi concluído em Março de 2011, o qual também utilizou o servidor produzido como sua bancada de testes (Figueiredo, 2011). Esta etapa teve por objetivo implementar um protótipo flexível, de fácil acesso através do LaSDPC sendo inspirado no trabalho desenvolvido por (Messias et al., 2007), onde foram revisados e adaptados várias das decisões tomadas naquele trabalho, à luz da realidade atual, onde foi disponível para este projeto um *hardware* dedicado.

Como produto final deste projeto tem-se a implementação de um servidor *Web* distribuído (estabelecendo uma plataforma de testes flexível e de fácil acesso) e a implementação real de algoritmos de controle de admissão de requisições, alguns deles já estudados em nível de simulação em trabalhos anteriores desenvolvidos no grupo.

Em (Estrella, 2006) foi proposto e testado utilizando a técnica de simulação um algoritmo de controle de admissão com o intuito de negociação. As requisições que não conseguem ser executadas no servidor em um primeiro instante, têm suas exigências de *QoS* rebaixadas mesmo sem o cliente saber dessa negociação.

Em (Barbato et al., 2006) foi simulada uma política que limita o tamanho da fila dos nós dos servidores. Esta política estabelece um tamanho máximo para as filas dos servidores do *cluster* e, caso a nova requisição seja atribuída a um nó cuja fila tenha atingido esse limite, ela será rejeitada, independentemente da sua classe.

Ao longo deste projeto de mestrado, também foi proposto um algoritmo de controle de admissão baseado em alocação dinâmica de conexões disponíveis no SWDS. Este algoritmo será descrito detalhadamente no Capítulo 4.

Com os resultados obtidos, objetiva-se mostrar o comportamento e as características de cada algoritmo implementado na arquitetura SWDS, avaliando o desempenho de cada um deles e calculando a influência dos fatores nas variáveis de respostas.

Esta dissertação está organizada da seguinte forma: o Capítulo 2 aborda qualidade de serviços, onde é apresentado *QoS* em nível de rede e *QoS* em nível de aplicação, a qual é utilizada nesse trabalho de mestrado. No capítulo 3 são abordados os conceitos de servidores *Web* e servidores *Web* distribuídos. Também é descrito neste Capítulo o servidor *Web* Apache, o qual serve de base para este projeto. Por fim é descrito o modelo do servidor SWDS. O Capítulo 4 apresenta os algoritmos propostos para este trabalho de mestrado. O Capítulo 5 descreve detalhadamente o ambiente de experimentos utilizado. O Capítulo 6 apresenta uma avaliação de desempenho detalhada dos algoritmos de controle de admissão, sendo que no fim desse capítulo é feito um estudo sobre a influência dos fatores nas variáveis de resposta. Por fim, o último capítulo apresenta as contribuições e conclusões deste trabalho e o que pode ser melhorado em trabalhos futuros.

CAPÍTULO

2

Qualidade de Serviços

2.1 Considerações Iniciais

Atualmente, no modelo da *Internet*, as requisições são tratadas de forma equivalente, ou seja, a entrega dos dados é realizada de modo igual a todos os usuários. Nesse modelo, os usuários compartilham a capacidade de transmissão com todos os fluxos de dados, os quais percorrem uma rota possível para chegar ao seu destino, de acordo com as rotas definidas e a capacidade de banda que estiverem disponíveis no momento.

Com este cenário da *Internet* atual, pode-se perceber que os usuários têm necessidades diferentes, e que a qualidade de serviço veio para descrever qual o tratamento que um usuário ou uma aplicação receberá em uma rede (Networks, 2003).

A qualidade de serviço é um dos principais tópicos de pesquisa e desenvolvimento na área de servidores *web* (Zhao et al., 2000). Por isso, este capítulo apresenta conceitos de *QoS* em nível de redes *IP*, procurando reutilizar esses conceitos também em nível de aplicação, para ser usado especificamente no Servidor *Web* com Diferenciação de Serviços.

2.2 Qualidade de Serviços

Atualmente, o tráfego na rede é bastante diversificado, com vários tipos de usuários. Cada tipo possui um requisito único em termos de largura de banda, atraso, perda e disponibilidade.

A maioria do tráfego hoje é baseada no protocolo *IP*. Ter um único protocolo de transporte é benéfico, pois os equipamentos de rede se tornam menos complexos e mais baratos. Entretanto,

este benefício é contestado pelo fato de que o *IP* é um protocolo sem conexões, isto é, os pacotes *IP* não possuem caminhos específicos à medida que trafegam pela rede. Isso resulta numa qualidade de serviço inadequada em uma rede *best effort* (Networks, 2003).

Sendo assim, para que a *Internet* possa evoluir, é preciso definir um método para prover serviços diferenciados. Este método pode ser entendido como a qualidade de serviço, a qual é definida na literatura de diferentes formas. O trabalho de (Zhao et al., 2000) define *QoS* como um mecanismo que provê diferenciação de serviços e garantias de desempenho para aplicações na *Internet*. A diferenciação de serviços provê serviços diferenciados para diferentes aplicações de acordo com seus requisitos. A *ISO* também define *QoS* como resultado coletivo de desempenho de um serviço, o qual determina o grau de satisfação de um usuário, ou seja *QoS* pode ser vista como a percepção do usuário quanto à eficiência de um serviço prestado a ele.

Analisando as redes de computadores, o conceito de qualidade de serviço tem aplicação em todos os níveis das arquiteturas convencionais, e parte do desenvolvimento nesse campo tem se dado na elaboração de técnicas aplicáveis em camadas inferiores, especialmente na camada de rede (*QoS* em nível de rede).

Contudo, pode-se observar que se as técnicas de provisão de *QoS* implementadas nas camadas inferiores fossem também implementadas nas camadas superiores, poderia se obter resultados mais expressivos de *QoS*. Oor exemplo, podem ser consideradas situações de sobrecarga de um servidor, onde as requisições não são tratadas de forma indiscriminada, o que não contribui para os esforços empreendidos ao longo da rede. Dessa forma, há um crescente interesse na provisão de *QoS* em Nível de Aplicação, e é sobre esse ponto de vista que se desenvolve este trabalho.

2.3 QoS em Nível de Rede

Em nível de rede, *QoS* pode ser definida como a capacidade de um elemento em prover certo grau de garantia do cumprimento dos requisitos de seu tráfego e serviços oferecidos, tais como níveis mínimos de perdas e atraso de dados requisitados pelo usuário (Saito, 2010).

A *IETF* propôs diversos modelos e mecanismos com o objetivo de oferecer serviços com *QoS* para as redes *IP*, entre as quais podem-se destacar: a arquitetura de Serviços Integrados (Braden et al., 1994), a arquitetura de Serviços Diferenciados (Blake et al., 1998), a Engenharia de Tráfego (Awduche et al., 1999) e o Roteamento Baseado em Restrições (Crawley et al., 1998), (Apostolopoulos et al., 1998).

Todas essas arquiteturas são utilizadas para prover *QoS* em nível de rede. Nas próximas seções são descritas as duas abordagens mais utilizadas para provisão de *QoS*: os Serviços Integrados e os Serviços Diferenciados (Zhao et al., 2000).

2.3.1 Serviços Integrados

Os serviços integrados são caracterizados pela reserva de recursos na rede. Antes de iniciar uma comunicação, o emissor solicita ao receptor a alocação de recursos para que a transação de dados da aplicação seja feita de modo a ter boa qualidade. Para tanto, tais aplicações utilizam o protocolo *RSVP (Resource Reservation Protocol)* que é um protocolo utilizado para troca de mensagens de alocação de recursos. Estes recursos dizem respeito à largura de banda e ao tempo em que será mantida a conexão (Braden et al., 1994). A Figura 2.1 ilustra o funcionamento do *RSVP*.

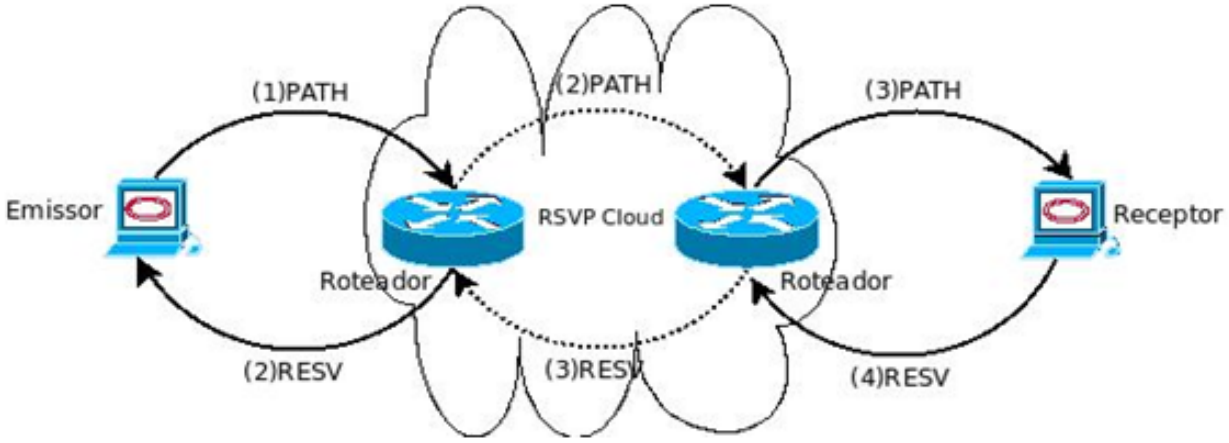


Figura 2.1: Funcionamento de sinalização do RSVP.

Suponha que o emissor seja uma Universidade 1 e o receptor uma Universidade 2, e a Universidade 1 deseja fazer uma videoconferência com a Universidade 2. Esta aplicação requer baixo atraso e baixa variação do atraso *(jitter)* para que sejam mantidos os requisitos de *QoS*. Deste modo a Universidade 1 manda uma mensagem por um caminho *(PATH)* para a Universidade 2, especificando as características do tráfego. Quando a mensagem chega à Universidade 2, inicia-se o procedimento de reserva de recursos (RESV) por todo o caminho entre esses dois nós da rede. Todos os roteadores entre as duas universidades passam pela etapa de alocação de recursos e qualquer um deles pode rejeitar a solicitação, informando para a Universidade 1 que a solicitação não foi aceita. Caso todos os roteadores tenham condições de disponibilizar os recursos solicitados, é alocada então a largura de banda e *buffer* necessários para a aplicação de videoconferência.

A reserva de recursos ao longo da rede gera uma grande desvantagem aos Serviços Integrados, que é a sua falta de escalabilidade. Em grandes redes, a necessidade de manter recursos reservados em todos os roteadores e o grande número de mensagens trocadas para reserva de recursos pelo protocolo *RSVP*, inviabilizam o seu uso (Kamienski e Sadok, 2000).

Atualmente pode-se encontrar na literatura trabalhos que usam algumas características do protocolo *IPV6* para melhorar a arquitetura de Serviços Integrados, como no artigo de (Padilla e Paradells, 2008) onde é utilizado o cabeçalho do protocolo *IPV6* para melhorar as configurações

padrão dos Serviços Integrados e com isso aumentar a velocidade de processamento da classificação dos pacotes dentro dos roteadores.

2.3.2 Serviços Diferenciados

Em função dos problemas existentes com a arquitetura de serviços integrados/*RSVP*, no que diz respeito à escalabilidade e à complexidade de implementação, surgiu outra proposta, denominada arquitetura de Serviços Diferenciados (Blake et al., 1998). A arquitetura de Serviços Diferenciados foi projetada para prover classes de serviços agregados, onde os fluxos individuais são agrupados de forma a serem tratados pela rede de acordo com a classe de serviço onde eles estão. Sendo assim, o gerenciamento pelos roteadores torna-se mais simples, pois os fluxos são agregados e os roteadores não precisam guardar o estado de cada fluxo individualmente, como é o caso dos Serviços Integrados.

Os Serviços Diferenciados são suportados com base em um campo denominado *DS (Differentiated Services)* que está presente no cabeçalho dos pacotes *IP*. O campo *DS* foi definido de forma a ser compatível com os octetos *ToS (Type of service)* do protocolo *IPv4* e *Traffic Class* do *IPv6*. É com base no *DS* que um conjunto de procedimentos distintos de envio de pacotes é especificado, e com isso oferecendo diferentes classes de serviços.

Em uma solicitação de serviço diferenciado é importante estabelecer um contrato de nível de serviço, firmado entre o usuário e o provedor de serviços de *Internet*. Para ter esse nível de serviço é preciso um componente denominado *SLA (Service Level Agreement)*. De acordo com (Bhoj et al., 1999), o *SLA* é um contrato de serviço entre um cliente e um provedor de serviços, o qual tem como função especificar para o cliente os detalhes da classificação de tráfego e o serviço de encaminhamento correspondente a ser recebido. Neste contexto, o provedor de serviço que mantém um contrato *SLA* com um cliente precisa garantir que o tráfego desse cliente obtenha a *QoS* contratada.

Um contrato *SLA* tem a função de determinar as classes de serviços suportadas e a quantidade de tráfego na rede entre os domínios. Esses contratos podem ser estáticos ou dinâmicos, sendo que no contrato estático os serviços são negociados de forma regular (mensalmente ou anualmente) e nos contratos dinâmicos os serviços podem ser solicitados sob demanda. Para isso, é necessária a utilização de algum protocolo de sinalização e controle (como o *RSVP*) de forma a indicar as necessidades do usuário em determinados momentos (Zhao et al., 2000).

As abordagens de *QoS* (Serviços Integrados e Serviços Diferenciados) podem ser encontradas no trabalho de (Wojtanowicz et al., 2009), onde é feita uma investigação sobre esses mecanismos considerando vários meios físicos de comunicação, do meio cabeado até a ambientes *wireless*, os quais colocam muitas restrições. Pode ser observado que protocolos que operam com confiança em meios cabeados demandam melhoramentos para lidar com ambientes *wireless*.

2.4 QoS em Nível de Aplicação

Na seção 2.3, foram definidas as características de qualidade de serviço em nível de rede. Como é de conhecimento, os servidores *Web* estão na camada de aplicação. Pode-se constatar que de nada adianta prover qualidade de serviço em nível de rede, se um servidor *Web* que está no nível de aplicação trata todas as suas requisições da mesma maneira. Portanto esta seção explora a qualidade de serviço em nível de aplicação.

Enquanto prever a demanda e aumentar a capacidade dos servidores em geral pode ser mais fácil, descobrir as expectativas de *QoS* de um usuário, saber qual será a demanda de novos usuários é mais custoso e apresenta maiores dificuldades. Para contornar este problema é necessário implementar a qualidade de serviço nos servidores *Web*, de modo que um servidor, quando sobrecarregado, tenha a capacidade de estabelecer prioridades entre as requisições e que possa atender as requisições mais importantes, conforme definido pelo operador do servidor.

Para o usuário final, deve-se ter um limite no tempo de resposta das requisições enviadas ao servidor e se esse limite for excedido, o usuário tende a ficar insatisfeito com os serviços prestados. Para que este problema não ocorra, vários trabalhos foram desenvolvidos no escopo de servidores *Web* com *QoS* quando sobrecarregados. Alguns deles serão citados nos próximos parágrafos.

O trabalho de (Kanodia e Knightly, 2003) propõe um modelo de serviço *Web* baseado na diferenciação de classes e no controle de admissão de forma a gerenciar os tempos médios de resposta contratados e maximizar a utilização de recursos.

Em (Estrella, 2006) é incluído um mecanismo de negociação no módulo de controle de admissão do modelo SWDS, com o objetivo de permitir que determinadas requisições sejam aceitas no servidor com suas exigências de *QoS* rebaixadas.

Em Blanquer et al. (2004) é mostrado como é possível conseguir a qualidade de serviço em nível de aplicação. Os autores argumentam que a melhor forma de se conseguir *QoS* nos servidores *Web* é a utilização de um mecanismo de controle de admissão e técnicas de controle de tráfego nos pontos de entrada dos *hosts* da *Internet*.

No trabalho de (Bochmann et al., 2001), os autores examinam o uso de técnicas de gerenciamento dinâmico de *QoS* em combinação com replicação em vários níveis de arquiteturas de aplicações para comércio eletrônico.

O trabalho de (Serra et al., 2005) propôs uma nova arquitetura denominada *WS-DASC* (*Web Servers - Differentiated Services Admission Control*), a qual provê QoS utilizando balanceamento de carga e um mecanismo para controle de admissão baseado no *Diffserv*. A plataforma é composta por um classificador, que é responsável pela classificação e pela admissão das requisições e um *gateway* que repassa as requisições para o nó com menor carga. A coleta de informações de carga é realizada por um monitor que está em cada nó do servidor, o qual mede a carga baseando-se na técnica de Coeficiente de Reatividade, que informa a tendência da carga de cada nó. A partir desse valor, o modelo opera em três modos distintos: exclusivo, que atende somente uma classe;

compartilhado, que atende outras classes; e saturado, que não atende novas requisições que chegam ao sistema.

2.5 Considerações Finais

Foi visto neste Capítulo que existe a necessidade de se implantar qualidade de serviços nas aplicações da *Internet*. Como observado nas seções 2.3.1 e 2.3.2, as abordagens de provisão de QoS na *Internet* mais utilizadas são as em nível de rede, serviços integrados e de serviços diferenciados. Além da *QoS* em nível de rede, foi apresentado também o quão é importante o conceito de *QoS* em nível de aplicação.

No próximo Capítulo serão apresentados os conceitos de servidores *Web* e o modelo SWDS, o qual tem por objetivo prover *QoS* à classes de serviços.

CAPÍTULO

3

Servidor Web com Diferenciação de Serviços (SWDS)

3.1 Considerações Iniciais

O objetivo deste Capítulo é apresentar a arquitetura do servidor *Web* com diferenciação de serviços proposta por (Teixeira, 2004). Para isso é necessário uma breve introdução sobre servidores *Web HTTP*, em especial o Apache, o qual serve como base para este projeto.

Também são apresentados alguns cenários de utilização de servidores *Web* com diferenciação de serviços e por fim alguns trabalhos relacionados a este trabalho de mestrado, que é o estudo de algoritmos de controle de admissão para o modelo SWDS.

3.2 Servidor Web

Em geral, a função de um servidor *Web* é esperar pelas requisições, tratá-las e em seguida respondê-las. Um cliente (normalmente um *browser*) requisita um recurso (página *HTML*, fotos, vídeos, arquivos, etc), o servidor examina a requisição, verifica se ele contém o que foi requisitado ou encaminha esta requisição a um programa ou a outro servidor que produzirá o dado requisitado. Finalmente, o cliente recebe uma resposta do servidor (Gröne et al., 2004).

Enquanto a funcionalidade básica do servidor *Web* é fornecer conteúdo, uma implementação mais completa também inclui um modo de receber conteúdo dos clientes. Esta característica é usada para submeter formulários e fazer o carregamento de arquivos.

Muitos servidores também incluem o processamento pelo servidor, isto é, criar documentos *HTML (hypertext markup language) on-the-fly* (em tempo de execução), em contrapartida a devolver os documentos fixos. Isto é referenciado como conteúdo dinâmico e estático, respectivamente.

3.2.1 Apache

O servidor *Web* Apache foi originalmente criado em 1995, sendo baseado e derivado do servidor *NCSA (National Center for Supercomputing Applications)*. O primeiro servidor produzido sob o nome Apache foi a versão 1.0.0, liberado em dezembro de 1995.

O Apache tinha muitas limitações na sua versão original e com o passar do tempo e com o advento de novas tecnologias, seus desenvolvedores começaram o desenvolvimento do Apache versão 2.0 em abril de 2002.

Estudos apontam que em 1995 o servidor *NCSA* dominava o mercado de servidores *Web* com 57%, em segundo lugar estava o *CERN* com 19%. Nesse mesmo ano, surgiu o Apache com 3,5% do uso. Atualmente o Apache lidera o mercado com 59,35%, seguido da Microsoft com 22,22% (Netcraft, 2011). A Figura 3.1 mostra um levantamento da utilização dos servidores nos últimos anos.

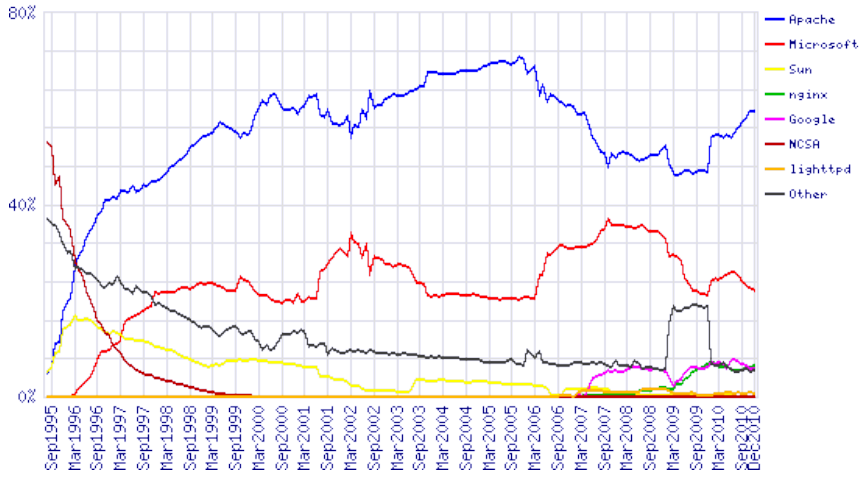


Figura 3.1: Levantamento dos servidores *Web* mais utilizados (Netcraft, 2011).

Para o desenvolvimento deste trabalho, o servidor *Web* Apache foi escolhido pela sua robustez, por ser código livre para fazer as alterações necessárias para a implementação do SWDS, por ter a possibilidade de inserir programas externos para a manipulação de requisições e principalmente por ser o servidor *HTTP* mais utilizado em todo o mundo de acordo com o site *netcraft.com*, o qual é referência em classificar popularidade de sites, servidores *Web*, entre outros.

Características Internas do Apache

Um servidor *Web* pode ser resumido a um sistema que escuta as requisições *HTTP*, processa o que lhe foi enviado e retorna a resposta a quem originou a requisição. No Apache esta é a tarefa do gerador de conteúdo, o "coração" do servidor *Web*. Para tratar cada requisição *HTTP* que chega ao servidor, é necessário que um gerador de conteúdo seja executado. Qualquer módulo pode registrar geradores de conteúdo, que geralmente são realizados pelas diretivas SetHandle ou AddHandle, que são inseridas no arquivo de configuração do Apache chamado httpd.conf.

Se uma requisição não tem um gerador de conteúdo associado a ela, esta é manipulada pelo gerador de conteúdo padrão do Apache, que vai somente retornar o arquivo indicado pela requisição para o sistema de arquivos.

Antes que as requisições cheguem ao gerador de conteúdo, o Apache divide as requisições em diferentes fases. Estas fases podem examinar e, talvez, manipular cabeçalhos das requisições, e determinar o que fazer com as requisições. Alguns exemplos dessas fases:

- Verificar se a *URL (Universal Resource Locator)* da requisição será compatível com a configuração para determinar qual gerador de conteúdo será utilizado;
- Mapear a *URL* para o sistema de arquivos (ex.: para um arquivo estático);
- Encontrar a versão que mais se enquadra com o navegador *Web*;
- Controlar o acesso ao servidor, com adição de algumas regras de autenticação e de acesso;
- Alterar a *URL* contida nas requisições segundo alguma estratégia.

Por fim ainda tem-se a fase de *log* que é executada antes do servidor enviar a mensagem de resposta ao cliente. Os módulos que vêm antes do gerador de conteúdos são chamados de módulos metadados e os que vêm depois são chamados de módulos de *log* (Semprebom, 2007).

Na Figura 3.2 são ilustradas as fases de processamento do Apache.

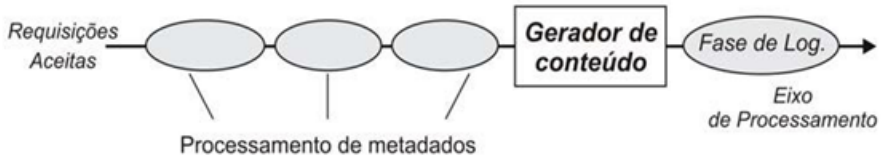


Figura 3.2: Fases de processamento do Apache (Semprebom, 2007).

Módulos de Multiprocessamento (MPM)

Na versão 2.x do Apache foi introduzido o modelo de processos *MPM* (Módulos Multiprocessamento), que são responsáveis por gerenciar as portas de comunicação, aceitar as conexões e alocar processos ou *threads* para atendimento das requisições (Gröne et al., 2004).

Quando é necessária a escolha de um *MPM* devem ser levadas em consideração várias características, como por exemplo, se o sistema operacional tem o suporte a *threads*, disponibilidade de memória no sistema, escalabilidade *versus* estabilidade. Existem três módulos *MPM* na versão 2.x do Apache, os quais são o *MPM Prefork*, *Worker* e o *Event*.

- O *MPM Prefork* é o modelo tradicional, sem *threads*. Ele manipula requisições de uma maneira similar ao Apache 1.3. O seu modelo é mais lento que modelos baseados em *threads*, pois cada requisição necessita de um processo para ser atendida. Ele é muito estável, pois cada requisição é gerenciada por um processo. Assim, se um processo falhar, este não afeta os outros processos em execução no servidor. Apesar de o *Prefork* utilizar muita memória, ele é adequado para sistemas tolerantes a falhas (DEBIAN, 2010). Na Figura 3.3 é exemplificado o *MPM prefork*.

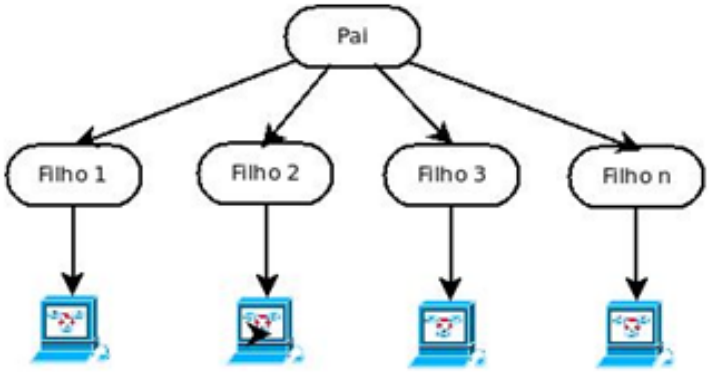


Figura 3.3: *MPM Prefork*.

- No modelo *MPM Worker* são utilizados vários processos filhos, onde cada filho dispara *threads* para o atendimento das requisições. Ele é mais escalável que o modelo *MPM Prefork* e possui um melhor desempenho, só que perde na confiabilidade, pois se um processo filho falhar, os processos e *threads* ligados a ele também irão falhar. Na Figura 3.4 é exemplificado o *MPM worker*.

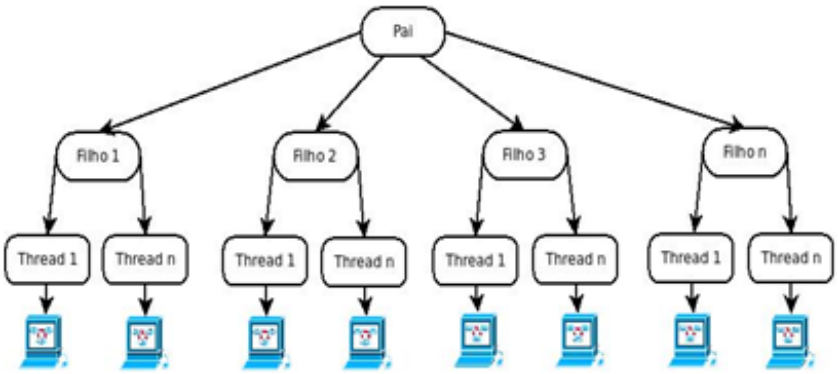


Figura 3.4: *MPM Worker*.

- O *MPM event* foi concebido para permitir que mais requisições sejam atendidas simultaneamente passando parte do trabalho de processamento para *threads* de suporte, liberando as *threads* principais para atender novas requisições. O *MPM Event* é indicado para os sites que possuem um tráfego com KeepAlive estendido, que é um recurso do Apache que permite que os clientes façam várias requisições utilizando a mesma conexão *TCP*.

Para o protótipo do SWDS foi utilizado o *MPM Worker* por ser mais escalável que o *MPM Prefork* e assim conseguir atender um maior número de requisições em menos tempo. Vale lembrar que o protótipo pode ser implementado tanto com *MPM Prefork* quanto *MPM Worker*, mas por fim decidiu-se utilizar o *MPM Worker* pelo fator escalabilidade.

Módulo mod_proxy

Para entender o mod_proxy é necessário saber o que é um *proxy*. *Proxy* é um servidor que atende as requisições, repassando os dados do cliente à frente. Um usuário (cliente) conecta-se a um servidor *proxy*, requisitando algum serviço como um arquivo, conexão, *website*, ou outro recurso disponível em outro servidor (Tanenbaum, 2003).

O módulo mod_proxy pode ser configurado de duas formas, como *proxy* de encaminhamento e *proxy* reverso (Foundation, 2010a).

O *proxy* de encaminhamento é um servidor intermediário que fica entre o cliente e o servidor original. Para obter um conteúdo do servidor de origem, os clientes mandam uma requisição para o *proxy* nomeando o servidor de origem como um alvo e o *proxy* então requisita o conteúdo do servidor original, retornando o resultado ao cliente. Um ambiente típico de uso do *proxy* de encaminhamento é prover acesso a *Internet* para clientes que estão por alguma maneira restringidos por um *firewall*. O *proxy* de encaminhamento também pode ser usado como *proxy cache* para reduzir o uso da rede.

Um *proxy* reverso, pelo contrário, aparece ao cliente como um servidor *Web* comum. O cliente faz solicitações normais de conteúdo ao *proxy* reverso. Então ele decide para onde enviar as solicitações e retorna o conteúdo como se fosse a própria origem. O *proxy* reverso pode ser utilizado em balanceamento de carga entre diversos servidores *backend*.

É importante ressaltar que o mod_proxy, por padrão, encaminha as requisições para o *backend* utilizando a política *round-robin*, ou seja, cada servidor recebe uma requisição por vez, fazendo com que a distribuição de requisições no *cluster* seja de forma circular.

Por fim, o *proxy* reverso é de grande importância para este projeto, pois é ele quem vai fazer com que o *frontend*[1] busque a página requisitada no *backend* e retorná-la ao cliente, sem este saber de qual *backend* veio a resposta.

[1] Servidor que encaminha requisições ao *backend*

Módulo mod_rewrite

Este módulo utiliza regras para reescrever *URLs* em tempo de execução. Ele suporta um número ilimitado de regras, para fornecer um mecanismo de manipulação de *URL* muito poderoso e flexível. As manipulações de *URL* podem depender de vários testes, das variáveis do servidor, variáveis de ambiente, os cabeçalhos *HTTP*, ou *time stamp*. A desvantagem do mod_rewrite é que ele é muito complexo, o que não é uma experiência muito boa para novatos no Apache (Foundation, 2010b).

Este módulo também é de grande importância no trabalho a ser desenvolvido, pois ele aceita manipulação de *URL* com programas externos, e envia as *URLs* escritas para o mod_prox para encaminhá-las ao *backend*.

Módulo mod_status

Este módulo possibilita recuperar métricas de desempenho do sistema. Essas iformações são extraídas através da apresentação de uma página HTML, a qual pode ser configurada para ser atualizada periodicamente. As informações recuperadas pelo mod_proxy são (Foundation, 2010c):

- Número de processos ociosos;
- O status de cada processo;
- O número total de acessos e a contagem de bytes processados;
- O tempo total de execução do servidor;
- A porcentagem de CPU usada por cada processo e o total executado pelo Apache;

Este módulo é importante para este trabalho, pelo fato de coletar informações para o monitoramento do protótipo.

Pool de Conexões

Como o servidor *Web* Apache trabalha com *pool* de conexões, é possível manipulá-las, podendo reservá-las para determinados tipos de usuários. Isto é um grande atrativo para este trabalho, pois no Algoritmo de Negociação, pode-se limitar o número máximo de conexões simultâneas e no Algoritmo de Reserva de Conexões, além de poder limitar o número de conexões simultâneas, também é possível manipular estas requisições reservando parte delas para a classe de maior prioridade.

No Capítulo 4, onde são descritos os algoritmos propostos, e no Capítulo 5, que descreve o ambiente de experimentos, é possível entender melhor como o *pool* de conexões funciona.

3.3 Servidor Web Distribuído

Com o aumento no número de acessos a sites da *Internet*, os servidores *Web* alocados em uma única máquina se tornam sobrecarregados e ficam ineficientes na manipulação das requisições *HTTP*. Uma solução para este problema é a alocação de mais máquinas para trabalharem em conjunto e distribuir a carga entre os diversos computadores conectados atuando como um só.

Para que isso ocorra, é necessário um mecanismo que controle o acesso aos servidores trabalhando em conjunto. Este mecanismo então fica encarregado de despachar a requisição para o servidor mais adequado, utilizando uma política de escolha do melhor servidor a ser utilizado no momento.

Um *cluster* de servidores *Web* é um grupo de servidores independentes que são gerenciados como um único sistema de alta disponibilidade e maior escalabilidade (Grant, 2000). O *cluster* funciona para o cliente como um sistema único, aumentando assim a disponibilidade total do sistema e o seu desempenho, mas pode ser difícil o gerenciamento do mesmo. Em um sistema de alto desempenho como o *mainframe*, o administrador precisa gerenciar apenas uma máquina. No *cluster* de servidores o administrador deve instalar os *softwares* necessários em vários servidores, configurar, gerenciar, operar cada um deles e garantir que os dados, tais como sites, arquivos de multimídia, e outros sejam os mesmos em cada sistema (servidor). Isso pode ser um desafio e é a razão pela qual as tecnologias de replicação de dados estão se tornando importante no mercado (Lima, 2003).

Segundo (Hirata, 2002), dado um conjunto de servidores que hospedam um site em uma única localização, pode-se identificar duas arquiteturas principais:

1. Arquitetura baseada em *cluster*: onde os clientes não conhecem os *IP's* dos nós do servidor. O único *IP* conhecido é um servidor virtual correspondente a um dispositivo (*hardware* ou *software*) que está localizado entre o cliente e o servidor, que é responsável pela distribuição de carga, então denominado distribuidor.

2. Arquitetura *Web* distribuída: quando os endereços *IP* reais dos servidores são visíveis às aplicações.

Muitos dos *clusters* de servidores *Web* fazem somente o balanceamento de carga. Isto quer dizer que o site replicado está em todos os servidores *Web* de um *cluster*. Isto é aceitável para sites de médio porte, mas além de certo tráfego, a conectividade deste site se torna um gargalo. Para solucionar este gargalo, grandes sites possuem múltiplos *clusters*, e estes clusters são distribuídos geograficamente. Com os clusters de servidores Web separados geograficamente, a requisição pode ser aceita em um *cluster* mais próximo e fazer o balanceamento de carga dentro do *cluster* selecionado. Obviamente que se todos os servidores em um *cluster* estiverem sobrecarregados outro *cluster* deve ser selecionado. Por fim o problema se torna mais complexo em tal ambiente (Swain e Kim, 2009).

Existem várias tecnologias para fazer balanceamento de carga de acordo com (Liu, 2005), (Shan et al., 2005), (Huang e Fang, 2004), (Banicescu et al., 2003)(Jiao e Wang, 2010) (Banicescu et al., 2003)

- Balanceamento baseado em *NAT*: O *Network Address Translation* converte endereços internos em endereços externos para fazer com que computadores de uma rede interna tenham acesso à rede externa, o oposto também pode ocorrer: redes externas podem visitar um endereço externo de um *gateway* fazendo com que pacotes da rede externa sejam encaminhados para computadores específicos da rede interna. Nesta analogia, pode-se distribuir requisições entre os diversos servidores de uma rede interna e com isso obter o balanceamento de cargas.

- Balanceamento baseado em *DNS*: O balanceamento de carga utilizando *DNS* é simples e amplamente utilizado. O servidor *DNS* pode ser configurado com um mesmo nome para vários endereços diferentes. Assim, diferentes servidores podem ter o mesmo nome no servidor de nomes, fazendo com que a carga imposta ao servidor seja distribuída pelo *DNS* de forma a ser balanceada entre os servidores que compõem o sistema.

- Balanceamento baseado em *IP tunneling*: Quando um servidor se encontra sobrecarregado, ele pode encaminhar suas requisições para outro servidor processar utilizando a tecnologia de *IP tunneling*. Para isto, é necessário que todos os servidores suportem esta tecnologia. Este tipo de balanceamento de carga pode trazer algumas desvantagens. É o caso de empacotamento e desempacotamento adicionais ao longo dos *tunnels* e também quando todo o sistema estiver sobrecarregado, pode ocorrer o *loop* de pacotes.

- Balanceamento baseado em *proxy* reverso: O *proxy* reverso é um servidor que fica a frente de vários servidores, os quais tem por objetivo atender as requisições encaminhadas pelo *proxy*. Todas as requisições do sistema devem passar pelo *proxy*, e este tem o papel de distribuir a carga no *backend*. O balanceamento de carga e o controle da carga podem ser atendidos também com o auxílio de algoritmos de controle de admissão e escalonamento de requisições no *backend*. Isto se torna um atrativo para este trabalho, pois o objetivo deste projeto é prover *QoS* em nível de aplicação para os diferentes níveis de classes de prioridades.

3.4 Modelo do Servidor SWDS

Grande parte dos servidores *Web* atuais tratam as requisições que recebem de acordo com a política *FCFS (first come first serve)*, onde as requisições que estão na fila são atendidas de acordo com suas ordens de chegada. Embora diferentes modelos de controle de concorrência possam ser implementados nos servidores, visando agilizar o atendimento das requisições, elas são atendidas, de modo geral, de maneira uniforme. Essa maneira uniforme não verifica as particularidades e a urgência de cada requisição (Teixeira, 2004).

Os objetivos do modelo SWDS são fornecer serviços diferenciados de acordo com a necessidade de cada usuário e requisitos de QoS previamente acertados.

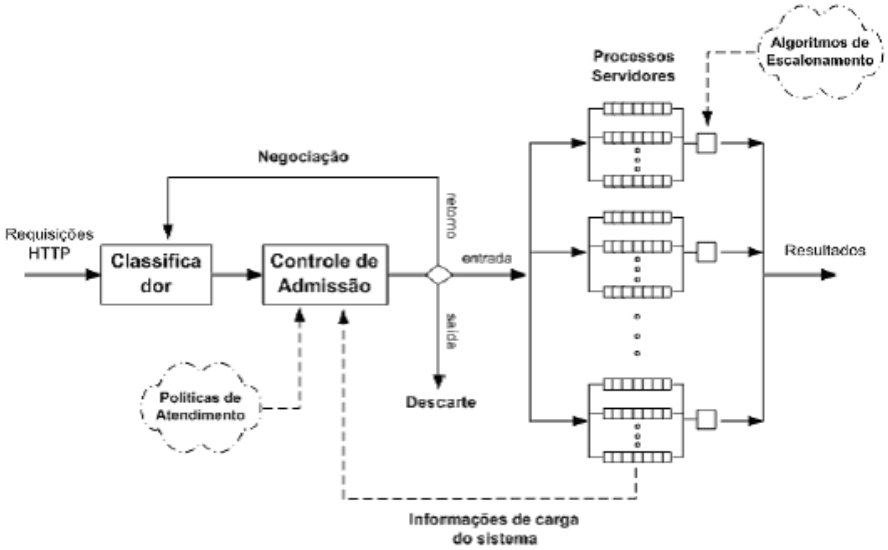


Figura 3.5: Módulos do SWDS(Teixeira, 2004).

A figura 3.5 apresenta a arquitetura do SWDS, onde podem ser observados os seguintes módulos:

- **Classificador**: Os clientes de um servidor *Web* têm diversas características com diferentes necessidades, com isso é necessário que as requisições *HTTP*, ao chegarem ao servidor *Web*, sejam subdivididas em classes de serviços. O componente responsável por receber as requisições e distribuí-las para as classes, seguindo regras pré-estabelecidas, é o classificador. Terminada esta fase, a requisição é alocada no sistema e recebe certo tratamento de acordo com a sua classe.

- **Controle de Admissão**: Assim que as requisições *HTTP* são devidamente classificadas, o módulo de controle de admissão é responsável por gerenciar a aceitação dessas requisições no servidor, levando em consideração as políticas de atendimento vigentes e as informações da carga de trabalho do sistema. Se o servidor estiver sobrecarregado, uma requisição poderá ser rejeitada (descarte), ou ter suas exigências de qualidade de serviço relaxadas (negociação), de modo que essa requisição possa ser aceita em uma classe de prioridade inferior. O controle de admissão é um módulo muito importante, pois é nele que se evita a sobrecarga do sistema, de modo a atender as exigências de *QoS*.

- ***Cluster* de Servidores *Web***: Depois que as requisições são admitidas pelo controle de admissão, o *cluster* de servidores *Web* fica responsável por atender as requisições de acordo com um algoritmo de escalonamento e enviar os resultados ao cliente que originou a solicitação.

Segundo (Teixeira, 2004), cada nó do *cluster* e o *Web switch* devem ser considerados como um servidor *Web* convencional composto por *CPU*, disco e interface de rede, já que o modelo não

exige uma arquitetura formada necessariamente por máquinas dispostas em um sistema distribuído. Assim, não é definida nenhuma plataforma de *hardware* para o funcionamento da arquitetura e nenhuma exigência quanto ao sistema operacional é especificada.

3.5 Mecanismos de Diferenciação de Serviços

Os mecanismos de diferenciação de serviços determinam como as requisições *HTTP* de um servidor *Web* serão atendidas de acordo com suas classes de serviço.

No trabalho de (Teixeira, 2004) foram implementados, utilizando simulações, três algoritmos de diferenciação de serviços, os quais podem ser divididos em duas categorias distintas de provisão de serviços diferenciados: enfileiramento baseado em classes (Reserva de Recursos) e escalonamento baseado em prioridade (Prioridade Rigoroso e Prioridade Adaptativo).

3.5.1 Reserva de Recursos

O algoritmo de reserva de recursos particiona o *cluster* de servidores *Web* e cada classe de requisições são associados a uma partição. Feito isso, cada classe terá uma parcela de processamento do servidor SWDS acarretando então o isolamento de desempenho entre elas.

Embora esse algoritmo seja adequado em alguns casos em que a carga de trabalho seja razoavelmente previsível e conhecida, para o ambiente da *Web* ele pode não ser bom, pois é um ambiente altamente dinâmico e imprevisível podendo fazer com que alguns recursos destinados a uma determinada classe fiquem sobrecarregados.

Em (Traldi et al., 2006) é proposto um algoritmo de Reserva Adaptativa de Recursos (RSVAdap). Esse algoritmo foi proposto com o intuito de adaptar o algoritmo de reserva de recursos para ambientes em que a carga é variável. Com isso o RSVAdap reserva os recursos do servidor sob demanda, de acordo com a carga de trabalho vigente.

Neste trabalho de mestrado é proposto um algoritmo parecido com o RSVAdap, porém com uma diferença, ao invés de alocar uma máquina para uma determinada classe e em um determinado momento, são alocadas somente as conexões necessárias para aquele momento, fazendo com que os recursos do servidor sejam melhores utilizados. No Capítulo 4 este algoritmo será mais detalhado.

3.5.2 Mecanismo de Prioridade Rigoroso

Este mecanismo utiliza prioridade nas filas do *cluster*, para que as requisições sejam atendidas por uma disciplina de prioridade rígida, ou seja, as requisições com maior prioridade são sempre atendidas primeiro.

Alguns resultados apresentados no trabalho de (Teixeira, 2004) mostram que, em diversas configurações de carga de trabalho e do servidor SWDS, o mecanismo de prioridade rigoroso permite

realizar a diferenciação de serviços de uma maneira simples e flexível. Porém, deve-se tomar cuidado com a atribuição de prioridade aos clientes, pois uma classe pode monopolizar o sistema fazendo com que clientes de prioridade mais baixa nunca sejam atendidos.

3.5.3 Mecanismo de Prioridade Adaptativo

O mecanismo de prioridade adaptativo proposto em (Teixeira et al., 2005) tem como objetivo flexibilizar a prioridade das requisições, definindo uma constante K (*look-ahead*), que dependendo do seu valor as requisições de prioridade alta terão mais ou menos importância. Este mecanismo funciona da seguinte maneira: as requisições que chegam são enfileiradas por ordem de chegada em cada nó do *cluster*, assim que o escalonador vai buscar uma requisição para ser atendida, ele percorre K posições na fila procurando por requisições da classe mais prioritária. Caso ele não encontre, o escalonador volta ao inicio da fila e começa a procurar requisições da classe imediatamente inferior e, assim sucessivamente.

Em outras palavras, a constante K (*look-ahead*) especifica o quão rigoroso será o esquema de priorização das requisições, fazendo com que clientes de alta prioridade não monopolizem o sistema (como ocorre no mecanismo anterior).

3.5.4 Cenários de Utilização

Existem vários cenários na *Web* onde é possível utilizar a abordagem de serviços diferenciados. Alguns deles serão mostrados a seguir:

- **Comércio eletrônico**: este cenário é um bom exemplo para se aplicar serviços diferenciados. Uma situação em que um usuário esteja na fase de conclusão do pedido e outro esteja somente navegando pelo *site* de comercio eletrônico, o usuário que está concluindo o seu pedido deveria ter uma importância maior do que o outro usuário, que está simplesmente navegando pelo site.

- **Sites de *downloads***: este ambiente também é propício para o uso de serviços diferenciados, pois usuários pagantes podem ter, por exemplo, uma taxa de transferência maior do que outro usuário comum, ou também usuários pagantes podem fazer *downloads* sem limites, enquanto outros clientes têm um limite diário de *downloads*.

Outros exemplos de aplicações que podem fazer o uso de serviços diferenciados são: *sites* acadêmicos, bancos, provedores de *Internet*, entre outros.

A próxima seção abordará o mecanismo de controle de admissão incluindo as características, funcionamento e a relevância do módulo de controle de admissão para o SDWS.

A seção de controle de admissão é muito importante, pois está ligada ao objetivo deste trabalho de mestrado, que é o estudo de algoritmos de controle de admissão para o servidor SWDS.

3.6 Controle de Admissão

O comportamento ideal de um servidor *Web* sobrecarregado é de se manter servindo requisições na sua capacidade máxima, mesmo quando a carga imposta é maior que a capacidade do servidor. Deve ser notado que qualquer aumento da carga causa um aumento no tempo de resposta. Isto resulta em uma situação em que usuários impacientes abandonam suas requisições, levando o servidor *Web* a processar requisições que já foram abandonadas pelos clientes (Sharifian et al., 2008). Estas requisições abandonadas consomem diversos recursos do servidor tais como processamento de conexão *TCP*, analisar e processar requisições, fazendo com que a produtividade do sistema caia drasticamente nas condições de sobrecarga, mesmo que a utilização dos recursos continue a 100%.

Para prevenir tais situações, um mecanismo de controle de sobrecarga deve ser incluído no algoritmo de balanceamento de carga. Diversos métodos como controle de admissão ou políticas de escalonamento especializadas (Sharifian et al., 2010) ou a combinação de ambos foram propostos.

Controles de admissão com mecanismos simples descartam algumas requisições somente quando a carga de trabalho do servidor começa a exceder seu limite. Porém nem todas as requisições podem ser tratadas de formas iguais, pois diferentes requisições têm diferentes impactos nos recursos do servidor *Web*. Por isso, essas diferenças são essenciais na implementação de um mecanismo de controle de admissão.

3.6.1 Mecanismos de Controle de Admissão

Os mecanismos de controle de admissão determinam o comportamento adotado pelo SWDS, quando novas solicitações são impostas sobre o mesmo.

No trabalho de (Teixeira, 2004) foram implementados três mecanismos de controle de admissão que são descritos a seguir:

- **Tamanho de filas**: pode-se dizer que esta abordagem é a mais simples, pois ela impõe um tamanho máximo para as filas dos servidores do *cluster*, recusando novas requisições quando este tamanho for excedido. Este mecanismo não oferece garantias de *QoS* aos clientes. No entanto a sua principal contribuição é diminuir a carga do sistema e consequentemente o tempo de resposta das requisições.

- **Tempo de resposta**: o objetivo desse mecanismo é tentar manter um tempo de resposta abaixo de certo limiar, utilizando *buffers* independentes para cada classe de serviços. As requisições de alta prioridade são sempre atendidas e as de baixa prioridade só são atendidas quando a carga de trabalho estiver abaixo de um limiar pré-estabelecido. Com este mecanismo, evita-se que o tempo de resposta apresente picos devido a aumentos na carga de trabalho, e com isso consegue-se uma maior estabilidade no sistema.

- **Utilização do sistema**: este mecanismo emprega a utilização média do *cluster* de servidores *Web* como métrica, a qual é calculada a partir de uma média exponencialmente ponderada e pode ser ajustada conforme valores atuais ou históricos da carga do sistema. Este mecanismo consegue manter a utilização do servidor SDWS, dentro de um limiar especificado de forma rigorosa. Isto permite oferecer um tratamento mais justo a todas as classes de requisições, fazendo com que solicitações que não podem ser atendidas, não entrem no sistema.

Alguns trabalhos relacionados ao mecanismo de controle de admissão de um servidor *Web* que provê diferenciação de serviços são citados a seguir:

- Em (Estrella, 2006) novas funcionalidades foram acrescentadas ao controle de admissão, sendo uma delas a capacidade de negociação. No modelo original proposto por (Teixeira, 2004), as requisições só passam pelo controle de admissão, sendo aceitas ou descartadas. Com a inclusão do mecanismo de negociação, requisições de classes de prioridade alta podem ter seus requisitos de *QoS* relaxados para serem aceitas no sistema.

- No trabalho de (Semprebom et al., 2006), além do mecanismo de controle de admissão, também foi empregado o conceito de escalonamento adaptativo por computação imprecisa. Com isso pode-se implementar páginas *Web* com duas versões: uma completa (precisa) e outra parcial (imprecisa). Um servidor projetado com esse modelo, em uma situação temporária de sobrecarga, pode responder a uma determinada requisição de um cliente com uma resposta parcial, que seja suficiente para o usuário saber que ele está sendo atendido, ainda que, com uma qualidade reduzida.

- No trabalho apresentado em (Lee et al., 2004) foram desenvolvidos dois algoritmos de controle de admissão. O primeiro algoritmo é o *Maximum Profit Algorithm (MPA)* que ordena os clientes de acordo com uma média máxima de tempo de espera do menor para o maior e depois dessa ordenação, assume-se que o cliente 1 tem a menor média máxima de tempo de espera e com isso ele deve receber melhor serviço do que clientes posteriores. O segundo algoritmo é o *Maximum Admission Algorithm (MAA)* que tem como objetivo admitir o maior número de requisições possíveis. A lógica desse algoritmo é que admitindo mais clientes, o serviço *Web* será mais popular e o provedor de conteúdo será capaz de gerar mais lucro ao longo do tempo.

- No trabalho de (Fu et al., 2008), foi desenvolvido um *framework* com controle de *QoS* para servidores *Web*. O *framework* é composto por 4 componentes no servidor *Web*: reconhecimento de estado, controle de admissão, classificação das requisições e escalonamento de filas.

  A classificação das requisições é baseada na prioridade dos clientes, conteúdos das páginas, e a prioridade da seção. Depois de classificadas as requisições são despachadas para diferentes filas de prioridade.

O controle de admissão do sistema é ativado quando este está com sintomas de sobrecarga. Para este trabalho foi proposta uma estratégia de controle de admissão denominada *Request Admission Feedback Control (RAFC)*. A estratégia garante que usuários de maior prioridade tenham seus requisitos de *QoS* alcançados. A finalidade do *RAFC* é aceitar requisições de alta prioridade primeiro e descartar as de baixa prioridade. O *RAFC* utiliza a carga real do servidor como parâmetros para regular dinamicamente seu mecanismo.

Para o escalonamento das filas, foi proposto o mecanismo *APSP (Adaptive Priority Scheduling Policy)*. O *APSP* tem como objetivo calcular uma prioridade levando em consideração muitos fatores como: o nível do conteúdo da mensagem, o tempo de chegada, o tempo de processamento, entre outros.

Os resultados foram obtidos através de simulação utilizando o *software Network Simulink*. Primeiramente, foram utilizados 3 clientes de diferentes prioridades e o algoritmo *FCFS*. Como resultado foi visto que o *FCFS* não tem ciência sobre a função de serviço e não pode assegurar que requisições de alta prioridade sejam executadas no tempo. Quando foi utilizado o algoritmo *APSP* para escalonar as requisições, foi observado que quando o servidor trabalha com 3 classes de prioridade, a classe mais prioritária consegue mais recursos do sistema para executar suas requisições.

- O trabalho de (Poggi et al., 2007) propõe uma técnica de aprendizado de máquina que consiste em estimar a probabilidade de usuários efetivarem compras em um site de e-commerce, utilizando dados passados de navegação. Esta técnica pode ser utilizada como um sistema de controle de admissão para priorizar sessões ou negá-las se não houver recursos disponíveis. Com isso é possível melhorar as vendas, pois clientes que compram mais têm mais prioridade sobre os que ficam a maioria do tempo somente navegando pelas páginas.

  Para concretizar o trabalho foi utilizado um *log* real de uma empresa de *e-commerce* (Atrapalo.com) e desenvolvida uma arquitetura denominada AUGURES, a qual implementa um preprocessamento que reescreve o *log* em um formato conveniente; um módulo onde se consegue separar os clientes em duas classes (compradores e não compradores); e um módulo que utiliza o pacote WEKA para aprendizado de máquina.

  Como resultado, foi comprovado que o AUGURES, utilizando aprendizado de máquina, pode admitir até 78% de clientes compradores contra 30% de uma estratégia randômica.

- O trabalho de (Sharifian et al., 2008) propõe um algoritmo de balanceamento de carga baseado em aproximação com um controle de admissão para servidores *Web* com cargas dinâmicas. Para desenvolver o trabalho foi criada uma arquitetura multicamada que é composta por servidores *Web*, servidores de aplicação, servidores de banco de dados, e um *Web-switch* na camada 7. O classificador, o algoritmo de balanceamento de cargas e o controle de admissão estão implementados no *Web-switch*. Quando uma requisição chega ao *Web-switch*, o classificador usa a *URL* requisitada para determinar qual a classe da requisição.

Assim que a requisição é classificada, ela entra em uma fila temporária correspondente à sua classe e o controle de admissão, utilizando a política *Round-Robin*, seleciona uma requisição de uma das filas e então decide aceitar ou não a requisição de acordo com um vetor de probabilidade adaptativa, o qual é atualizado pelo monitor que estima a carga do sistema.

O Controle de admissão só entra em funcionamento quando a média das cargas das *CPUs* está acima de 70%. As classes que terão maior prioridade em uma situação de sobrecarga são as que têm a maior fila e a menor demanda por *CPU*.

Depois que a requisição é aceita, o algoritmo de balanceamento de carga vai direcionar a requisição a um dos servidores tendo como base um algoritmo de balanceamento de cargas baseado em aproximação. Este algoritmo seleciona o servidor menos sobrecarregado para atender a requisição.

Foram estipuladas 5 classes de serviços em que requisições que demandam menos processamento são mais prioritárias que requisições que demandam mais processamento. Nos resultados finais pode-se observar que, utilizando os algoritmos de controle de admissão e balanceamento de cargas, consegue-se manter um *througput* superior e um tempo de resposta inferior aos algoritmos *WRR* e *CAP*.

3.7 Implementação da Arquitetura SWDS

A arquitetura SWDS foi implementada neste trabalho com base no trabalho de (Andreolini et al., 2004) e a sua organização está apresentada na Figura 3.6.

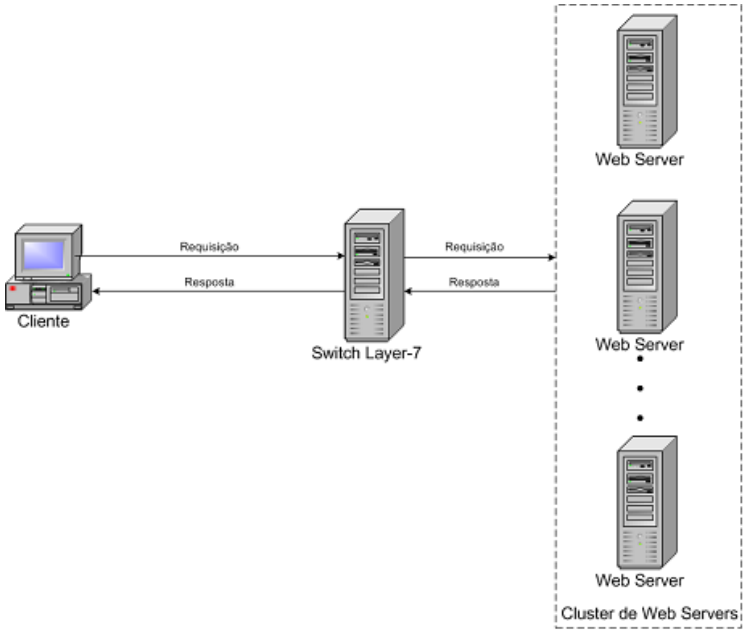


Figura 3.6: Arquitetura do SWDS.

Para implementar a arquitetura SWDS foram utilizados o Apache como *software* servidor e alguns módulos citados nas seções anteriores (mod_rewrite, mod_proxy e mod_status) (Foundation, 2010a), (Foundation, 2010b), (Foundation, 2010c).

A Figura 3.7, mostra o funcionamento da arquitetura SWDS implementada:

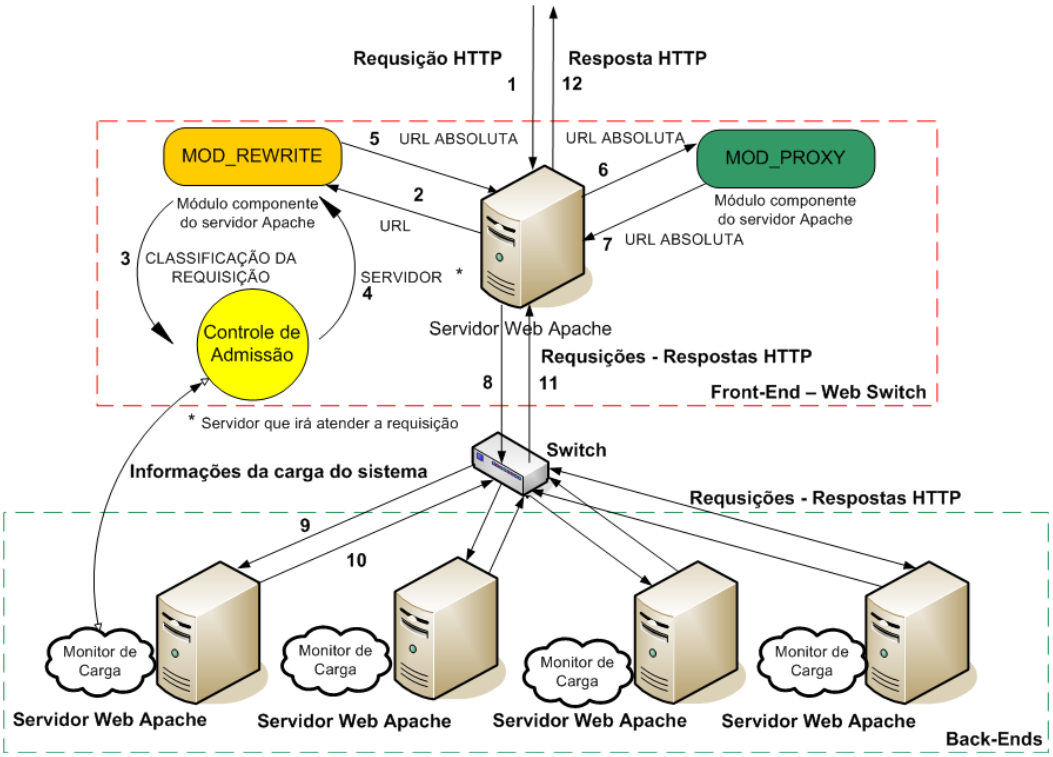


Figura 3.7: Arquitetura do SWDS implementada.

O servidor *Web* (Frontend) recebe a requisição do cliente (1) e em seguida repassa ao mod_rewrite (2). O mod_rewrite tem por função classificar a requisição no sistema de acordo com a URI do cliente (/1 Classe 1 e /2 Classe 2). Em seguita o mod_rewrite encaminha a requisição para um programa externo (3) chamado AdmControl, o qual é responsável pelo controle de admissão. O AdmControl possui 3 funções: a primeira é receber informações do número de conexões ocupadas no *Backend*, a segunda é realizar os cálculos dos algoritmos para aceitar ou não a requisição, e por fim, encaminhar a requisição ao *Backend* de acordo com a política implementada (4). São utilizados os manipuladores de arquivos stdin/stdout para a comunicação entre o mod_rewrite e o AdmControl. As informações de conexões ocupadas/livres são fornecidas com o auxílio do mod_status. O mod_status coleta informações de carga dos servidores (*Backends*) e repassa para o *Frontend* tilizando arquivos via *sockets*. Após a tomada de decisão, a requisição é repassada para o mod_proxy (5, 6) que posteriormente encaminhará a requisição para o servidor que será especificado por alguma política de escalonamento (7, 8, 9). Finalmente a requisição é atendida por um dos servidores e em seguida a resposta é despachada para o *Frontend*, o qual retorna a resposta ao cliente (10, 11, 12).

3.8 Considerações Finais

O objetivo deste Capítulo foi apresentar uma visão geral do comportamento de um servidor *Web* e um servidor *Web* distribuído. Para isso foi necessário relatar sobre o servidor *Web* Apache, que é o mais utilizado no mundo e que serviu de base para a execução deste trabalho de mestrado.

Também foi apresentada a arquitetura do SWDS e seus respectivos módulos, dando ênfase no controle de admissão. Por fim, foram apresentados alguns trabalhos relacionados a controle de admissão em servidores *Web* distribuídos.

No Capítulo 4 serão apresentado os algoritmos propostos e implementados na arquitetura SWDS.

CAPÍTULO

4

Algoritmos de Controle de Admissão Propostos

4.1 Considerações iniciais

Este Capítulo tem por objetivo descrever o Algoritmo de Negociação, o qual foi proposto e avaliado em nível de simulação no trabalho de (Estrella, 2006), e o Algoritmo de Reserva de Conexões que foi proposto neste trabalho de mestrado. Estes algoritmos foram implementados e testados no protótipo da arquitetura SWDS.

4.2 Algoritmo de Negociação

O algoritmo de negociação implementado neste trabalho de mestrado rebaixa os critérios de *QoS* sem o cliente saber desta negociação, ou seja, o cliente pode ser atendido em uma classe inferior sem saber que foi rebaixado. Esta solução é um atrativo, pois requisições que seriam descartadas em uma classe mais prioritária, têm outra chance de serem atendidas em uma classe menos prioritária, não descartando estas requisições em um primeiro instante.

Para a implementação deste algoritmo optou-se por duas classes de serviço, de modo a validar o algoritmo implementado, sendo a Classe 1 de maior prioridade e a Classe 2 de menor prioridade. Pode estender o número de Classes para N, porém é necessário um estudo bem detalhado para verificar o impacto do aumento de classes no desempenho da arquitetura SWDS.

O algoritmo de negociação oferece uma nova chance para que requisições da classe mais prioritária sejam readmitdas em uma classe menos prioritária, caso sua classe de serviço esteja sobrecarregada, com isso rebaixando os seus critérios de *QoS*.

Para que sejam novamente admitidas no sistema, as requisições devem retornar ao controle de admissão, e este, tomando como base o número de conexões simultâneas ocupadas, decide ou não a aceitação da requisição no sistema. A Figura 4.1 ilustra o funcionamento do algoritmo de negociação no protótipo do SWDS.

Como mencionado, existem duas classes de serviços: Classe 1 e Classe 2. As requisições de Classe 1 têm uma parte dos recursos computacionais (máquinas) alocados exclusivamente para elas , portanto as requisições de Classe 1 têm mais chances de serem admitidas no sistema em relação as requisições de Classe 2. As requisições de Classe 2 disputam o restante dos recursos com as requisições de Classe 1, que são negociadas para a Classe 2.

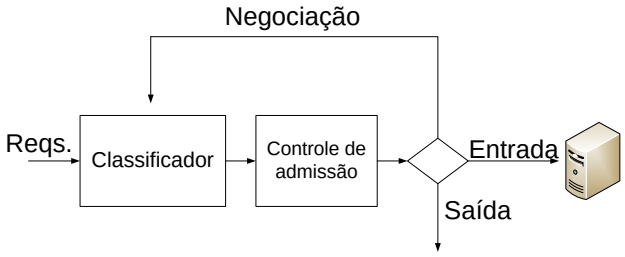


Figura 4.1: Algoritmo de Negociação.

O algoritmo funciona da seguinte forma: quando chega uma requisição de Classe 1, o controle de admissão a direciona para os recursos alocados à Classe 1. Caso estes recursos não estejam sobrecarregados, a requisição é aceita no sistema como Classe 1; por outro lado, se estes recursos já estiverem sobrecarregados, a requisição volta ao classificador e em seguida ela é marcada como uma requisição de Classe 2. Caso o conjunto de recursos da Classe 2 não esteja sobrecarregado, ela é aceita no sistema como Classe 2, rebaixando seus requisitos de QoS, senão é descartada. Com isso é possível observar que requisições de Classe 1 têm duas chances de serem admitidas no sistema e requisições de Classe 2 só podem ser aceitas ou descartadas, ou seja, elas só têm uma chance de serem admitidas. Esta característica da negociação pode ser observada na Figura 4.2.

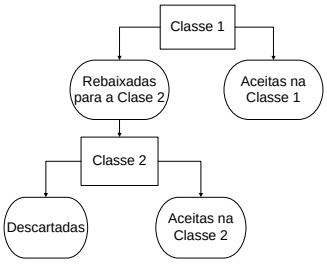


Figura 4.2: Fluxograma da Negociação.

O algoritmo de negociação implementado no protótipo deste trabalho de mestrado é melhor caracterizado na Figura 4.3.

```
int media-classe1, media-classe2;
media-classe1 = número de conexões classe 1 ocupadas;
media-classe2 = número de conexões classe 2 ocupadas;
    switch (classe)

        case 1:
            if (media-class1 < LIMIT)
                Escalona para recursos da classe 1;
            else if (media-classe2 < LIMIT)
                Escalona para recursos da classe 2;
            else
                Descarta;

        case 2:
            if (media-classe2 < LIMIT)
                Escalona para recursos da classe 2;
            else
                Descarta;
```

Figura 4.3: Pseudo-código do algoritmo de negociação

4.3 Algoritmo de Reserva de Conexões

O objetivo desta seção é apresentar o algoritmo proposto por este trabalho de mestrado, o qual é denominado algoritmo de reserva de conexões.

Um dos motivos para a implementação desta proposta no servidor SWDS é que, além de servir como um controle de admissão que tem por objetivo não deixar que o servidor fique sobrecarregado, ele traz para o controle de admissão parte da tarefa do escalonador de requisições, pois é possível alocar conexões para determinada classe dinamicamente.

Outra motivação é que, ao invés de alocar um recurso inteiro, (no caso uma máquina) pode-se ir mais a fundo e alocar somente as conexões necessárias para cada classe por um determinado tempo. Com isso é possível o melhor aproveitamento do cluster de servidores.

O algoritmo de reserva de conexões, como no algoritmo de negociação, utiliza duas Classes de serviços e funciona da seguinte forma: cada requisição que entra no sistema é contabilizada, e com isso é feito um cálculo para verificar a porcentagem de requisições de cada classe que está no sistema. A partir deste cálculo, a cada segundo, uma *thread* é disparada para fazer a reserva das conexões. As requisições de Classe 1 podem utilizar o sistema até o limite máximo de conexões disponíveis; já o limite de aceitação de requisições de Classe 2 é alterado de tempos em tempos de acordo com a carga que chega ao sistema. A Equação 4.1 representa a reserva de conexões:

$$LC2 = LIMIT * (1 - PC1 * (PR1/0,5)) \tag{4.1}$$

Sendo que:

- **LC2**: Quantas conexões a Classe 2 poderá utilizar concorrentemente com a Classe 1;

- **LIMIT**: Limite máximo de conexões simultâneas disponíveis no sistema;
- **PC1**: Porcentagem de conexões exclusivas da Classe 1 (de 0 a 1) no cenário inicial de 50% de carga para a Classe 1 e 50% de carga para a Classe 2. Este valor fica fixo para as demais cargas que chegarem ao sistema em um mesmo experimento;
- **PR1**: Porcentagem de requisições de Classe 1 que entram no sistema (de 0 a 1);

Como exemplo de execução do algoritmo, pode-se considerar os seguintes valores: um limite (LIMIT) de 600 conexões simultâneas no servidor SWDS, sendo 100 conexões simultâneas em cada servidor *Backend*. Para oferecer um melhor serviço à Classe 1, no cenário inicial de 50% de carga para a Classe 1(PR1) e 50% para a Classe 2, a porcentagem de conexões exclusivas da Classe 1 (PC1) é de 0,625, o restante é disputado pelas duas classes. Utilizando a Equação 1, é verificado que a Classe 2 pode utilizar as conexões simultâneas concorrentemente com a Classe 1 até o limite de 225 (LC2) conexões. Mudando a Carga para 75% para a Classe 1 e 25% para a Classe 2, a Classe 2 poderá utilizar o sistema até 38 (LC2) conexões simultâneas. E por fim se a carga da Classe 1 for de 25% e a carga da Classe 2 75%, a Classe 2 poderá atender requisições até o limite de 412 (LC2) conexões simultâneas.

Com o LC2 calculado, é necessário verificar se a carga do sistema não é maior que LC2. Se a carga é menor que LC2, a requisição de Classe 2 é aceita, senão é descartada.

Sendo a Classe 1 de maior prioridade, ela pode atender suas requisições até o limite máximo de conexões disponíveis, ou seja, até 600 conexões disponíveis (levando em conta o exemplo citado).

Pode-se destacar que o objetivo do algoritmo é priorizar requisições de Classe 1 reservando menos recursos para a classe de menor prioridade e fazendo com que requisições de Classe 1 dominem a maior parte do sistema. Este objetivo pode ser melhor visualizado na Figura 4.4.

Como pode ser observado na Figura 4.4, a primeira coluna representa uma carga de 50% para a Classe 1 e 50% para a Classe 2, a segunda coluna representa uma carga de 75% para a Classe 1 e 25% para a Classe 2 e a última coluna representa uma carga de 25% para a Classe 1 e 75% para a Classe 2. Pode-se observar nas três colunas que a Classe 2 utiliza as conexões simultâneas somente até o limite azul. A parte vermelha das colunas são as conexões reservadas exclusivamente para a Classe 1.

É importante ressaltar que esta explicação é para fins de exemplificação, e que o algoritmo pode alocar as conexões dinamicamente para qualquer tipo de carga de ambas as classes.

4.4 Considerações Finais

Este Capítulo apresentou os algoritmos de controle de admissão propostos para este trabalho de mestrado, suas características principais e o funcionamento detalhado.

No próximo Capítulo, será apresentado o ambiente de experimentos, mostrando as ferramentas utilizadas para geração de carga de trabalho e o tipo de aplicativo que é servido pelo SWDS.

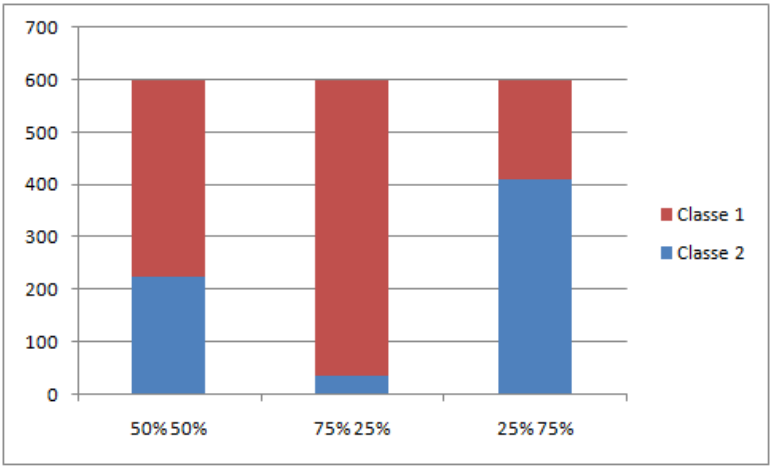


Figura 4.4: Ilustração do exemplo de reserva de conexões

CAPÍTULO

5

Ambiente de Experimentos

5.1 Considerações Iniciais

Este capítulo aborda quais os elementos necessários para a criação do ambiente de experimentos do SWDS, bem como *software* e *hardware* utilizados. Também é apresentado o gerador de carga de trabalho que foi utilizado para a realização dos testes e por fim é apresentado o tipo de aplicação que o *backend* utiliza para exercitar o SWDS.

5.2 Ambiente de Experimentos

Tão logo a arquitetura SWDS foi devidamente implementada, definiu-se um ambiente de testes para a avaliação da arquitetura, utilizando os algoritmos de controle de admissão descritos no Capítulo 4 desta dissertação.

Como pode ser observado, e já citado anteriormente em capítulos anteriores, a arquitetura SWDS é um protótipo, o qual possui uma máquina *frontend* e seis máquinas do grupo *backend*.

Avaliar algoritmos em protótipos pode ser interessante pelo fato do ambiente não ser tão controlado e assim tem mais influência de fatores externos, como por exemplo a latência da rede, ocupação dos servidores e outros, os quais acabam muitas vezes sendo ignorados ou mal quantificados em ambientes simulados.

Para avaliar a arquitetura com os algoritmos de controle de admissão propostos, foram consideradas duas classes de serviços: Classe 1, a mais prioritária, e Classe 2, a menos prioritária,

sendo que a Classe 1, independentemente da configuração de carga dos clientes, sempre terá mais recursos que a Classe 2.

São nove as máquinas que compõe o protótipo do SWDS, as quais são dispostas da seguinte forma: duas máquinas como cliente (uma máquina para cada classe), uma máquina *frontend*, que tem como função classificar, admitir e/ou descartar requisições, e por fim seis máquinas que compõem o grupo de *backend*, as quais têm por função processar as requisições que são enviadas ao grupo. A Figura 5.1 ilustra a arquitetura devidamente implementada para a execução dos testes.

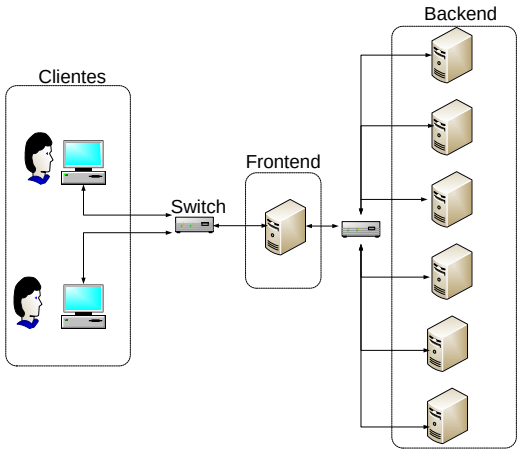


Figura 5.1: Ambiente de testes.

As máquinas utilizadas para a implementação da arquitetura SWDS são homogêneas, exceto o *frontend*, o qual possui uma configuração mais potente, pelo fato de ele ser o gargalo do sistema. A Tabela 5.1 apresenta as configurações de hardware e software das máquinas utilizadas.

Tabela 5.1: Configuração de Software e Hardware

Finalidade	Configuração de Hardware.	Configuração de Software
Clientes	Core 2 Quad Q9400@2.66GHz 4GB RAM	Ubuntu 9.04 HTTPerf
Frontend	Core 2 Quad Q6600@2.40GHz 8GB RAM	Ubuntu Server 9.04 Apache HTTPD v2.2.2
Backend	Core 2 Quad Q6600@2.40GHz 2GB RAM	Ubuntu Server 9.04 Apache HTTPD v2.2.2

Para a geração da carga de trabalho dos experimentos, utilizou-se uma ferramenta chamada Httperf (Mosberger e Jin, 1998). Esta ferramenta representa cargas de trabalho em rajadas, com distribuição exponencial, entre outras. Ela é utilizada em vários trabalhos científicos que pesquisam *Web cluster*, como em (Sharifian et al., 2008), (Sharifian et al., 2010), (Shan et al., 2002), (Shan et al., 2005).

Após testes preliminares utilizando o Httperf, foi observado que o tempo de resposta de requisições que foram descartadas era contabilizado e este tempo é muito próximo de zero. Com isso,

na etapa em que o Httperf gerava as estatísticas finas, podia-se notar que o tempo médio de resposta das requisições de Classe 2 eram bem menores que o tempo médio de resposta de requisições de Classe 1. Isso ocorria pois, para calcular o tempo médio de resposta, era feito uma somatória de todos os tempos de resposta e então dividia-se este valor pelo número total de requisições, fazendo com que requisições de Classe 2 tivessem um tempo médio de resposta bem menor, pelo fato de descartar mais requisições que a Classe 1 e, com isso contabilizar na somatória de tempo de resposta valores perto de zero.

Percebendo este detalhe, o código do Httperf foi alterado com o objetivo de contabilizar o tempo de resposta das requisições aceitas pelo sistema e assim desprezar o tempo de resposta das requisições que eram descartadas.

Para que o *cluster* de servidores pudesse ser exercitado, optou-se por utilizar uma página *Web* dinâmica, pois em testes preliminares foi observado que páginas estáticas eram mantidas em cache e não existia processamento. Com isso foi desenvolvida, em PHP, uma aplicação *CPU-Bound* e *IO-Bound*, a qual utiliza muito processamento e entrada e saída.

5.3 Configuração dos Algoritmos

Para a execução dos experimentos, foi necessária a configuração dos algoritmos para o ambiente de testes, no caso, o protótipo do SWDS.

O algoritmo de negociação foi configurado da seguinte forma: no grupo *backend* existem seis máquinas servidoras, as quais têm um limite máximo de conexões simultâneas, sendo que estes limites são variados ao longo dos experimentos.

Para escolher um número ideal de máquinas a serem reservadas exclusivamente para a Classe 1 durante os experimentos, foram realizados 3 experimentos alterando o número de máquinas exclusivas em 2,3 e 4 máquinas, respectivamente. A partir destes experimentos, foi possível verificar quantas máquinas reservadas para a Classe 1 seria ideal para o algoritmo de negociação.

Foram executadas três baterias de 3 experimentos, sendo que cada experimento era executado dez vezes para obter um intervalo de confiança de 95% de acordo com a tabela *T-Student*. As configurações de carga que foram utilizadas nestes experimentos podem ser observadas na seção 6.2 do Capítulo de Análise de Resultados e o limite de conexões simultâneas foi fixado em 400 para estes experimentos de escolha do número ideal de máquinas para a Classe 1.

Os gráficos que representam tempo médio de resposta podem ser analisados tomando como base os eixos X e Y. O eixo X corresponde às configurações de carga, ou seja, o primeiro grupo de colunas corresponde a uma carga de 25% de requisições para a Classe 1 e 75% de requisições para a Classe 2, o segundo grupo de colunas corresponde a 50% de carga para a Classe 1 e para a Classe 2 e por fim o último grupo de colunas apresenta uma carga de 75% para a Classe 1 e 25% para a Classe 2. O eixo Y corresponde ao tempo médio de resposta de cada requisição (em milissegundos).

É possível analisar nas Figuras 5.2, 5.5, 5.3, 5.6, 5.4 e 5.7 de tempo médio de resposta do primeiro *byte* e total do algoritmo de negociação, que os tempos seguem praticamente o mesmo comportamento, diferenciando somente quando a Classe 1 tem uma carga pequena. Com isso as requisições de Classe 1 terão um tempo médio de resposta do primeiro *byte* e total menor à medida que o número de máquinas exclusivas a esta Classe aumenta.

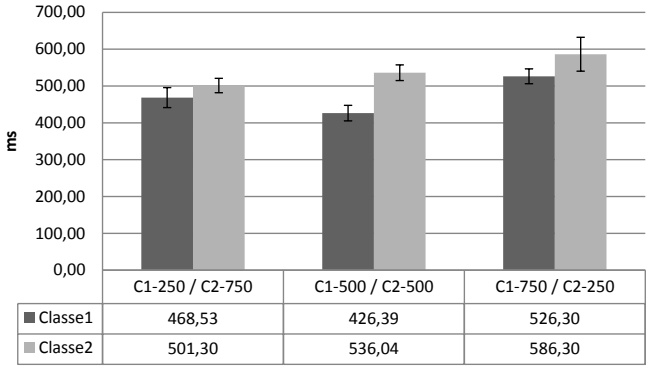


Figura 5.2: Tempo médio de resposta do primeiro byte - Negociação com 2 máquinas exclusivas para a Classe 1

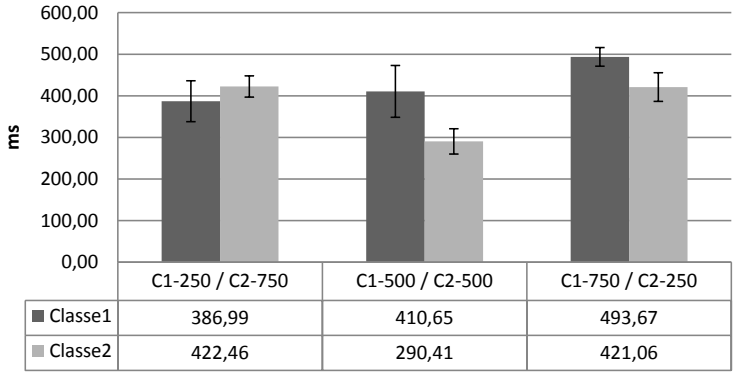


Figura 5.3: Tempo médio de resposta do primeiro byte - Negociação com 3 máquinas exclusivas para a Classe 1

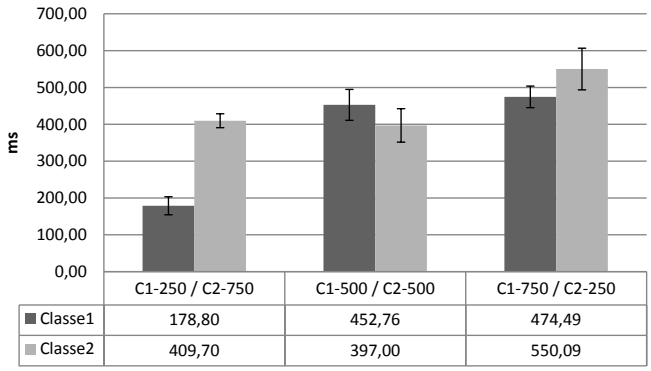


Figura 5.4: Tempo médio de resposta do primeiro byte - Negociação com 4 máquinas exclusivas para a Classe 1

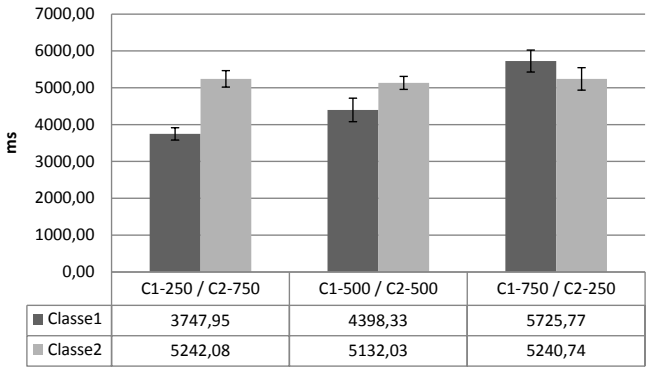


Figura 5.5: Tempo médio de resposta total - Negociação com 2 máquinas exclusivas para a Classe 1

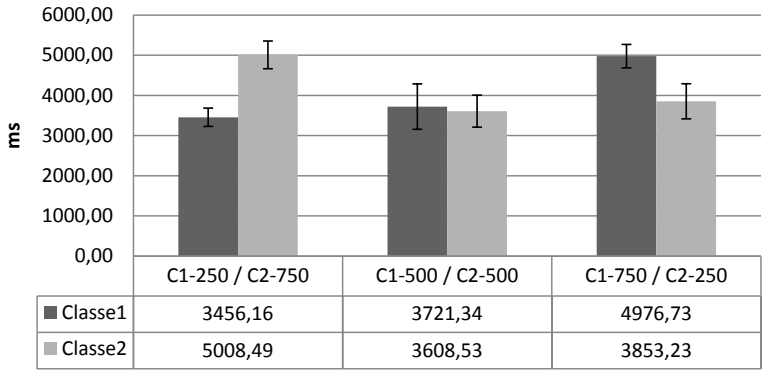


Figura 5.6: Tempo médio de resposta total - Negociação com 3 máquinas exclusivas para a Classe 1

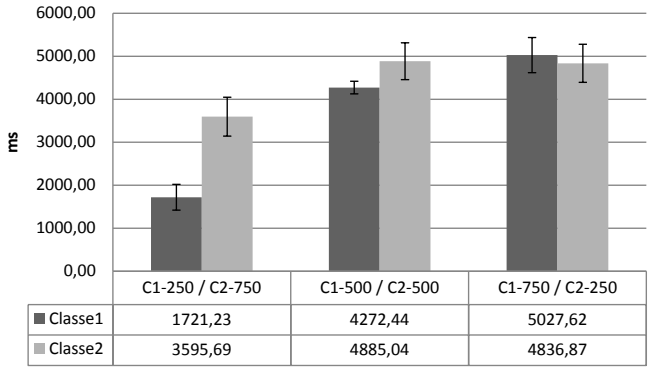


Figura 5.7: Tempo médio de resposta total - Negociação com 4 máquinas exclusivas para a Classe 1

Os gráficos que apresentam as porcentagens de requisições aceitas do algoritmo de negociação podem ser analisados da seguinte forma: a primeira e a segunda colunas correspondem a uma carga de 25% para a Classe 1 e de 75% para a Classe 2, a terceira e quarta colunas correspondem a uma carga de 50% para a Classe 1 e 50% para a Classe 2 e a quita e a sexta colunas correspondem a

uma carga de 75% para a Classe 1 e 25% para a Classe 2. Sendo que a barra vermelha corresponde às requisições que foram negociadas para a Classe 2.

Verificando a porcentagem de requisições aceitas no algoritmo de negociação, nota-se na Figura 5.8, a qual apresenta as requisições aceitas com duas máquinas exclusivas para a Classe 1, que a Classe 2 atende uma porcentagem razoavelmente grande de requisições em relação à Classe 1. Nota-se também que quando a carga da Classe 1 é igual a carga da Classe 2 e quando a carga da Classe 1 é maior que a carga da Classe 2 (terceira, quarta, quinta e sexta colunas), se não houvesse negociação, a Classe 2 teria uma porcentagem de requisições aceitas maior que a Classe 1. Portanto, reservar duas máquinas exclusivas para a Classe 1 não seria um grande atrativo.

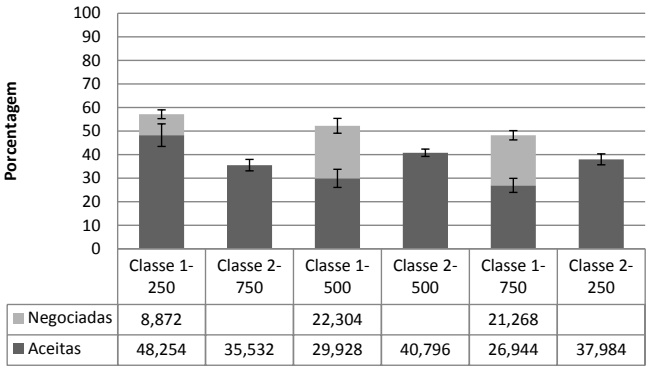


Figura 5.8: Porcentagem de requisições aceitas - Negociação com 2 máquinas exclusivas para a Classe 1

Ao analisar a Figura 5.9, a qual apresenta as requisições aceitas com 3 máquinas exclusivas para a Classe 1, nota-se que a Classe 1, em todas as configurações de carga, consegue atender uma porcentagem maior de requisições em relação ao experimento anterior. Isto ocorre porque a Classe 1 está com uma máquina a mais que o experimento anterior.

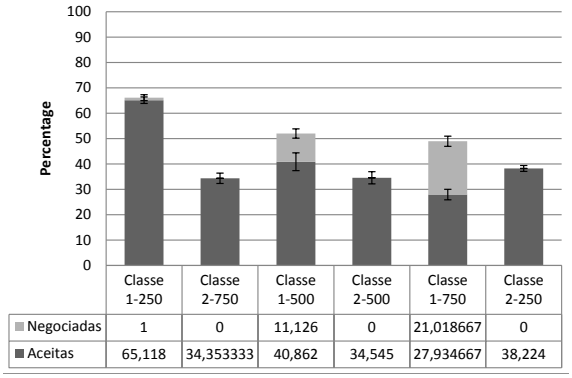


Figura 5.9: Porcentagem de requisições aceitas - Negociação com 3 máquinas exclusivas para a Classe 1

A Figura 5.10 apresenta as porcentagens de requisições aceitas com 4 máquinas para a Classe 1. Pode-se notar que, quando a Classe 1 tem uma carga pequena, uma parcela dos recursos com-

putacionais ficam subutilizados pelo fato da Classe 1 ter 4 máquinas disponíveis para poucas requisições (primeira e segunda colunas), fazendo com que a maioria das requisições de Classe 2 fossem descartadas.

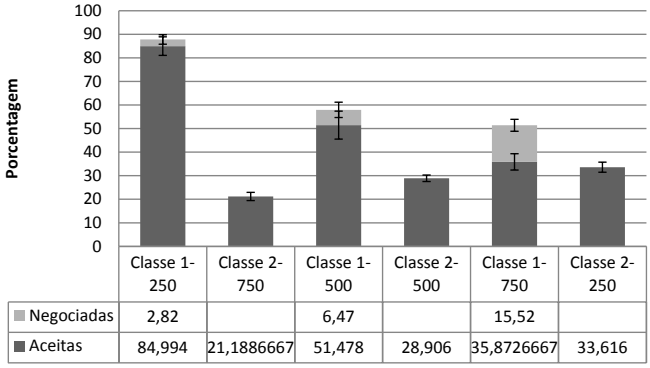


Figura 5.10: Porcentagem de requisições aceitas - Negociação com 4 máquinas exclusivas para a Classe 1

Após analisar os gráficos referentes a tempo médio de resposta e porcentagem de requisições aceitas, optou-se por fixar 3 máquinas para a Classe 1 no algoritmo de negociação durante todos os experimentos, pelo fato de não subutilizar os recursos da Classe 1 quando a carga desta Classe for pequena.

O algoritmo de reserva de conexões utiliza o mesmo ambiente de experimentos do algoritmo de negociação, só que com uma diferença: não existe máquina reservada para a Classe 1, e sim conexões exclusivas ao longo do experimento.

Para saber qual a melhor porcentagem de PR1, que foi definido na Seção 4.3, foram feitas três baterias de experimentos, as quais foram fixadas para PR1 50%, 62,5% e 75%, respectivamente. Como no algoritmo de negociação, o limite de conexões simultâneas foi fixado em 400 e cada bateria de experimentos foi executada dez vezes para obter um intervalo de confiança de 95% de acordo com a tabela *T-Student*.

De modo geral, as Figuras 5.11, 5.14, 5.12, 5.15, 5.13, 5.16 mostram o tempo médio de resposta do primeiro *byte* e total das requisições que foram aceitas no sistema utilizando o algoritmo de reserva de conexões. Nota-se que o tempo médio de resposta do primeiro *byte* acompanha o tempo médio de resposta total em praticamente todas as configurações de cargas dos experimentos.

É possível observar que, à medida que o valor de PR1 aumenta, o tempo médio de resposta da Classe 2 tende a ser menor que o da Classe 1 quando se tem uma carga pequena para a Classe 2. Isto ocorre pelo fato da Classe 2 disputar cada vez menos com as requisições de Classe 1 justamente pelo aumento do PR1.

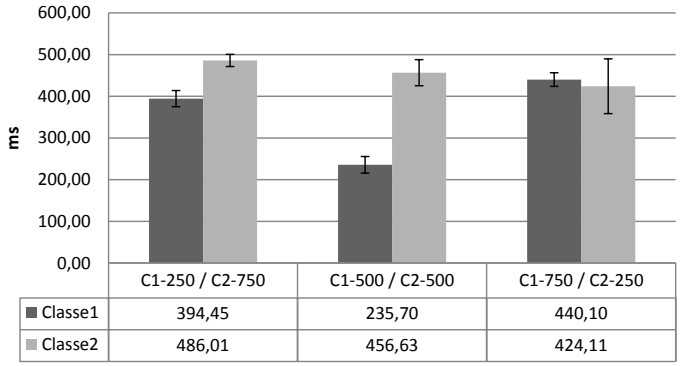


Figura 5.11: Tempo médio de resposta do primeiro byte - Reserva de conexões com PR1 fixado em 50%

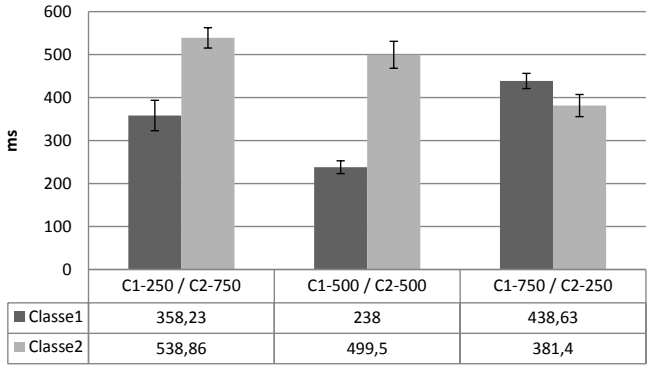


Figura 5.12: Tempo médio de resposta do primeiro byte - Reserva de conexões com PR1 fixado em 62,5%

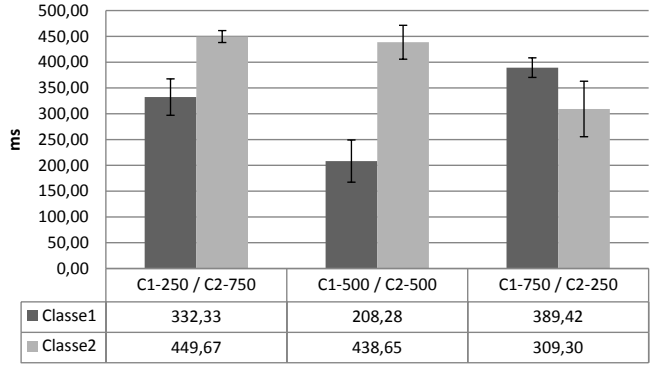


Figura 5.13: Tempo médio de resposta do primeiro byte - Reserva de conexões com PR1 fixado em 75%

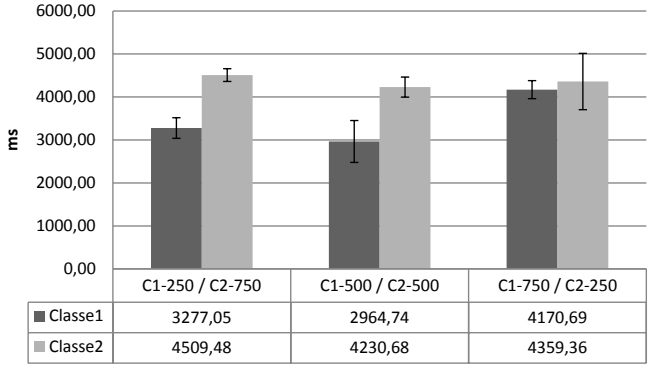


Figura 5.14: Tempo médio de resposta total - Reserva de conexões com PR1 fixado em 50%

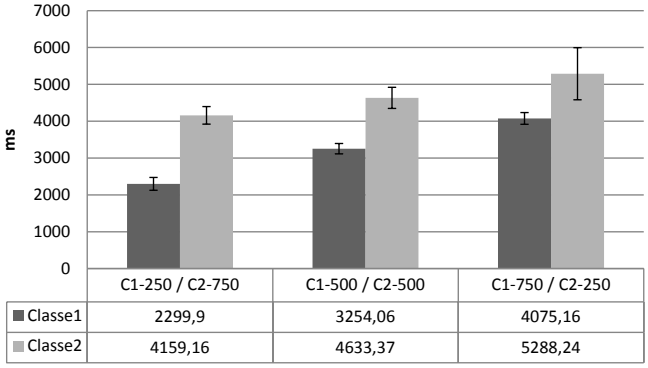


Figura 5.15: Tempo médio de resposta total - Reserva de conexões com PR1 fixado em 62,5%

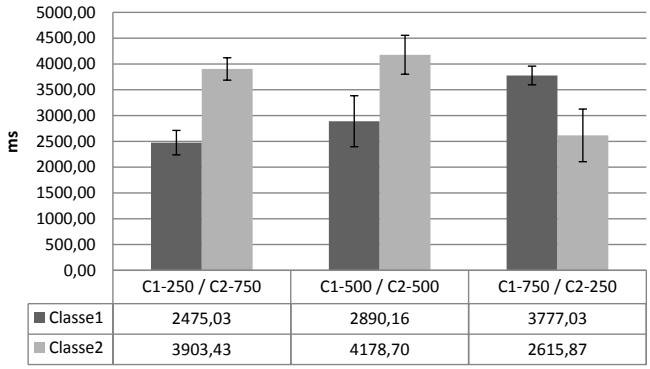


Figura 5.16: Tempo médio de resposta total - Reserva de conexões com PR1 fixado em 75%

Analisando a porcentagem de requisições aceitas pelas duas Classes na Figura 5.17, pode-se notar que quando o PR1 é fixado em 50% e a carga da Classe 1 é pequena (primeiro grupo de colunas), a porcentagem de requisições aceitas de Classe 2 fica próxima da porcentagem de Classe 1. Isto ocorre pelo fato do servidor alocar poucas conexões exclusivas para a Classe 1, fazendo com que a Classe 2 dispute mais conexões no servidor.

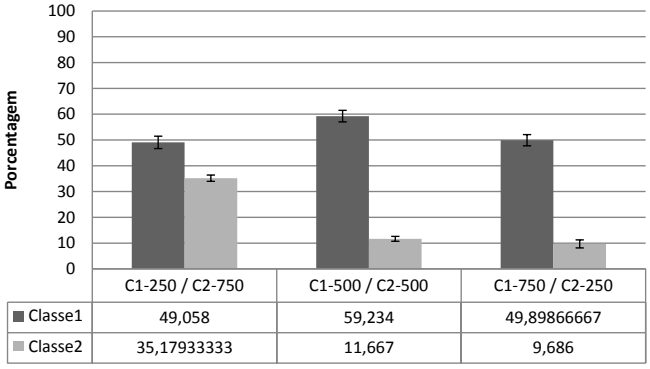


Figura 5.17: Porcentagem de requisições aceitas - Reserva de conexões com PR1 fixado em 50%

A Figura 5.18 apresenta a porcentagem de requisições aceitas com PR1 fixado em 62,5%. É possível notar que com o PR1 maior, uma porcentagem maior de requisições de Classe 1 são atendidas, se relacionado com o experimento anterior. Isto ocorre, pois quanto maior o PR1, mais conexões simultâneas são alocadas exclusivamente para a Classe 1.

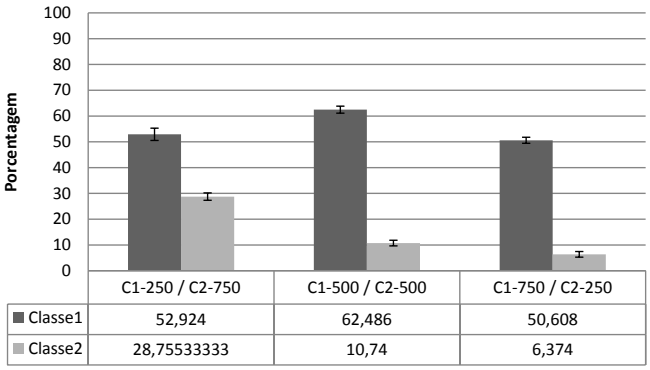


Figura 5.18: Porcentagem de requisições aceitas - Reserva de conexões com PR1 fixado em 62,5%

A Figura 5.19 segue a tendência da Figura 5.18, somente aumentando o atendimento de requisições de Classe 1 e diminuindo o atendimento da Classe 2.

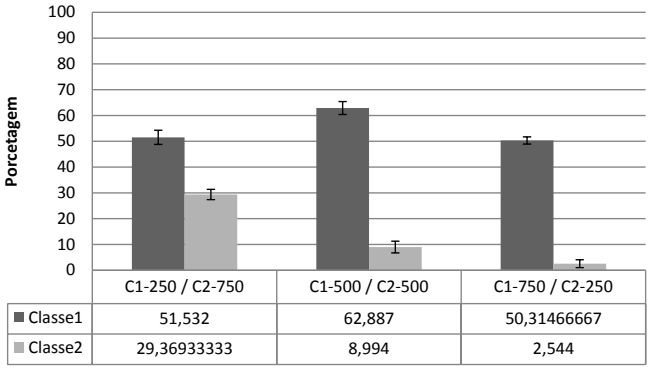


Figura 5.19: Porcentagem de requisições aceitas - Reserva de conexões com PR1 fixado em 75%

A partir da análise destes experimentos modificando o valor de PR1 para o algoritmo de reserva de conexões, optou-se por fixar PR1 em 62,5%, pois com o PR1 fixado em 50%, muitas requisições de Classe 1 eram descartadas quando a carga da Classe 2 era grande e com o PR1 fixado 75%, quase não existia atendimento de requisições de Classe 2 quando a carga da Classe 1 era grande, pelo fato das requisições de Classe 1 dominarem o sistema.

5.4 Considerações Finais

Este Capítulo apresentou os tipos e as configurações de *software* e *hardware* utilizados na construção do ambiente de experimentos. Também foi citada a utilização da ferramenta denominada Httperf, a qual tem por objetivo gerar cargas de trabalho para servidores *Web*. Foi apresentado o tipo de aplicação que foi usada nos experimentos. E por fim foi mostrado uma análise para a escolha dos números ideais para a configuração dos algoritmos.

No próximo Capítulo serão apresentados o planejamento de experimentos e a análise dos resultados obtidos.

CAPÍTULO

6

Análise de Resultados

6.1 Considerações Iniciais

Este capítulo aborda o planejamento de experimentos, a avaliação de desempenho e a influência dos fatores da arquitetura SWDS utilizando os algoritmos descritos no capítulo 4 desta monografia. O objetivo principal deste capítulo é mostrar o comportamento dos algoritmos de controle de admissão propostos.

6.2 Planejamento de Experimentos

Os experimentos foram planejados como pode ser observado na Tabela 6.1, onde o fator **Algoritmo** corresponde aos algoritmos propostos neste trabalho, o fator **Limite** corresponde ao limite máximo de conexões simultâneas que podem ser utilizadas em cada máquina do grupo *backend*, e por fim o fator **Carga** corresponde à porcentagem de carga submetida para cada classe.

A carga de trabalho corresponde a um montante de 20.000 requisições enviadas ao servidor em uma rajada de 1.000 requisições por segundo, ou seja, quando é enviada uma carga de 50% de requisições para a Classe 1 e 50% de carga para a Classe 2, quer dizer que estão sendo enviadas 10.000 requisições com rajadas de 500 requisições por segundo para a Classe 1 e 10.000 requisições com rajadas de 500 requisições por segundo para a Classe 2. Já quando é enviada uma carga de 75% de carga para a Classe 1 e 25% de carga para a Classe 2, pode-se entender que estão sendo enviadas 15.000 requisições com rajadas de 750 requisições por segundo para a Classe 1 e 5.000 requisições com rajadas de 250 requisições por segundo para a Classe 2, e assim sucessivamente.

Tabela 6.1: Fatores e níveis do planejamento de experimentos

Fatores	Níveis.
Algoritmo	algoritmo de negociação algoritmo de reserva de conexões
Limite	400 600 800 1000 1200
Carga	50% reqs. Classe 1 e 50% reqs. Classe 2 75% reqs. Classe 1 e 25% reqs. Classe 2 25% reqs. Classe 1 e 75% reqs. Classe 2

A metodologia escolhida para a realização dos experimentos foi fatorial completo, ou seja, foram combinados todos os níveis do fator **Algoritmo** com todos os níveis do fator **Limite** e com todos os níveis do fator **Carga**, totalizando trinta experimentos. Cada combinação de experimentos foi executada dez vezes de modo a obter um intervalo de confiança de 95% das variáveis de resposta.

Como variáveis de resposta para a análise dos resultados, foram escolhidos o tempo médio de resposta do primeiro *byte* de cada requisição, o tempo médio de resposta total de cada requisição e a porcentagem de requisições aceitas no sistema.

6.3 Avaliação Comportamental

Nesta seção, serão apresentados e analisados os resultados obtidos dos experimentos planejados na seção 6.2.

As subseções 6.3.1 e 6.3.2 apresentam os resultados referentes ao algoritmo de negociação e algoritmo de reserva de conexões respectivamente, mostrando os tempos médios de resposta do primeiro byte e os tempos médios de resposta total de requisições de Classe 1 e Classe 2. Também serão apresentadas as porcentagens de requisições aceitas e negociadas.

6.3.1 Algoritmo de Negociação

O algoritmo de negociação aqui analisado foi configurado com três servidores para a Classe 1 e três servidores para a Classe 2. Os limites de conexões simultâneas para cada servidor variam entre 400, 600, 800, 1.000, 1.200, e as cargas impostas nos experimentos são: 25% para a Classe 1 e 75% para a Classe 2, 50% para a Classe 1 e 50% para a Classe 2 e 75% para a Classe 1 e 25% para a Classe 2.

As seguintes subseções apresentam a análise dos resultados.

Negociação com Limite de 400 Conexões Simultâneas

Analisando os tempos médios de resposta do primeiro *byte* e total, notou-se que o comportamento do tempo do primeiro *byte* acompanhava o tempo de resposta total das requisições. Isto ocorreu pelo fato da página requisitada pelos clientes ser dinâmica, ou seja, necessitava de processamento por parte do servidor para entregar a resposta. Com isso media-se um tempo quando o primeiro *byte* chegava ao cliente e media-se o tempo de chegada do último *byte*.

A Figura 6.1 e a Figura 6.2 apresentam os tempos médios de resposta do primeiro *byte* e total, respectivamente, com um limite de conexões simultâneas fixado em 400. É possível observar que o tempo de resposta do primeiro *byte* é menor que o tempo de resposta total. Isto ocorre pelo fato da página requisitada ser dinâmica, necessitando de um tempo para ser processada no servidor. Nota-se também que o comportamento de ambos os tempos médios de resposta têm comportamentos parecidos. Com isso, para a análise dos resultados, utilizou-se somente o tempo médio de resposta total, pelo fato de seu comportamento ser parecido com o tempo médio de resposta do primeiro *byte*.

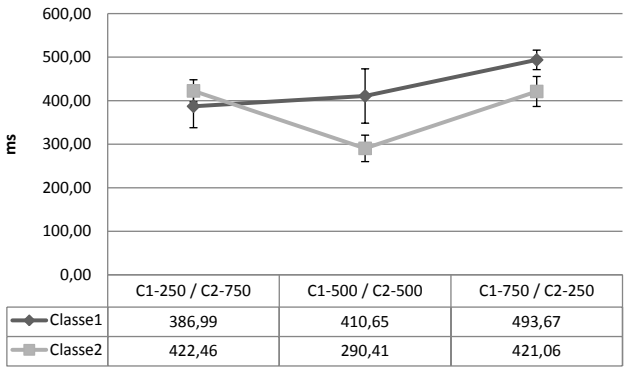


Figura 6.1: Tempo médio de resposta do primeiro byte - negociação - limite 400.

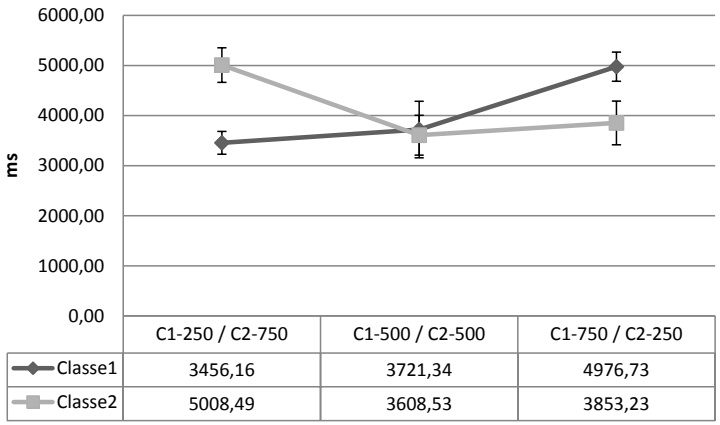


Figura 6.2: Tempo médio de resposta total - negociação - limite 400.

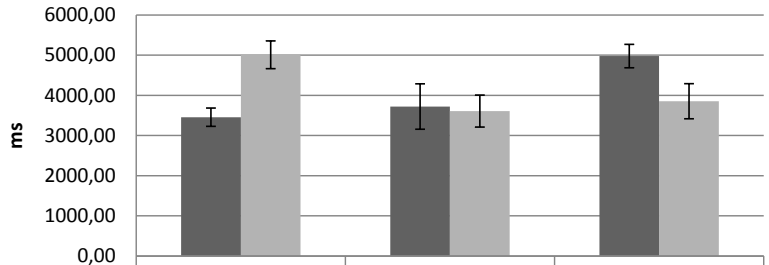
O gráfico da Figura 6.3 contém os tempos médios de resposta total de cada requisição atendida da Classe 1 e da Classe 2, com um limite de 400 conexões simultâneas para cada servidor do *backend*.

O primeiro grupo de colunas corresponde a 25% de carga para a Classe 1 e 75% de carga para a Classe 2, o segundo grupo de colunas corresponde a 50% de carga para a Classe 1 e 50% de carga para a Classe 2 e o terceiro grupo de colunas corresponde a 75% de carga para a Classe 1 e 25% de carga para a Classe 2. Este padrão se manterá para todas as análises de tempo médio de resposta.

Pode-se observar no primeiro grupo de colunas da Figura 6.3 que o tempo médio de resposta total da Classe 1 é menor que o tempo médio de resposta da Classe 2. Isso pode ser explicado pelo fato da Classe 1 ter um número de requisições menor no sistema (Classe 1 com 25% de carga e Classe 2 com 75%), fazendo com que o tempo de resposta destas requisições seja menor que o tempo de resposta das requisições da Classe 2.

No segundo grupo de colunas da Figura 6.3, verifica-se que o tempo de resposta das duas classes é estatisticamente igual, pois o intervalo de confiança da Classe 1 sobrepõe a média da Classe 2 e vise-versa. Isso ocorre, pois a carga imposta para cada classe é de 50%, e também porque ocorre negociação, como pode ser verificado na Figura 6.4. Quando ocorre a negociação para a Classe 2, o tempo médio de resposta total das requisições de Classe 2 aumenta, pelo fato dos recursos da Classe 2 serem compartilhados com a Classe 1.

Por último, no terceiro grupo de colunas da Figura 6.3, observa-se que o tempo médio de resposta da Classe 1 é maior que o tempo médio da Classe 2. Isto ocorre por consequência da negociação, pois a carga da Classe 1 é bem maior em relação à carga da Classe 2 (75% de carga para a Classe 1 e 25% de carga para a Classe 2). É importante notar que apesar da carga da Classe 2 ser pequena, o seu tempo médio de resposta foi relativamente grande se comparado com a Classe 1, pois as requisições que são negociadas para a Classe 2 disputam recursos com requisições de Classe 2, fazendo com que o tempo médio de resposta total aumente.



	C1-250 / C2-750	C1-500 / C2-500	C1-750 / C2-250
Classe1	3456,16	3721,34	4976,73
Classe2	5008,49	3608,53	3853,23

Figura 6.3: Tempo médio de resposta total - negociação - limite 400.

A Figura 6.4 mostra a porcentagem de requisições aceitas de Classe 1 e Classe 2, sendo que a primeira e a segunda coluna correspondem a uma carga de 25% para a Classe 1 e de 75% para a

Classe 2, a terceira e quarta coluna correspondem a uma carga de 50% para a Classe 1 e 50% para a Classe 2 e a quita e a sexta coluna correspondem a uma carga de 75% para a Classe 1 e 25% para a Classe 2. Este padrão se mantém para todas as figuras que representam requisições aceitas do algoritmo de negociação.

Pode-se verificar na Figura 6.4 que quando se tem uma carga de 25% para a Classe 1 e 75% para a Classe 2 (primeira e segunda coluna), estatisticamente não existe negociação. Isto se deve ao fato do limite de conexões simultâneas ser pequeno, fazendo com que os recursos da Classe 2 fiquem muito sobrecarregados com 75% de carga, não sobrando espaço para negociação.

Quando são analisadas as terceira e quarta colunas, verifica-se que, com cargas iguais para as duas classes, ocorre 11,12% de negociação para a Classe 2. Isto significa que 11,12% de requisições que iriam ser descartadas foram atendidas na Classe 2, pelo fato do limite de conexões simultâneas ser pequeno. Com isso, assim que os recursos da Classe 1 se encontram sobrecarregados, acontece a negociação.

Nas duas últimas colunas da Figura 6.4, pode-se notar uma porcentagem muito grande de requisições negociadas para a Classe 2. Como a carga submetida à Classe 1 é maior em relação a carga da Classe 2, os recursos da Classe 2 ficam menos ocupados que os recursos exclusivos da Classe 1, ocorrendo a negociação de requisições da Classe 1 para a Classe 2.

É importante salientar que proporcionalmente, a Classe 1, em todas as configurações de carga, atende mais requisições que a Classe 2, fornecendo para clientes de Classe 1 uma maior qualidade de serviço.

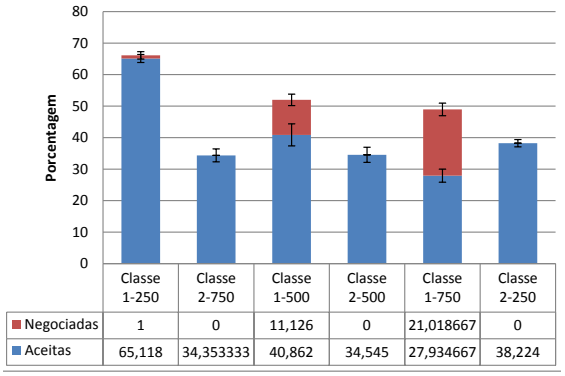


Figura 6.4: Porcentagem de requisições aceitas - negociação - limite 400.

Negociação com Limite de 600 Conexões Simultâneas

Analisando agora os experimentos do algoritmo de negociação com limite de 600 conexões simultâneas, a Figura 6.5 apresenta o tempo médio de resposta total para ambas as classes. Pode-se notar que o tempo médio de resposta total para um limite de 600 conexões simultâneas é maior quando comparado com um limite de 400 conexões por segundo (Figura 6.3). Isto ocorre, pois o processamento de 600 conexões simultâneas leva mais tempo e processamento que 400 conexões simultâneas.

Na Figura 6.5 pode-se observar que a carga influencia em todos os tempos médios de resposta das Classes. Quando se tem carga pequena para a Classe 1, o tempo da Classe 1 fica menor que o da Classe 2, quando se tem carga pequena para a Classe 2, o tempo de resposta da Classe 2 fica menor que o da Classe 1 e finalmente quando a carga é igual para ambas as classes, apesar de os tempos de resposta serem quase iguais, não se pode afirmar nada, pois os seus intervalos de confiança se sobrepõem. Também pode-se notar que o tempo médio de resposta da Classe 2 aumenta em relação à carga imposta a ela durante os experimentos. Este comportamento é obtido pelo fato de se ter alocado três recursos para cada classe, podendo haver a negociação da Classe 1 para a Classe 2.

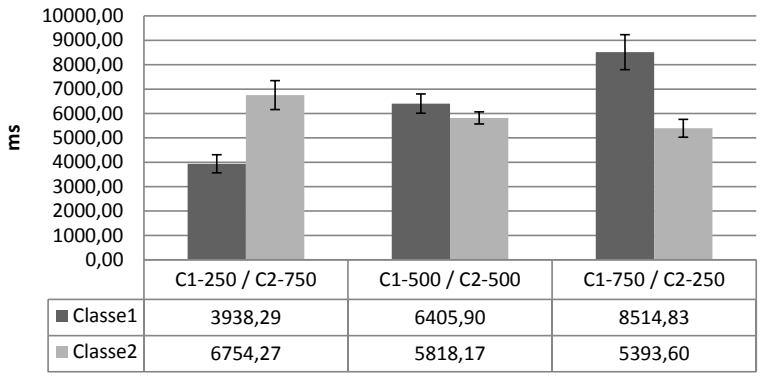


Figura 6.5: Tempo médio de resposta total - negociação - limite 600.

Na figura 6.6, pode-se verificar que houve uma maior porcentagem de requisições aceitas pelas duas Classes. Isto ocorreu pelo mesmo motivo do aumento do tempo médio de resposta, que é o aumento do número máximo de conexões. Quanto maior o número de conexões simultâneas, maior é o tempo médio de resposta e a porcentagem de requisições atendidas.

Analisando detalhadamente a Figura 6.6, pode-se notar que ocorre negociação quando a carga imposta ao sistema é de 25% para a Classe 1 e 75% para a Classe 2. Os motivos que levam à negociação é que o limite de conexões simultâneas aumenta e que a carga se mantém (em relação a primeira e segunda coluna da Figura 6.4), fazendo com que a capacidade de atender requisições também aumente nas duas classes, e com isso em determinados momentos ocorre a negociação.

Verificando as terceira e quarta colunas da Figura 6.6, nota-se um ligeiro aumento na negociação em relação à Figura 6.4. Isto é devido ao número de conexões simultâneas ter aumentado e com isso aumentando o número de requisições aceitas tanto na Classe 1 como na Classe 2. É importante ressaltar que, apesar do limite de conexões ter aumentado, a carga imposta ao servidor ainda é grande, fazendo com que haja descarte e negociação.

Na quinta e sexta coluna da Figura 6.6 é observado que, se não houvesse a negociação, a Classe 2 poderia ter mais requisições atendidas, pelo fato da carga da Classe 1 ser maior que a da Classe 2 e o número de recursos reservado para cada classe ser o mesmo. Porém quando acontece a negociação, pode-se notar que 14,01% de requisições de Classe 1 são executadas na Classe 2 e assim aumentando o número de requisições aceitas da Classe 1 para 51,9%.

Finalmente, percebe-se que em todas as configurações de carga, a Classe 1 é melhor que a Classe 2 quando se analisa a porcentagem de requisições aceitas, fazendo com que usuários de Classe 1 tenham uma maior probabilidade de serem atendidos.

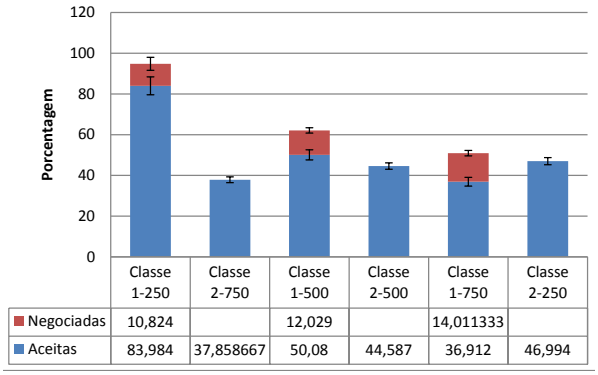


Figura 6.6: Porcentagem de requisições aceitas - negociação - limite 600.

Negociação com Limite de 800 Conexões Simultâneas

A Figura 6.7 apresenta os tempos médios de resposta total para todas as variações de carga com limite de 800 conexões simultâneas para cada servidor do *backend*.

No primeiro grupo de colunas da Figura 6.7, verifica-se um tempo médio de resposta menor para a Classe 1, pois a carga da Classe 1 se torna menor conforme o limite de conexões simultâneas aumenta, fazendo com que a Classe 1 tenha um tempo de resposta inferior comparando com a Classe 2.

O segundo grupo de colunas mantém a igualdade no tempo médio de resposta das duas Classes, pois seus intervalos de confiança sobrepõem as médias. Isto ocorre pelo fato da carga ser igual e o limite máximo de conexões ser maior, com isso a carga imposta no experimento faz as duas classes se sobrecarregarem igualmente.

Por fim, o terceiro grupo de colunas da Figura 6.7 mostra que o tempo médio de resposta da Classe 1 fica maior que o tempo da Classe 2 e o tempo médio de resposta da Classe 2 aumenta proporcionalmente em relação à carga (o tempo aumenta com a carga menor). Isto ocorre, pois a negociação é feita com mais frequência, fazendo a Classe 2 compartilhar seus recursos com a Classe 1 e com isso descartar mais requisições de Classe 2.

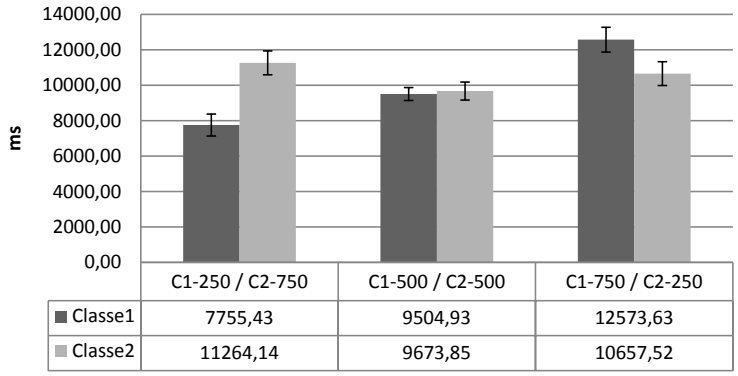


Figura 6.7: Tempo médio de resposta total - negociação - limite 800.

A Figura 6.8 apresenta porcentagem de requisições aceitas utilizando o algoritmo de negociação com limite de 800 conexões simultâneas em cada servidor.

Na primeira e na segunda coluna, pode-se notar que a Classe 1 negociou 9,64% de requisições com a Classe 2, fazendo com que quase todas as requisições fossem atendidas (95,33%). Isto ocorre, pois o limite de conexões simultâneas aumenta para 800, disponibilizando mais recursos para ambas as classes, porém aumentando o tempo médio de resposta.

Observando a terceira e quarta coluna da Figura 6.8, verifica-se que a porcentagem de requisições de Classe 1 que foram atendidas na Classe 1 é estatisticamente igual a Classe 2. Ocorre menos negociação que a Figura 6.6 por um fato interessante: a partir do limite de 800 conexões simultâneas, quando se tem cargas iguais para as duas classes, a tendência é diminuir o número de requisições negociadas. Isto se deve ao fato de demorar mais para os grupos de máquinas das duas classes se sobrecarregarem, ou seja, menos requisições de Classe 1 irão disputar recursos com a Classe 2.

Nas duas últimas colunas da Figura 6.8, observa-se que a Classe 1 fica sobrecarregada, forçando a negociação com a Classe 2. Verifica-se também que se não houvesse negociação, a Classe 2 teria uma porcentagem maior de requisições aceitas no sistema.

Novamente é possível observar que a Classe 1, proporcionalmente, tem um melhor atendimento de requisições de que a Classe 2 se for levado em conta a porcentagem de requisições aceitas no sistema.

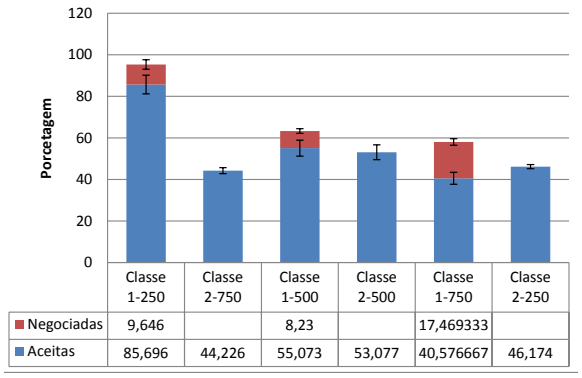


Figura 6.8: Porcentagem de requisições aceitas - negociação - limite 800.

Negociação com Limite de 1000 Conexões Simultâneas

A Figura 6.9 apresenta o tempo médio de resposta total de ambas as classes, com todos os níveis de carga, com limite de 1000 conexões simultâneas.

É observado que em relação a Figura 6.7, os tempos médios de resposta são maiores para as Classes. Isto é decorrente do limite de conexões simultâneas ser maior.

No primeiro grupo de colunas da Figura 6.9, pode-se observar um tempo médio de resposta menor da Classe 1 em relação à Classe 2. Isto ocorre pelo fato da Classe 1 ter uma carga menor, porém com a mesma quantidade de recursos que a Classe 2 e com a vantagem de poder fazer negociação.

O terceiro grupo de colunas mostra um tempo médio de resposta para a Classe 2 ligeiramente menor que o tempo da Classe 1. Isto é decorrente da negociação, pois a carga da Classe 2 é pequena, fazendo com que requisições de Classe 1 sejam negociadas para a Classe 2 quando o limite da Classe 1 é atingido, aumentando o tempo médio de resposta da Classe 2.

Novamente, é possível observar que o tempo médio de resposta da Classe 2 aumenta em relação à carga a ela imposta, justamente pelo fato de existir negociação.

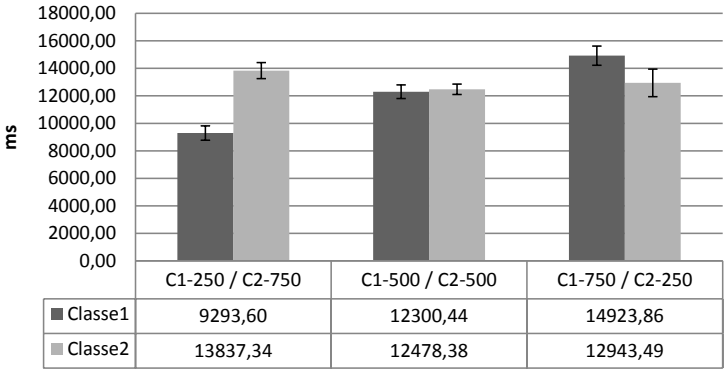


Figura 6.9: Tempo médio de resposta total - negociação - limite 1000.

A Figura 6.10 mostra as porcentagens de requisições aceitas e negociadas da Classe 1 e aceitas da Classe 2, com todos os níveis de carga utilizando um limite de 1000 conexões simultâneas em cada servidor.

Nas primeira e segunda colunas da Figura 6.10 pode-se notar que a Classe 1 atende, estatisticamente, 100% de suas requisições e a Classe 2 atende somente 52,89% em média. Isto ocorre, pois a carga da Classe 1 é pequena e o limite de conexões simultâneas é grande e também pelas duas classes terem o mesmo número de recursos. Pode-se observar também que não há negociação, pois os recursos de Classe 1 são suficientes para atender todas as requisições, embora o tempo de resposta aumente em relação a experimentos com limite de conexões simultâneas menores.

As terceira e quarta colunas da Figura 6.10 mostram que as duas classes atenderam, estatisticamente, o mesmo número de requisições, porém a Classe 1 conseguiu ser melhor pelo fato de 7% de suas requisições serem negociadas para a Classe 2 e com isso aumentando a porcentagem de requisições aceitas da Classe 1. As porcentagens de requisições aceitas das duas Classes foram parecidas, pelo fato dos grupos de servidores delas sobrecarregarem juntos, pois a carga é a mesma para ambas as classes, ficando mais difícil de ocorrer negociação.

Nas quinta e sexta colunas pode ser verificada que a Classe 1 atende uma porcentagem menor de requisições no seu grupo de máquinas em relação à Classe 2. Isto ocorre, pois a carga da Classe 1 é maior que a Carga da Classe 2, fazendo com que requisições de Classe 1 sejam negociadas para a Classe 2. Esta negociação para a Classe 2 faz com que as requisições que iriam ser descartadas na Classe 1 disputem recursos com a Classe 2, aumentando a porcentagem de requisições de Classe 1 aceitas pelo sistema e em consequência descartando uma boa porcentagem de requisições de Classe 2.

Apesar de em alguns casos da Figura 6.10 a Classe 1 atender uma porcentagem de requisições menor em seus servidores em relação à Classe 2, a negociação faz com que este cenário mude, pois requisições de Classe 1 que seriam descartadas são aceitas no grupo de servidores Classe 2, fazendo com que em nenhuma configuração de carga, a Classe 1 seja pior que a Classe 2 em termos de proporção requisições aceitas.

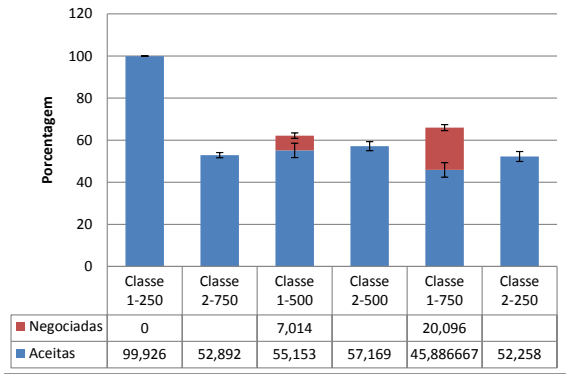


Figura 6.10: Porcentagem de requisições aceitas - negociação - limite 1000.

Negociação com Limite de 1200 Conexões Simultâneas

A Figura 6.11 mostra o tempo médio de resposta total para requisições de Classe 1 e Classe 2 utilizando o algoritmo de negociação com todos os níveis de carga e com limite de 1200 conexões simultâneas para cada servidor do *backend*.

Observando o primeiro conjunto de colunas mostra que o tempo médio de resposta da Classe 1 é menor em relação a Classe 2 pelo fato da carga da Classe 1 ser menor que a carga da Classe 2.

Analisando o segundo conjunto de colunas da Figura 6.11, os tempos médios de resposta são quase iguais. Isto se dá pelo fato da carga imposta as duas classes serem iguais. Contudo, não se pode afirmar nada, pois o intervalo de confiança inferior da Classe 2 não sobrepõe a média da Classe 1.

Já o último conjunto de colunas mostra que o tempo médio de resposta da Classe 1 é maior que o da Classe 2, pelo fato da Classe 1 ter uma carga maior. Apesar do tempo médio de resposta da Classe 2 ser menor que o da Classe 1, ele aumentou em relação aos outros tempos da Classe 2, pois aconteceu uma boa porcentagem de negociação (Figura 6.12).

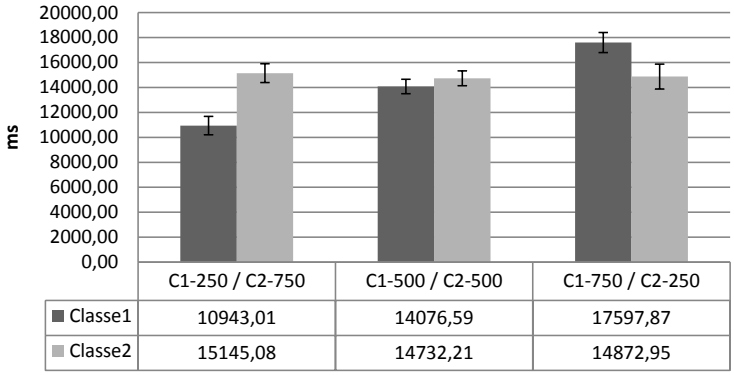


Figura 6.11: Tempo médio de resposta total - negociação - limite 1200.

A Figura 6.12 mostra as porcentagens de requisições aceitas e negociadas da Classe 1 e aceitas da Classe 2, com todos os níveis de carga utilizando o algoritmo de negociação e com limite de 1200 conexões simultâneas.

As primeira e segunda colunas da Figura 6.12 mostram que, estatisticamente, todas as requisições de Classe 1 são atendidas. Isto é ocasionado pelo aumento do limite máximo de conexões e porque a carga da Classe 1 é menor que a Classe 2, fazendo com que haja descarte na Classe 2.

As terceira e quarta colunas apresentam que as porcentagens de requisições aceitas de Classe 1 e Classe 2 são quase as mesmas. Isto ocorre, pois o limite de conexões simultâneas aumenta, fazendo com que os servidores demorem mais para se sobrecarregarem, com isso, quando requisições de Classe 1 vão ser negociadas, as máquinas da Classe 2 já estão sobrecarregas.

As quinta e sexta colunas da Figura 6.12 mostram que a Classe 1 atendeu uma porcentagem menor de requisições em seus servidores, se comparado com a Classe 2. No entanto, 22,92% de requisições de Classe 1 foram negociadas para a Classe 2. Isto ocorre, pois a Classe 2 não se

sobrecarrega somente com a sua carga, fazendo com que as requisições que iriam ser descartadas na Classe 1, disputem recursos com a Classe 2 e sejam atendidas.

Figura 6.12: Porcentagem de requisições aceitas - negociação - limite 1200.

6.3.2 Algoritmo de Reserva de Conexões

Para avaliar o Algoritmo de Reserva de Conexões foram variados seus limites de conexões simultâneas de cada servidor em: 400, 600, 800, 1.000, 1.200, e as cargas impostas nos experimentos foram: 25% para a Classe 1 e 75% para a Classe 2, 50% para a Classe 1 e 50% para a Classe 2 e 75% para a Classe 1 e 25% para a Classe 2.

As seguintes subseções apresentam a análise dos resultados.

Reserva de Conexões com Limite de 400 Conexões Simultâneas

Analisando os tempos médios de resposta do primeiro *byte* e total, pôde-se observar que o tempo de resposta do primeiro *byte* acompanhava o tempo de resposta total (como no algoritmo de negociação). Isto ocorreu pelo fato da página requisitada ser dinâmica. Com isso o processamento da requisição demanda certo tempo.

A Figura 6.13 e 6.14 mostram os tempos médios de resposta do primeiro *byte* e total do algoritmo de reserva de conexões com o limite de conexões simultâneas fixado em 400. Nota-se que o tempo médio de resposta do primeiro *byte* é menor que o tempo médio de resposta total. Isto se dá pelo fato da página requisitada ser dinâmica, necessitando de processamento. Nota-se também que o tempo médio de resposta total (como no algoritmo de negociação) acompanha o tempo médio de resposta do primeiro *byte*. Por isso optou-se por apresentar somente os tempos médios de resposta total das requisições de Classe 1 e Classe2.

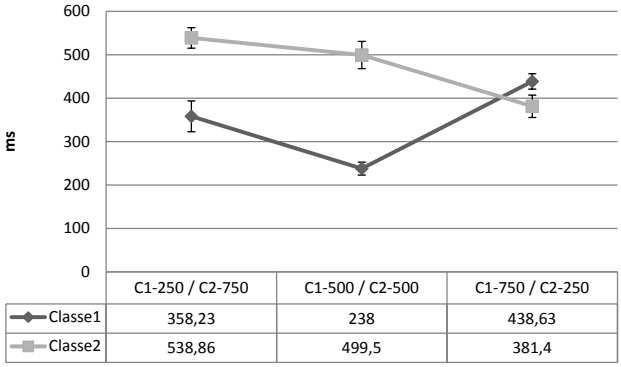


Figura 6.13: Tempo médio de resposta do primeiro *byte* - reserva de conexões - limite 400.

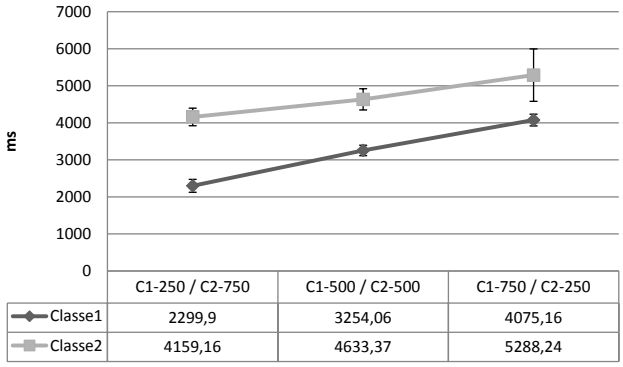


Figura 6.14: Tempo médio de resposta total - reserva de conexões - limite 400.

A Figura 6.15 apresenta os tempos médios de resposta total de cada requisição da Classe 1 e da Classe 2 com um limite de 400 conexões simultâneas para cada servidor do *backend*.

No primeiro conjunto de colunas, é possível verificar que o tempo médio de resposta das requisições de Classe 1 é menor que os da Classe 2. Isto ocorre pelo fato da carga da Classe 2 ser maior que a carga da Classe 1 e também por a Classe 1 ter uma porcentagem de conexões exclusivas.

No segundo conjunto de colunas da Figura 6.15 é possível observar que a Classe 1 também tem também tem um menor tempo de resposta em relação à Classe 2. O motivo que levou a este resultado é o servidor não se sobrecarregar com o limite pequeno de conexões simultâneas. Sendo assim, mesmo com cargas iguais, a Classe 1 é melhor que a Classe 2, pois ela tem uma porcentagem maior de conexões reservadas. Este comportamento tende a mudar conforme o limite de conexões aumenta (será observado posteriormente).

No terceiro grupo de colunas, verifica-se que o tempo médio de resposta total da Classe 2 foi ligeiramente maior que o da Classe 1. Isto ocorre pelo fato do servidor não se sobrecarregar com um limite de 400 conexões simultâneas em cada *Backend*, fazendo com que os tempos médios de resposta das duas Classes sejam parecidos, mesmo aceitando poucas requisições de Classe 2.

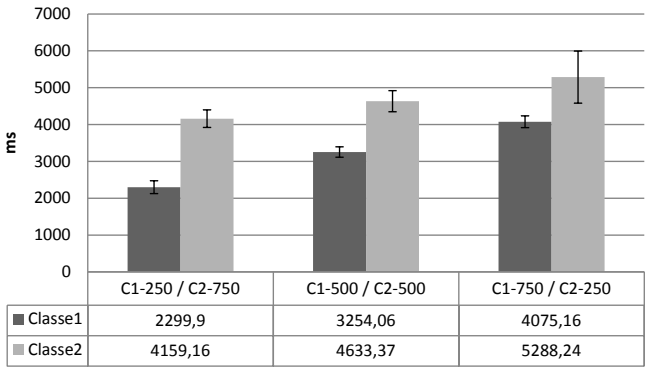


Figura 6.15: Tempo médio de resposta total - reserva de conexões - limite 400.

A Figura 6.16 apresenta a porcentagem de requisições aceitas pela Classe 1 e pela Classe 2, utilizando todos os níveis de carga com limite de 400 conexões simultâneas por servidor.

É possível observar que a Classe 1 atende uma porcentagem maior de requisições que a Classe 2 em todas as configurações de carga. Notas-se que quanto maior a carga da Classe 2, maior será o percentual de requisições aceitas por esta Classe, pelo fato de ser alocado um número maior de conexões a ela. Por fim é possível observar que, tendo um limite de conexões simultâneas pequeno, a porcentagem de requisições aceitas tanto da Classe 1 quanto da Classe 2, será pequena também.

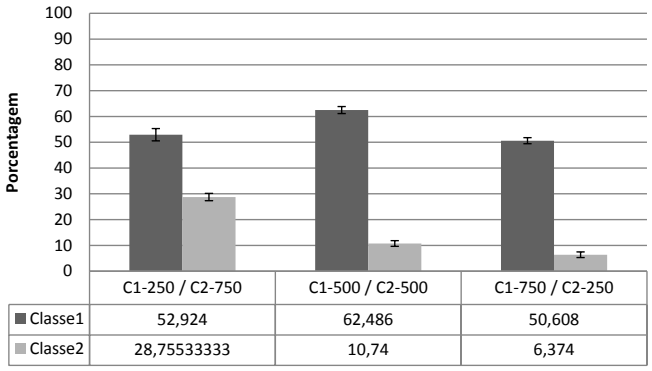


Figura 6.16: Porcentagem de requisições aceitas - reserva de conexões - limite 400.

Reserva de Conexões com Limite de 600 Conexões Simultâneas

A Figura 6.17 apresenta o tempo médio de resposta total para as duas classes com limite de 600 conexões simultâneas para cada servidor do *backend*.

A Figura 6.17 mostra que, quando o limite de conexões simultâneas aumenta, o tempo médio de resposta também aumenta. Isto é evidente pelo fato do sistema ter que gerenciar um número maior de conexões simultâneas.

Apesar de o tempo médio de resposta da Classe 2 ser maior que o da Classe 1, quando a carga da Classe 1 é maior que a carga da Classe 2 (terceiro conjunto de colunas), este comportamento tende a mudar quando o limite de conexões simultâneas fica maior. Isto será observado mais detalhadamente nas próximas figuras.

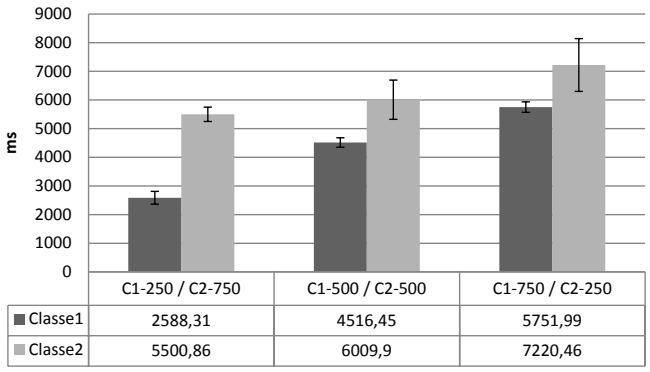


Figura 6.17: Tempo médio de resposta total - reserva de conexões - limite 600.

A Figura 6.18 apresenta as porcentagens de requisições aceitas para a Classe 1 e Classe 2 com limite de 600 conexões simultâneas.

Nota-se na Figura 6.18, em uma visão geral, que a porcentagem de requisições aceitas em ambas as Classes é maior do que quando se utilizou o limite de 400 conexões simultâneas. Este comportamento se deve ao fato de ter 200 conexões simultâneas a mais em cada servidor, aumentando então o número de requisições aceitas, em contrapartida aumentando também o tempo médio de resposta de cada requisição.

Com 600 conexões simultâneas em cada servidor, a Classe 1 ainda não consegue atender todas as suas requisições quando são enviadas cargas pequenas (5000 requisições) e médias (10000 requisições). Isto ocorre porque o limite de conexões simultâneas ainda não suporta todas as rajadas de requisições.

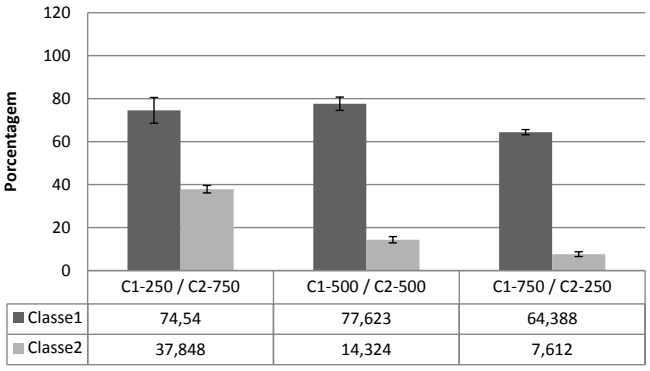


Figura 6.18: Porcentagem de requisições aceitas - reserva de conexões - limite 600.

Reserva de Conexões com Limite de 800 Conexões Simultâneas

A Figura 6.19 apresenta o tempo médio de resposta total para a Classe 1 e Classe 2 com limite de 800 conexões simultâneas.

O primeiro grupo de colunas mostra que a Classe 1 tem um menor tempo de resposta total que a Classe 2, mesmo com o aumento de conexões simultâneas. Isto ocorre em todas as alterações

de limite, pelo fato da carga Classe 1 ser menor e por ter uma porcentagem exclusiva de conexões para a Classe 1.

No segundo grupo de colunas da Figura 6.19 observa-se que o tempo médio de resposta total de ambas as Classes ficaram estatisticamente iguais. Isto se deve ao fato do aumento do limite de conexões simultâneas. Com isso, quando o sistema começa a atender somente requisições de Classe 1 (em torno de 37,5% de carga do sistema), as requisições de Classe 2 que já estão no servidor demoram um tempo maior para serem escalonadas novamente, pelo aumento da carga de Classe 1 no sistema.

Verificando o terceiro grupo de colunas da Figura 6.19, nota-se uma característica importante do algoritmo. Apesar de não poder afirmar estatisticamente, pelo fato dos intervalos de confiança das duas Classes se sobreporem, quando se tem uma carga grande para a Classe 1, uma carga pequena para a Classe 2 e o número de conexões dos servidores passa de um certo limite, a tendência é aumentar a diferença entre o tempo médio de resposta da Classe 1 com relação à Classe 2. Isto ocorre pelo fato da Classe 2 poder atender requisições somente quando o servidor não está sobrecarregado, então estas poucas requisições de Classe 2 que são atendidas, ficam com o tempo médio de resposta baixo em relação à Classe 1 que atende requisições até o limite máximo de conexões simultâneas.

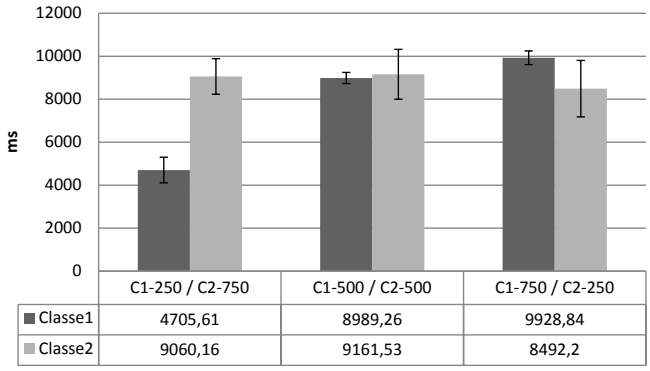


Figura 6.19: Tempo médio de resposta total - reserva de conexões - limite 800.

A Figura 6.20 apresenta as porcentagens de requisições aceitas pela Classe 1 e 2 utilizando um limite de 800 conexões simultâneas para cada servidor do *backend*.

Pode-se notar nos dois primeiros conjuntos de colunas da Figura 6.20 que quando se tem uma carga pequena (5000 requisições) e uma carga média (10000 requisições) para a Classe 1, quase todas as suas requisições são atendidas. Isto é decorrente do aumento do limite de conexões simultâneas.

No último grupo de colunas da Figura 6.20 é possível notar que a Classe 1 consegue atender aproximadamente 70% de suas requisições. Isto ocorre, pois a carga da Classe 1 ainda é suficiente para sobrecarregar as conexões disponíveis.

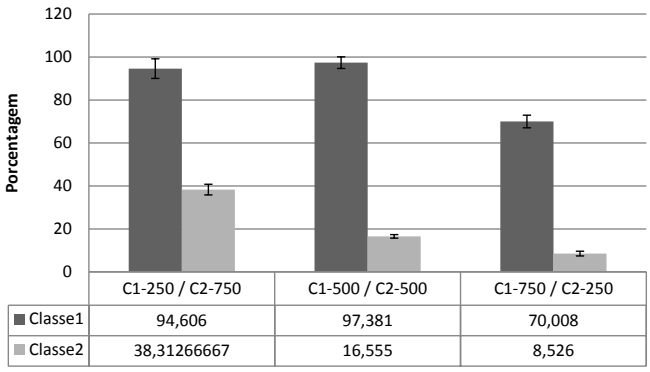


Figura 6.20: Porcentagem de requisições aceitas - reserva de conexões - limite 800.

Reserva de Conexões com Limite de 1000 Conexões Simultâneas

O gráfico da Figura 6.21 apresenta os tempos médios de resposta total para a Classe 1 e 2 utilizando um limite de 1000 conexões simultâneas para cada servidor do *backend*.

No segundo grupo conjunto de colunas da Figura 6.21 é possível observar que quanto maior o limite de conexões simultâneas, maior o tempo da Classe 1. A partir de certo limite de conexões simultâneas, o comportamento do algoritmo, quando se tem cargas iguais para as duas Classes, será semelhante quando se tem uma carga grande para a Classe 1 e uma carga pequena para a Classe 2.

No terceiro conjunto de colunas da Figura 6.21 é mais notório a diminuição do tempo médio de resposta da Classe 2 em relação à Classe 1. Isto ocorre, pois aumentando o limite de conexões simultâneas e mantendo a carga (75% para a Classe 1 e 25% para a Classe 2), as requisições de Classe 2 não sobrecarregam as máquinas, pois elas só podem ser aceitas em alguns instantes, ou seja, só até o limite em que, em média, 6,25% das conexões estão ocupadas. Deste modo, diminuí-se o tempo de resposta, pois o hardware consegue manter estas conexões sem sobrecarga.

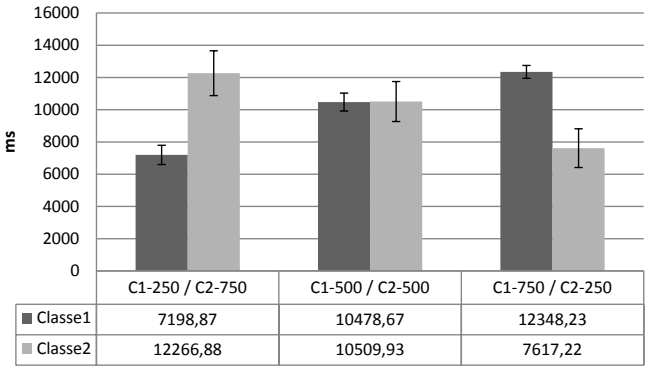


Figura 6.21: Tempo médio de resposta total - reserva de conexões - limite 1000.

A Figura 6.22 apresenta as porcentagens de requisições aceitas pelas classes, utilizando um limite de 1000 conexões simultâneas para cada servidor do *backend*.

É verificado na Figura 6.22 que quando se tem um limite de conexões grande, praticamente todas as requisições de Classe 1 são atendidas quando as cargas da Classe 1 são pequena e média. Isto ocorre porque a carga é mantida e o limite aumenta. Com isso os servidores comportam todas estas requisições. Já com a carga da Classe 1 grande (terceiro conjunto de colunas), o servidor não consegue atender todas as requisições de Classe 1. No entanto, a porcentagem de requisições aceitas desta Classe é maior do que quando o limite era menor.

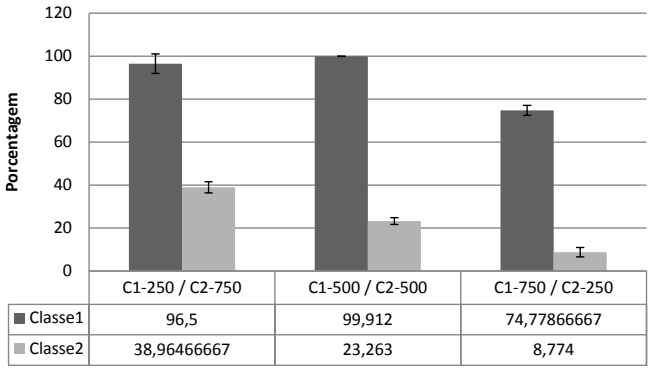


Figura 6.22: Porcentagem de requisições aceitas - reserva de conexões - limite 1000.

Reserva de conexões com limite de 1200 conexões simultâneas

A Figura 6.23 apresenta os tempos médios de resposta total para as duas classes, utilizando 1200 conexões simultâneas para cada servidor.

Figura 6.23 pode ser observado no terceiro grupo de colunas que quanto maior o limite de conexões, maior é a diferença do tempo médio de resposta entre as duas Classes. Isto foi explicado anteriormente.

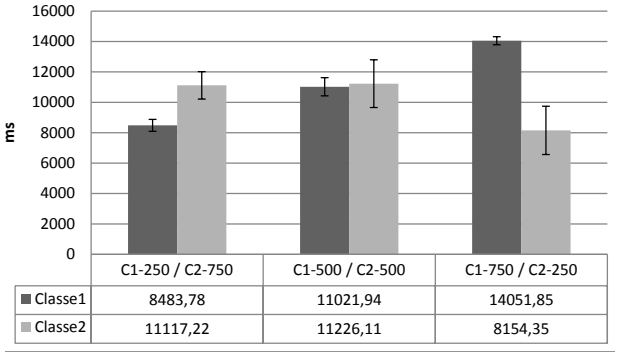


Figura 6.23: Tempo médio de resposta total - reserva de conexões - limite 1200.

A Figura 6.24 mostra a porcentagem de requisições aceitas de Classe 1 e Classe 2 utilizando 1200 conexões simultâneas em cada servidor.

A Figura 6.24 mostra, nos dois primeiros conjuntos de colunas, que a Classe 1 conseguiu atender 100% de suas requisições. Isto ocorreu pelo aumento do número de conexões simultâneas e pelo fato da carga ter se mantido durante todos os experimentos.

No terceiro grupo de colunas da Figura 6.24 nota-se que a Classe 1 não conseguiu atender todas as suas requisições. Este comportamento era esperado, pelo fato da carga da Classe 1 ser grande, com isso quase todas as requisições são armazenadas para a Classe 1, porém nem todas as requisições conseguem ser atendidas.

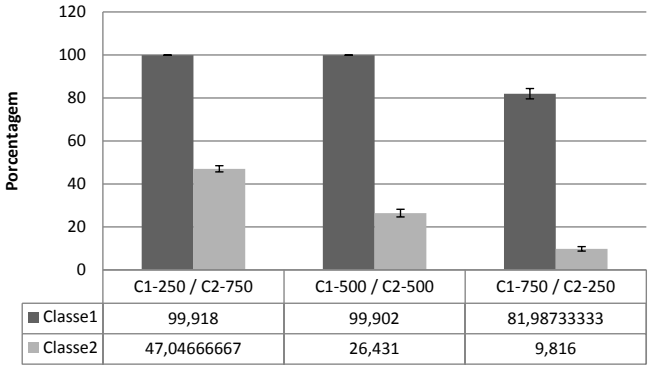


Figura 6.24: Porcentagem de requisições aceitas - reserva de conexões - limite 1200.

6.3.3 Comparativo Entre os Algoritmos

Esta seção tem como objetivo fazer uma análise comparativa entre os algoritmos propostos. Para isso, foram elaboradas 3 tabelas, as quais agrupam todas as variáveis de respostas.

Durante a análise dos resultados, pôde-se notar que os algoritmos têm comportamentos diferentes. O algoritmo de negociação possui 3 máquinas exclusivas para a Classe 1, ou seja, quando estas máquinas estão sobrecarregadas, requisições que iriam ser descartadas da Classe 1 são marcadas como Classe 2 e disputam as outras 3 máquinas com requisições de Classe 2. Este algoritmo pode não ser uma boa alternativa quando a carga da Classe 1 é muito pequena, pelo fato dos recursos alocados à esta Classe ficarem subutilizados.

O algoritmo de reserva de conexões, ao invés de alocar máquinas para determinada classe, aloca somente as conexões necessárias para cada Classe dinamicamente, fazendo um melhor uso dos recursos disponíveis. As conexões são alocadas de acordo com a carga que chega ao sistema. Se a carga da Classe 1 é grande, mais conexões exclusivas ficam alocadas para esta Classe. Por outro lado, se a carga da Classe 1 for pequena, menos conexões são alocadas para a Classe 1. Com isso pode-se notar que, quanto maior a carga da Classe 1, menos requisições de Classe 2 são aceitas no sistema.

As Tabelas 6.2, 6.3 e 6.4 mostram um comparativo numérico das variáveis de respostas. A Tabela 6.2 apresenta os resultados de 25% de carga para a Classe 1 e 75% de carga para a Classe 2. A Tabela 6.3 apresenta os resultados de 50% de carga para a Classe 1 e 50% de carga para a Classe 2. E por fim, a Tabela 6.4 apresenta os resultados de 75% de carga para a Classe 1 e 25% para a Classe 2. As sigla TMRT presente nas Tabelas significa Tempo Médio de Resposta Total.

Nota-se que, apesar do algoritmo de reserva de conexões atender uma porcentagem menor de requisições na maioria dos experimentos, ele obtém um tempo médio de resposta menor e consegue

Tabela 6.2: Comparação entre os algoritmos - Carga 25% para a Classe 1 e 75% para a Classe 2

Comparativo	TMRT		% Req. Aceitas	
	Classe 1	Classe 2	Classe 1	Classe 2
Negociação-400	3456,16	5008,49	66,11	34,35
Reserva-400	2299,9	4159,16	52,92	28,75
Negociação-600	3938,29	6754,27	94,8	37,85
Reserva-600	2588,31	5500,86	74,54	37,84
Negociação-800	7755,43	11264,14	95,34	44,22
Reserva-800	4705,61	9060,16	94,606	38,31
Negociação-1000	9293,60	13837,34	99,92	52,89
Reserva-1000	7198,87	12266,88	96,5	38,96
Negociação-1200	10943,01	15145,08	99,91	56,48
Reserva-1200	8483,78	11117,22	99,91	47,04

Tabela 6.3: Comparação entre os algoritmos - Carga 50% para a Classe 1 e 50% para a Classe 2

Comparativo	TMRT		% Req. Aceitas	
	Classe 1	Classe 2	Classe 1	Classe 2
Negociação-400	3721,34	3608,53	51,98	34,54
Reserva-400	3254,06	4633,37	62,48	10,74
Negociação-600	6405,9	5818,17	62,10	44,58
Reserva-600	4516,45	6009,9	77,62	14,32
Negociação-800	9504,93	9673,85	63,3	53,07
Reserva-800	8989,26	9161,53	97,38	16,55
Negociação-1000	12300,44	12478,38	62,16	57,16
Reserva-1000	10478,67	10509,93	99,91	23,26
Negociação-1200	14076,59	14732,21	66,06	62,17
Reserva-1200	11021,94	11226,11	99,90	26,43

priorizar mais as requisições de Classe 1. Em contrapartida, o algoritmo de negociação atende mais requisições em um maior tempo e, em algumas configurações de carga, a Classe 2 fica próxima da Classe 1 levando-se em consideração a porcentagem de requisições aceitas.

6.4 Estudo da Influência dos Fatores

Esta seção apresenta um estudo da influência dos fatores no tempo médio de resposta total e na porcentagem de requisições aceitas. Os fatores representados como A, B e C são caracterizados da seguinte forma:

- **A**: Algoritmo de controle de admissão;
- **B**: Limite máximo de conexões simultâneas;

Tabela 6.4: Comparação entre os algoritmos - Carga 75% para a Classe 1 e 25% para a Classe 2

Comparativo	TMRT		% Req. Aceitas	
	Classe 1	Classe 2	Classe 1	Classe 2
Negociação-400	4976,73	3853,23	48,5	38,2
Reserva-400	4075,16	5288,24	50,6	6,37
Negociação-600	8514,83	5393,6	50,9	46,99
Reserva-600	5751,99	7220,46	64,38	7,61
Negociação-800	12573,63	10657,52	50,04	46,17
Reserva-800	9928,84	8492,2	70	8,52
Negociação-1000	14923,86	12943,49	65,97	52,25
Reserva-1000	12348,23	7617,22	74,77	8,77
Negociação-1200	17597,87	14872,95	75,13	61,09
Reserva-1200	14051,85	8154,35	81,98	9,81

- **C**: Carga imposta ao servidor;

Para este estudo de influência dos fatores, foi reduzido o número de níveis do fator B para dois, pelo fato da complexidade de se calcular a influência dos fatores com N níveis e também porque é possível observar a influência do fator B utilizando somente os dois níveis extremos (400 e 1200).

Pode ser observado também que no fator Carga (fator C), utilizou-se 3 níveis para realizar o cálculo da influência dos fatores. Estes níveis serão combinados dois a dois para diminuir o nível da complexidade dos cálculos.

6.4.1 Influência dos Fatores no Tempo Médio de Resposta

As Figuras 6.25, 6.26, 6.27 apresentam a influência dos fatores no tempo médio de resposta da Classe 1 combinando as cargas duas a duas.

A Figura 6.25 combina as cargas de 50% para a Classe 1 - 50% para a Classe 2 e 75% para a Classe 1 - 25% para a Classe 2. Pode-se notar que o fator B (Fator limite de conexões simultâneas) tem uma influência predominante. Isto se deve ao fato de quando o limite máximo de conexões aumenta, o tempo médio de resposta também aumenta proporcionalmente para ambos os algoritmos utilizados.

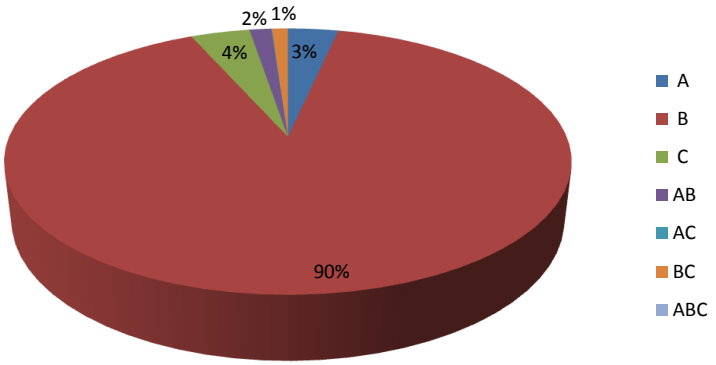


Figura 6.25: Influência dos fatores no tempo médio de resposta da Classe 1.

A Figura 6.26 mostra a combinação das cargas de 50% para a Classe 1 - 50% para a Classe 2 e 25% para a Classe 1 - 75% para a Classe 2. Nota-se que nesta figura o Fator B também foi o que mais influenciou no tempo médio de resposta da Classe 1. Isto é decorrente do mesmo motivo citado para a Figura 6.25.

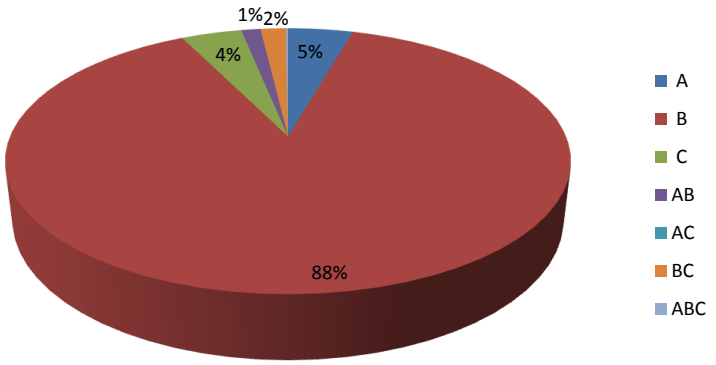


Figura 6.26: Influência dos fatores no tempo médio de resposta da Classe 1.

A Figura 6.27 mostra a combinação das cargas de 75% para a Classe 1 - 25% para a Classe 2 e 25% para a Classe 1 - 75% para a Classe 2. Observa-se que para estas combinações de carga, o fator C (carga) teve uma maior influência e o fator B (limite) teve uma menor influência em relação à Figura 6.26. Isto se deve ao fato do tempo médio de resposta ter aumentado desproporcionalmente, quando se utilizou o algoritmo de negociação com limite de 1200 em relação ao algoritmo de reserva de conexões com o mesmo limite.

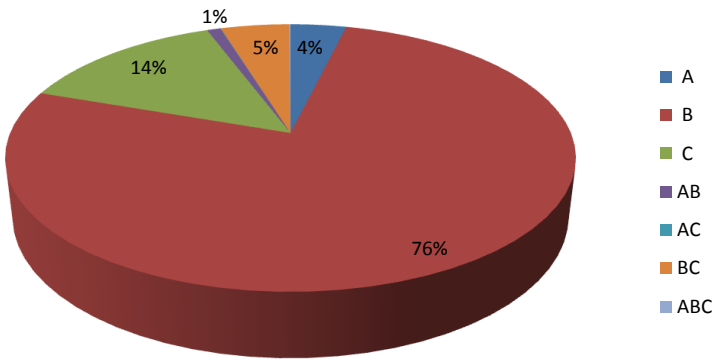


Figura 6.27: Influência dos fatores no tempo médio de resposta da Classe 1.

As Figuras 6.28, 6.29, 6.30 apresentam a influência dos fatores no tempo médio de resposta da Classe 2 combinando as cargas duas a duas.

A Figura 6.28 combina as cargas de 50% para a Classe 1 - 50% para a Classe 2 e 75% para a Classe 1 - 25% para a Classe 2. É possível verificar que o limite é o fator que mais influencia o tempo médio de resposta, pelo fato deste aumentar proporcionalmente quando o limite é fixado em 1200, exceto quando se utiliza o algoritmo de reserva de conexões, combinado com um limite de 1200 e com 75% de carga para a Classe 1 e 25% de carga para a Classe 2. Isto ocorre pois, quanto menor a carga e maior o limite de conexões simultâneas, utilizando o algoritmo de reserva de conexões, o tempo médio de resposta da Classe 2 tende a ser menor que o da Classe 1, pelo fato do servidor não se sobrecarregar com o limite de aceitação de requisições de Classe 2. Com isso é possível notar 5% de influência do fator algoritmo e 13% de influência combinando o fator algoritmo com o fator limite.

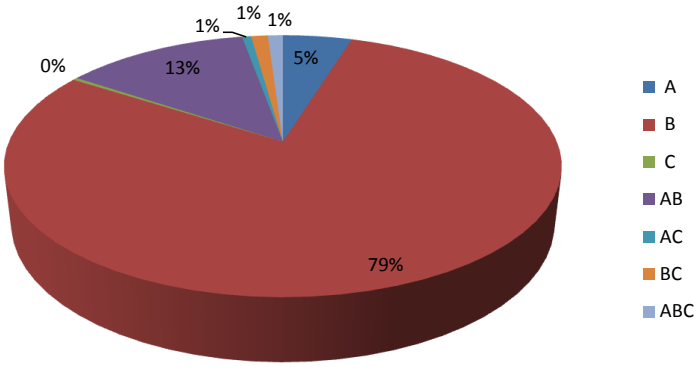


Figura 6.28: Influência dos fatores no tempo médio de resposta da Classe 2.

A Figura 6.29 mostra a combinação das cargas de 50% para a Classe 1 - 50% para a Classe 2 e 25% para a Classe 1 - 75% para a Classe 2. É possível notar que a maior influencia no tempo médio de resposta foi fator B (limite). Isto ocorre, pois com o aumento do limite de conexões simultâneas para 1200, os tempos médios de resposta aumentam proporcionalmente, tanto para o algoritmo de negociação quanto para o algoritmo de reserva de conexões.

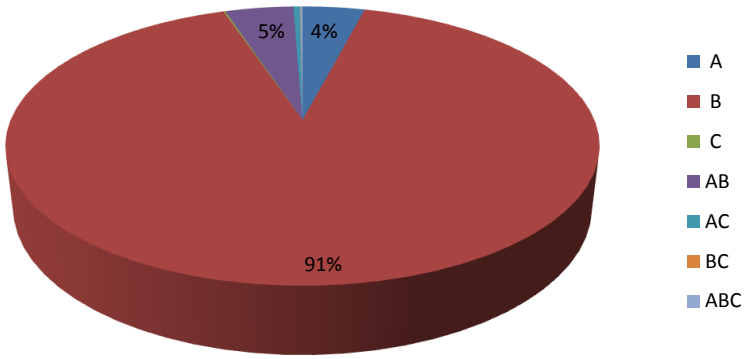


Figura 6.29: Influência dos fatores no tempo médio de resposta da Classe 2.

A Figura 6.30 mostra a combinação das cargas de 75% para a Classe 1 - 25% para a Classe 2 e 25% para a Classe 1 - 75% para a Classe 2. Nota-se neste gráfico que ainda o fator que mais influencia é o limite de conexões. Isto pode ser explicado tomando como base a explicação dada para a Figura 6.28.

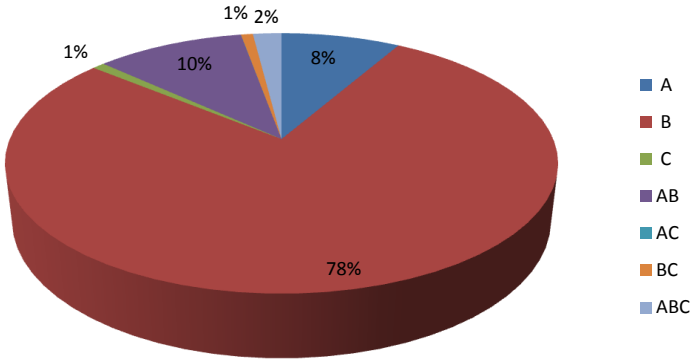


Figura 6.30: Influência dos fatores no tempo médio de resposta da Classe 2.

6.4.2 Influência dos Fatores na Porcentagem de Requisições Aceitas

As Figuras 6.31, 6.32, 6.33 apresentam a influência dos fatores na porcentagem de requisições aceitas da Classe 1 combinando as cargas duas a duas.

A Figura 6.31 combina as cargas de 50% para a Classe 1 - 50% para a Classe 2 e 75% para a Classe 1 - 25% para a Classe 2. Nota-se que o fator que mais influência a porcentagem de requisições aceitas da Classe 1 é o limite. Isto ocorre, pois a porcentagem de requisições aceitas no sistema aumenta proporcionalmente para os dois algoritmos, quando o limite de conexões simultâneas aumenta.

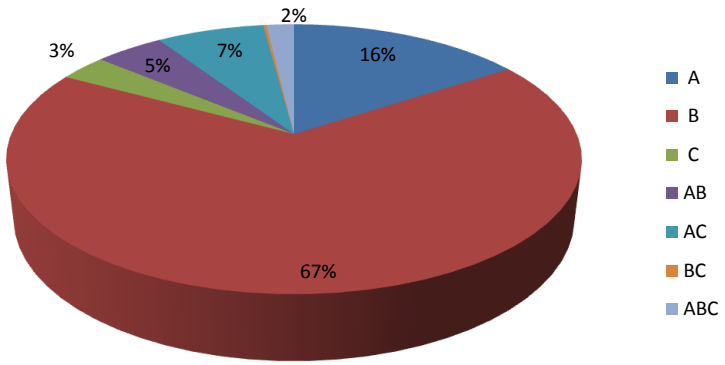


Figura 6.31: Influência dos fatores na porcentagem de requisições aceitas da Classe 1.

As Figuras 6.32 e 6.33 mostram a combinação das cargas de 50% para a Classe 1 - 50% para a Classe 2 e 25% para a Classe 1 - 75% para a Classe 2, 75% para a Classe 2 - 25% para a Classe 2 e 25% para a Classe 1 - 75% para a Classe 2, respectivamente. Pode-se observar que o limite de conexões tem a maior influência na variável de resposta. Este comportamento pode ser explicado pela Figura 6.31

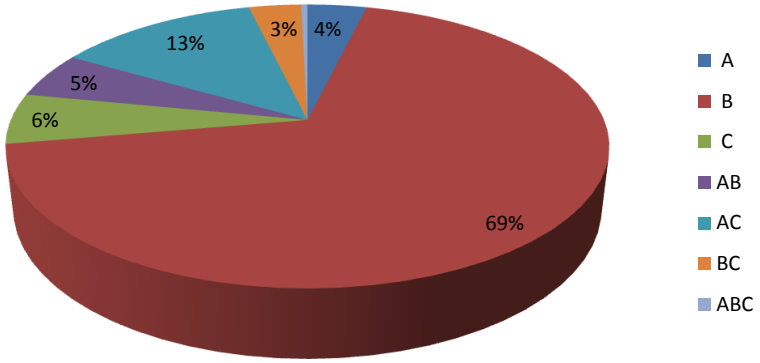


Figura 6.32: Influência dos fatores na porcentagem de requisições aceitas da Classe 1.

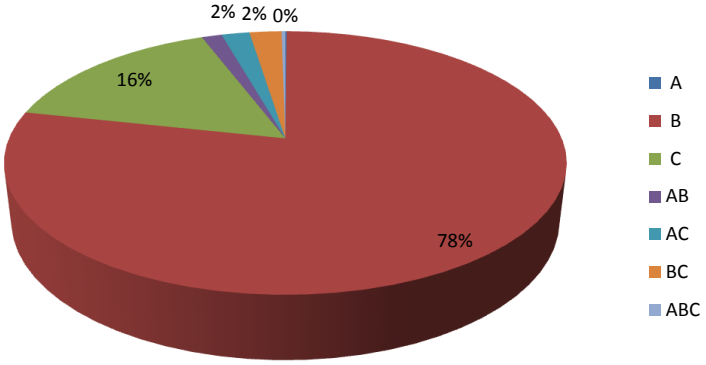


Figura 6.33: Influência dos fatores na porcentagem de requisições aceitas da Classe 1.

As Figuras 6.34, 6.35, 6.36 apresentam a influência dos fatores na porcentagem de requisições aceitas da Classe 2 combinando as cargas duas a duas.

A Figura 6.34 combina as cargas de 50% para a Classe 1 - 50% para a Classe 2 e 75% para a Classe 1 - 25% para a Classe 2. Nota-se que o fator A (algoritmo), tem uma grande influência na porcentagem de requisições aceitas pela Classe 2. Isto ocorre, pois quanto menor a carga da Classe 2, menor será a porcentagem de requisições aceitas, quando utilizado o algoritmo de reserva de conexões. Em contra partida, o algoritmo de negociação mantém a proporção de requisições aceitas quando a carga da Classe 2 diminui.

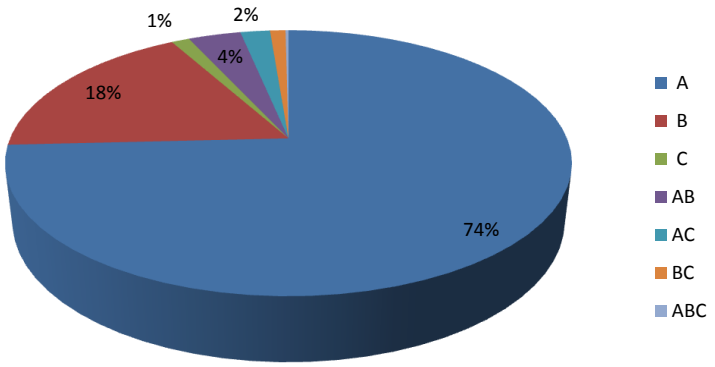


Figura 6.34: Influência dos fatores na porcentagem de requisições aceitas da Classe 2.

As Figuras 6.35 e 6.36 combinam as cargas de 50% para a Classe 1 - 50% para a Classe 2 e 25% para a Classe 1 - 75% para a Classe 2 e 75% para a Classe 1 - 25% para a Classe 2 e 25% para a Classe 1 - 75% para a Classe 2, respectivamente. Pode se notar que tanto o algoritmo quanto o limite influenciam na porcentagem de requisições aceitas da Classe 2. Isto ocorre, pois quando se aumenta o limite de conexões simultâneas, ambos os algoritmos atendem mais requisições, porém de formas diferentes, ou seja, o algoritmo de negociação atende quase a mesma proporção de requisições de Classe 2 em todas as configurações de carga. Já o algoritmo de reserva de conexões atende, proporcionalmente, menos requisições de Classe 2 quando a carga desta Classe diminui.

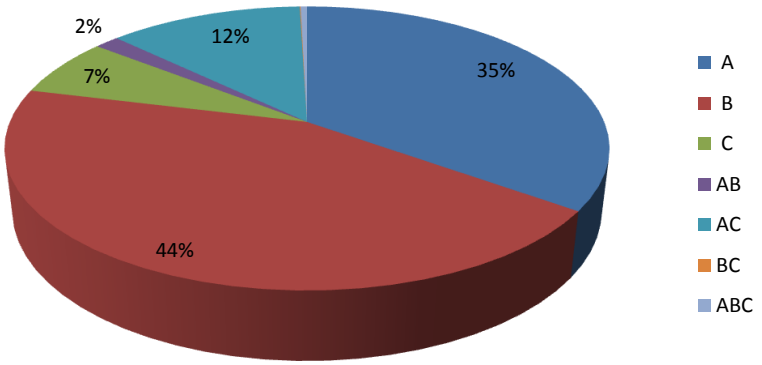


Figura 6.35: Influência dos fatores na porcentagem de requisições aceitas da Classe 2.

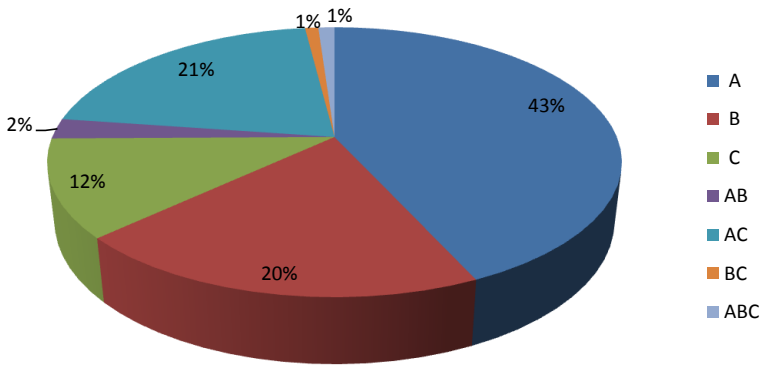


Figura 6.36: Influência dos fatores na porcentagem de requisições aceitas da Classe 2.

6.5 Considerações Finais

Este Capítulo apresentou o planejamento de experimentos, combinando limites de conexões, cargas de trabalhos e algoritmos de controle de admissão. Após a análise dos tempos médios de resposta e das porcentagens de requisições aceitas, foi possível observar as características de cada algoritmo de controle de admissão.

Por fim, foi apresentada a influência dos fatores nas variáveis de respostas. Pôde-se observar que o fator Limite influencia mais, pois aumentando o limite de aceitação, mais requisições são atendidas no sistema, e o tempo médio de resposta das requisições aumenta.

A partir desta avaliação de desempenho, é possível chegar a algumas conclusões. O próximo Capítulo apresentará estas conclusões e também alguns possíveis trabalhos futuros.

CAPÍTULO

7

Conclusões e Trabalhos Futuros

7.1 Conclusões

A motivação para este trabalho é decorrente do crescente aumento de utilização da Internet, fazendo com que milhares de atividades e/ou aplicações, que antes eram realizadas de forma convencional, migrassem para o domínio da Internet. Com isso houve a necessidade de se elaborar técnicas que suportassem essa demanda de aplicações, com o intuito de não sobrecarregar os servidores. Uma das possibilidades é o descarte de requisições quando o servidor se encontra sobrecarregado. Porém, criar mecanismos que simplesmente descartem requisições sem análise prévia não é um grande atrativo, pelo fato das aplicações *Web* fornecerem, a cada dia mais, diferenciação de serviços na Internet (usuários prioritários têm mais vantagens).

Para a realização deste projeto de mestrado, duas importantes atividades foram desenvolvidas. A primeira consistiu na construção de um protótipo de servidores *Web* com capacidade de diferenciação de serviços. Para atingir este objetivo, foi usado como base o modelo SWDS proposto por (Teixeira, 2004), o qual é composto por um classificador, controle de admissão e um *cluster* de servidores *Web*. A segunda atividade consistiu na proposta e desenvolvimento de algoritmos de controle de admissão para serem implementados no protótipo do servidor SWDS.

Verificando esta demanda por serviços diferenciados, este trabalho propôs dois algoritmos de controle de admissão. O primeiro foi baseado no trabalho de Estrella (2006), o qual propôs e implementou um algoritmo de negociação em nível de simulação e que neste trabalho foi adaptado para o protótipo do SWDS. O segundo algoritmo proposto neste trabalho, denominado algoritmo de reserva de conexões, tem como objetivo reservar uma porcentagem maior de conexões

simultâneas para a classe mais prioritária, de acordo com a carga de trabalho que é imposta ao servidor.

Estes algoritmos foram implementados, testados e avaliados no protótipo do SWDS, utilizando rajadas de carga sintética para sobrecarregar o servidor e assim verificar o comportamento de cada algoritmo.

Pôde-se notar que o algoritmo de negociação tem por objetivo negociar as requisições de Classe 1 que seriam descartadas em um primeiro momento rebaixando-as para a Classe 2. Este algoritmo tem um número fixo de máquinas para a Classe 1 e o restante das máquinas ficam à disposição da Classe 2. Quando as máquinas de Classe 1 estão sobrecarregadas, as requisições desta Classe são enviadas para as máquinas de Classe 2 e atendidas segundo os critérios de QoS dessa Classe.

Nota-se também no algoritmo de negociação que em todas as configurações de carga, a Classe 1 teve uma porcentagem maior de requisições aceitas. Isto não ocorreria se a Classe 1 não negociasse com a Classe 2 nos casos em que a carga da Classe 1 era grande, pelo fato dos recursos da Classe 1 não serem suficientes.

O algoritmo de negociação é uma boa alternativa para servidores em que o tráfego de requisições é similar, ou seja, todos os dias a característica de carga que o servidor recebe é a mesma (previsível). Com isso, é possível reservar o número de máquinas ideal para a Classe 1. Como é muito difícil de prever a característica de carga da *Web*, este mecanismo pode não ser uma boa alternativa para servidores *Web*, pelo fato da carga ser altamente variável.

O algoritmo de reserva de conexões aloca parte dos recursos disponíveis para a Classe de maior prioridade utilizando a equação que foi descrita no Capítulo 4. Com esta equação, pode-se reservar conexões para a Classe 1 dinamicamente, fazendo com que não ocorra desperdício de recursos como acontece no algoritmo de negociação quando se tem uma carga pequena para a Classe 1.

Pode-ser observado também que à medida que a Classe 1 recebe uma carga maior que a Classe 2, a Classe 1 tende a dominar o sistema. No entanto a partir de um certo limite, é reservado para Classe 2 um número de conexões fixas. Desta forma, a Classe 2 não descarta todas as suas requisições. A intensão do algoritmo de reserva de conexões é fornecer total prioridade para requisições de Classe 1 quando a carga desta Classe é alta, assumindo que requisições de Classe 2 não sejam importantes em situações de sobrecarga da Classe 1.

Uma característica que foi observada ao longo dos experimentos com o algoritmo de reserva de conexões é que, quando se tem uma carga grande para a Classe 1 e uma carga pequena para a Classe 2 e o número de conexões dos servidores passa de um certo limite, é possível notar que a Classe 1 obtém um tempo de resposta maior que o tempo da Classe 2, pelo fato dos servidores não estarem sobrecarregados quando as requisições de Classe 2 eram aceitas no sistema.

Apesar do algoritmo de reserva de conexões atender um número menor de requisições (na maioria dos casos), ele consegue um melhor tempo de resposta em relação ao outro algoritmo.

Pode-se concluir que o algoritmo de reserva de conexões é recomendado para o ambiente da *Web* onde a característica da carga é dinâmica, pois o algoritmo consegue se adaptar a qualquer tipo de carga.

7.2 Contribuições

As principais contribuições realizadas com o desenvolvimento deste trabalho de mestrados estão listadas a seguir:

- **Implementação do protótipo da arquitetura SWDS**: Com base nos trabalhos relacionados, foi implementado o protótipo do SWDS, o qual oferece qualidade de serviço em nível de aplicação. Para que este protótipo fosse construído, foi necessária a utilização de um *cluster* de máquinas homogêneas, utilizando o *software* Apache configurado como *proxy* reverso, as quais respondem a um servidor *frontend*, que tem o objetivo de receber e responder requisições aos clientes.

- **Proposta de algoritmos de controle de admissão**: A partir de uma ampla revisão bibliográfica, foi possível propor algoritmos de controle de admissão que cobrissem algumas lacunas na área de algoritmos de controle de admissão com diferenciação de serviços. O algoritmo de negociação proposto por (Estrella, 2006) foi adaptado para o protótipo do SWDS. Ele tem por objetivo rebaixar requisições de uma classe mais prioritária quando esta se encontra sobrecarregada. Também foi proposto neste trabalho outro algoritmo, denominado algoritmo de reserva de conexões. Este algoritmo, diferente de vários algoritmos de reserva de recursos (alocação de máquinas), aloca somente as conexões necessárias para cada classe, fazendo com que os recursos sejam melhores utilizados.

- **Avaliação de desempenho dos algoritmos**: para avaliar o desempenho dos algoritmos foi realizado um planejamento de experimentos, utilizando a metodologia de fatorial completo. Na análise dos resultados, foi possível observar as características de cada algoritmo de controle de admissão. Por fim foi realizado um estudo da influência de cada fator nas variáveis de resposta do sistema.

7.3 Trabalhos Futuros

Este trabalho apresentou a implementação de algoritmos de controle de admissão em um protótipo do SWDS. Pode-se destacar algumas melhorias para trabalhos futuros:

- Utilizar uma carga de trabalho mais realista, como por exemplo, *logs* de sites de *e-commerce*. Esta tarefa não é trivial pelo fato da maioria das empresas não disponibilizarem seus *logs*, e os que estão disponíveis já estarem defasados.

- Fazer experimentos com modificações nos algoritmos, como por exemplo: no algoritmo de reserva de conexões, alterar a proporção de conexões que serão alocadas para a classe de maior prioridade. E no algoritmo de negociação, alterar o número de máquinas para a classe mais prioritária.

- Utilizar mais de duas classes e verificar o impacto do número de classes no desempenho do sistema.

Referências Bibliográficas

ABRAÃO, I. C. *Desempenho de servidores web: Estratégias de cache e diferenciação de serviços*. Tese de Doutoramento, Universidade de São Paulo, 2008.

ANDREOLINI, M.; CASALICCHIO, E.; COLAJANNI, M.; MAMBELLI, M. A cluster-based web system providing differentiated and guaranteed services. *Cluster Computing*, v. 7, p. 7–19, 2004.
Disponível em: http://portal.acm.org/citation.cfm?id=953113.953137

APOSTOLOPOULOS, G.; GUÉRIN, R.; KAMAT, S.; ORDA, A.; PRZYGIENDA, T.; WILLIAMS, D.; ET AL. QoS routing mechanisms and OSPF extensions. 1998.

AWDUCHE, D.; MALCOLM, J.; AGOGBUA, J.; O'DELL, M.; MCMANUS, J. Rfc2702: Requirements for traffic engineering over mpls. *RFC Editor United States*, 1999.

BANICESCU, I.; VELUSAMY, V.; DEVAPRASAD, J. On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring. *Cluster Computing*, v. 6, n. 3, p. 215–226, 2003.

BARBATO, A. K. *Políticas para servidores web baseados em sessões visando qualidade e diferenciação de serviços*. Dissertação de Mestrado, Universidade de São Paulo, 2008.

BARBATO, A. K.; SANTANA, M. J.; SANTANA, R. H. C.; TRALDI, O. A.; TEIXEIRA, M. A. M. Avaliação de políticas de controle de admissão para servidores web com serviços diferenciados. *Anais da 58Â^{a} Reunião Anual da SBPC*, 2006.

BARBATO, A. K.; SANTANA, R. H. C.; TRALDI, O. A.; SANTANA, M. J.; TEIXEIRA, M. A. M. Avaliação de desempenho: Modelagem e simulação - servidor web com serviços diferenciados e algoritmo de reserva adaptativa de recursos (rsvadap). *XII Simpósio Internacional de Iniciação Científica da Universidade de São Paulo (SIICUSP)*, 2004.

BHOJ, P.; SINGHAL, S.; CHUTANI, S. SLA management in federated environments. In: *Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on*, IEEE, 1999, p. 293–308.

BLAKE, S.; BLACK, D.; CARLSON, M.; DAVIES, E.; WANG, Z.; WEISS, W. An architecture for differentiated service. 1998.

BOCHMANN, G.; KERHERVÉ, B.; LUTFIYYA, H.; SALEM, M.; YE, H. Introducing QoS to electronic commerce applications. *Electronic Commerce Technologies*, p. 138–147, 2001.

BRADEN, R.; CLARK, D.; SHENKER, S. Integrated services in the internet architecture: an overview. 1994.

CRAWLEY, E.; NAIR, R.; RAJAGOPALAN, B.; SANDICK, H. *A framework for QoS-based routing in the internet–RFC no. 2386*. Relatório Técnico, Internet RFC, August 1998, 1998.

DEBIAN Servidor apache http - modelo tradicional sem "threads". 2010.
Disponível em: http://packages.debian.org/pt-br/lenny/apache2-mpm-prefork

DOVROLIS, C.; STILIADIS, D. Relative differentiated services in the Internet: issues and mechanisms. In: *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, ACM, 1999, p. 204–205.

ESTRELLA, J. C. *Mecanismos de negociação no módulo de controle de admissão da arquitetura de servidor web com diferenciação de servicos (swds)*. Dissertação de Mestrado, ICMC - USP, 2006.

FIGUEIREDO, R. N. *Avaliação de algoritmos de controle de congestionamento como controle de admissão em um modelo de servidores web com diferenciação de serviços*. Dissertação de Mestrado, Universidade de São Paulo, 2011.

FOUNDATION Apache http server. 2010a.
Disponível em: http://httpd.apache.org/docs/2.0/mod/mod_proxy.html

FOUNDATION Apache http server. 2010b.
Disponível em: http://httpd.apache.org/docs/2.0/misc/rewriteguide.html

FOUNDATION Apache http server. 2010c.
Disponível em: http://httpd.apache.org/docs/2.0/mod/mod_status.html

FU, D.; MIN, L.; QIUYONG, Z.; YUEQIN, Z. A Framework of Web QoS Control through Self-Tuning Based on the Request Classification. In: *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM'08. 4th International Conference on*, IEEE, 2008, p. 1–4.

GRANT, D. Web server clustering. 2000.
Disponível em: http://technet.microsoft.com/en-us/library/bb742600.aspx#ECAA

GRÖNE, B.; KNÖPFEL, A.; KUGEL, R.; SCHMIDT, O. *The Apache modeling project*. Univ.-Verl., 2004.

HIRATA, R. Otimizando servidores web de alta demanda. 2002.

HUANG, Y.; FANG, C. Load balancing for clusters of VOD servers. *Information Sciences*, v. 164, n. 1-4, p. 113–138, 2004.

JIAO, Y.; WANG, W. Design and Implementation of Load Balancing of Distributed-system-based Web Server. In: *Electronic Commerce and Security (ISECS), 2010 Third International Symposium on*, IEEE, 2010, p. 337–342.

KAMIENSKI, C.; SADOK, D. Qualidade de Serviço na Internet. *minicurso, 18o SBRC, Belo Horizonte/MG*, 2000.

KANODIA, V.; KNIGHTLY, E. Ensuring latency targets in multiclass Web servers. *Parallel and Distributed Systems, IEEE Transactions on*, v. 14, n. 1, p. 84–93, 2003.

LEE, S.; LUI, J.; YAU, D. A proportional-delay DiffServ-enabled Web server: admission control and dynamic adaptation. *Parallel and Distributed Systems, IEEE Transactions on*, v. 15, n. 5, p. 385–400, 2004.

LI, J.; WANG, Y.; ZHANG, J. Research of web qos control model based on dynamic resource reallocation scheme. In: *Proceedings of the 2008 International Symposium on Information Science and Engieering - Volume 02*, Washington, DC, USA: IEEE Computer Society, 2008, p. 103–106.
Disponível em: http://portal.acm.org/citation.cfm?id=1493611.1493887

LIMA, C. ORPIS: um modelo de consistência de conteúdo replicado em servidores Web distribuídos. 2003.

LIU, G. A Dynamic Load-Balancing Algorithm for Heterogeneous Web Server Cluster [J]. *Chinese Journal of Computers*, v. 2, 2005.

MESSIAS, V.; ESTRELLA, J.; SANTANA, M.; SANTANA, R.; BRUSCHI, S.; TEIXEIRA, M. Implementação de um Protótipo de Servidor Web Distribuído com Diferenciação de Serviços. *XIII Simpósio Brasileiro de Sistemas Multimídia e Web WebMedia*, v. 2007, 2007.

MOSBERGER, D.; JIN, T. httperf – a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, v. 26, p. 31–37, 1998.
Disponível em: http://doi.acm.org/10.1145/306225.306235

MOURÃO, H.; SANTANA, M.; SANTANA, R.; BARBATO, A. Reconhecimento de sessões em um modelo para servidor web com diferenciação de serviços. 2007.

NETCRAFT Popularidade dos servidores web. 2011.
Disponível em: http://news.netcraft.com

NETWORKS Introduction to quality of service (qos). 2003.
Disponível em: http://www.nortel.com

PADILLA, J.; PARADELLS, J. Communication Networks IntServ6: an approach to support QoS over IPv6 wired and wireless networks. *European Transactions on Telecommunications*, v. 19, n. 6, p. 635–652, 2008.

POGGI, N.; MORENO, T.; BERRAL, J. L.; GAVALDÀ, R.; TORRES, J. Web customer modeling for automated session prioritization on high traffic sites. In: *Proceedings of the 11th international conference on User Modeling*, UM '07, Berlin, Heidelberg: Springer-Verlag, 2007, p. 450–454 (*UM '07*,).
Disponível em: http://dx.doi.org/10.1007/978-3-540-73078-1_63

SAITO, P. T. M. *Provisão integrada de qos relativa e absoluta em serviços computacionais interativos com requisitos de responsividade de tempo real*. Dissertação de Mestrado, Universidade de São Paulo, 2010.

SEMPREBOM, T. *Diferenciação de serviços em sevidores web baseada em escalonamento adaptativo e controle de admissão*. Dissertação de Mestrado, Universidade Federal de Santa Cartatina, 2007.

SEMPREBOM, T.; OLIVEIRA, R.; MONTEZ, C. Classes de Serviço em Servidores Web Apache Através de Escalonamento Adaptativo e Controle de Admissão. *Proc. In: XII Simpósio Brasileiro de Sistemas Multimídia e Web*, p. 273–282, 2006.

SERRA, A.; GAITI, D.; BARROSO, G.; BOUDY, J. Assuring qos differentiation and load balancing on web servers clusters. In: *Control Applications, 2005. CCA 2005. Proceedings of 2005 IEEE Conference on*, 2005, p. 885 –890.

SHAN, Z.; LIN, C.; MARINESCU, D. C.; YANG, Y. Modeling and performance analysis of qos-aware load balancing of web-server clusters. *Comput. Netw.*, v. 40, p. 235–256, 2002.
Disponível em: http://portal.acm.org/citation.cfm?id=635909.635912

SHAN, Z.; LIN, C.; WEI, Y. Prototype implementation and performance evaluation of a qos-based web server. In: *Proceedings of the IEEE International Workshop*, Washington, DC, USA: IEEE Computer Society, 2005, p. 199–206.
Disponível em: http://portal.acm.org/citation.cfm?id=1105001.1105565

SHARIFIAN, S.; MOTAMEDI, S. A.; AKBARI, M. K. A content-based load balancing algorithm with admission control for cluster web servers. *Future Gener. Comput. Syst.*, v. 24, p. 775–787, 2008.
Disponível em: http://portal.acm.org/citation.cfm?id=1390856.1390912

SHARIFIAN, S.; MOTAMEDI, S. A.; AKBARI, M. K. An approximation-based load-balancing algorithm with admission control for cluster web servers with dynamic workloads. *J. Supercomput.*, v. 53, p. 440–463, 2010.
Disponível em: http://dx.doi.org/10.1007/s11227-009-0303-8

SILVA, L. H. C. *W4gen: Um gerador de cargas de trabalho sintéticas para a avaliação de desempenho de modelos de servidores web*. Dissertação de Mestrado, Universidade de São Paulo, 2006.

SWAIN, M.; KIM, Y. A study of data source selection in distributed web server system. In: *Southeastcon, 2009. SOUTHEASTCON'09. IEEE*, IEEE, 2009, p. 311–316.

TANENBAUM, A. *Computer Networks. 4th*. 2003.

TEIXEIRA, M.; SANTANA, M.; SANTANA, R. Using adaptive priority scheduling for service differentiation QoS-aware Web servers. In: *Performance, Computing, and Communications, 2004 IEEE International Conference on*, IEEE, 2005, p. 279–285.

TEIXEIRA, M. M. *Suporte a serviços diferenciados em servidores web: Modelos e algoritmos*. Tese de Doutoramento, ICMC - USP, 2004.

TRALDI, O.; BARBATO, A.; SANTANA, R. Service differentiating algorithms for QoS-enabled web servers. In: *Proceedings of the 12th Brazilian Symposium on Multimedia and the web*, ACM, 2006, p. 263–272.

WOJTANOWICZ, K.; ZABIEROWSKI, W.; NAPIERALSKI, A. Quality of Service in wireless networks. In: *CAD Systems in Microelectronics, 2009. CADSM 2009. 10th International Conference-The Experience of Designing and Application of*, IEEE, 2009, p. 168–170.

ZHAO, W.; OLSHEFSKI, D.; SCHULZRINNE, H. Internet quality of service: An overview. *Columbia University, New York, New York, Technical Report CUCS-003-00*, 2000.