UM ESTUDO DO SERVIDOR RIO EM UMA REDE DE ALTA VELOCIDADE

Marcello Ribeiro Valle

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Edmundo Albuquerque de Souza e Silva, Ph.D.

Profa. Rosa Maria Meri Leão, Dr.

Prof. José Ferreira de Rezende, Dr.

Prof. Daniel Ratton Figueiredo, Ph.D.

Profa. Anna Dolejsi Santos, DSc.

RIO DE JANEIRO, RJ - BRASIL

AGOSTO DE 2007

VALLE, MARCELLO RIBEIRO

Um estudo do servidor RIO em uma rede de alta velocidade [Rio de Janeiro] 2007

XVII, 95 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2007)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Educação a Distância
2. Avaliação de Desempenho
3. Qualidade de Serviço
4. Servidores Multimídia
5. Redes de Computadores

I. COPPE/UFRJ II. Título (Série)

Agradecimentos

Ao fim de mais uma etapa não posso deixar de agradecer a todos do LAND - – Laboratório de Modelagem, Análise e Desenvolvimento de Redes e Sistemas de Computação. Em especial aos professores Rosa Maria Meri Leão e Edmundo A. de Souza e Silva que me orientaram durante este trabalho.

Agradeço aos meus pais, Orlando e Marlene, pelo apoio e encorajamento, que foram fundamentais para que eu chegasse até aqui. Também não posso esquecer da minha namorada, Fabiana, pela paciência e compreensão durante os momentos de ausência.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UM ESTUDO DO SERVIDOR RIO EM UMA REDE DE ALTA VELOCIDADE

Marcello Ribeiro Valle

Agosto/2007

Orientadores: Edmundo de Souza e Silva
Rosa Maria Meri Leão

Programa: Engenharia de Sistemas e Computação

O trabalho apresentado nessa dissertação é um estudo sobre o servidor multimídia RIO em uma rede de alta velocidade usando uma carga real de usuários interativos. Foram coletados dados de experimentos reais relativos à qualidade de serviço percebida pelos clientes bem como ao tráfego gerado por um dos nós de armazenamento. Com essas medidas foi possível avaliar o impacto de escolhas de projeto tais como: a utilização de *cache* pelos clientes, a replicação de conteúdo do servidor e a distribuição de nós de armazenamento em redes heterogêneas. Foi realizada também uma comparação entre os resultados obtidos com os experimentos com uma carga de trabalho real e uma sintética, gerada por um modelo desenvolvido em um trabalho paralelo a este. As principais conclusões foram: A adição de um *storage server* em uma rede inferior as demais pode trazer ganhos para o sistema em alguns cenários; apesar do bom balanceamento de carga promovido pela técnica de alocação aleatória de blocos é possível que ocorra o desbalanceamento de curto prazo, o que pode aumentar a latência do sistema, porém, a replicação do conteúdo pode minimizar o desbalanceamento de curto prazo; os resultados obtidos com a carga sintética foram bem próximos dos obtidos com carga real, o que demonstra a boa acurácia do modelo.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A STUDY OF RIO SERVER ON A HIGH SPEED NETWORK

Marcello Ribeiro Valle

August/2007

Advisors: Edmundo de Souza e Silva
Rosa Maria Meri Leão

Department: Computer and System Engineering

The work performed in this dissertation is the performance study of the multimedia server RIO on a high speed network submitted to a real interactive workload. We collected data necessary to evaluate the QoS perceived by the users as well as the traffic originated from the storage nodes. From the collected measures we evaluated the impact of different server configurations such as: a) the use of cache in the client nodes; b)content replication and c) distribution of storage nodes through heterogeneus networks. we have also compared the results obtained from experiments using a real workload and a synthetic workload, generated by a model developed in another work. The most relevant conclusions are: a) the addition of a storage server located in the regular Internet (not the giga net) can improve the performance in some scenarios, b) despite the load balancing resulting from the random data allocation, we can observe short-term unbalancing that may increase the system's latency. However, content replication can minimize short-temp unbalancing. The results obtained from the use of synthetic workload are almost identical to those obtained from the real workload and therefore we conclude that the synthetic workload can be used in the large scale stress performance experiments of the RIO server.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário de Redes

ACK	:	Mensagem de Confirmação (*Acknowledge*).
Canal	:	Meio através do qual trafegam os pacotes (*Link*).
Codec	:	Codificador/decodificador de sinais de áudio ou vídeo.
FEC	:	Mecanismo de Correção de Erros (*Forward Error Correction*).
Internet	:	A rede de computadores mais popular atualmente, composta por um conjunto de canais (*links*), computadores (*hosts*) e roteadores (*routers*), prestando serviços como correio eletrônico (*e-mail*), web e suporte a video-conferências, dentre outros, para milhões de usuários ao redor do mundo.
Jitter	:	A medida da variação do atraso em pacotes enviados sucessivamente por uma rede.
MOS	:	Nota de opinião média – um indicador subjetivo de QoS (*mean opinion score*).
QoS	:	Qualidade de Serviço (*Quality of Service*).
Roteador	:	Um dispositivo que recebe mensagens e as encaminha para seus destinos, procurando selecionar a melhor rota disponível.
RTP	:	Protocolo de Tempo-Real (*Real-Time Protocol*).
RTT	:	Tempo para um pacote trafegar da origem ao destino, e voltar do destino para a origem (*Round Trip Time*).
Taxa de recepção	:	Taxa, em bits por segundo, com a qual os dados são recebidos por um computador na rede.

TCP : Protocolo de Controle de Transmissão (*Transmission Control Protocol*). O protocolo de transmissão de dados mais utilizado na Internet, que oferece garantia de entrega dos dados, controle de congestionamento e controle de fluxo.

UDP : Protocolo de Datagrama do Usuário (*User Datagram Protocol*). Protocolo de transmissão de dados minimalista, que não oferece garantia de entrega dos dados, controle de congestionamento ou controle de fluxo. É usado primordialmente para transmissão de dados multimídia como vídeo e voz.

Capítulo 1

Introdução

A queda do custo de conexões banda larga e o aumento da diversidade de serviços oferecidos através da Rede são fatores que contribuem para consagrar a Internet como uma nova mídia de comunicação, entretenimento e educação ao alcance de uma parcela significativa da população.

Essa popularização, aliada aos avanços na tecnologia de redes e a maturidade das técnicas de codificação de objetos multimídia, possibilitou o surgimento de novos serviços relacionados à transmissão multimídia via Internet, tais como ensino a distância, aplicações de voz, visualização de vídeos, videoconferência, entre outras. Dois exemplos desse tipo de serviço são o YouTube[17], página que permite aos usuários o compartilhamento de vídeos, e o Skype[16], que permite a comunicação via Internet utilizando voz e vídeo.

Dos serviços citados, vale um destaque para o ensino a distância por estar intimamente relacionado à proposta deste trabalho e por permitir a disseminação do conhecimento de maneira mais democrática, atingindo uma parcela da população que, por diversos fatores, não têm acesso aos métodos de ensino convencionais.

Devido a sua natureza, a transmissão de conteúdo multimídia possui requisitos de qualidade de serviço (QoS) muito diferentes do conteúdo *web* tradicional (texto e imagem). Para transmissão de conteúdo multimídia, seria ideal um tratamento diferenciado entre pacotes gerados por aplicações multimídia e por aplicações *web*

tradicionais. Porém, a infraestrutura atual da Internet não oferece garantias de qualidade e nem mecanismos para diferenciação de serviços.

Um objeto multimídia consiste em uma seqüência de dados (geralmente quadros de um vídeo ou amostras de áudio) que deve ser transmitida respeitando rígidas restrições de tempo. Perdas durante a transmissão são aceitáveis, mas se ocorrerem em excesso podem comprometer a qualidade de reprodução.

Devido a essa limitação, para atingir a QoS necessária para aplicações multimídia diversos mecanismos foram desenvolvidos na camada de aplicação. Por isso, um desafio a ser superado está relacionado ao projeto de servidores que suportem esse tipo de aplicação, um servidor multimídia deve ser robusto o suficiente para armazenar e manipular uma grande quantidade de informação obedecendo a restrições de tempo muito rígidas.

A cada dia novos provedores de serviços multimídia surgem e o número de usuários não para de crescer. Isso torna necessário o desenvolvimento e aprimoramento de técnicas buscando atender essa demanda de maneira satisfatória. Como exemplo, em um artigo do *Wall Street Jounal*[33] publicada em 2006, foi observado, em um mês, um aumento de 20% na quantidade de vídeos hospedados no Youtube.

1.1 Objetivo e Motivação

Nesse trabalho, é feito um estudo do comportamento RIO(*Ramdomized I/O Multimedia Storage Server*). Ele foi inicialmente desenvolvido pela UCLA(*University of California Los Angeles*) e totalmente re-projetado e aprimorado através de um programa de cooperação UFRJ/UCLA. É um servidor multimídia distribuído que tem como principal característica a alocação aleatória dos blocos de cada objeto armazenado entre os discos que compõe o sistema. Atualmente, o RIO é desenvolvido no LAND (Laboratório de Análise e Desenvolvimento de Sistemas) onde passa por um processo contínuo de aprimoramento e adição de novas funcionalidades. O RIO também vem sendo utilizado em um projeto de ensino a distância do CEDERJ.

Serão realizados vários experimentos com o servidor RIO avaliando escolhas de projeto tais como: utilização de *cache* pelos clientes, replicação de conteúdo e o posicionamento de nós de armazenamento em redes com características diferentes. O objetivo do trabalho é definir o quanto cada escolha de projeto pode impactar na qualidade do serviço prestado. A carga utilizada para a realização dos experimento foi obtida a partir da coleta de *logs* de comportamento de alunos do CEDERJ (Centro de Educação Superior a Distância do Estado do Rio de Janeiro).

O servidor multimídia RIO vai atuar distribuído entre algumas instituições de ensino, a saber: UFF (Universidade Federal Fluminense), Fiocruz, UFMG (Universidade Federal de Minas Gerais) e UFRJ (Universidade Federal do RIO de Janeiro). Os experimentos foram realizados com o servidor operando em uma rede com alta capacidade, a Rede Giga, coordenada pela RNP(Rede Nacional de Pesquisa) e o CPqD(Centro de Pesquisas e Desenvolvimento). A Rede Giga disponibiliza taxas de até 10 Gbit/s para as aplicações que a utilizam e o seu objetivo é auxiliar no desenvolvimento de projetos em redes de comunicação de dados.

A ligação entre todas as instituições de ensino é feita através da Rede Giga, exceto a UFMG que será acessada através da Internet. Cada instituição vai possuir um servidor de armazenamento, de modo que o conteúdo acessado vai estar distribuído entre os servidores localizados nessas instituições.

Esse trabalho faz parte do projeto DIVERGE (Distribuição de Vídeos em Larga Escala sobre Redes Giga, com Aplicações a Educação), uma descrição mais detalhada da proposta desse projeto será apresentada no capítulo 4.

Os *logs* de comportamento foram usados diretamente para a realização dos experimentos e também de maneira indireta, através da alimentação de um modelo capaz de gerar carga sintética para o servidor RIO. Foi necessária a utilização de um emulador desenvolvido durante um trabalho anterior [34]. Esse emulador é alimentado pelos logs de comportamento e, de maneira semelhante à um cliente comum, tem a função de enviar as requisições de blocos para o servidor e esperar pela recepção dos mesmos. Sua principal diferença é o fato de que o vídeo não é exibido na tela, dessa maneira a máquina não tem a sobrecarga inerente a decodificação do vídeo. Isso

permite que vários emuladores sejam disparados em cada máquina participante, o que é fundamental para a viabilidade dos testes. Durante o desenvolvimento desse trabalho, as funcionalidades do emulador foram estendidas para calcular a estatísticas relativas à quantidade de informação recebida pelo cliente. Essas estatísticas são importantes para avaliar a qualidade do serviço prestado aos clientes.

A utilização da Rede Giga é fundamental para a realização dos experimentos em larga escala. Permitindo também a geração de uma carga considerável bem como a participação de outras instituições de pesquisa nos experimentos. Sem a Rede Giga, a distribuição do servidor não teria resultados satisfatórios devido ao retardo de transmissão em redes tradicionais. Em um dos ambientes de teste, um dos nós do servidor vai estar funcionando na UFMG e os dados serão trocados via Internet. Esse cenário vai contribuir para aprofundar o conhecimento sobre o funcionamento do servidor RIO em ambientes heterogêneos.

Um trabalho anterior já foi desenvolvido visando estudar o desempenho do servidor em um ambiente heterogêneo [21], naquele trabalho os clientes eram caracterizados por um acesso seqüencial de curta duração ao conteúdo do servidor. O trabalho dessa dissertação se diferencia do anterior porque agora foram utilizados clientes com comportamento interativo e de longa duração e pela coleta de medidas do lado servidor.

1.2 Contribuição

Uma das contribuições deste trabalho está em uma melhor compreensão do comportamento do servidor RIO graças à experimentos em uma rede de alta velocidade, usando uma carga real, coletada de um projeto de ensino distância, que é caracterizada por clientes com um comportamento interativo. Os experimentos possibilitaram:

- A avaliação da escalabilidade do servidor;
- A avaliação de estratégias de uso de *cache* no cliente;

- A avaliação de uso de replicação de blocos no servidor;
- A avaliação do tráfego enviado pelos servidores de armazenamento;
- O estudo da acurácia de um modelo para geração de carga sintética.

Outras contribuições importantes foram:

- A implantação de uma infra-estrutura para a realização de experimentos em larga-escala, utilizando a rede giga. Que agora está disponível para ser utilizada em trabalhos futuros;
- A adição de novas funcionalidades ao emulador de clientes, de maneira que agora este pode ser utilizado como uma ferramenta para avaliação da qualidade do serviço prestado pelo servidor;
- A detecção e correção de problemas no código do cliente e do servidor RIO.

1.3 Organização do trabalho

A organização deste trabalho segue o esquema descrito a seguir: No capítulo 2 é feita uma revisão de propostas relacionadas a transmissão de conteúdo multimídia na Internet. No capítulo 3 é apresentada uma descrição da arquitetura do RIO. Uma descrição do ambiente onde os experimentos foram realizados pode ser encontrada no capítulo 4. Os resultados obtidos a partir dos experimentos bem como uma análise dos mesmos pode ser encontrada no capítulo 5. Por fim, as conclusões e propostas para trabalhos futuros podem ser encontrados no capítulo 6.

Capítulo 2

Aplicações Multimídia na Internet

A popularização da Internet, somada aos avanços na tecnologia de redes, à maturidade das técnicas de codificação de objetos multimídia e ao aumento do poder de processamento dos computadores, possibilitou o surgimento de aplicações de mídia contínua tais como áudio e vídeo sob demanda, bibliotecas digitais, ambientes de realidade virtual, educação a distância, entre outros. Essa demanda nem sempre pode ser atendida de maneira satisfatória já que a infra-estrutura da Internet não é a ideal para o suporte a esses tipos de aplicações.

Os objetos multimídia tais como vídeos, trechos de áudio, textos e figuras geralmente são codificados e armazenados em discos de um servidor multimídia. Devido a sua natureza, esses objetos podem apresentar uma grande variabilidade no seu tamanho de armazenamento: em um mesmo servidor podem ser encontrados objetos com apenas alguns kilobytes ou centenas de megabytes de tamanho. Para objetos multimídia, que geralmente possuem um tamanho mais elevado, a transmissão completa para o cliente, como acontece com objetos *web* tradicionais, pode provocar um atraso inaceitável para o início da sua reprodução. Visando contornar esse problema, muitos estudos têm sido realizados buscando desenvolver protocolos de *streaming*, esses tipos de protocolo permitem que o cliente decodifique e visualize o objeto multimídia, a medida que os pacotes de dados são enviador pelo provedor.

A utilização de protocolos de *streaming* impõe rígidos limites de tempo para a

recepção dos blocos no cliente, se esses limites não forem respeitados, a qualidade de exibição pode ficar comprometida. Essa característica dos protocolos de *streaming* é algo que deve ser levado em consideração durante o desenvolvimento de qualquer servidor multimídia. Outro fator complicador é o de que a maioria dos vídeos é codificada com taxas variáveis (VBR - *Variable Bit Rate*), o que torna os requisitos de tempo ainda mais complexos.

Também deve ser observada a alta carga imposta à rede por aplicações multimídia sob o ponto de vista de uma aplicação individual ou de um agregado das mesmas. Por este motivo, várias propostas vem sendo feitas com a intenção de utilizar os recursos da rede de maneira mais inteligente, através de técnicas tais como compartilhamento de fluxos, replicação de conteúdo próximo aos clientes, troca de conteúdo entre clientes, entre outras.

As aplicações multimídia podem ser classificadas de acordo com vários critérios, a seguir serão apresentados alguns desses critérios e as classificações possíveis a partir de então.

2.1 Classificações quanto a natureza do serviço prestado

Em [40], as aplicações multimídia são divididas de acordo com a natureza do serviço prestado, assim temos as seguintes classes.

2.1.1 Transmissão de Áudio e Vídeo Pré-Armazenados

Nessa classe de serviço, o conteúdo já foi previamente codificado e armazenado no servidor. O cliente então se conecta ao servidor e pode executar comandos como pausa, avanço e retrocesso. Isso é feito através do envio de requisições ao servidor que em resposta envia os blocos solicitados pelo cliente.

2.1.2 Transmissão de Áudio e Vídeo ao Vivo

Nessa classe de serviço, o conteúdo é codificado e transmitido ao mesmo tempo em que está sendo gerado. Por ser um conteúdo que está sendo gerado dinamicamente, o cliente tem um nível de interatividade limitado, só é possível fazer uma movimentação pelo conteúdo que já foi recebido caso este tenha sido armazenado.

2.1.3 Transmissão de Áudio e Vídeo Interativos em Tempo Real

Essa classe de serviço possibilita a comunicação entre diversos clientes utilizando áudio e vídeo. O conteúdo é codificado e transmitido ao mesmo tempo que é gerado. Devido a sua natureza, esse serviço possui as restrições de tempo muito estreitas já que, para evitar desconforto dos clientes, o tempo de envio dos pacotes tem que ser da ordem de milissegundos.

2.2 Classificações quanto a distribuição dos componentes

Os servidores multimídia também podem ser classificados de acordo com a distribuição e a função de cada componente participante do sistema.

2.2.1 VoD

A arquitetura mais simples para a transmissão de um fluxo multimídia é composta por um único servidor. Esse servidor será responsável pela indexação do conteúdo multimídia, atendimento das requisições dos clientes e envio dos dados. Esse tipo de arquitetura pode ser visto em trabalhos como [20] e [57] e está ilustrado na Figura 2.1.

Uma das limitações apresentadas por essa arquitetura diz respeito à possuir

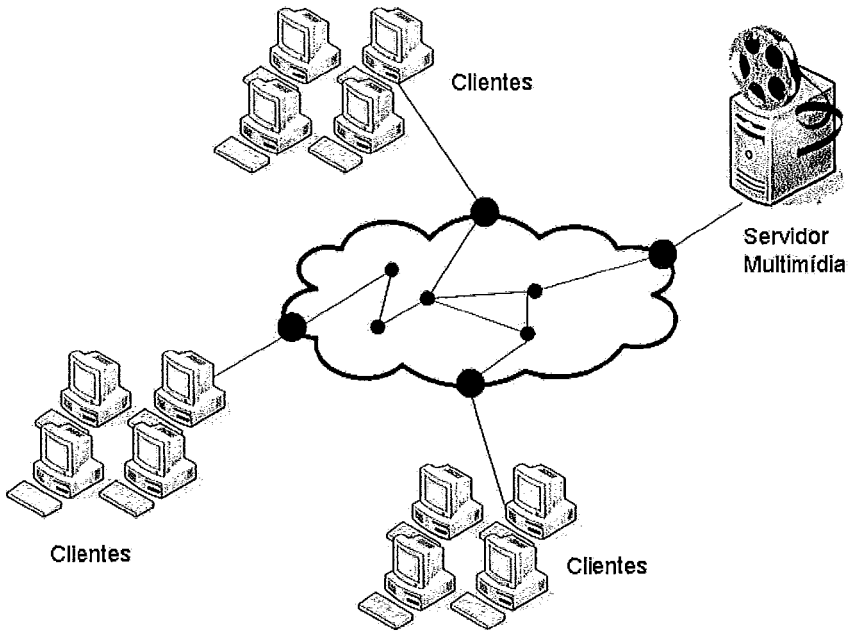

Figura 2.1: Arquitetura VOD

um único ponto de falha. Se o servidor multimídia falhar, todo o sistema ficará comprometido.

Para atender às necessidades de largura de banda e capacidade de armazenamento, um servidor desse tipo pode possuir centenas ou até milhares de discos rígidos. O tempo médio de falha em um sistema com essa proporção de discos pode ser muito baixo, comprometendo a disponibilidade do sistema [20]. Outra limitação relevante diz respeito à escalabilidade desse tipo de sistema. A transmissão de objetos multimídia pode necessitar de uma largura de banda de alguns Gbps (gigabits por segundo). Devido aos custos de transmissão, pode ser economicamente inviável atender à uma quantidade elevada de clientes com uma QoS aceitável utilizando um sistema baseado em uma arquitetura deste tipo [26].

2.2.2 DVoD

Em um servidor multimídia tradicional, o dispositivo onde os objetos multimídia estão armazenados pode possuir uma largura de banda limitada e geralmente está distante dos clientes em potencial. Visando atender um número maior de usuários

e diminuir os a largura de banda necessária é comum a divisão da responsabilidade pelo armazenamento e transmissão dos objetos entre vários servidores localizados próximos aos clientes. Esse tipo de arquitetura é chamada de servidor de vídeo distribuído ou DVoD (*Distributed Video on Demand*).

A principal vantagem desta técnica é o fato de que o custo de transmissão entre os servidores distribuídos e os clientes ser muito pequeno em comparação ao custo de transmissão a partir de um servidor central, o que possibilita um aumento na escalabilidade do sistema. Em [19] foi feito uma análise comparativa entre arquiteturas centralizadas e distribuídas.

Nesta técnica, os servidores podem funcionar como *proxies*, armazenando dinamicamente partes do conteúdo de um repositório central, de acordo com a requisições dos clientes mais próximos[22]. Também é possível que não exista um repositório central. Dessa maneira, os vídeos são distribuídos entre todos os servidores do sistema e podem ser replicados ou não. Esse é o caso do servidor RIO que será detalhado no capítulo 3.

Proxies

Uma das propostas de arquitetura DVoD faz uso de um repositório central onde os vídeos são armazenados e vários servidores distribuídos (*proxies*) posicionados próximos aos clientes em potencial. Esses servidores funcionam replicando dinamicamente o conteúdo do servidor central, de acordo com as requisições dos clientes. Dessa maneira, existe uma economia da largura de banda requerida entre o repositório central e os clientes em troca de um requerimento maior de capacidade de armazenamento. Essa troca geralmente é lucrativa, devido ao baixo custo dos dispositivos de armazenamento. Sendo assim, esses *proxies* devem armazenar o conteúdo que é acessado com mais freqüência e devem possuir uma capacidade de armazenamento e largura de banda suficiente para aliviar de maneira satisfatória a carga imposta sobre o repositório central. A Figura 2.2 representa uma arquitetura baseada em *proxies*.

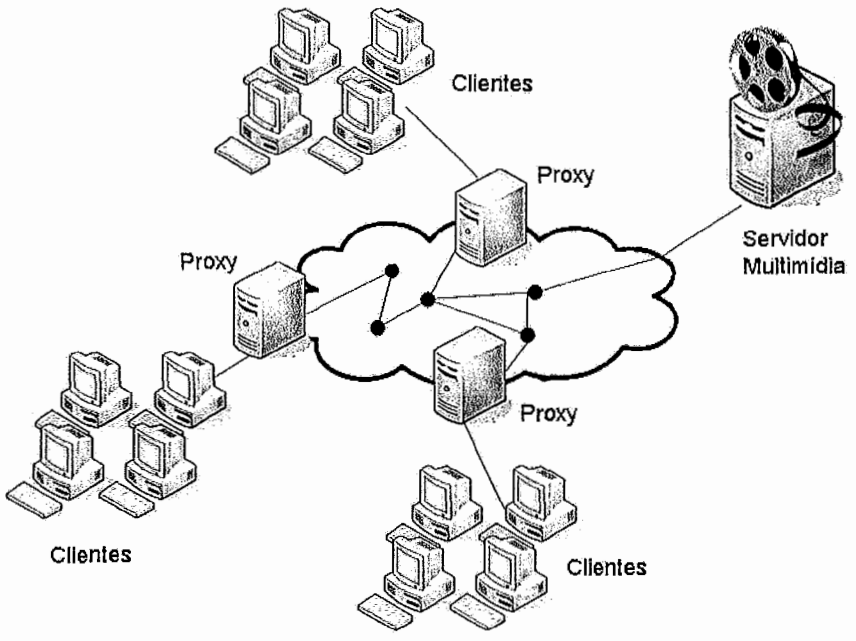

Figura 2.2: Arquitetura baseada em Proxies

O custo/benefício de uma arquitetura que usa *proxies* é extremamente dependente de fatores que devem ser levados em conta durante o projeto do sistema. Alguns fatores mais importantes são: o meio de transmissão do repositório central para os *proxies* (*unicast* ou *multicast*), possibilidade de trocas de conteúdo entre *proxies*, uso de técnicas de compartilhamento de banda, trecho do vídeo a ser armazenado no *proxy* (prefixo ou vídeo inteiro) e etc.

Existem vários estudos na literatura tentando definir qual a melhor escolha ao se desenvolver esse tipo de sistema, em [22], é feito um estudo de algumas variações na arquitetura: são estudados casos onde o repositório pode transmitir dados via *multicast* ou somente *unicast*, também são estudados casos onde os *proxies* podem trocar o conteúdo entre si. Algumas discussões a respeito de quais trechos de cada vídeo devem ser armazenados no *proxy* também são feitas.

Em [18], é feito um estudo sobre o melhor conteúdo a ser armazenado em cada *proxy* em função da capacidade de transmissão *multicast* entre o repositório central e os *proxies*. Naquele trabalho também são feitos estudos com a utilização do protocolo *badwidth skimming* para compartilhamento de fluxos.

Em [56], são feitos estudos para o caso em que o meio de transmissão entre os repositórios e o *proxy* é somente *unicast*. A transmissão dos dados entre os *proxies* e os clientes pode ser *multicast*. Também foi estudada a utilização de alguns protocolos de compartilhamento de fluxo (*batching*, *patching* e *stream merging*) para a transmissão entre o *proxy* e o cliente.

Em [35], é considerado o caso onde o repositório central transmite o sufixo do vídeo via *multicast* em intervalos de tempo regulares (difusão periódica). Cada *proxy* utiliza um esquema que combina as técnicas de *patching* e difusão periódica. Naquele trabalho, os *proxies* tem a capacidade de alterar o esquema de transmissão dinamicamente de acordo com a popularidade dos vídeos. Mais discussões sobre esse tema podem ser encontradas em [23] e [30].

Redes de distribuição de conteúdo

As redes de distribuição de conteúdo (*Content Delivery Networks* ou CDNs) se caracteriza por uma coleção de servidores dedicados e estrategicamente localizados através da Internet. Esses servidores têm a função de hospedar e distribuir conteúdos de qualquer tipo, desde páginas da *web* até conteúdo multimídia. Dessa maneira, esses provedores podem delegar a responsabilidade de armazenamento e distribuição do conteúdo para as CDNs [49]. As CDNs de grande porte podem possuir milhares de servidores geograficamente distribuídos. Como exemplo, a empresa Akamai [1] publicou a informação de que possui 18.000 servidores espalhados por 70 países, de maneira a ficarem próximos de clientes em potencial.

Além do tamanho, outra característica que diferencia uma CDN de um servidor VoD com *proxies* é o fato dos servidores serem povoados pró-ativamente, buscando antecipar futuras demandas [29], enquanto arquiteturas com *proxies* se baseiam em requisições do cliente. Outra característica diz respeito ao mecanismo que redireciona os clientes para um servidor "bom", que não faz parte de arquiteturas VoD com *proxies*. A Figura 2.3 representa a arquitetura de uma CDN genérica.

Para aumentar a disponibilidade do conteúdo armazenado, é comum a replica-

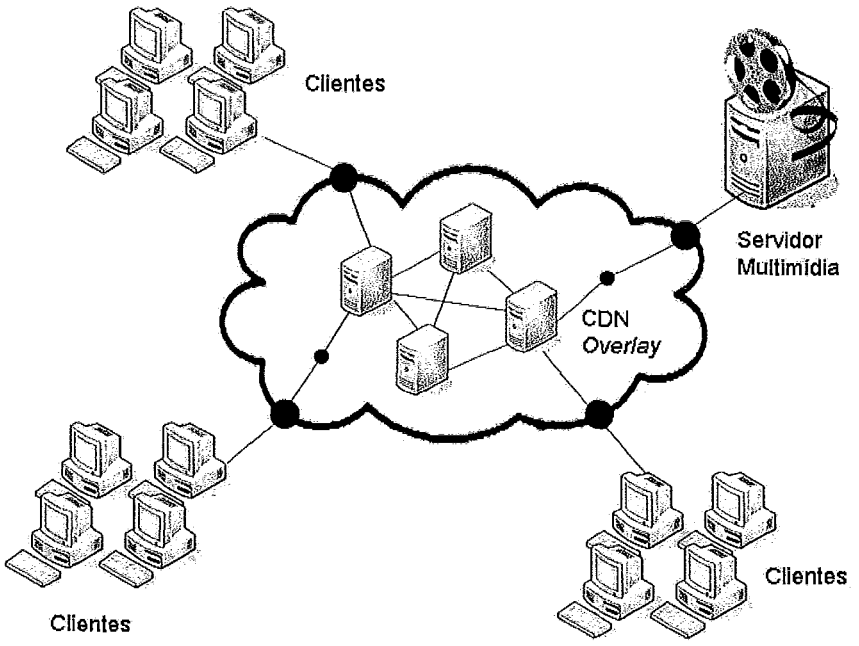

Figura 2.3: *Rede de distribuição de conteúdo*

ção dos dados em vários servidores diferentes. O cliente, ao solicitar um conteúdo qualquer, é redirecionado para o servidor que possuir a menor carga. Para realizar esse trabalho, uma CDN deve analisar informações a respeito das condições da rede, carga de cada servidor que possui a réplica, além de informações geográficas e da topologia da rede. O redirecionamento de clientes normalmente é feito via resolução de DNS [50, 39].

Alguns trabalhos como [38] têm mostrado que as CDNs dificilmente redirecionam um cliente para o servidor com menor carga, o que pode ser observado é que essas arquiteturas evitam o redirecionamento para o pior servidor (com maior carga de trabalho). Apesar de possuírem uma tecnologia fechada, estudos sobre o comportamento das CDNs foram possíveis graças a engenharia reversa, como em [29].

2.2.3 VoD Cooperativo

Em um sistema P2P, cada membro colabora para a formação de um sistema distribuído visando a troca de conteúdo. Cada participante do sistema pode funcionar

como servidor e cliente, o conteúdo baixado por um participante torna-se disponível para outros participantes do sistema.

Sistemas de compartilhamento de arquivos P2P ou *Peer to Peer* (Par à Par) tem se tornado cada vez mais populares nos últimos anos, sistemas tais como *Napster* [11], *Kazaa* [9], *Gnutella* [8] e *BitTorrent* [2] têm atraído cada vez mais usuários. Estudos têm mostrado que o tráfego gerado por aplicações P2P já são responsáveis pelo consumo de uma parcela significativa da banda em algumas redes privadas [49].

Originalmente, os sistemas P2P não foram desenvolvidos visando a transmissão de fluxos multimídia mas, recentemente, algumas arquiteturas têm sido propostas visando a transmissão de tais fluxos de maneira cooperativa [27, 54, 53, 36]. Para permitir a colaboração, cada cliente possui uma *cache* com capacidade para armazenar uma parte do vídeo que está sendo transmitido. Esse cliente pode transmitir o que está em sua *cache* para novos clientes se possuir largura de banda suficiente. Essa arquitetura tem como vantagens a redução da carga imposta ao servidor central e a alta disponibilidade do conteúdo. Uma desvantagem desse sistema diz respeito à latência para o início da exibição do vídeo. Ao chegar no sistema, um novo cliente deve entrar em contato com alguns dos nós participantes para saber se deve receber o fluxo direto do servidor ou de outro participante e o tempo gasto com essa comunicação pode ser considerável [27]. Outro ponto que deve ser observado diz respeito à possibilidade de algum participante se tornar "órfão", isto é, o participante do qual ele está recebendo o fluxo deixar o sistema. Devem existir meios para que um participante órfão possa receber o fluxo de outra fonte sem causar uma interrupção no vídeo exibido.

2.3 Classificações quanto ao nível de interatividade oferecida

Os servidores multimídia também podem ser classificados de acordo com o nível de interatividade oferecida aos clientes [42]. Os tipos mais comuns são:

- No-VoD: Esse tipo é bem similar a uma transmissão *broadcast*, onde nenhuma interatividade é oferecida ao cliente e este é apenas um participante passivo do sistema.

- QVoD (*Quasi-Video-On-Demand*): Os clientes são agrupados de acordo com seus interesses e algumas operações de movimentação podem ser executadas trocando o cliente de grupos.

- NVod (*Near-Video-On-Demand*): Algumas operações de movimentação podem ser efetuadas em tempo discreto. Esse tipo de serviço geralmente é implementado utilizando vários canais com a mesma programação deslocada em intervalos regulares de tempo.

- TVod (*True-Video-On-Demand*): O usuário controla completamente o fluxo do vídeo, podendo se movimentar livremente em seu conteúdo.

2.4 Arquitetura Geral de um servidor multimídia

Um servidor multimídia é fundamental para a transmissão de áudio/vídeo sob demanda. Mesmo com a utilização de uma rede de alta capacidade o servidor precisa ser robusto o suficiente para suportar a alta carga imposta por esse tipo de aplicação. O servidor é responsável pela recepção dos pedidos feitos pelos clientes, pela transmissão dos dados requisitados e pelo armazenamento dos dados nos discos do sistema.

Para otimizar o processo de transmissão de um objeto multimídia é comum que o servidor divida cada objeto em pedaços menores, chamados de blocos. Essa técnica simplifica o processo de armazenamento do objeto e também a sua transmissão. Cabe ao cliente, após a recepção de cada bloco, reconstruir o objeto original.

As aplicações multimídia possuem severas restrições de tempo na entrega de blocos multimídia. O tempo de entrega de um bloco pode ser influenciado por diversos fatores que não podem ser controlados, tais como o atraso durante a transmissão na

rede. Por isso, é de suma importância que um servidor multimídia seja projetado de maneira a contornar esses problemas visando oferecer uma **QoS** satisfatória.

2.4.1 Armazenamento em Disco

Servidores multimídia devem armazenar um número razoável de objetos que geralmente possuem um tamanho em disco muito elevado. Em muitas situações, para aumentar a confiabilidade do sistema costuma-se replicar alguns blocos do vídeo em discos diferentes. Com isso faz-se necessária a utilização de vários discos em um único servidor multimídia, o que pode resolver não somente problemas com relação à capacidade de armazenamento mas também com relação à banda exigida dos dispositivos de entrada/saída [26].

Para evitar as limitações de largura de banda de disco e capacidade de armazenamento, é comum que um servidor possua vários discos em paralelo para possibilitar o aumento do número de clientes atendidos simultaneamente.

O conjunto de setores enviados pelo servidor é chamado de bloco de dados. Esses precisam ser armazenados em um *buffer* no cliente para serem consumidos. Enquanto um cliente consome esse bloco de dados, o servidor pode servir a outros clientes. Essa técnica permite que o servidor divida a banda de disco entre todos os clientes ativos e é bastante eficaz, porque a largura de banda dos discos excede em muito a taxa de exibição dos vídeos assistidos pelos clientes.

Para que o sistema funcione corretamente, é fundamental que os discos tenham uma carga relativamente igual, ou seja, não pode haver um disco que possua uma carga muito pequena em detrimento de outro que esteja sobrecarregado. Isso pode ser obtido através da distribuição dos blocos do objeto multimídia entre diversos discos buscando evitar que um usuário só faça requisições aos blocos de um mesmo disco durante a transmissão de um objeto multimídia.

A definição de um padrão para a distribuição dos blocos nos discos é crítica para a obtenção de um bom desempenho do sistema. A abordagem mais simples seria a de se armazenar todos os blocos de objeto multimídia em um mesmo disco.

Essa abordagem tem como principal vantagem simplicidade de implementação. Em contrapartida, ela apresenta a seguinte desvantagem: Se um objeto receber uma carga muito alta de pedidos, o disco pode ficar sobrecarregado, o que vai causar o desbalanceamento da carga e prejudicar o desempenho do mesmo. Existem algumas técnicas para distribuição dos blocos no disco, entre elas, as mais populares são: a técnica *Data Striping*[48] e *Random Data Allocation* [45].

A técnica *Data Striping* visa a distribuição dos blocos de dados entre todos os discos do sistema seguindo uma política *Round Robin*. Essa distribuição é feita de maneira a possibilitar que as requisições dos clientes se movam através de discos adjacentes. Dessa maneira a largura de banda total do sistema é dividida igualmente entre os discos. Essa técnica permite um balanceamento de carga mais efetivo no sistema e conseqüentemente um número maior de clientes pode ser atendido simultaneamente. A utilização da técnica de *striping* pressupõe que o acesso aos objetos multimídia será seqüencial na maioria das vezes. Se houver um comportamento muito interativo por parte dos clientes, pode ocorrer o desbalanceamento dos discos do sistema. A figura 2.4 ilustra como funciona a técnica.

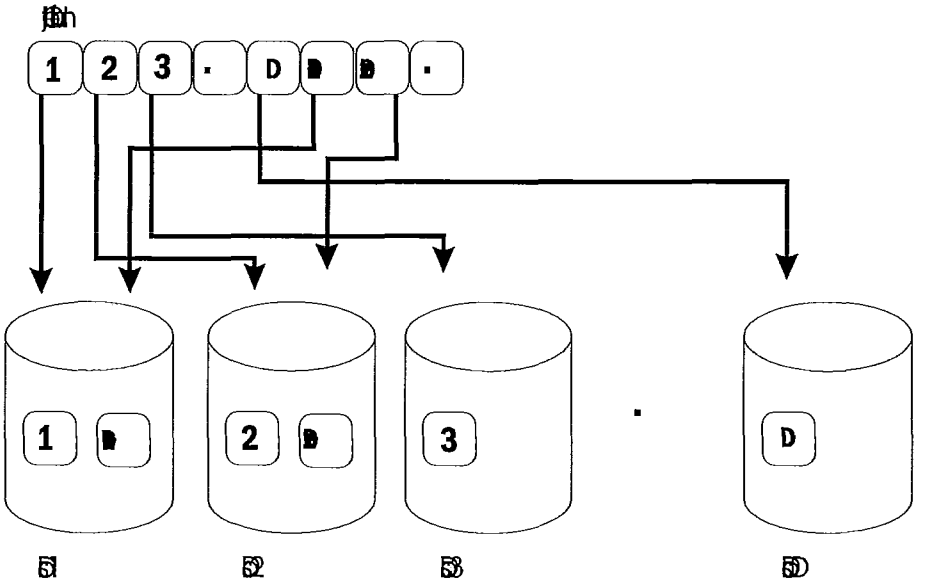

Figura 2.4: Técnica *striping*

Na técnica *random data allocation*, ou alocação aleatória de blocos, cada bloco é armazenado aleatoriamente em um disco do sistema. Essa técnica não possui o inconveniente apresentado pela técnica de *striping* no que diz respeito à interatividade

dos clientes. A localização dos blocos de um objeto é independente do padrão de acesso. A implementação da técnica de alocação aleatória dos blocos é mais simples e esta possui uma performance igual ou superior à técnica de *striping* [48]. Em [45] é feito um estudo comparativo das técnicas de *striping* e *random data allocation* em ambientes onde o modo de operação é dirigido pelo servidor (baseado em ciclos) ou pelo cliente (baseado em requisições do cliente). Este trabalho chega à conclusão de que a alocação aleatória é a mais apropriada para lidar com diferentes padrões de acesso e interatividade dos clientes. Na Figura 2.5 é ilustrada como funciona a técnica de alocação aleatória.

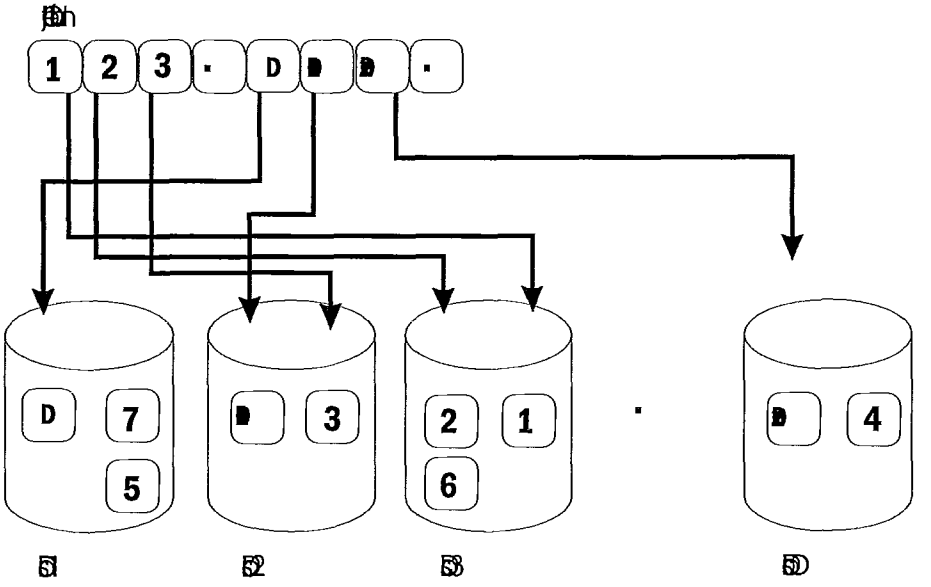

Figura 2.5: Alocação aleatória de blocos

Devido as suas características, a alocação aleatória provê um bom balanceamento de carga quando utilizada. Isso ocorre devido a inexistência de um padrão de acesso aos discos, o que garante que a carga será distribuída de maneira igualitária entre os discos a longo prazo.

Apesar disso, estatisticamente é possível que em um dado momento, um número de requisições elevado seja direcionado a um único disco do sistema, o que acarretaria em uma carga desigual entre os discos durante um curto intervalo de tempo[46]. Esse fenômeno é conhecido como desbalanceamento de carga a curto prazo e pode causar um aumento na latência do sistema, o que é prejudicial ao desempenho do mesmo. A Figura 2.6 mostra uma situação onde, em um determinado instante de tempo, a

fila de requisições do quarto disco está mais cheia do que os demais.

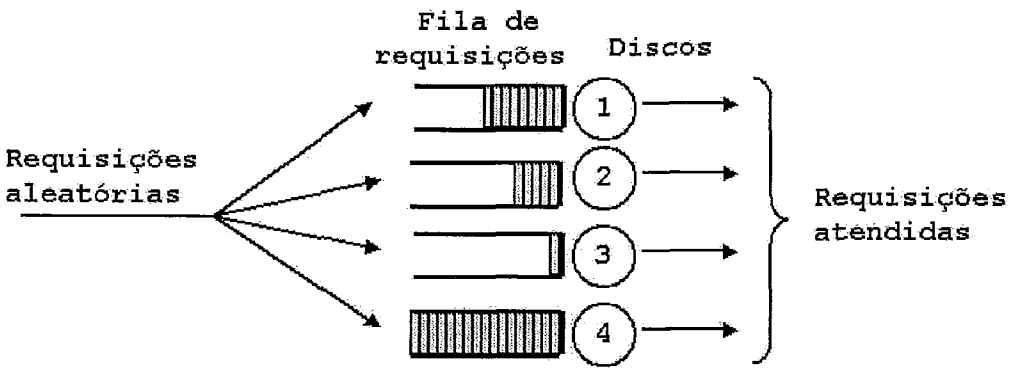

Figura 2.6: Desbalanceamento de carga a curto prazo

Visando diminuir os efeitos causados pelo desbalanceamento a curto prazo foram propostas algumas técnicas baseadas em replicação de conteúdo. Dessa maneira, é possível ao servidor escolher o disco com menor carga no momento em que um bloco replicado é pedido. Em [24], é feita um estudo dos benefícios obtidos com replicação de conteúdo. Em [28], é apresentado um estudo sobre os benefícios da replicação de uma fração dos blocos armazenados.

2.4.2 Tolerância a Falhas de Disco

Um aspecto relacionado à servidores multimídia de extrema importância diz respeito à manutenção do seu funcionamento e da acessibilidade do seu conteúdo. Duas técnicas muito utilizadas para o aumento da tolerância a falhas em disco são: checagem de paridade e replicação [20].

Na checagem de paridade, os D discos do sistema são agrupados de maneira que $d_g = D/n_g$ onde n_g representa o número de grupos e d_g o número de discos por grupo. Para cada grupo, um disco é escolhido para guardar as informações de paridade. A informação a ser armazenada neste disco é obtida através da operação de ou-exclusivo dos outros $d_g - 1$ discos do grupo. Se algum destes $d_g - 1$ discos do grupo falhar, os dados que foram perdidos podem ser recuperados através da operação ou-exclusivo entre os discos restantes e o disco de paridade. Técnicas de

verificação de paridade só conseguem recuperar dados se houver falha em um único disco. Dessa maneira, se mais de um disco falhar, a informação não poderá ser recuperada.

As técnicas de replicação utilizam espaço adicional para duplicar as informações armazenadas no servidor multimídia. Essa replicação pode ser completa ou parcial. Na replicação completa, todos os blocos são duplicados. Na replicação parcial um bloco pode ser replicado com probabilidade p, onde $0 \leq p \leq 1$. Essa técnica oferece algumas vantagens. A primeira delas é o ganho em desempenho, já que o pedido para um bloco duplicado pode ser direcionado para o disco com menor carga. Outra vantagem diz respeito aos tipos de falhas que podem ser recuperadas. Em algumas situações, as técnicas de replicação podem recuperar as informações perdidas no caso de uma falha em mais de um nó de armazenamento.

2.4.3 Múltiplos Caminhos

A transmissão de objetos multimídia via Internet também apresenta alguns desafios que não estão diretamente relacionados à limitações no servidor multimídia. O caminho seguido pelos pacotes durante a transmissão de um objeto provavelmente será compartilhado por outras aplicações. Por isso, ocorrerão influências externas que vão influenciar o *jitter* e a taxa de perda. Aplicações multimídia são extremamente sensíveis a esses dois fatores, principalmente no que diz respeito à rajada de perdas [41], que pode comprometer significativamente a QoS percebida pelo usuário.

Uma técnica utilizada para minorar esses problemas consiste em enviar os pacotes através de caminhos diferentes, visando diminuir o impacto causado por gargalos na rede. Vários estudos mostraram que essa técnica pode diminuir consideravelmente o tamanho da rajada de perda e o tempo de espera na fila [52, 37]. Um atrativo desta técnica, consiste no fato de não serem necessárias alterações nos serviços prestados pela rede nem em sua estrutura. Para isso, os mecanismos para prover a diversidade de caminhos tem que ser implementados na camada de aplicação. A solução mais simples consiste em replicar os dados em servidores com localizações físicas diferen-

tes, de maneira que sejam acessados por caminhos heterogêneos. Quando os dados não podem ser replicados, uma possível solução é enviar o fluxo por mais de um caminho.

Para que essa técnica obtenha resultados satisfatórios, é necessário que o benefício obtido compense o esforço dispensado para encontrar novos caminhos. Também é preciso que o processo de perda entre os caminhos possua baixa correlação, ou seja, que os caminhos escolhidos não tenham gargalos em comum. Foram realizados vários estudos visando desenvolver maneiras mais simples para encontrar novos caminhos e mensurar a escalabilidade da utilização de vários caminhos.

O servidor RIO, quando funcionando de maneira distribuída, pode efetivamente enviar o conteúdo por mais de uma caminho. Por isso é interessante falar um pouco mais sobre o que vem sendo produzido na literatura com relação a esse tema.

Em [32], é feito um estudo sobre a diversidade de caminhos para vídeos pré-armazenados. Naquele trabalho são utilizados apenas dois caminhos e as métricas avaliadas são: a distribuição do tamanho da rajada de perda e a correlação entre os processos de perda dos caminhos. Os autores chegaram a conclusão de que a utilização de dois caminhos apresenta melhorias na distribuição da tamanho das rajadas de perda se comparado à utilização de apenas um caminho.

Em [44], é feito um estudo da escalabilidade da técnica quando mais de dois caminhos são utilizados. Também é feito um estudo para o caso de muitas aplicações utilizarem esse recurso em uma mesma rede. Naquele trabalho, se concluiu que o aumento indiscriminado do número de caminhos pode ter efeitos adversos sobre a aplicação e prejudicar o desempenho da mesma. Também foi concluído que essa técnica é efetiva somente se poucas aplicações se utilizarem dela. Isso porque com aumento do número de aplicações que a utilizam, ocorre o aumento da correlação entre o processo de perda dos caminhos utilizados.

2.4.4 Considerações do lado Cliente: *Buffers*, Latência Inicial e *Cache*

Durante a recepção dos blocos do servidor, o cliente precisa manter um *buffer* para armazenar uma porção inicial do objeto multimídia antes de começar a exibição do mesmo. Este armazenamento é necessário para absorver o *jitter* e permitir a exibição do vídeo de maneira contínua. O preenchimento desse *buffer* causa um atraso no início da reprodução do objeto chamado de *startup delay*. O dimensionamento deste é de extrema importância. Se esse tamanho for muito grande, o cliente vai ter que esperar muito tempo para o início da reprodução. Se for muito pequeno, pode acarretar em interrupções na reprodução devido a falta de algum bloco, que não pôde ser enviado pelo servidor a tempo. O servidor, por sua vez, tem que armazenar os blocos em memória antes de enviá-los ao cliente. Isso faz necessária a utilização de *buffers* no servidor.

Ao receber um bloco de dados, o cliente começa a reproduzir a informação que recebeu. O cliente precisa então requisitar o próximo bloco antes do final da reprodução do bloco corrente. Caso contrário, podem ocorrer interrupções na reprodução do objeto. O cliente também precisa decidir o que fazer com os blocos que já foram recebidos: ele pode descartar esses blocos, armazenar todos os blocos em disco ou armazenar os blocos recebidos mais recentemente. Esse armazenamento local é útil em situações onde o usuário apresenta um comportamento muito interativo. Evitando novos pedidos ao servidor quando se deseja assistir uma trecho que já foi exibido anteriormente. Essa decisão do que fazer com um bloco já exibido requer uma troca entre espaço de armazenamento disponível e largura de banda exigida.

Capítulo 3

Servidor Multimídia RIO

Neste capítulo será apresentada a arquitetura do servidor multimídia RIO (*Randomized I/O Multimedia Storage Server*) que foi o objeto de estudo deste trabalho. A seguir serão apresentados tópicos descrevendo de maneira mais detalhada os componentes principais do servidor bem como suas funcionalidades.

3.1 Visão Geral

O RIO é um servidor multimídia universal. Por definição, um servidor universal dá suporte a mídias de vários tipos: vídeo, áudio, texto e imagem. Além disso, o servidor RIO é capaz de lidar com mídias com ou sem restrição de tempo.

O servidor RIO foi inicialmente elaborado na *University of California at Los Angeles* (UCLA)[47]. Na época da sua criação, o RIO era utilizado como um simulador de ambientes 3D. O servidor foi re-projetado e re-feito através de um projeto de cooperação UCLA/UFRJ e atualmente seu desenvolvimento continua no laboratório LAND/UFRJ.

Durante a gravação em disco, o servidor não faz distinção entre os tipos de mídia, todas são consideradas objetos e armazenadas da mesma maneira. Os objetos são divididos em blocos de dados do mesmo tamanho e armazenados de forma aleatória entre os nós do sistema. A escolha da técnica de armazenamento aleatório foi uma

alternativa às técnicas tradicionais de *striping*. O tamanho de cada bloco de dados pode ser configurado no servidor.

Estudos preliminares comprovaram a viabilidade do servidor através de simulações [28] e um protótipo foi implementado para máquinas SUN E4000 e utilizado para exibição de vídeos MPEG e simulação 3D. A seguir, foi implementado um modelo para um cluster de PCs utilizando o sistema operacional Linux. Essa versão foi implementada utilizando C++. Posteriormente, o servidor foi re-projetado e refeito sendo que agora seu desenvolvimento atual é realizado pelo grupo de pesquisas do LAND [10], onde foi desenvolvido um novo módulo cliente, o *RioMMClient*, e adicionadas novas funcionalidades [43, 25]. O *RioMMClient* possui uma interface gráfica que permite ao usuário a interação com o vídeo. Essas mudanças foram feitas visando a utilização do servidor em um ambiente de ensino a distância. Sua primeira aplicação está ligada ao ensino a distância, com o curso de Tecnologia em Sistemas de Computação oferecido pelo CEDERJ [3].

Dentre as funcionalidades adicionadas ao servidor que foram desenvolvidas pelo LAND [10] podem ser citadas:

- *RioMMClient*, um cliente gráfico para exibição dos vídeos que permite ao usuário a execução de comandos VCR, utilizando o *MPlayer* para a visualização dos vídeos [31];
- um módulo que permite o sincronismo de transparências com o vídeo que está sendo exibido, de maneira que quando uma operação de VCR é executada a transparência muda de acordo com o conteúdo solicitado;
- um módulo para *Patching* Interativo [43], que permite a utilização da técnica de compartilhamento de banda *patching* otimizada para situações onde os usuários executam operações VCR;
- o *riosh*, uma interface em modo texto e gráfico que permite a gerência dos objetos no servidor;
- um *buffer* no lado cliente totalmente novo incluído para amenizar os efeitos causados pelos atrasos na rede (*jitter*) [25];

- um mecanismo no servidor que, em caso de acesso seqüencial a um vídeo, pode antecipar os pedidos dos clientes (*prefetching*) e otimizar o desempenho do sistema [25];
- um novo algoritmo de admissão de usuários [25];
- um módulo para coletar medidas de desempenho no servidor [25];
- implementação de técnicas de *smoothing* no servidor;
- melhoria no sistema de redundância;
- um módulo para coletar dados relativos ao comportamento do usuário no *RioMMClient.*

Na Figura 3.1 é apresentado um esquema em alto nível da arquitetura do servidor. Como pode ser visto, existe um único nó servidor. Esse nó é o centro do sistema e tem a função de receber e gerenciar as requisições dos clientes. Outra entidade participante do sistema é o *storage server* (servidor de armazenamento) que mantém os blocos de cada objeto multimídia e são controlados pelo nó servidor. Podem existir um ou mais *Storage Servers* nos sistema e esses podem ser distribuídos por qualquer ponto na rede. As mensagens de controle trocadas entre todos os participantes do sistema utilizam o protocolo TCP. Algumas mensagens comuns são: pedido de uma nova conexão por parte do cliente, visualização do conteúdo do servidor, gravação de blocos nos nós de armazenamento, pedido de blocos pelo cliente, etc. A transmissão do conteúdo multimídia utiliza o protocolo UDP e ela é feita diretamente entre os clientes e os nós de armazenamento.

3.2 Visão Detalhada do servidor RIO

Nesta seção, o servidor RIO será apresentado de maneira mais detalhada. Aqui serão apresentados os componentes que fazem parte da estrutura do servidor, suas funcionalidades e relacionamentos. Na Figura 3.2 é apresentada a organização da estrutura do servidor.

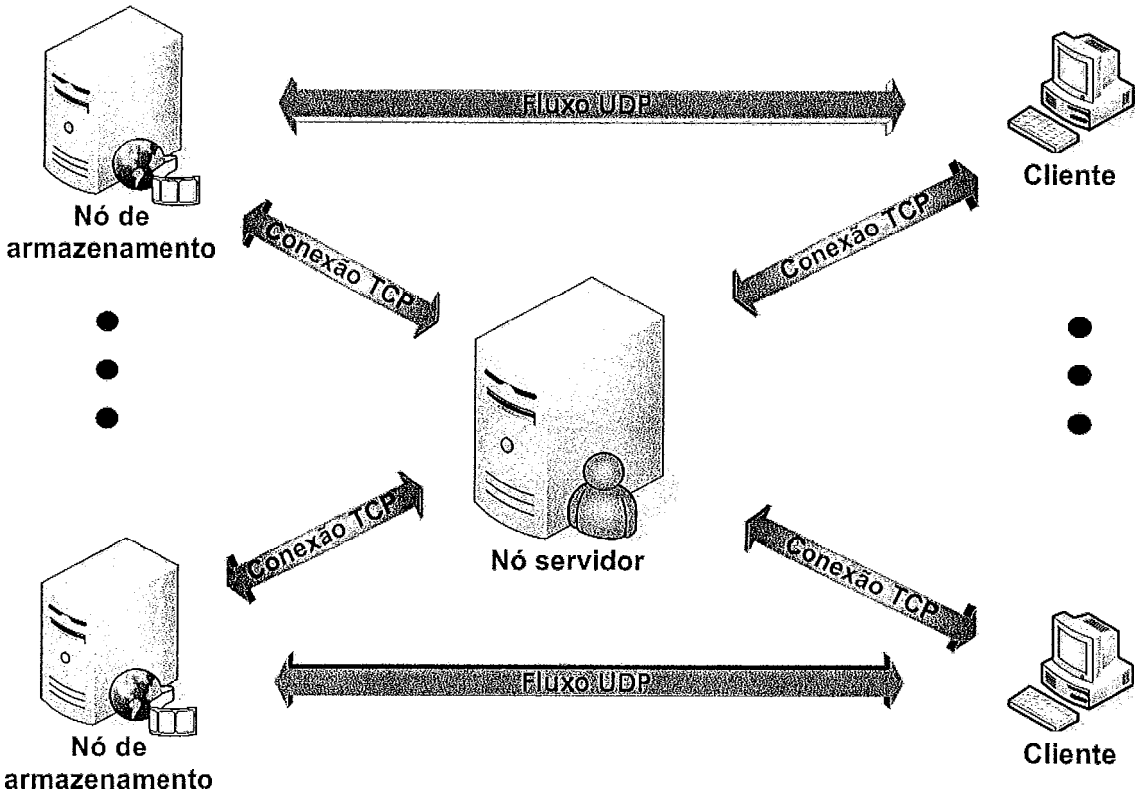

Figura 3.1: Visão simplificada do servidor RIO

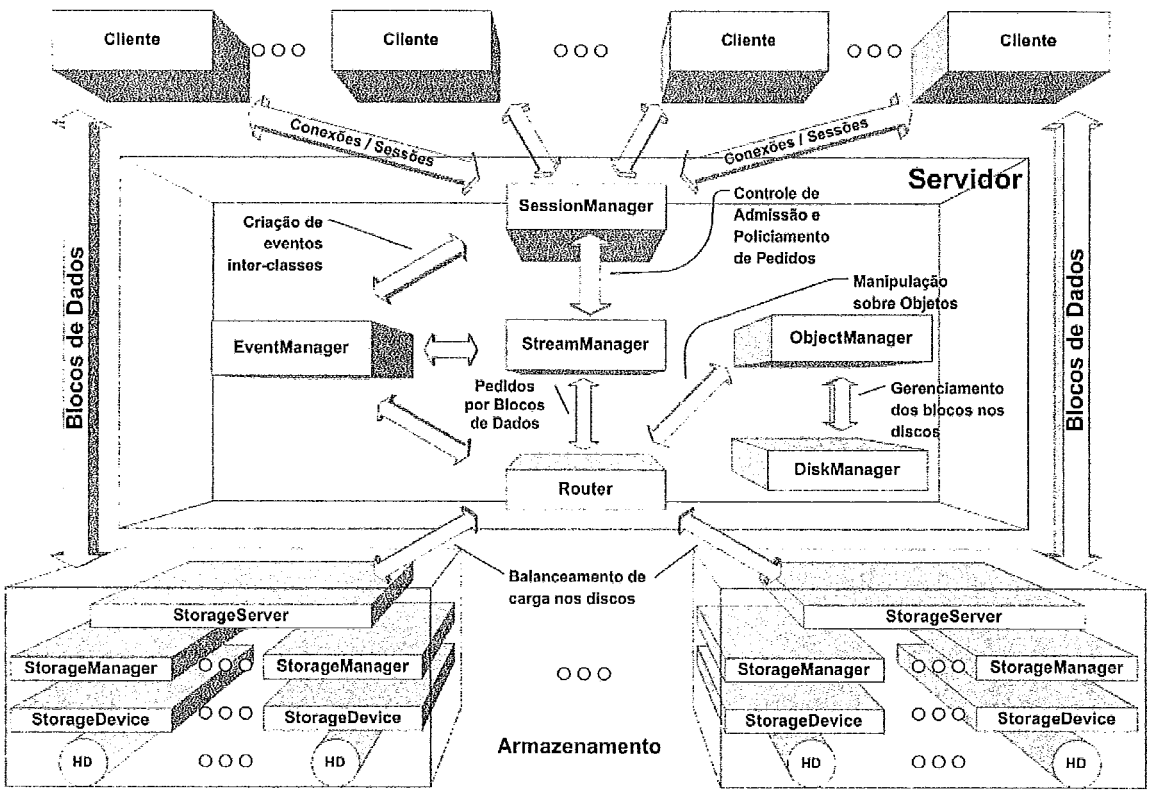

Figura 3.2: Visão detalhada do servidor RIO

3.2.1 Gerenciador

O RIO possui apenas um gerenciador. Seus principais componentes são SessionManager, ObjectManager, StreamManager, EventManager, DiskManager e Router. Estes serão descritos a seguir.

Gerenciador de Sessão (*Session Manager*)

O *Session Manager* é o ponto de conexão entre o cliente e o servidor. Todo pedido do cliente é de responsabilidade deste módulo, que cria uma nova *thread* para atender aos pedidos desse cliente. O *Session Manager* então repassa esse pedido ao *Stream Manager*. Outra função importante do *Session Manager* é a de verificar a corretude de todos os parâmetros do servidor, como, por exemplo, verificar se o número de réplicas de um bloco não ultrapassa o número de nós de armazenamento.

Gerenciador de Fluxos (*Stream Manager*)

O *Stream Manager* é responsável pelo gerenciamento e abertura dos fluxos entre o servidor e os clientes. É ele quem executa o controle de admissão descrito na seção 3.2.3.

O *Stream Manager* também é responsável pelo policiamento de pedidos descrito na seção 3.2.4. Após esses controles, o *Stream Manager* cria um evento no *Event Manager* e encaminha o pedido ao *Router*.

Gerenciador de Eventos (*Event Manager*)

Esse componente tem a função de gerar os eventos que serão utilizados para a comunicação entre os vários componentes da arquitetura do servidor. Ao ser criado, cada evento é instanciado com informações inerentes à cada tipo de evento. A seguir, ele é posto na fila de requisições não atendidas. Eventos comuns no servidor RIO são: requisição do envio de um bloco de um nó de armazenamento a um cliente,

requisição de armazenamento de um bloco em um nó de armazenamento, entre outros.

Gerenciador de Objetos (*Object Manager*)

O *Object Manager* tem a função de gerenciar os metadados dos objetos armazenados no servidor. Operações tais como: abertura, fechamento, criação e exclusão de objetos também são de responsabilidade do *Object Manager*. Quando existe uma conexão do cliente com o servidor visando a criação de um novo objeto, cabe ao *Object Manager* a alocação dos blocos de todas as suas cópias e atualização dos metadados dos objetos nos discos. O mesmo é válido durante a exclusão de um objeto.

Gerenciador de Disco (*Disk Manager*)

O *Disk Manager* é responsável pela alocação física dos blocos no disco. Sua função é retornar uma posição livre em um dos discos sempre que é solicitado ao *Object Manager* a criação de um novo bloco. A escolha do disco e da posição livre no mesmo é feita de forma aleatória. No caso de réplica, o *Disk Manager* retorna uma posição livre para cada cópia.

Roteador (*Router*)

O *Router* é responsável pela recepção dos pedidos de leitura/escrita gerados pelos fluxos, escolha dos dispositivos para atender aos pedidos, envio dos comandos para os nós de armazenamento e recepção das mensagens de controle enviadas por estes nós.

No *Router* são encontradas duas filas para cada disco do sistema: uma para pedidos com restrição de tempo e outra para pedidos sem restrição de tempo, conforme mostrado na Figura 3.3. A fila com restrição de tempo tem prioridade sobre a fila sem restrição de tempo. O atendimento de cada fila é FIFO.

Para atender aos pedidos gerados pelos fluxos, o *Router* faz um mapeamento do bloco, usando o *Object Manager*. Para um pedido de escrita, o *Router* precisa atualizar todas as réplicas do bloco. Para um pedido de leitura, o *Router* utiliza o algoritmo de balanceamento de carga para averiguar qual disco possui a menor fila (dentre os discos que possuem uma cópia do bloco). No caso de tráfego com restrição de tempo, o *Router* só considera a fila dedicada a esse tráfego, caso contrário, a soma das duas filas é considerada.

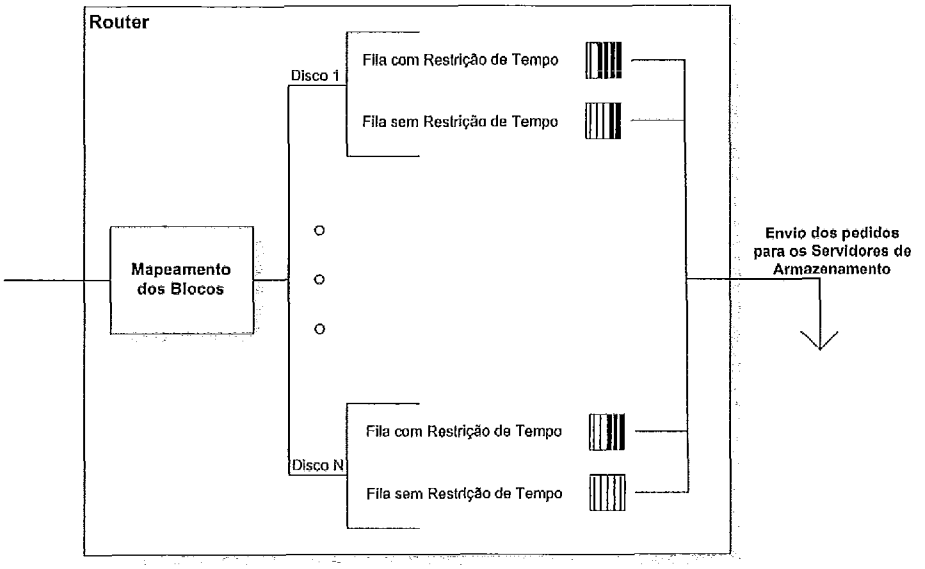

Figura 3.3: Fluxo Lógico do Router

3.2.2 Servidor de armazenamento

Cada nó de armazenamento (*Storage Server*) é responsável pelo armazenamento dos blocos (Figura 3.4). As seguintes classes são encontradas no servidor de armazenamento:

Interface com o Roteador (*RouterInterface*)

É através do *RouterInterface* que ocorre a comunicação entre o nó de armazenamento e o nó servidor. As mensagens de controle entre esses dois componentes são trocadas utilizando o protocolo TCP. As informações de controle podem ser:

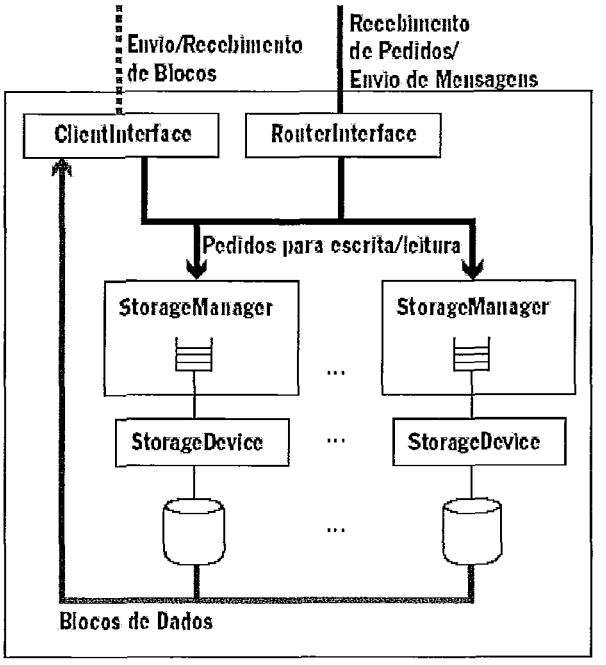

Figura 3.4: Estrutura do Storage Server

pedidos de envio de bloco de dados para um cliente, confirmação de envio de um pedido, entre outros. Se a informação de controle for um envio de pedido, ela é repassada para o *StorageManager*, caso contrário, ela é enviado ao *ClientInterface*.

Dispositivo de Armazenamento (*StorageDevice*)

Para cada disco no sistema existe uma instância do *StorageDevice*, responsável pelas operações de entrada e saída. O RIO pode manipular discos diretamente, ou então armazenar o conteúdo em arquivos. Se forem utilizados arquivos então é necessário se fazer chamadas ao sistema operacional.

Gerenciador de Armazenamento (*StorageManager*)

Para cada *StorageDevice* existe um *StorageManager* para gerenciá-lo. O *StorageManager* é responsável pelo escalonamento dos pedidos feitos ao disco que gerencia. Para cada pedido de leitura, é associado um buffer para o armazenamento do bloco solicitado e ele é enviado ao cliente pelo *ClientInterface*.

Interface com o Cliente (*ClientInterface*)

O componente *ClientInterface* é responsável pelo envio e recebimento dos blocos de dados utilizando o protocolo UDP. Durante o envio, cada bloco é fragmentado de acordo com algumas variáveis, a saber: quantidade de fragmentos que compõem um bloco, endereço do receptor, formado pelo IP, porta e identificação da requisição.

3.2.3 Controle de admissão

O *StreamManager* (gerenciador de fluxo) é o responsável pelo controle de admissão no servidor RIO. Ele é executado sempre que um cliente abre um novo fluxo de dados.

O controle de admissão original é baseado nas taxas solicitadas por cada cliente. No momento em que o servidor é inicializado, um arquivo de configurações é lido. Entre as informações contidas neste arquivo existem a **taxa total** suportada pelo servidor e a **taxa reservada** para fluxos sem restrição de tempo. A segunda informação existe para garantir que os fluxos sem restrição de tempo sejam atendidos mesmo na presença de fluxos com restrição de tempo. O servidor ainda mantém uma variável **taxa alocada**. Essa variável é incrementada de acordo com os requisitos de cada cliente. Para que o cliente seja admitido, ele envia informações sobre o tipo de tráfego (com ou sem restrição de tempo), tipo de operação (escrita ou leitura) e a taxa solicitada. O controle considera somente os fluxos com restrição de tempo e um novo usuário é aceito se a seguinte condição for verdadeira:

(taxa alocada + taxa solicitada) $\leq$ (taxa total - taxa reservada)

Se essa condição for verdadeira, a **taxa alocada** será acrescida da **taxa solicitada**, caso contrário, o novo fluxo não será aceito e o cliente não será admitido.

Em [25], foi implementado um outro controle de admissão mais complexo. Este porém tem um custo computacional elevado. Neste trabalho foi utilizado o controle de admissão original que é foi descrito nesta seção.

3.2.4 Policiamento de pedidos

Se um cliente começar a fazer muitos pedidos em um curto intervalo de tempo (rajadas de pedidos), pode haver uma influência negativa no tempo de atendimento aos pedidos de outros clientes do sistema. Para evitar que isto ocorra, o *StreamManager* policia os pedidos feitos por cada cliente.

É utilizado um *leaky bucket* [51] para fazer o policiamento. Este mecanismo possui dois parâmetros: a taxa de geração de fichas r e a capacidade F de fichas que o cliente pode acumular quando está transmitindo a uma taxa menor do que a de geração de fichas. A cada pedido enviado para o *Router*, uma ficha é consumida. Se no momento da chegada de um pedido não existir nenhuma ficha, o pedido ficará armazenado em uma fila até que uma nova ficha seja gerada. A quantidade de fichas vai sendo incrementada a uma taxa r e pode chegar até o valor F. Este valor é configurado na inicialização do servidor enquanto que a taxa de geração de fichas é configurada na **taxa solicitada** pelo cliente. Sendo assim, a quantidade máxima de pedidos que serão repassados em qualquer intervalo de tempo t é rt + F.

Esse mecanismo foi implementado em [25]. Ele permite o acúmulo de fichas por parte do cliente durante um intervalo de tempo onde não ocorrem requisições (uma pausa, por exemplo). O acúmulo de fichas permite o preenchimento mais rápido dos buffers quando o cliente executa operações VCR.

3.3 Clientes RIO

3.3.1 Cliente Riosh

Essa é a interface utilizada pelos clientes para modificarem o conteúdo do servidor, disponível em modo gráfico e modo texto. Através do riosh é possível copiar um vídeo para o servidor, criar/apagar diretórios, listar o conteúdo de um diretório, dentre outras funções. Para utilizar o riosh, o cliente precisa possuir um login e senha.

3.3.2 Cliente *RioMMClient*

O *RioMMClient* é o cliente de visualização do servidor RIO. Através dele é possível realizar operações VCR tais como: avanço, retrocesso e pausa. O *RioMMclient* também oferece o recurso de sincronização de transparências com o vídeo que está sendo exibido, recurso muito útil quando se trata de ensino a distância.

O *RioMMClient* utiliza o *MPlayer* [31] para exibir os vídeos e sua interface gráfica foi desenvolvida em C++. Na figura 3.5 é apresentada a interface do RioMMClient. Essa interface, além de apresentar o vídeo da aula, apresenta também os *slides* sincronizados bem como uma lista de tópicos para facilitar a navegação.

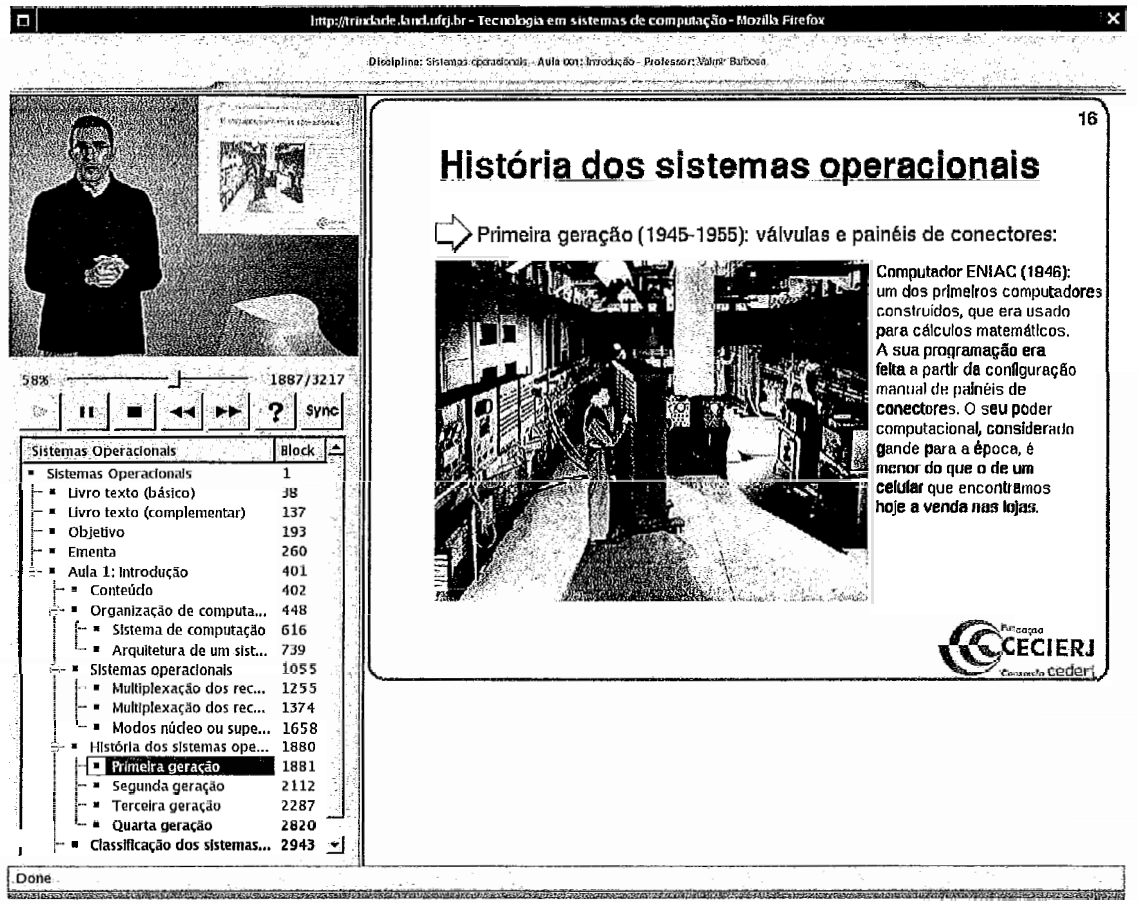

Figura 3.5: Interface do RioMMClient

É possível executar o *RioMMClient* com a utilização de *cache*. Dessa maneira, todo bloco que já foi requisitado pelo cliente fica armazenado localmente. Se, durante a exibição do vídeo, o cliente necessitar de algum bloco já acessado, este é recuperado

do disco do cliente, evitando uma nova requisição ao servidor e diminuindo o tráfego na rede. Não existe um limite definido para a quantidade de informação que o cliente pode armazenar em *cache*, sendo que esse limite é capacidade do disco.

Cada cliente pode estar em um dos três estados a seguir. É oportuna uma discussão de como esses estados influenciam o tráfego experimentado pela rede.

- Ativo
- Inativo
- Em movimentação

No estado **Ativo**, o cliente está assistindo ao vídeo normalmente. A cada bloco que é exibido, é feita a requisição de um novo bloco. Isso representa um tráfego mais suave que só fica sujeito as variações na taxa de bits do vídeo que está sendo requisitado. Se o cliente estiver utilizando cache, pode ser que, depois de uma movimentação , a exibição do vídeo chegue em um trecho que já está armazenado em *cache* e por isso, o cliente vai acessar as informações do seu disco sem a necessidade de fazer novos pedidos ao servidor e sem a geração de tráfego adicional.

No estado **Inativo**, o cliente está com o vídeo em pausa, por isso não está fazendo requisições ao servidor e conseqüentemente não gera tráfego algum.

No estado **Em movimentação**, o cliente executou um salto no vídeo, esse salto pode ser para uma posição no vídeo posterior ou anterior a atual. Mesmo tendo uma duração muito rápida, esse estado é de suma importância. No momento do salto, o cliente esvazia o seu *buffer* e requisita todos os blocos necessários para o preenchê-lo novamente, só depois de preencher o *buffer* é que o cliente re-inicia a exibição do vídeo. No momento do salto, o cliente faz uma rajada de pedidos ao servidor, o que implica em um tráfego mais elevado durante um certo intervalo de tempo. Sem a utilização de *cache*, os saltos para trás ou para frente vão causar rajadas de pedidos ao servidor. Quando se está utilizando cache, pode ser que o salto seja feito até um ponto que já está armazenado em *cache*, o que não irá causar o aumento no tráfego observado.

3.3.3 Emulador de clientes

Para permitir experimentos mais abrangentes com o servidor RIO, foi desenvolvido um emulador de clientes. O emulador possui os mesmos mecanismos de um cliente comum. A diferença está no fato de não exibir o vídeo que está sendo recebido. Dessa maneira, o gasto computacional em decodificar o vídeo é eliminado e assim, um número maior de emuladores pode ser disparado em cada máquina. O emulador lê um arquivo que informa o tempo de duração de cada bloco do vídeo. Dessa maneira, o emulador sabe qual o momento certo para fazer o pedido de um bloco.

Cada instância do emulador representa um cliente no sistema. Para simular operações de movimentação no vídeo, cada emulador recebe como parâmetro de entrada um arquivo contendo o tempo de cada operação, como mostrado na Figura 3.6. Esse arquivo será chamado daqui em diante de *log* de comportamento.

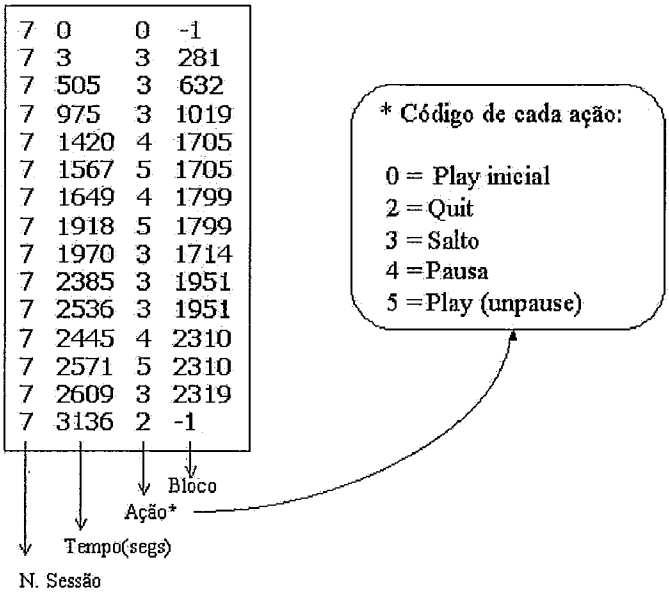

Figura 3.6: Formato típico de um log de comportamento do emulador

Para os experimentos desse trabalho, cada log de comportamento exibe as ações executadas por um aluno enquanto estava assistindo ao conteúdo do servidor. Cada linha do *log* possui 4 campos: O número de sessão do usuário, o tempo em que ocorreu uma ação (segundos), o código da ação e o número do bloco no vídeo (para

as ações **play inicial** e **quit** o número de bloco informado é sempre -1). As ações possíveis são:

- Abertura de sessão (play inicial)
- Finalização de sessão (quit)
- Pausa
- Play (unpause)
- Salto

Essas ações são as mesmas que um cliente real pode executar enquanto estiver assistindo a um vídeo. Ao ser finalizado, o emulador de clientes gera um *log* contendo as seguintes informações

- Tempo de requisição, recepção e exibição de cada bloco;
- Relação dos blocos perdidos;
- Quantidade de informação que foi requisitada do servidor, recuperada da cache, recebida e exibida;

Todos os mecanismos de coleta de dados foram implementados durante este trabalho. Um *log* típico gerado pelo emulador de clientes pode ser visto na figura 3.7. De agora em diante, devido a sua função, o emulador será chamado simplesmente de cliente dentro deste trabalho.

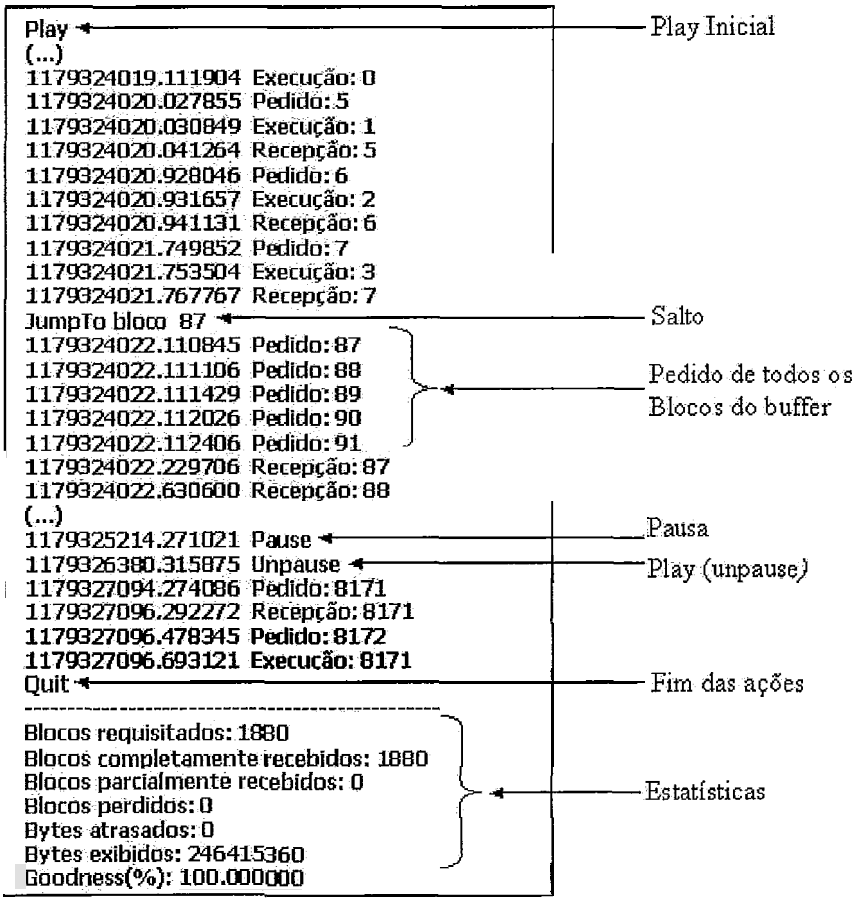

Figura 3.7: Formato típico de um log de saída do emulador

Capítulo 4

Ambiente de Testes

Neste capítulo será apresentada uma descrição do ambiente utilizado para os testes. Devido a sua relevância, será feita uma breve descrição da Rede Giga, bem como da proposta do projeto DIVERGE. Uma breve descrição do *software* utilizado para a coleta das medidas de interesse será feita e por fim, será apresentada a metodologia para a obtenção dos resultados.

4.1 Rege Giga

A Rede Giga é uma rede experimental de alta velocidade que faz parte do Projeto Giga [13] coordenado pela RNP (Rede Nacional de Ensino e Pesquisa) [15] e pelo CPqD (Centro de Pesquisa e Desenvolvimento em Telecomunicações) [4] com recursos financeiros do FUNTTEL (Fundo para o Desenvolvimento Tecnológico das Telecomunicações) e apoio da FINEP (Financiadora de Estudos e Projetos) [7].

O objetivo do Projeto GIGA é o desenvolvimento de tecnologias de Redes e de Serviços de Telecomunicações voltadas para IP/WDM (*wavelength-division multiplexing*) diretamente sobre Redes Ópticas e também serviços e aplicações de banda larga. A Rede Giga interliga entidades de pesquisa nas cidades de Campinas, São Paulo, São José dos Campos, Cachoeira Paulista, Rio de Janeiro e Petrópolis.

Este trabalho foi realizado com a participação das seguintes instituições: Uni-

versidade Federal do Rio de Janeiro (UFRJ), a Universidade Federal Fluminense (UFF), Fiocruz, onde o switch giga foi alocado dentro do Canal Saúde e UFMG.

A Rede Giga pode ser considerada como uma VPN (*Virtual Private Network* ou Rede Virtual Privada) que interconecta a UFRJ, UFF e Fiocruz. Na Figura 4.1 pode ser vista a arquitetura da rede Giga com as instituições participantes.

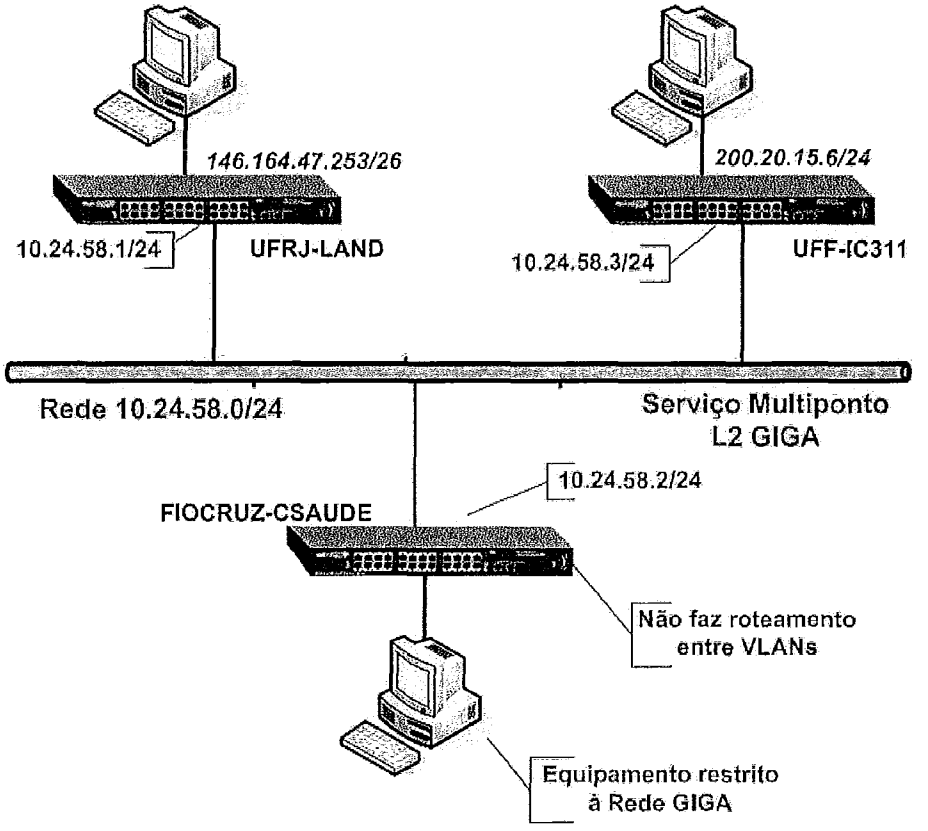

Figura 4.1: Arquitetura da rede Giga para os testes

4.2 Projeto DIVERGE

O trabalho que está sendo apresentado serviu para atingir alguns dos objetivos traçados pelo projeto DIVERGE (Distribuição de Vídeo em larga Escala sobre Redes Giga, com Aplicações a Educação).

O projeto DIVERGE é um sub-projeto integrante do Projeto Giga. Ele versa sobre a transmissão de mídia contínua em tempo real. O seu foco é a transmissão de vídeo e voz, incluindo tele-conferência e vídeo em larga escala, através de servidores

multimídia voltados para aplicações em educação. A figura 4.2 apresenta a proposta original do projeto DIVERGE para a distribuição dos componentes do servidor RIO.

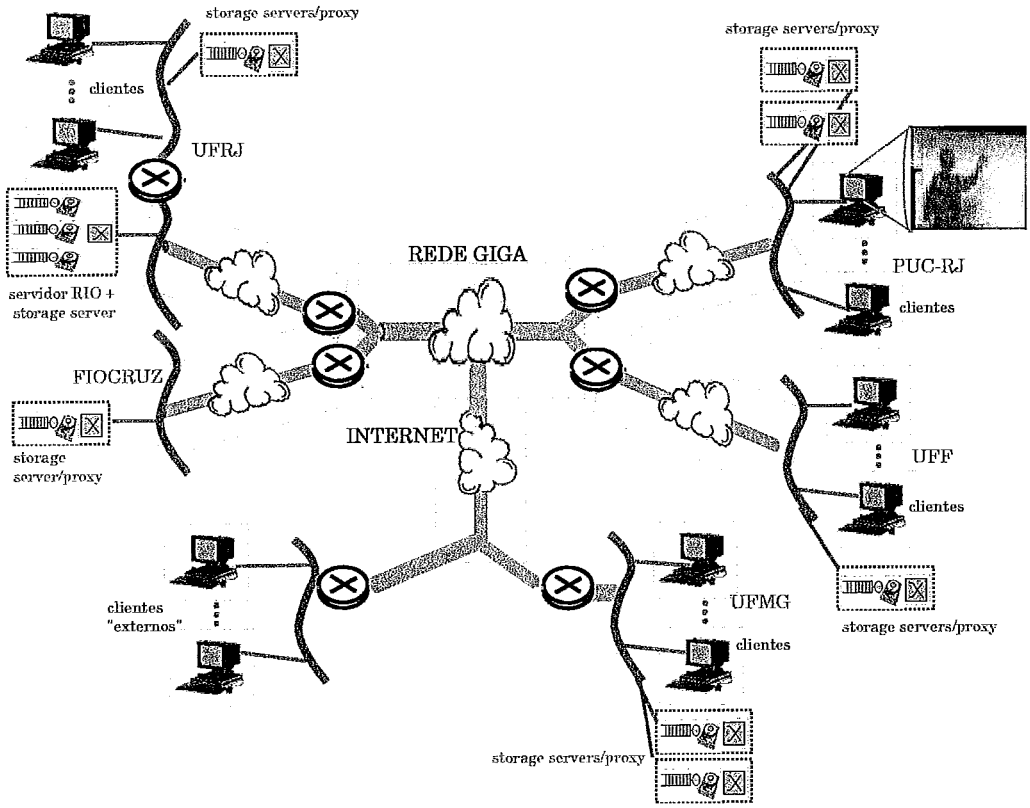

Figura 4.2: Proposta original do projeto DIVERGE

A participação da Rede Giga, de alta velocidade, é essencial por permitir a experimentação em larga escala e identificação de problemas que podem vir a ocorrer em cenários desse porte. A utilização da Rede Giga também permite arquiteturas inovadoras tais como a distribuição de *storage servers* na Fiocruz e UFF.

Dentre os objetivos do DIVERGE, podem ser citados:

- Implementação de um serviço de distribuição multimídia em larga escala.
- Utilização de protocolos de compartilhamento de banda e replicação de conteúdo.
- Caracterização do tráfego gerado pelas aplicações.
- Estudo da escalabilidade de servidores multimídia.

- Estudo de comportamento do protocolo TCP em redes giga.
- Determinar o benefício de uma rede giga para o acesso de um grande número de usuários e comparar esse resultado com outras propostas de distribuição de vídeos.
- Experimentos com carga sintética e real.

O projeto DIVERGE tem algumas metas em comum com o trabalho apresentado aqui. Outras metas do projeto vem sendo buscadas através da realização de outros trabalhos, paralelos a este.

4.2.1 Equipamentos utilizados

Neste tópico serão apresentados alguns dos equipamentos utilizados durante os testes bem como as configurações dos mesmos.

A rede Giga utiliza em suas bordas *switches* do modelo *Summit 200-24* produzido pela *Extreme Networks* [6]. Esse modelo possui 24 portas *ethernet* 10/100 MBits por segundo (chamadas portas 100) e também 2 portas giga MBits por segundo (chamadas portas giga).

Na Tabela 4.1 é possível encontrar a relação do *hardware* das máquinas utilizadas como *storage server*. A máquina urca, localizada na UFRJ foi escolhida para rodar um *storage server* e o servidor simultaneamente. Todas as máquinas utilizadas possuem o sistema operacional *Scientific Linux* 4.3 ou superior instalado.

Tabela 4.1: Relação do *hardware* das máquinas servidoras.

Máquina	Local	Processador	Memória	Disco	Placa de rede
urca	UFRJ	Intel(R) Pentium(R) 4 CPU 3.60GHz	1 GB	(SATA-40GB)51 MB/s	1 Gbit/s
pontal	UFRJ	Intel(R) Pentium(R) 4 CPU 2.40GHz	1 GB	(IDE-80GB)35 MB/s	1 Gbit/s
fiomach	FIOCRUZ	Intel(R) Pentium(R) 4 CPU 3.20GHz	512 MB	(SATA-160GB)53 MB/s	1 GBit/s
ped2	UFF	Intel(R) Pentium(R) 4 CPU 2.40GHz	512 MB	(IDE-80GB)28 MB/s	100 Mbit/s
tornado	UFMG	Intel(R) Pentium(R) 4 CPU 3.20GHz	2 GB	(SATA-80GB)55 MB/s	1GBit/s

As demais máquinas, utilizadas para disparar os clientes têm como configuração mínima: Pentium IV 2 GHz, 1 GB de RAM, HD IDE 20 MB/s.

Na UFRJ, todas as máquinas foram conectadas a uma das portas giga do *summit* através de um switch *Dell Powerconnect 2724* [5]. Na UFF, a máquina utilizada foi ligada diretamente a porta 1000 do *summit*, sendo que uma das portas 100 é utilizada para conexão ao switch do laboratório, o que permite acesso a Internet. Na Fiocruz a máquina está ligada diretamente a porta giga do *summit* e não possui acesso a Internet. A segunda porta giga disponível em todos os *summit* é utilizada para a conexão com a fibra-óptica. Na UFMG a máquina está ligada ao *switch* do laboratório e tem acesso a Internet.

Na UFF, Fiocruz e UFMG existe apenas uma máquina disponível para testes, elas serão utilizadas como *storage servers*. Na UFRJ existem 13 máquinas, sendo duas reservadas para *storage servers* e servidor, uma para o *software* de monitoramento da rede e as restantes para disparo de clientes. Na Figura 4.3 pode ser visto a arquitetura da rede GIGA encontrada na UFRJ.

Foram feitos experimentos com até cinco *storage servers*, sendo que dois desses estão na UFRJ, um na UFF e um na Fiocruz. O quinto storage se encontra na UFMG e será acessado via Internet (externamente a Rede Giga). O *storage* da UFMG foi adicionado visando obter mais informações a respeito do funcionamento do servidor RIO em um ambiente heterogêneo.

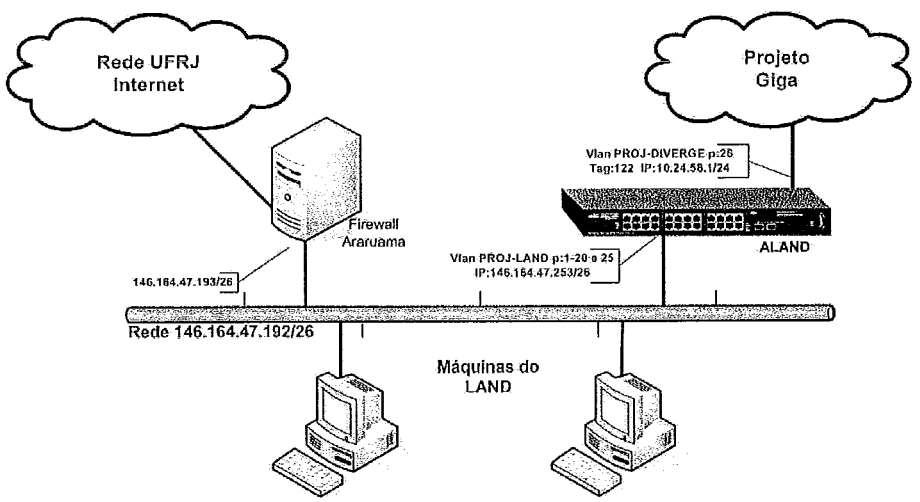

Figura 4.3: Arquitetura da rede Giga na UFRJ

Devido à restrições do equipamento disponível, foi decidido que a UFRJ deveria

possuir dois *storage servers* e as demais instituições um *storage server*. O servidor sempre ficará na UFRJ. Os clientes foram todos disparados da UFRJ.

4.3 NeTraMet

Para coleta do tráfego gerado por um dos *storage servers* foi escolhido o *software* o *NeTraMet* [12]. Esse *software* foi desenvolvido na Universidade de Auckland e é uma implementação de código-aberto da arquitetura RTFM (*Realtime Traffic Flow Measurement*) [14] para medição de tráfego em redes.

A arquitetura RTFM define uma entidade chamada *meter* ou medidor, que fica localizado em algum ponto da rede medindo fluxos de dados e armazenando a informação obtida em tabelas. No contexto da arquitetura RTMF, os fluxos podem ser identificados por mais de 40 atributos, sendo que os principais são: o protocolo, endereço ip do fonte e do destino, número da porta da fonte e do destino. Baseado nesses atributos, é possível a criação de regras que permitam aos medidores observar somente os fluxos de interesse.

A arquitetura RTMF também define as entidades *manager* (gerenciador) e *meter reader*. A primeira é responsável pela gerência dos medidores, o que abrange a passagem do conjunto de regras de fluxos a serem medidos, o tipo de amostragem, etc. A segunda entidade é responsável pela leitura dos dados gerados pelo medidor.

O NeTraMet, mais especificamente, é a implementação de um medidor, definido pela arquitetura RTMF. Neste trabalho, o NeTraMet vai estar posicionado na saída de um dos *storage servers* e, baseado nas regras que lhe serão passadas, vai medir somente os fluxos que nos interessam. Para passagem das regras ao NeTraMet e coleta dos resultados, será utilizado o *software* NeMaC que funciona como *manager* e também como *meter reader*.

4.4 Metodologia dos Experimentos

Para a realização dos experimentos deste trabalho, foi utilizada uma configuração distribuída para o servidor RIO. Para isso foram avaliados cenários com 1, 2, 3, 4 e 5 *storage servers*. Em todos os cenários, o servidor está na UFRJ. Os *storage servers* variam de acordo com o esquema da Tabela 4.2.

Tabela 4.2: Distribuição dos Storage Servers.

Cenário	Número de storages por instituição			
	UFRJ	Fiocruz	UFF	UFMG
1	1	0	0	0
2	2	0	0	0
3	2	1	0	0
4	2	1	1	0
5	2	1	1	1

Como pode ser visto, cada cenário apresenta a adição de um *storage server*, sendo que os dois primeiros estarão na UFRJ, o terceiro na Fiocruz e o quarto na UFF, utilizando todas as instituições interligadas pela Rede Giga. O quinto cenário contempla um *Storage Server* na UFMG, instituição que não participa da Rede Giga e será acessada através da Internet, essa configuração visa avaliar o desempenho do servidor quando um de seus nós está localizado em uma rede com características de banda e latência diferente das demais.

Para cada cenário, serão feitos vários experimentos com um número incremental de clientes. O número inicial de clientes é 25 e cada incremento é da ordem de 25. Ao final de cada experimento, as medidas desejadas serão coletadas.

As medidas coletadas no lado cliente foram escolhidas para permitir uma avaliação da qualidade percebida, com essa informação será possível avaliar a escalabilidade do servidor, quantificar os ganhos obtidos com diferentes escolhas de projeto (*cache*, replicação, utilização de redes com qualidade inferior) bem como avaliar a acurácia do modelo de comportamento estudado. As medidas são:

- Percentual de bytes recebidos a tempo de serem exibidos, que aqui será chamado de *goodness*. A partir dessa medida, será computada a média do *goodness*

obtido por todos os clientes, que será chamado de *goodness* médio.

- Percentual de bytes perdidos.
- Percentual de bytes atrasados.[1]

Utilizando um *software* de monitoramento de redes, será coletado todo tráfego de saída de um dos *storage servers* localizado na UFRJ. A carga em todos os *storage servers* é semelhante dado o balanceamento de carga obtido com a técnica de alocação aleatória de blocos. Dessa maneira, será possível avaliar a carga imposta a cada *storage server* individualmente. Essa informação será extremamente útil para avaliar a escalabilidade do servidor. Duas medidas serão extraídas a partida coleta desse tráfego:

- A taxa média observada.
- A variância da taxa média observada.

A taxa de saída de um *storage server* está diretamente relacionada com a taxa de requisições a qual ele está submetido (quanto maior a taxa de requisições, maior a taxa de saída e vice-versa). Com essas medidas será possível ter uma idéia da carga a qual está submetido um dos *storage servers*, assim será possível obter mais informações acerca do balanceamento de carga entre os *storage servers* bem como detectar cenários onde o *storage server* esteja mais sujeito à rajadas de pedidos.

Para geração de carga, foram utilizados *logs* de comportamento obtidos a partir das sessões dos alunos do curso de Tecnologia em Sistemas de Computação, oferecido pelo CEDERJ. Para a administração desse curso existem pólos, distribuídos pelas cidades do interior do Rio de Janeiro, onde são disponibilizados laboratórios para que os alunos possam assistir as aulas. Em cada pólo, existe um servidor RIO que fica na mesma rede dos computadores do laboratório. Dessa maneira, os alunos podem acessar o conteúdo do servidor sem os inconvenientes de uma rede lenta,

[1] Uma unidade de informação é considerada atrasada quando ela chega ao cliente depois do momento em que ela deveria ser exibida, esse atraso geralmente é causado pela latência na rede.

o que poderia influenciar no comportamento dos mesmos. Para a realização dos experimentos foram escolhidos *logs* com duração entre 30 minutos e 1 hora.

Os passos necessários para a realização dos experimentos são:

1. Configuração física das máquinas envolvidas no experimento;
2. Configuração das portas das placas de rede nas máquinas da UFF e Fiocruz;
3. Parametrização do servidor RIO de forma distribuída, utilizando N *Storage Servers*;
4. Parametrização e inicialização da ferramenta de coleta de tráfego;
5. Parametrização e execução de X clientes para um número N de *Storage Servers*;
6. Coleta e análise dos logs gerados pelos clientes e pela ferramenta de coleta de tráfego;
7. Aumentar o número de X de clientes e voltar ao passo 4;
8. Voltar ao passo 3.

O passo 1 envolve a conexão das máquinas ao *Summit*, seja diretamente, como ocorre na UFF e Fiocruz, ou através de um *switch*, como ocorre na UFRJ.

No passo 2, ocorre a configuração das máquinas para permitir o envio de dados pela Rede Giga: nas máquinas ligadas diretamente ao *switch Summit*, é necessário desligar a auto-negociação por questões de compatibilidade. Em todas as máquinas é necessária a configuração de rotas para permitir a troca de dados entre as instituições pela Rede Giga ao invés da Internet tradicional, a única instituição onde isso não precisa ser feito é a UFMG.

O passo 3 envolve a definição de algumas variáveis de ambiente relativas ao servidor RIO. São essas variáveis que definem quantos são e onde estão os *Storage Servers* e se replicação utilizada. Nessa etapa também é necessária a cópia dos vídeos que serão acessados durante o experimento. Foram selecionados os 20 vídeos

mais acessados que possuem o conteúdo das aulas disponibilizadas pelo CEDERJ. Esses vídeos possuem tempo de duração que varia de 40 minutos a 1 hora e estão codificados em MPEG-2 com uma taxa que varia de 1 a 1,25 MBits por segundo. Como foi dito anteriormente, cada vídeo é dividido em blocos de tamanho fixo. Devido a isso, o tempo de exibição de cada bloco é variável. Calculando o tempo médio de exibição dos blocos de cada vídeo, é possível dizer que existem vídeos com tempo médio variando de 0,8 a 1,1 segundos.

No passo 4, o *software* responsável pela coleta de informações relativas ao tráfego em um dos *storage servers* será parametrizado e disparado. O *NeTraMet* vai ser disparado em uma máquina específica, nessa máquina não será disparado nenhum cliente ou *storage server*, isso porque o NeTraMet consome uma quantidade razoável da capacidade da CPU. No *switch Dell* do laboratório, será feito um espelhamento da porta onde um dos *storage servers* está conectado. Dessa maneira, todo o tráfego que passar por essa porta será reproduzido em uma segunda porta pré-determinada. Nesta segunda porta estará conectada a máquina onde o *NeTraMet* estará em execução.

Assim sendo, o *NeTraMet* será configurado com regras que permitam a filtragem de qualquer tráfego que não interesse para o experimento. O único tráfego que será contabilizado será do tipo UDP que tiver como endereço de envio a máquina onde estará rodando o *Storage Server* e como endereço de destino uma das máquinas onde houverem clientes sendo disparados. O *software* vai monitorar então todo o tráfego que chegar a interface e vai retornar a taxa observada a cada segundo.

No passo 5, primeiramente é definido se os clientes irão utilizar *cache* ou não. Após essa definição, são disparados X clientes distribuídos por C máquinas dentro do LAND. Em cada máquina, os clientes são disparados com um atraso distribuído aleatoriamente entre 1 e 5 segundos. Cada cliente procura preencher seu *buffer* no início da sua execução e por isso, esse atraso tem o objetivo de evitar que todos os clientes acessem o servidor em um intervalo de tempo muito curto, o que iria gerar uma carga momentânea extremamente alta, prejudicando os resultados. Dessa maneira é possível gerar um tráfego mais suave no início do experimento.

Visando diminuir o número de perdas causadas pela latência da rede, cada cliente vai trabalhar com um *buffer* inicial do tamanho de 5 blocos. A Tabela 4.3 apresenta os resultados relativos à latência no tempo de chegada dos blocos. Foram utilizadas configurações com apenas um *storage server* na UFF e depois com apenas um na UFMG. Somente um cliente foi disparado. É possível notar a diferença de uma ordem de grandeza nos valores obtidos e o maior atraso observado na Internet foi de aproximadamente 4,3 segundos. O tempo médio de duração dos blocos dos vídeos escolhidos é aproximadamente 0,85 segundos. A escolha de um *buffer* inicial de tamanho 5 pode armazenar um trecho inicial de cada vídeo que varia entre 4 a 5 segundos, tempo suficiente para absorver a maioria dos atrasos provocados pela rede.

Tabela 4.3: Comparação dos atrasos medidos na Rede Giga e na Internet, em milissegundos.

	Média	Variância	Maior
Rede Giga	50,529	52,875	2205,85
Internet	376,395	995,441	4296,08

A partir do seu disparo, cada cliente passa a fazer requisições ao servidor de acordo com um *log* de comportamento. Foram selecionados logs de comportamento com duração de no mínimo 30 minutos e no máximo 1 hora. Esses limites foram necessários porque uma quantidade razoável de logs coletados tem duração muito curta, também foram coletados alguns logs com uma duração acima de 2 horas. A medida que executa suas ações, o cliente gera um *log* de saída conforme descrito na subseção 3.3.3.

O passo 6 ocorre após 70 minutos do disparo do primeiro cliente, tempo necessário para o fim da execução de todos os clientes. Os *logs* gerados serão coletados a analisados.

Os seguintes experimentos serão realizados: com e sem utilização de cache do lado cliente; com e sem replicação de conteúdo; com carga real e também com carga sintética. Com esses experimentos será possível avaliar os ganhos obtidos com a utilização de *cache* e replicação. Uma comparação entre os resultados obtidos com carga sintética e real também será útil para a avaliação da acurácia do modelo

proposto em [55] para a geração de carga.

No passo 7, o número X de clientes é incrementado e o experimento recomeça.

No passo 8, é feita uma nova configuração do servidor e o experimento recomeça.

A Figura 4.4 mostra um esquema mais detalhado da arquitetura dos experimentos.

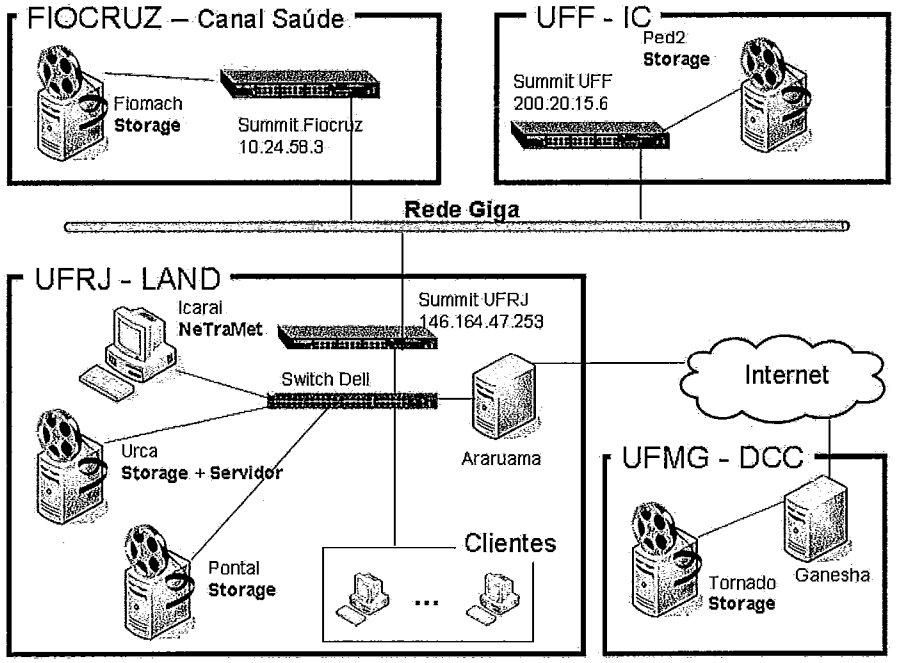

Figura 4.4: Arquitetura usada nos experimentos

Para um melhor entendimento dos resultados obtidos a figura 4.5 destaca os quatro principais pontos de perda no caminho entre o *storage server* e o cliente. Estes são:

1. A interface de rede do *storage server*;
2. Os *buffers* dos equipamentos de borda da rede Giga;
3. A rede do LAND, onde todos os clientes estão localizados;
4. A interface de rede dos clientes.

Nesse ponto, vale ressaltar um problema, relacionado ao número máximo de clientes interativos suportados por cada *storage server*, que foi encontrado durante

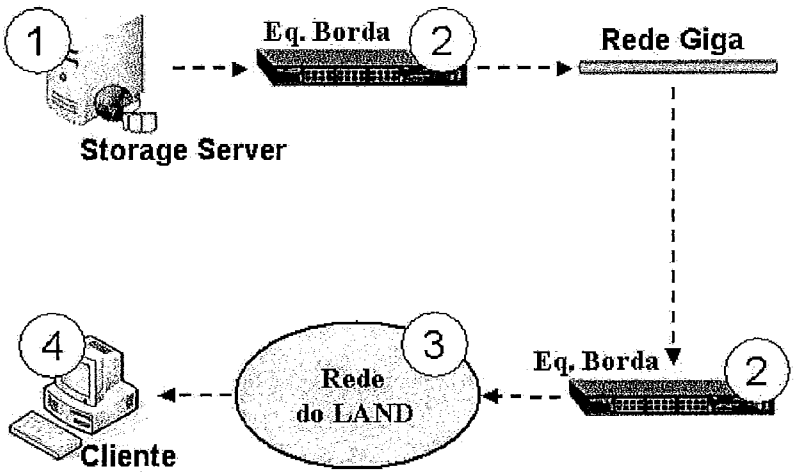

Figura 4.5: Pontos de perda

a realização dos experimentos. O servidor RIO cria uma *thread* para cada pedido de leitura em disco. Por padrão, o RIO estabelece 300 como o número máximo de *threads* para leitura. Caso esse número seja excedido, o sistema tem que esperar por uma *thread* disponível para efetuar a leitura ao disco.

Logo no início dos experimentos foi possível perceber que, quando era disparado um número superior a 50 clientes por *storage server*, o tempo de leitura dos blocos no disco começava a subir. Em casos extremos esse tempo chegava ao valor de 40 segundos para a leitura de um bloco de dados de apenas 128 KB. Esse problema costumava ocorrer entre 10 e 25 minutos após o início do experimento e, se não fosse resolvido, poderia comprometer todo o processo de experimentação.

Foi implementado um protótipo cujo o objetivo era realizar acessos ao disco de maneira semelhante ao RIO, ou seja: lendo blocos de 128 KB do disco de maneira aleatória com a abertura de uma nova *thread* para cada pedido. Durante a execução do programa, utilizando um limite máximo de 300 *threads*, foi possível constatar, de maneira semelhante ao que ocorreu com o RIO, a demora no tempo de leitura dos blocos. Esse programa foi executado em distribuições diferentes do Linux a saber: Scientific Linux 4.4, Scientific Linux 5, Ubuntu 7.04 e Mandrake 10. Em todas essas distribuições foi observado o mesmo problema. Foram realizados novos

experimentos, só que dessa vez, variando o número de *threads* disponíveis. A partir dos resultados foi constatado que quando se utilizava um número inferior a 5 *threads* o problema de lentidão na leitura dos blocos desaparecia.

Decidiu-se então limitar o número de *threads* utilizadas pelo RIO em 5. Foi possível notar uma melhoria nos resultados e se chegou ao número de até 100 clientes por *storage server*. Ao tentar disparar mais de 100 clientes por *storage*, era possível notar que agora, muitos ficavam esperando por *threads* disponíveis durante um longo tempo. Estava ocorrendo uma troca: a partir de um certo ponto, um número de *threads* menor iria fazer com que alguns pedidos de leitura ao disco tivessem uma latência maior para serem atendidos mas, em contrapartida, evitaria uma sobrecarga no disco. Um número maior de *threads* iria permitir que uma quantidade maior de pedidos fosse atendida, mas acarretaria na demora no tempo de leitura dos discos e o que causaria prejuízos ao sistema. Por isso optou-se pela utilização de apenas 5 *threads* para o atendimento de pedidos ao disco. Vale ressaltar que, devido a restrições de tempo, a escolha desse valor se baseou nos resultados empíricos obtidos com o protótipo desenvolvido. É extremamente necessário um estudo mais aprofundado sobre o número ideal de *threads* utilizadas pelo servidor RIO durante o seu funcionamento.

Observados esses detalhes relativos a execução dos experimentos, agora já é possível a apresentação dos resultados obtidos.

Capítulo 5

Resultados

A apresentação dos resultados obtidos com cada experimento será o tema deste capítulo. Para cada experimento serão apresentados os seus objetivos bem como os resultados esperados. Detalhes específicos de cada um também serão discutidos.

5.1 Objetivo Geral dos Experimentos

Os experimentos realizados, apesar de possuírem metas específicas, em conjunto compartilham o mesmo objetivo geral: avaliar o desempenho do servidor RIO em uma rede de alta velocidade com uma carga real de clientes interativos. Serão analisados aspectos tais como a escalabilidade do servidor, representada pelo ganho obtido com a adição de mais *storage servers*, ganhos relativos a utilização de *cache* e replicação de conteúdo. A coleta de estatísticas do lado cliente e servidor é pertinente porque permite mensurar a qualidade do serviço percebida pelos clientes bem como a quantidade de informação gerada pelos *storage servers*. Essas duas informações estão intimamente ligadas e juntas darão um panorama bem mais completo do que está ocorrendo no ambiente de experimentação.

5.2 Primeiro Experimento: RIO submetido a uma carga real de usuários interativos sem a utilização de cache

5.2.1 Descrição e Objetivos

O primeiro experimento é caracterizado pela utilização de uma carga real de usuários interativos para estressar o servidor. Foi feita uma coleta dos *logs* gerados durante as sessões dos alunos do CEDERJ e estes foram utilizados para alimentar os emuladores de clientes. Outra característica desse experimento é a não utilização de *cache* por parte dos clientes. Dessa maneira, será possível avaliar nos próximos experimentos os ganhos obtidos com o uso de *cache* para uma carga de trabalho semelhante a gerada pelos alunos do CEDERJ.

Para esse experimento não foi utilizado nenhum tipo de replicação do conteúdo armazenado pelo servidor.

A carga do *storage server* será avaliada através de duas métricas importantes: a taxa média e o desvio padrão do tráfego gerado para os clientes. Pretendemos avaliar o comportamento do desvio padrão com o aumento no número de usuários interativos sendo atendidos pelo sistema. Esta avaliação é importante pois um desvio padrão alto indica grandes variações na taxa média. Esta variação é sinal de que o *Storage Server* está sujeito a rajadas de requisições, o que causará um atraso maior no atendimento das requisições do cliente.

5.2.2 Descrição dos Resultados Esperados

No primeiro experimento espera-se obter ganhos de desempenho com a adição de novos *storage servers*, mesmo que este não esteja na Rede Giga (como é o caso da UFMG). Esse ganho não será linear, o motivo disso será explicado mais adiante.

Também é esperado que a distribuição da carga entre os *storage servers* seja

homogênea, dada a utilização da técnica de alocação aleatória de blocos. Como só pudemos medir o tráfego enviado por um dos *storage servers*, será possível confirmar o balanceamento de carga em cenários com 2 *storage servers* baseado no valor da taxa média observada. Em cenários com mais de 2 *storage servers* não é possível garantir que a carga esteja dividida de maneira homogênea entre os *storage servers* não observados.Porém, nestes cenários esperamos que a taxa do *storage server* observado não seja muito diferente do proporção entre a carga total gerada e o número de *storage servers*.

5.2.3 Resultados e Análise

Na Figura 5.1 é apresentado o gráfico com os resultados obtidos, o gráfico é composto no eixo Y pela porcentagem de *bytes* recebidos em tempo hábil pelos clientes, aqui chamado de *goodness*. O eixo X representa o número de clientes participantes no experimento.

Cada curva representa uma configuração do servidor, variando de 1 a 5 *storage servers*, localizados conforme descrito na seção 4.4. O intervalo de confiança é de 95%, esse intervalo foi calculado considerando o resultado de cada cliente como uma amostra. Apesar do intervalo de confiança ser muito grande, na média a adição de novos *storage servers* possibilitou o atendimento de um número maior de clientes. Até mesmo a adição de um nó na UFMG enviando dados através da Internet contribuiu para o aumento do *goodness*. O ganho obtido com a adição de novos *storage servers* fica mais evidente a medida que a população de clientes aumenta. É possível observar uma queda do *goodness*com o aumento do número de clientes, essa é causada principalmente pelo aumento no número de clientes disparados em cada máquina, o que começou a onerar a interface de rede das máquinas.

Outro fator que contribuiu para a queda do *goodness* observado é o aumento da variabilidade na taxa de requisições ao *storage server*, esse fenômeno será explicado mais a frente.

O gráfico da figura 5.2 apresenta a porcentagem de *bytes* perdidos e atrasados

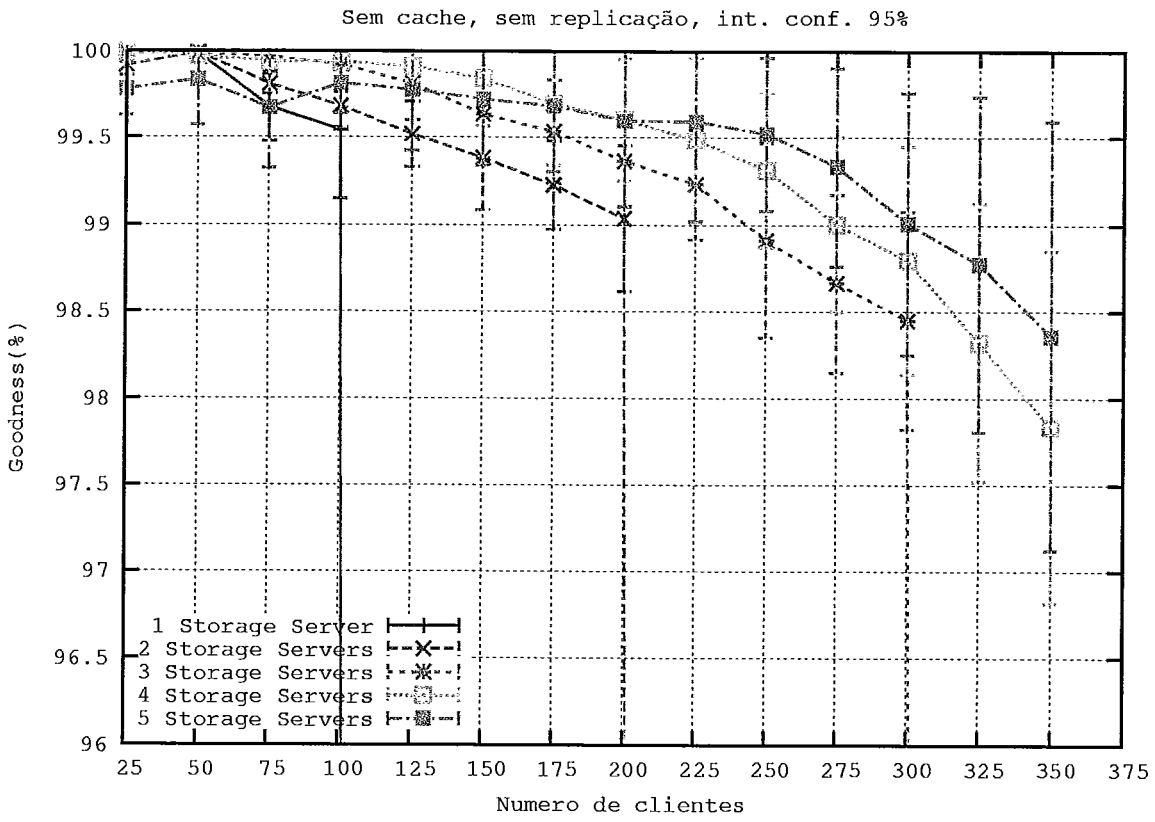

Figura 5.1: *Goodness* em função do número de clientes. 1 a 5 *storage servers*, sem utilização de *cache* e sem replicação.

para os experimentos com 3 e 5 *storage servers.*

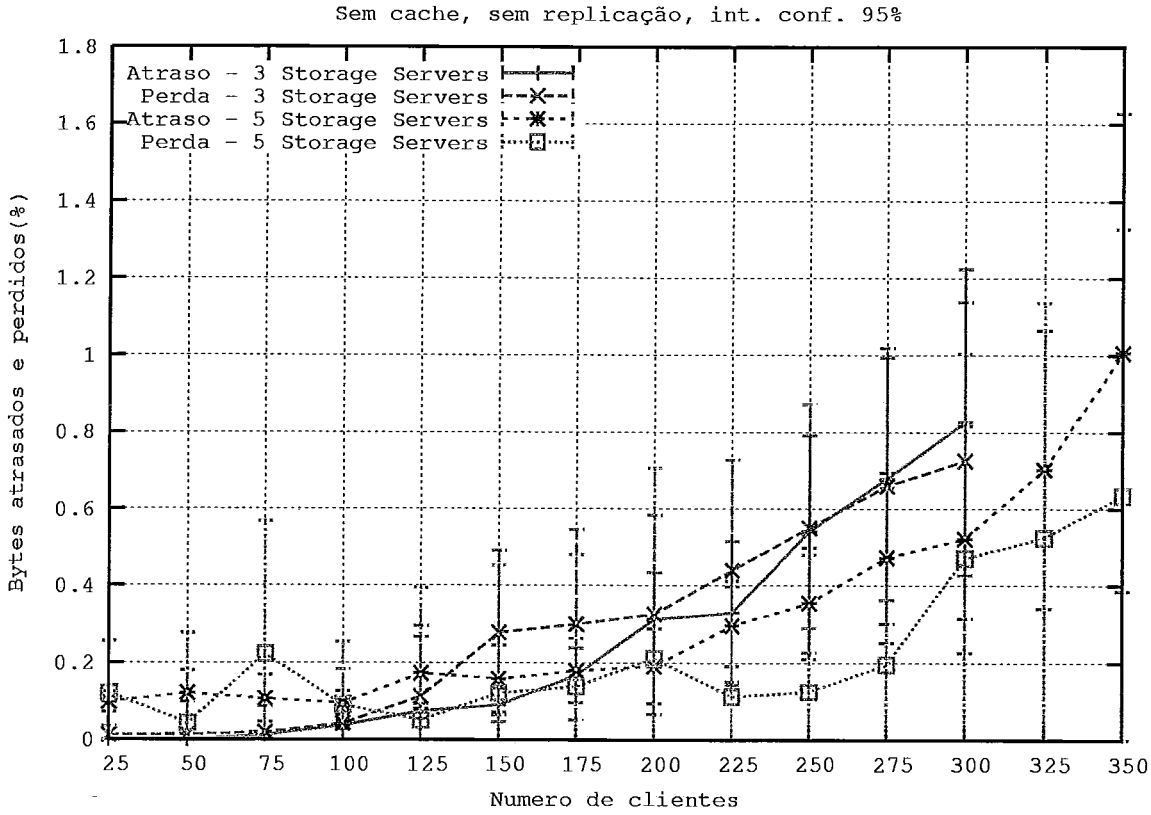

Figura 5.2: Porcentagem de *bytes* perdidos e atrasados em função do número de clientes X. 3 e 5 storage servers, sem utilização de *cache* e sem replicação.

Com relação ao experimento com 5 *Storage Servers*, vale a pena chamar a atenção para os resultados obtidos nos experimentos com menos de 200 clientes. Para esta população, foi obtido um *goodness* inferior com 5 *storage servers* em relação ao obtido para a configuração com 4 *storage servers*, indicando que o uso de um quinto *storage server* na Internet (fora da rede Giga) só apresenta vantagem a partir de uma certa população de usuários. O que ocorre é que parte das perdas é inerente a natureza da rede pelo qual o tráfego originado na UFMG teve que percorrer. Ou seja, independente do número de clientes, sempre vai haver uma perda significativa no caminho entre UFMG e os clientes. Medições feitas durante o experimento chegaram a uma perda entre UFMG e UFRJ de aproximadamente 3%. Mesmo com esse índice de perdas no canal, a partir de 200 clientes se torna vantajosa a utilização do quinto *Storage Server* na UFMG. Isso ocorreu porque, a partir de 200 clientes, a carga poupada com a adição de um quinto *storage server* foi suficiente para compensar as

perdas na rede entre a UFMG e UFRJ.

Foi possível simular até 350 clientes no LAND, um número superior a esse poderia prejudicar os resultados devido a exigências na capacidade de processamento nas máquinas clientes. No gráfico da Figura 5.3 é possível observar o número de clientes atendidos com *goodness* médio acima de 98% e 99% respectivamente. No eixo X é apresentado o número de clientes e no eixo Y o número de *Storage Servers.*

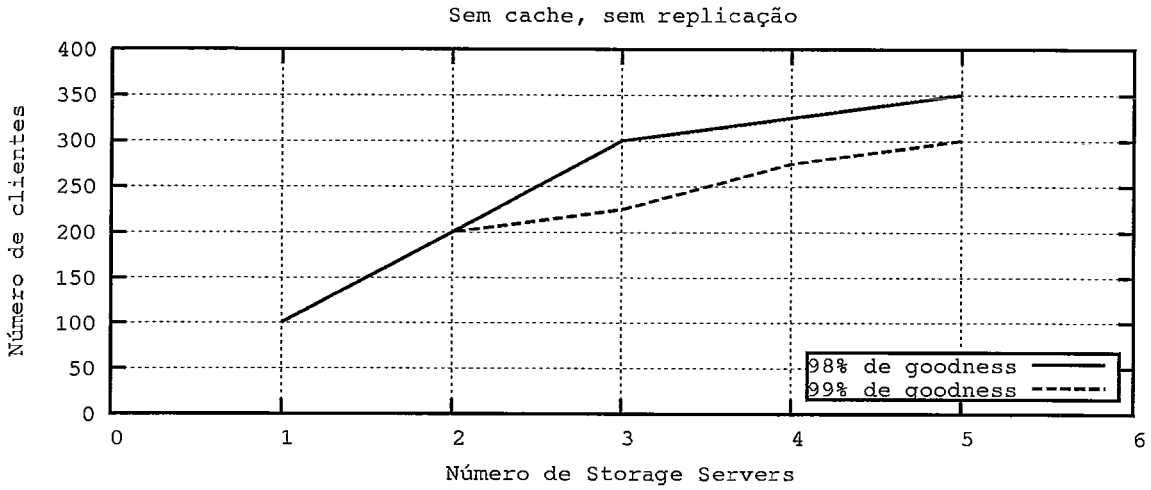

Figura 5.3: Gráfico que exibe, para cada configuração do servidor, o número de clientes atendidos com *goodness* médio de até 98% e 99% respectivamente.

Os gráficos das Figuras 5.4, 5.5 e 5.6 representam a taxa de saída de um dos *storage servers* medida durante três experimentos. Esses experimentos não têm nenhuma relação direta entre si, o objetivo da apresentação desses gráficos é mostrar que apesar serem experimentos distintos, todos os três gráficos possuem três intervalos com o comportamento em comum:

- Uma subida brusca no início: devido a entrada gradual e contínua de novos clientes no sistema;
- Um trecho relativamente estável com oscilações: localizado normalmente entre 300 e 1800 segundos, onde todos os clientes estão participando do sistema mas não necessariamente ativos;
- Um trecho com uma queda contínua e suave: logo após 1800 segundos até aproximadamente 3600 segundos, causada pela saída gradual dos clientes, dado que

foram selecionados *logs* de comportamento com duração de 30 a 60 minutos.

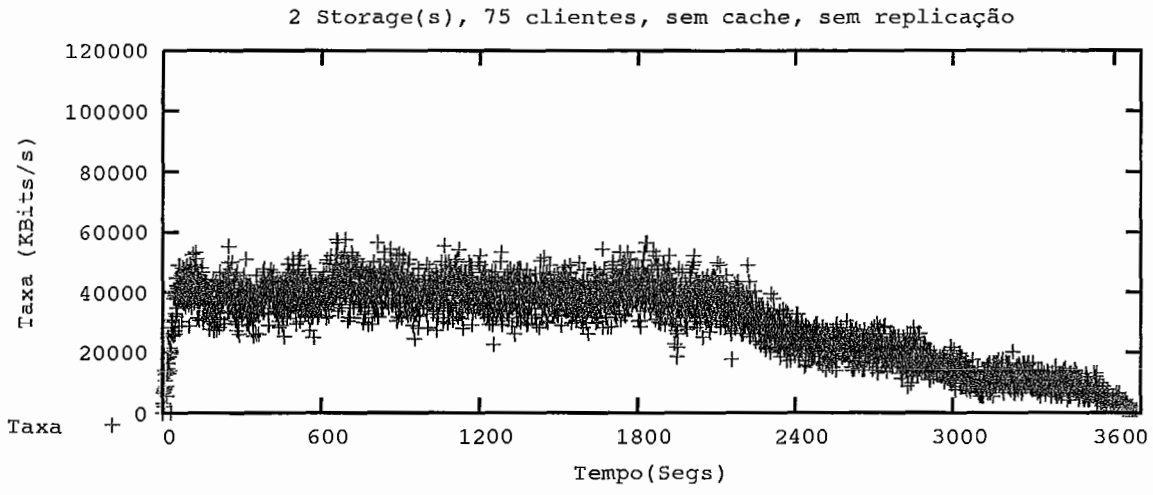

Figura 5.4: Gráfico do tráfego gerado no experimento com 2 *Storages* e 75 clientes. Sem replicação e sem uso de *cache*.

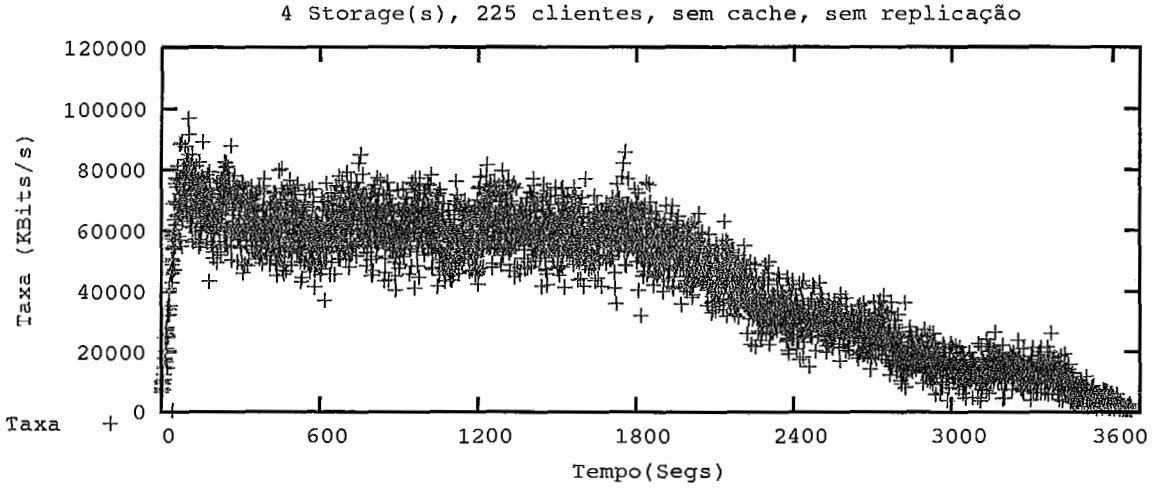

Figura 5.5: Gráfico do tráfego gerado no experimento com 4 *Storages* e 225 clientes. Sem replicação e sem uso de *cache*.

Tendo esses resultados em mãos e visando promover um entendimento mais amplo sobre o que está ocorrendo nos *Storage Servers* o intervalo que vai de 300 até 1800 segundos será estudado de maneira mais detalhada. A justificativa para o estudo desse trecho é a certeza de que todos os clientes já foram iniciados e estão conectados ao servidor nesse intervalo de tempo. Vale notar que isso não significa que todos os clientes estejam no estado ativo, eles podem estar em estado de pausa

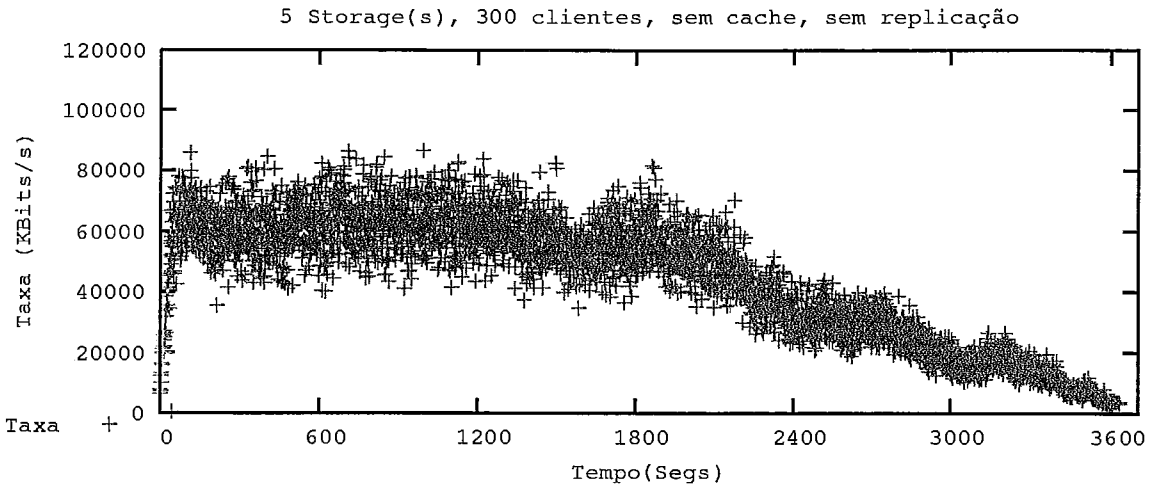

Figura 5.6: Gráfico do tráfego gerado no experimento com 5 *Storages* e 300 clientes. Sem replicação e sem uso de *cache*.

ou realizando saltos. A conseqüência dessa interatividade pode ser observada nas oscilações nas taxas dentro do intervalo especificado.

O primeiro estudo diz respeito as médias das taxas e seus respectivos desvios-padrão. Na Tabela 5.1 estes valores são apresentados. É possível tirar algumas conclusões após a apresentação desses resultados. A primeira delas diz respeito ao balanceamento de carga: nota-se que para qualquer número de *storage servers*, a taxa média é representada aproximadamente pela razão entre a carga total e o número de *storage servers*. Levando em consideração que os vídeos são codificados a uma taxa próxima de 1 Mbit/s é razoável supor que a carga total de cada experimento é um valor próximo ao produto entre o número de clientes e a taxa do vídeo.

Com esses resultados é possível garantir que em cenários com 2 *storage servers* está ocorrendo um balanceamento de carga efetivo. Em cenários com um número maior de *storage servers* isso não pode ser confirmado, dado que não temos informações da situação nos outros *storage servers*. Porém, como em todos os cenários o *storage server* observado apresentou uma taxa média próxima ao esperado é de se esperar que o mesmo esteja acontecendo com os demais *storage servers*.

Uma segunda observação diz respeito aos desvios-padrão encontrados. É fácil

notar que dentro de uma mesma configuração do servidor (valores representados pelas colunas da Tabela 5.1) , a medida que aumentamos o número de clientes temos obviamente um aumento na taxa média bem como no desvio padrão. Comparando os experimentos onde a quantidade de clientes atendidos é igual mas o número de *storage servers* varia (valores representados pelas linhas da Tabela 5.1), nota-se que na maioria das vezes se observa uma queda do desvio padrão.

A última observação é mais sutil e diz respeito a experimentos com configurações diferentes mas com taxas médias semelhantes. Esses experimentos são aqueles no qual os *storage serves* geram uma taxa média muito parecida (independente da configuração utilizada). Por exemplo, os experimentos com 1 *storage* e 25 clientes e 2 *storages* e 50 clientes são semelhantes porque possuem uma média de 25 clientes por *storage*. É possível observar na Tabela 5.1 que as taxas médias desses tipos experimentos são bem próximas mas o desvio padrão não é.

Tabela 5.1: Média e Desvio Padrão nas taxas geradas pelo Storage Server sem a utilização de *cache no cliente* e sem replicação, MBits/Seg. Amostras coletadas entre 300 e 1800 segundos de duração do experimento.

	1 Storage		2 Storages		3 Storages		4 Storages		5 Storages	
Clientes	Média	D.Pad.	Média	D.Pad.	Média	D.Pad.	Média	D.Pad.	Média	D.Pad.
25	26,116	2,329	13,666	2,536	8,853	3,136	6,875	2,315	5,585	2,068
50	52,506	3,102	26,537	3,952	17,809	3,980	13,443	3,352	10,549	3,261
75	79,896	4,981	40,453	4,897	25,937	4,886	20,549	4,212	15,762	4,045
100	102,830	6,955	53,657	5,953	37,292	5,665	27,540	5,571	22,111	4,222
125	X	X	63,738	7,720	43,150	6,217	33,335	5,776	26,852	5,908
150	X	X	79,932	8,155	54,332	6,994	38,381	6,049	31,524	5,916
175	X	X	96,166	8,747	61,943	7,195	44,835	6,520	35,741	6,021
200	X	X	104,504	10,160	69,894	7,596	53,598	7,358	41,020	6,764
225	X	X	X	X	78,661	8,551	60,787	7,762	46,663	6,848
250	X	X	X	X	88,160	9,111	68,603	8,506	52,414	7,530
275	X	X	X	X	94,533	10,714	73,609	9,813	57,532	8,192
300	X	X	X	X	108,156	11,753	76,396	10,095	61,618	8,622
325	X	X	X	X	X	X	86,472	10,559	65,829	9,625
350	X	X	X	X	X	X	91,841	11,663	71,041	10,848

Os gráficos das Figuras 5.7, 5.8, 5.9 e 5.10 exibem a taxa gerada por um *storage server* no intervalo de 300 a 1800 segundos. Os gráficos correspondem aos experimentos com média de 25, 50, 75 e 100 clientes atendidos por *storage server*

respectivamente.

É possível observar que com o aumento do número de *storage servers*, mantendo-se o mesmo número médio de clientes atendidos por um *storage server*, ocorre um aumento na variabilidade da taxa gerada pelo *storage server* sendo que a taxa média sofre poucas alterações. A Tabela 5.2 apresenta os coeficientes de variação para os experimentos com 25, 50, 75 e 100 clientes atendidos em média por *storage server*. Aqui vale a pena reforçar o fato de que, para manter a mesma média de clientes atendidos por *storage server* é necessário aumentar a população de clientes atendidos a medida que se adicionam novos *storage servers*.

Tabela 5.2: Coeficiente de variação observados em experimentos com a mesma média de clientes por *Storage Server*

Clientes/Storage	1 Storage	2 Storages	3 Storages	4 Storages	5 Storages
25	0,089	0,149	0,178	0,202	0,220
50	0,059	0,111	0,129	0,137	0,144
75	0,062	0,102	0,109	0,132	X
100	0,068	0,097	0,109	X	X

A Tabela 5.3 agrupa os experimentos com o mesmo número de clientes por *Storage Server* e apresenta a taxa média observada e o limite sob o qual se encontram 95% das amostras[1].

Os gráficos das figuras 5.11, 5.12, 5.13 e 5.14 apresentam os histogramas relativos as distribuições das taxas. Para a confecção dos histogramas, o intervalo entre a maior e a menor taxa foi dividido em 50 sub-intervalos. A partir de então, foi feita a contagem do número de amostras contidas em cada sub-intervalo. Nota-se que, a medida que o número de *storage servers* aumenta, mantendo a proporção de clientes por *storage server*, uma quantidade maior de amostras pode ser encontrada afastada da média.

[1] Ou seja: o 95o percentil

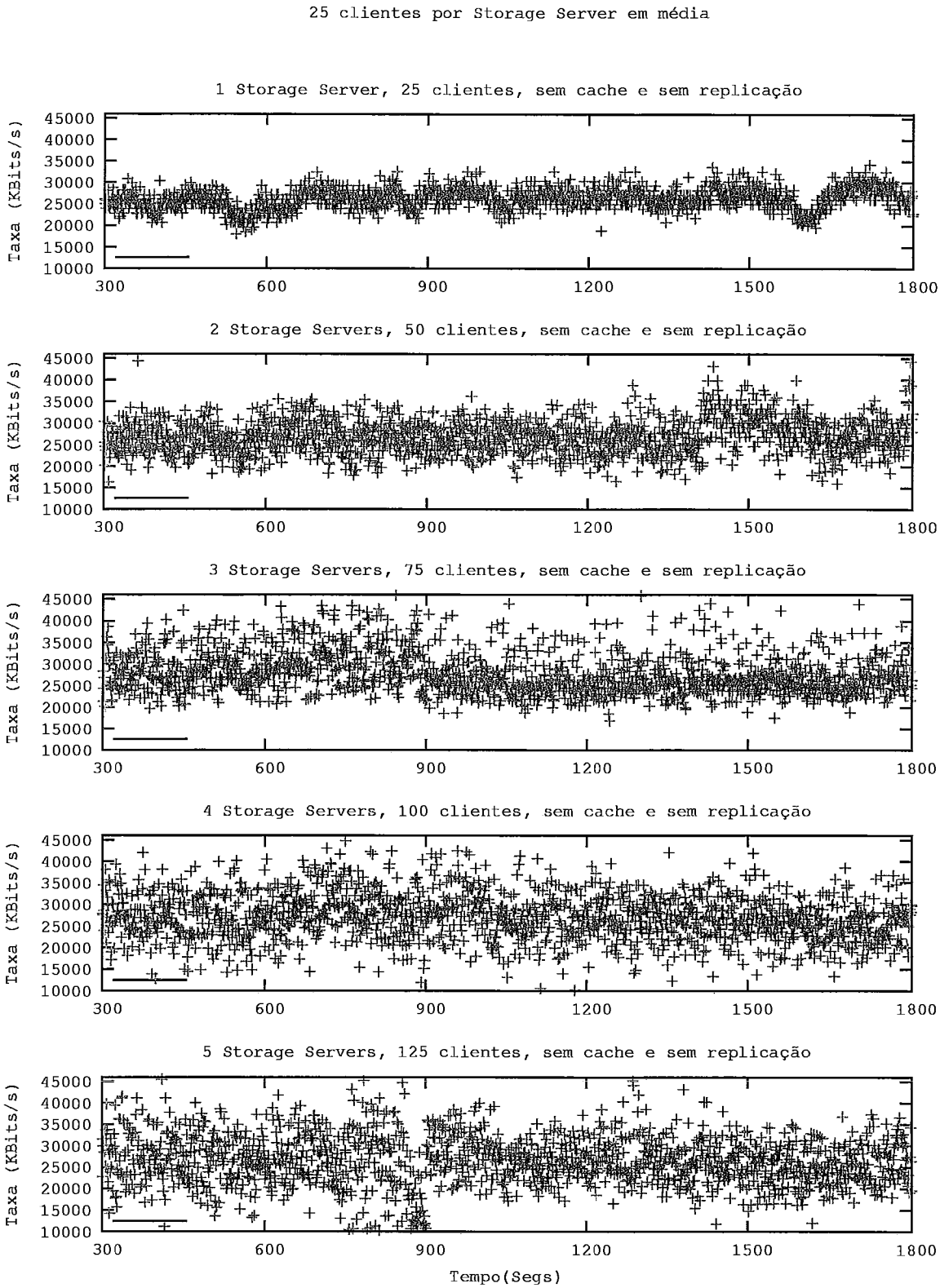

Figura 5.7: Gráficos dos experimentos com as configurações onde existem 25 clientes em média por *Storage Server*.

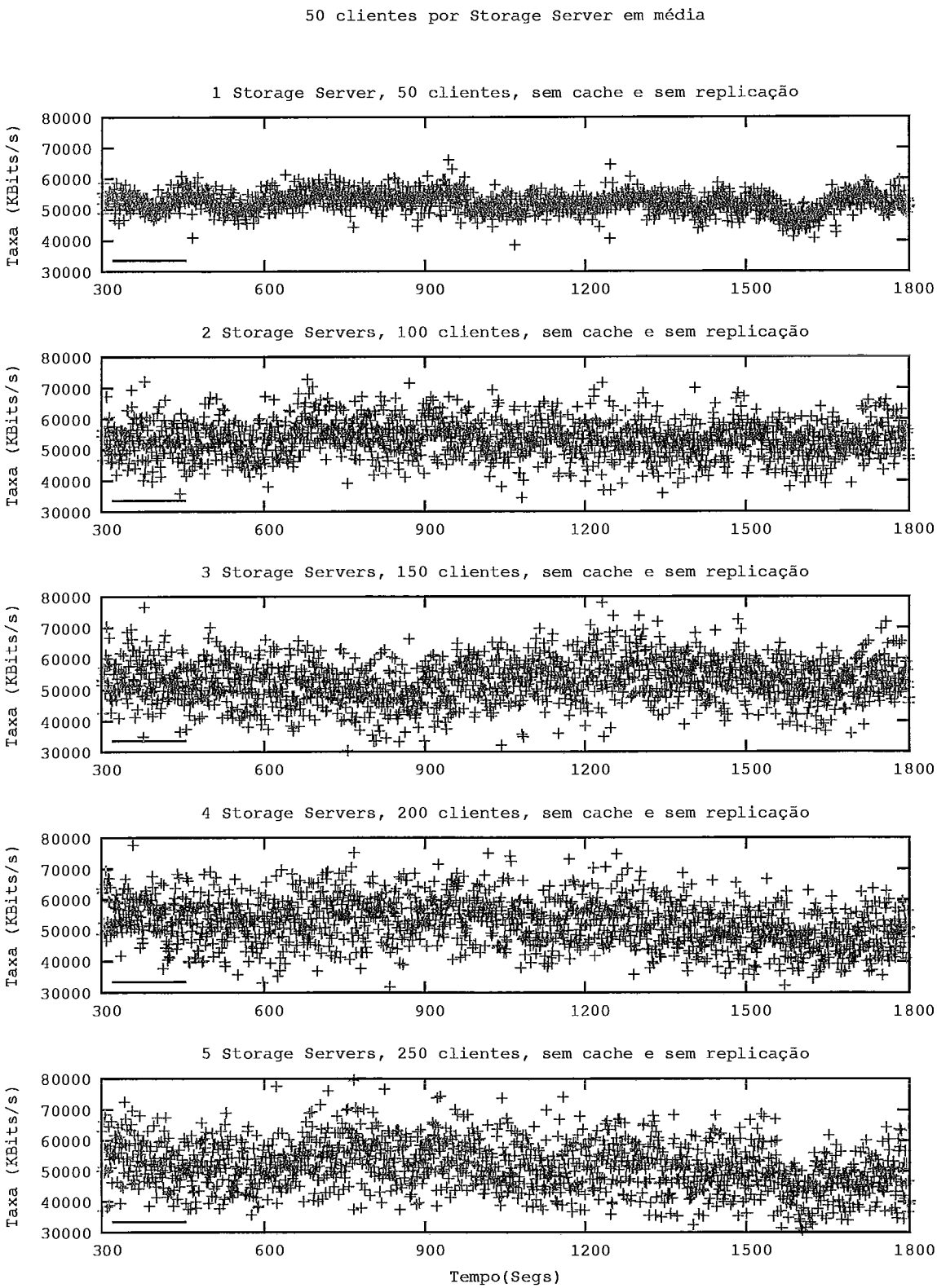

Figura 5.8: Gráficos dos experimentos com as configurações onde existem 50 clientes em média por *Storage Server*.

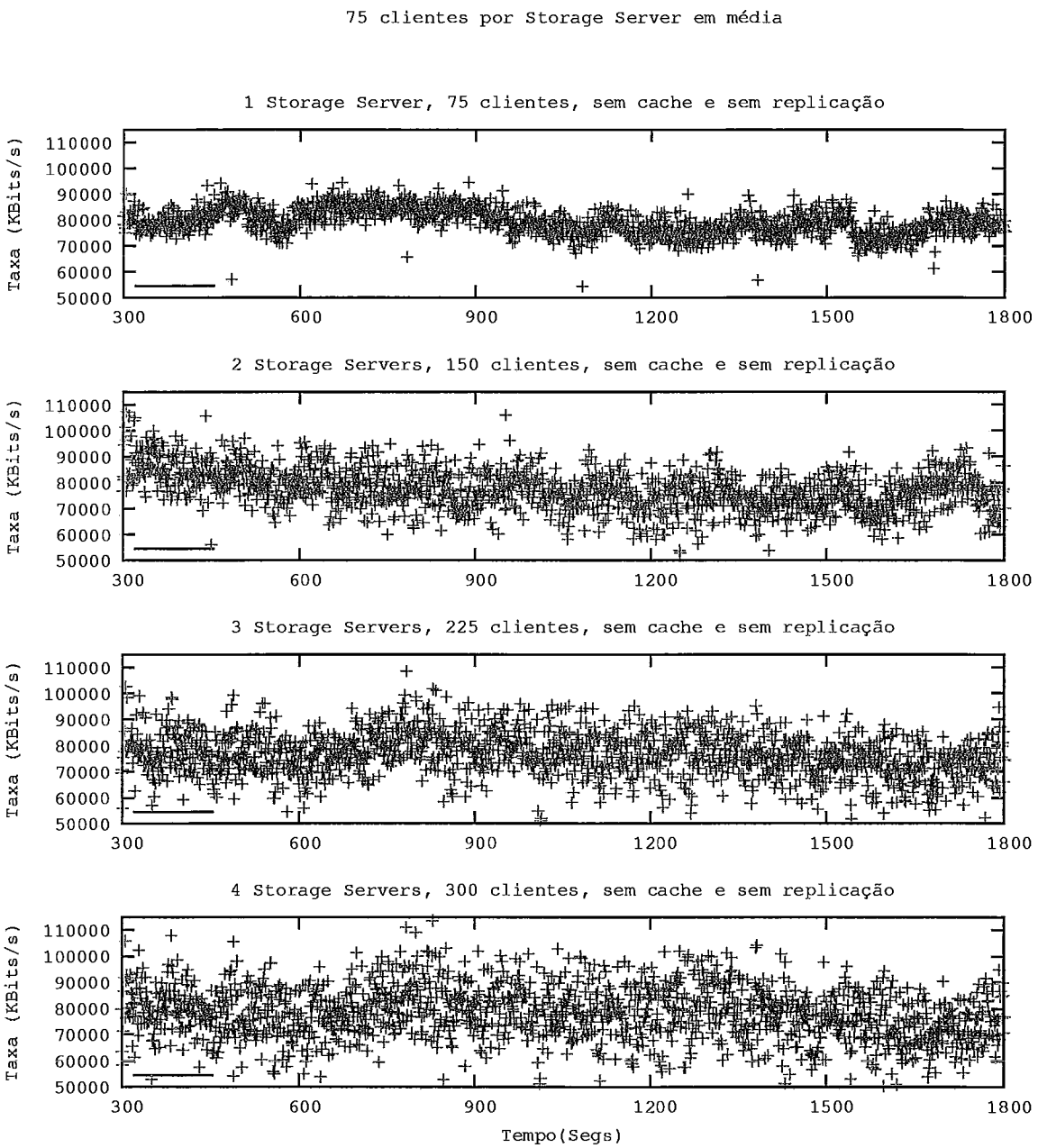

Figura 5.9: Gráficos dos experimentos com as configurações onde existem 75 clientes em média por *Storage Server*.

Tabela 5.3: Taxa média observada e limites sob qual se localizam 95% das amostras coletadas (KBits/seg). Experimentos com 25, 50, 75 e 100 clientes por *Storage Server*

	25 clientes por *Storage Server*		50 clientes por *Storage Server*	
Num. Storages	Média	Limite superior	Média	Limite superior
1	26116	30371	52506	57488
2	26537	33461	53657	63892
3	25937	35440	54332	64614
4	27540	36371	53598	65084
5	26852	36966	52414	65234
	75 clientes por *Storage Server*		100 clientes por *Storage Server*	
Num. Storages	Média	Limite superior	Média	Limite superior
1	79896	87423	102830	117563
2	79932	90690	104504	123180
3	78661	92302	108156	127452
4	76396	95250	X	X

Os resultados apresentados até aqui corroboram a expectativa de que a adição de novos *storage servers* pode efetivamente dividir o tráfego gerado por cada *storage* mas tem como conseqüência o aumento da variação desse tráfego quando a proporção de clientes por *storage server* é mantido constante. Como os blocos são distribuídos aleatoriamente entre os *storage servers* é possível que em alguns instantes de tempo muitos clientes façam pedidos por blocos que estejam em um mesmo *storage server*. Este fato pode ser explicado com modelos simples uma vez que a carga no nó é poisson.

Um aumento na variabilidade da taxa de requisições significa que o sistema vai estar mais sujeito a rajadas de pedidos. Isso pode ser prejudicial porque a fila de requisições de um *storage server* pode ficar mais cheia em alguns momentos, o que vai causar um aumento no tempo de atendimento aos pedidos.

No gráfico da figura 5.15 é apresentado o *goodness* médio dos experimentos que possuem 50 clientes por *Storage Server*. Devido ao aumento na variabilidade da taxa de requisições, é possível notar que a medida que o número de *storage servers* aumenta, há um decréscimo do *goodness* médio percebido.

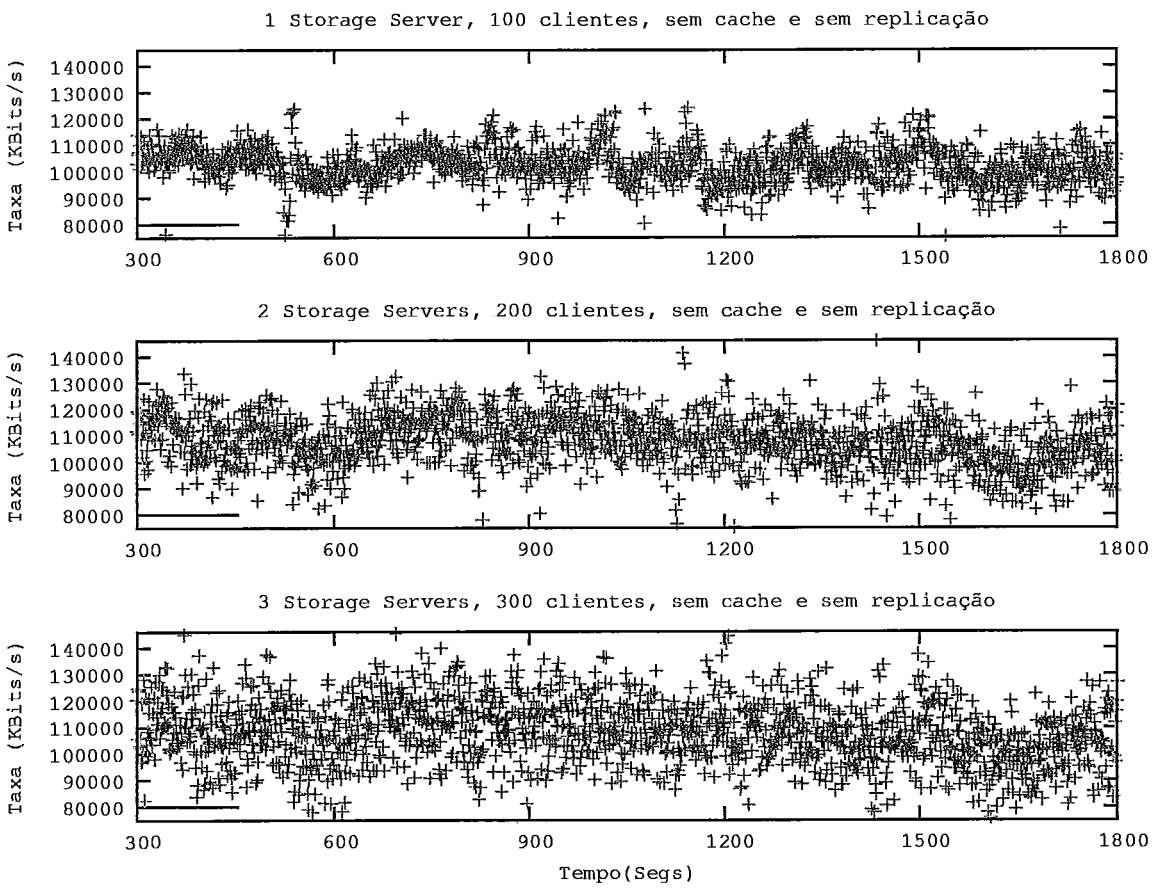

Figura 5.10: Gráficos dos experimentos com as configurações onde existem 100 clientes em média por *Storage Server*.

5.3 Segundo Experimento: RIO submetido a uma carga real de usuários interativos com a utilização de cache

5.3.1 Descrição e Objetivos

Esse experimento é muito semelhante ao anterior, a diferença fundamental é a utilização de *cache* por parte dos clientes. Aqui, também serão utilizados *logs* de comportamento coletados durante as sessões dos alunos do CEDERJ. A *cache* de cada cliente não possui limitação de tamanho sendo que ela pode armazenar todos

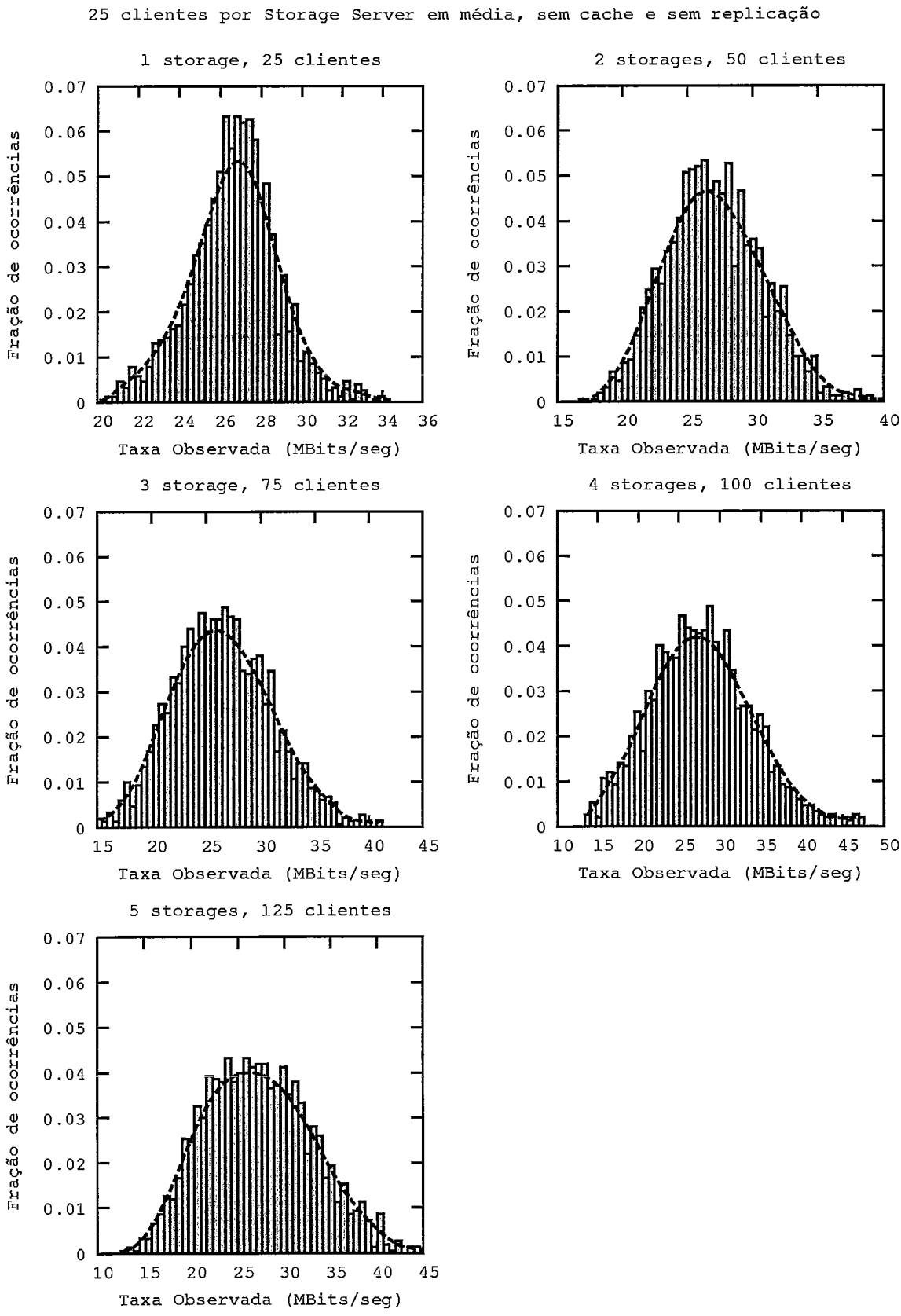

Figura 5.11: Histogramas gerados a partir dos experimentos com 25 clientes em média por *Storage Server*, trechos de 300 a 1800 segundos.

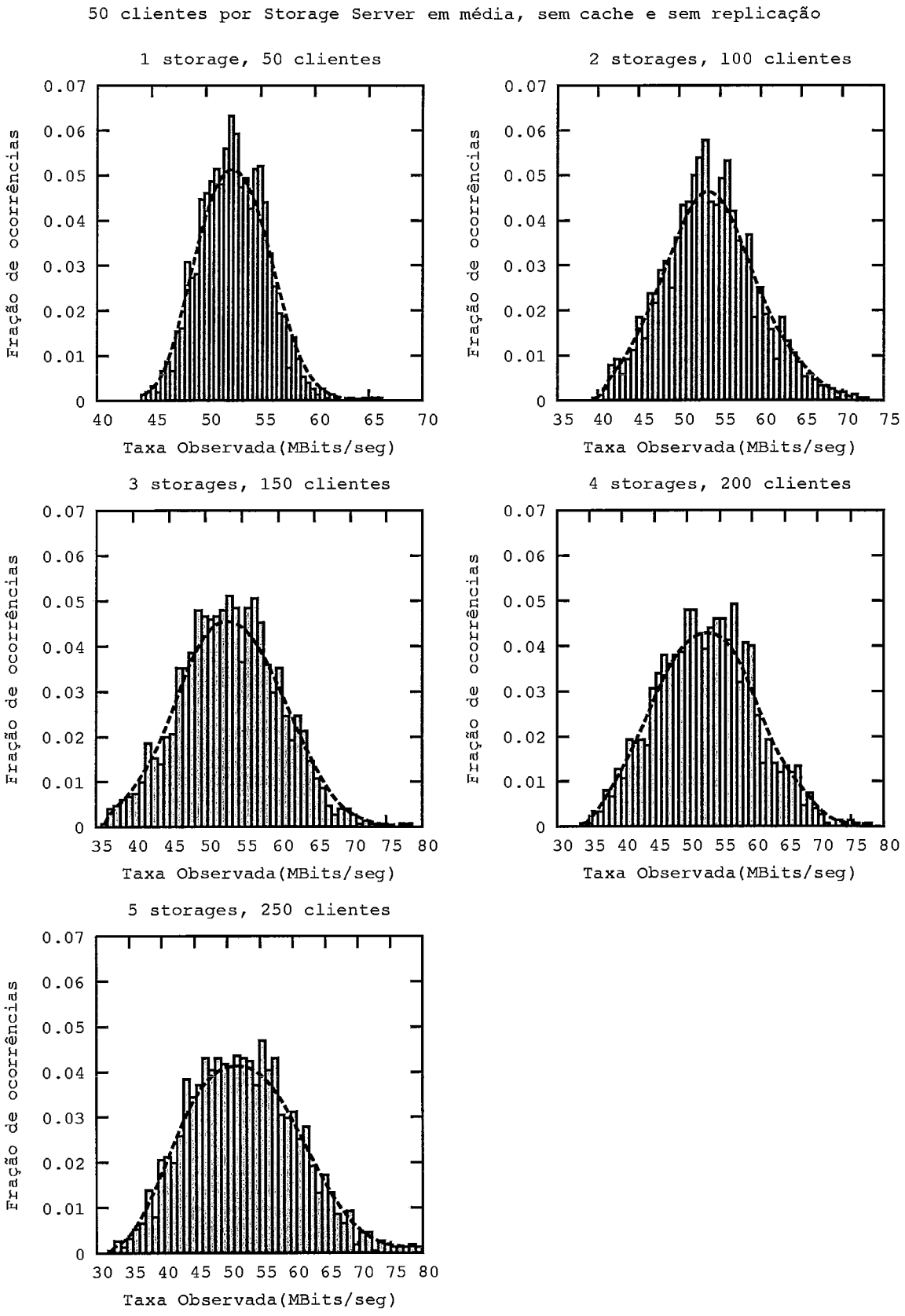

Figura 5.12: Histogramas gerados a partir dos experimentos com 50 clientes em média por *Storage Server*, trechos de 300 a 1800 segundos.

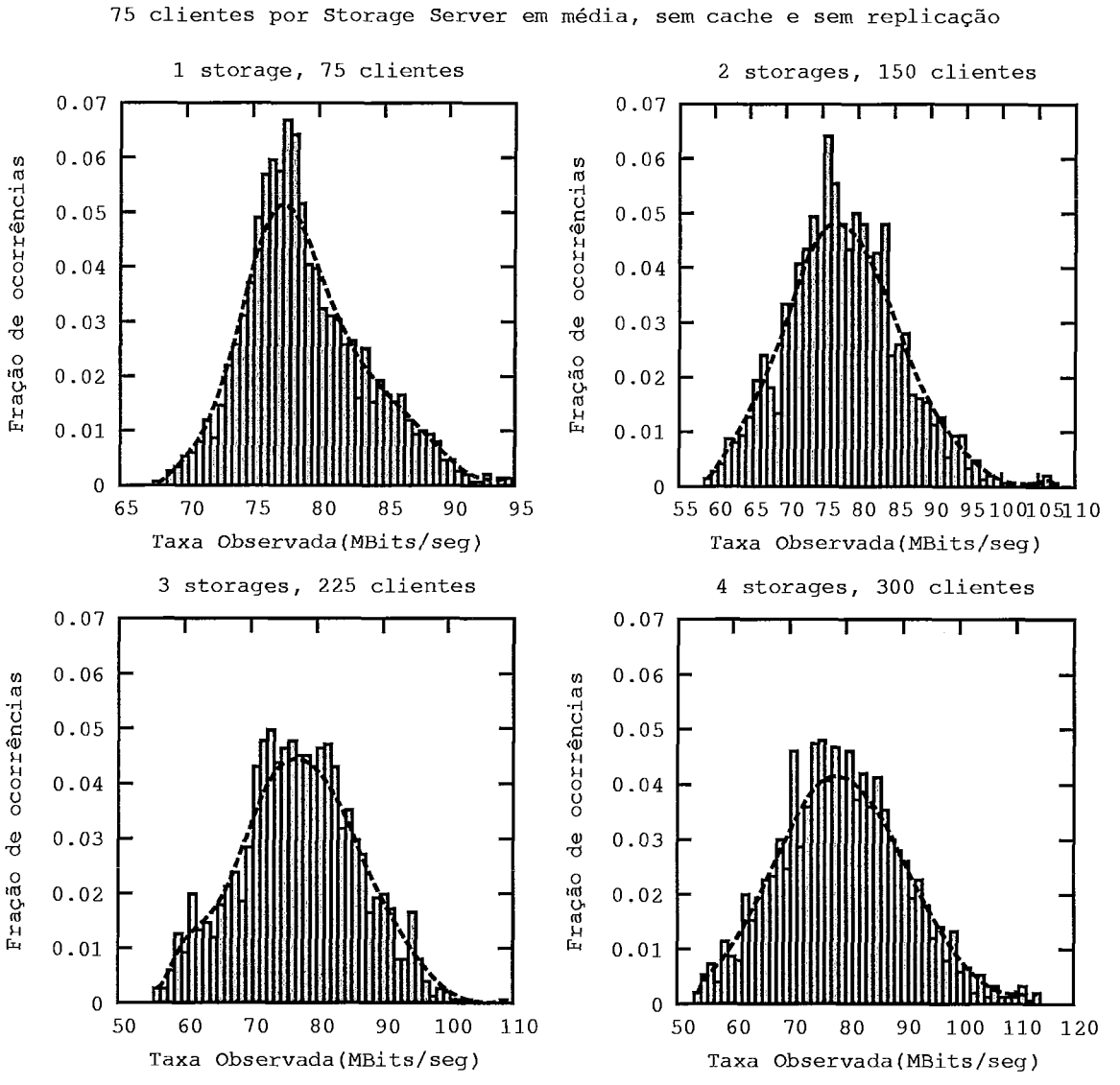

Figura 5.13: Histogramas gerados a partir dos experimentos com 75 clientes em média por *Storage Server*, trechos de 300 a 1800 segundos.

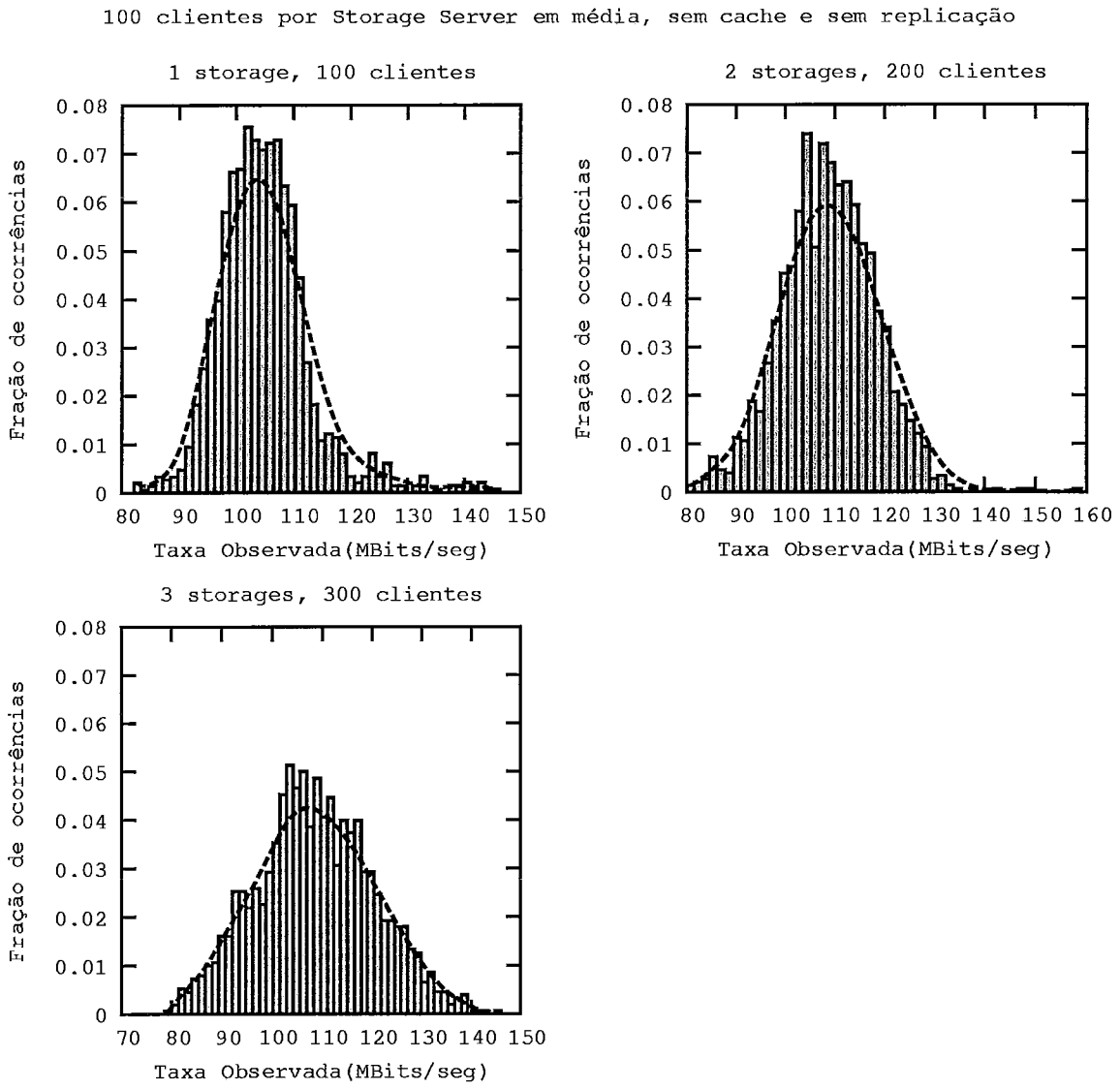

Figura 5.14: Histogramas gerados a partir dos experimentos com 100 clientes em média por *Storage Server*, trechos de 300 a 1800 segundos.

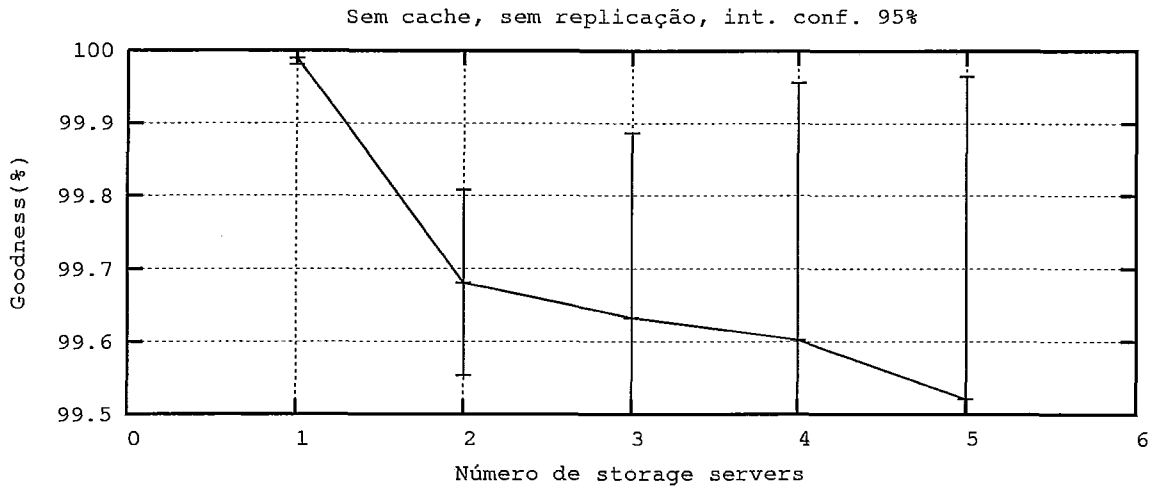

Figura 5.15: *Goodness* médio dos experimentos com 50 clientes por *storage server.*

o conteúdo assistido durante uma sessão.

Para esse experimento não foi utilizado nenhum tipo de replicação do conteúdo armazenado pelo servidor. Os experimentos serão executados conforme descrito na seção 4.4.

5.3.2 Descrição dos Resultados Esperados

Nesse experimento espera-se obter melhores resultados em relação ao experimento anterior. Espera-se obter ganhos relativos ao *goodness* do lado cliente e também observar uma queda na taxa média gerada pelo *storage server.*

5.3.3 Resultados e Análise

O gráfico da Figura 5.16 apresenta o *goodness* médio percebido pelos clientes. O eixo X representa o número de clientes e o eixo Y o *goodness* obtido. Cada curva corresponde a uma configuração distinta do servidor. Novamente é possível perceber perdas a medida que aumentamos o número de clientes que são causadas devido ao aumento no número de clientes disparado por máquina bem como pelo aumento na variabilidade da taxa de requisições. Como no experimento anterior, é possível notar ganhos com a adição de novos *Storage Servers*, mesmo que o *storage server* esteja

localizado na UFMG.

Neste segundo experimento também é possível observar que na configuração com 5 *storage servers* ocorrem perdas mesmo com poucos clientes no sistema. Como já foi explicado anteriormente, essas perdas não dependem da carga do sistema e são inerentes ao canal de comunicação entre UFMG e UFRJ. Apesar dessas perdas, com uma população de 225 clientes já se torna vantajosa a utilização do quinto *storage server.*

Nos gráficos da Figura 5.17 é possível comparar o *goodness* médio obtido nos experimentos com e sem *cache.* É possível notar que o ganho com a utilização de *cache* aumenta a medida que a população de clientes cresce. Foi possível observar uma melhoria de até 0,5% no *goodness* médio de uma população de 350 clientes. Esse valor é pequeno, mas é possível que com um número maior de clientes os ganhos se tornem mais visíveis.

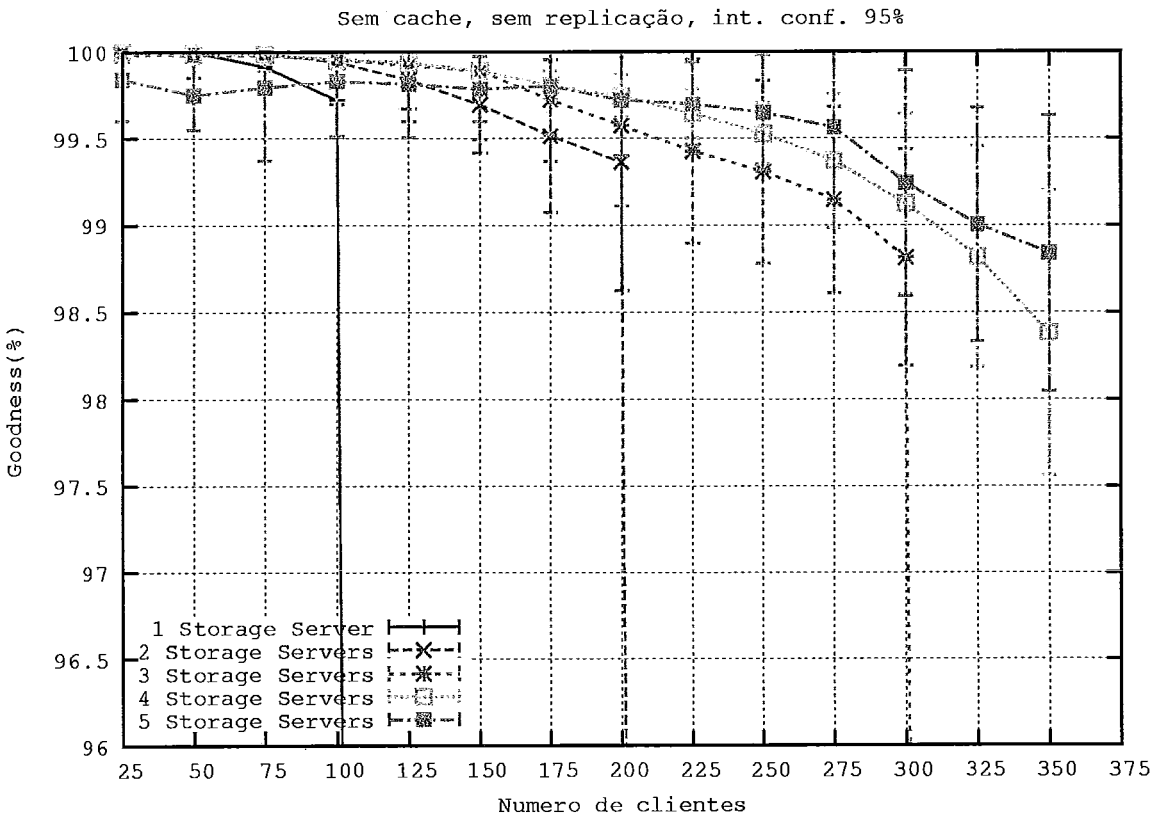

Figura 5.16: Gráfico do número de clientes X *Goodness*, 1 a 5 storage servers, com utilização de *cache* e sem replicação.

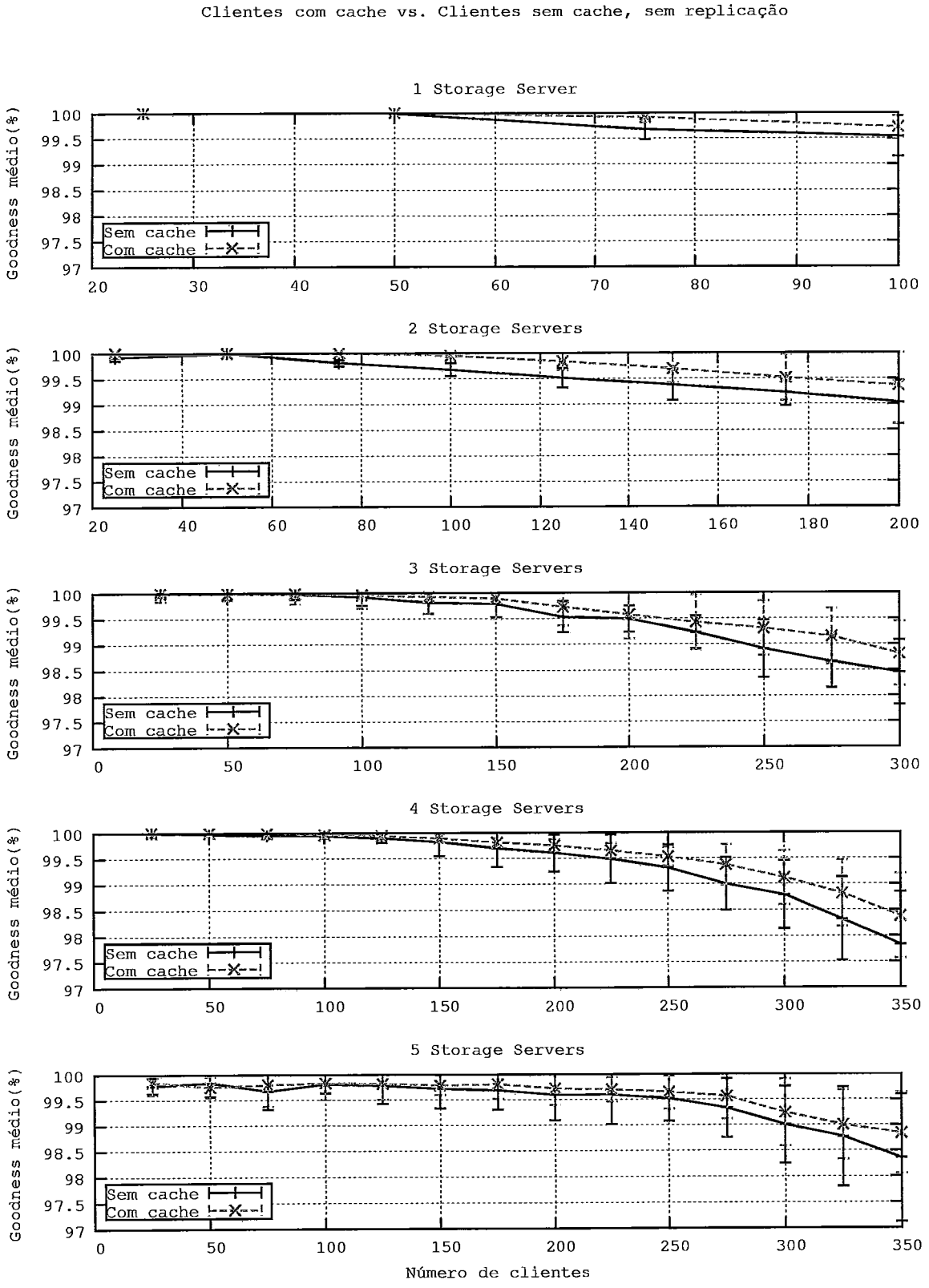

Figura 5.17: Comparação entre o *goodness* médio percebido pelos clientes com e sem a utilização de *cache*

Para completar as informações coletadas do lado cliente, o gráfico da Figura 5.18 apresenta o número de clientes atendidos, em cada configuração do servidor, com *goodness* médio acima de 98% e 99%, respectivamente. Em muitos cenários, ao se comparar com o experimento anterior, foi possível atender a cerca de 25 clientes a mais mantendo o *goodness* médio acima do limite estabelecido. No caso mais extremo, utilizando 3 *storage servers*, foi possível atender até 275 clientes com *goodness* médio acima de 99%. No experimento anterior esse valor foi de 225 clientes.

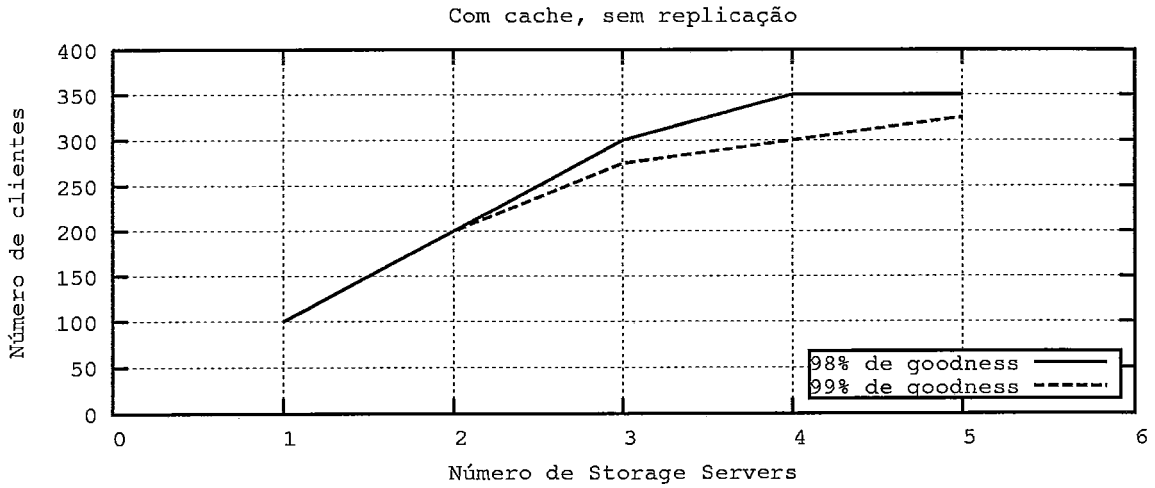

Figura 5.18: Gráfico que exibe, para cada configuração do servidor, o número de clientes atendidos com *goodness* médio de até 98% e 99%, respectivamente.

A partir de agora, é feito um estudo dos resultados obtidos com relação as medidas coletadas do lado servidor. Na Tabela 5.4 são apresentados os resultados para a média e desvio padrão da carga no *storage server*. Comparando com os resultados da seção 5.2, é possível perceber uma economia de banda com a utilização de *cache* nos clientes. A Tabela 5.5 apresenta a porcentagem da economia obtida entre os experimentos com e sem *cache*. É possível perceber valores que variam na faixa de 2% a 10%. Vale lembrar que essa economia de banda está diretamente ligada ao padrão de interatividade dos alunos do CEDERJ.

Apesar dos ganhos relativos a economia de banda, é possível notar que a utilização de *cache* não apresentou resultados muito diferentes no que diz respeito a variabilidade do tráfego gerado pelo *Storage Server*. Na Tabela 5.6 são apresentados os coeficientes de variação obtidos nos testes com e sem utilização de *cache*. Isso está

ocorrendo mesmo com a utilização de *cache*, em determinados instantes de tempo, o *storage server* recebe uma grande quantidade de requisições devido a interatividade dos usuários.

Ao final desse experimento pode-se afirmar que a utilização de *cache* nos clientes contribui para o decréscimo dos requisitos de largura de banda e conseqüentemente para a melhoria da qualidade do vídeo assistido por cada cliente. Apesar disso, a utilização de *cache* não trouxe ganhos significativos no que diz respeito a variabilidade no tráfego enviado pelo *storage server*.

Tabela 5.4: Média e Desvio Padrão das taxas geradas pelo Storage Server com a utilização de *cache no cliente*, MBits/Seg. Amostras coletadas entre 300 e 1800 segundos de duração do experimento.

	1 Storage		2 Storages		3 Storages		4 Storages		5 Storages	
Clientes	Média	D.Pad.	Média	D.Pad.	Média	D.Pad.	Média	D.Pad.	Média	D.Pad.
25	25,429	2,302	11,051	2,429	8,120	3,085	6,143	2,159	5,294	2,192
50	50,821	3,023	25,651	3,798	16,506	3,782	12,767	3,198	10,199	2,900
75	75,664	4,911	38,968	4,816	24,931	4,654	19,274	4,115	15,077	3,647
100	91,180	6,803	50,660	5,923	34,798	5,553	25,004	5,369	21,051	3,794
125	X	X	59,452	7,610	39,653	6,088	29,418	5,598	25,893	5,628
150	X	X	74,814	7,972	49,740	6,769	35,013	5,859	29,916	5,791
175	X	X	92,776	8,623	56,623	7,074	40,843	6,265	34,209	5,955
200	X	X	99,948	10,055	64,925	7,384	48,375	7,228	39,019	6,423
225	X	X	X	X	74,120	8,468	57,377	7,616	44,782	6,614
250	X	X	X	X	82,415	8,834	61,732	8,292	49,594	7,229
275	X	X	X	X	91,798	10,575	70,393	9,664	54,389	7,972
300	X	X	X	X	101,445	11,475	75,746	9,879	58,132	8,475
325	X	X	X	X	X	X	82,684	10,419	63,671	9,465
350	X	X	X	X	X	X	88,562	11,037	66,550	10,595

5.4 Terceiro Experimento: RIO submetido a uma carga real com replicação de conteúdo

5.4.1 Descrição e Objetivos

Este experimento será caracterizado pela replicação do conteúdo contido nos *Storage Servers*. Cada bloco terá uma segunda cópia que será colocada em um *Storage*

Tabela 5.5: Economia de banda com a utilização de *cache* em relação aos experimentos sem utilização de *cache*

Clientes	1 Storage	2 Storages	3 Storages	4 Storages	5 Storages
25	2,631%	4,969%	3,878%	7,738%	5,210%
50	3,209%	3,339%	7,317%	5,029%	3,318%
75	5,297%	3,671%	9,134%	6,205%	4,346%
100	4,522%	5,585%	6,688%	9,208%	4,794%
125	X	6,724%	8,104%	8,751%	3,571%
150	X	6,403%	8,452%	8,775%	5,101%
175	X	5,491%	8,589%	8,904%	4,286%
200	X	4,360%	7,109%	9,745%	4,878%
225	X	X	5,773%	5,610%	4,031%
250	X	X	6,517%	7,100%	5,380%
275	X	X	3,634%	4,369%	5,463%
300	X	X	6,205%	3,338%	5,657%
325	X	X	X	4,381%	3,278%
350	X	X	X	3,570%	6,322%

Tabela 5.6: Comparação entre o coeficiente de variação das taxas em experimentos sem e com utilização de *cache*

Clientes	1 Storage		2 Storages		3 Storages		4 Storages		5 Storages	
	S *cache*	C *cache*	S *cache*	C *cache*	S *cache*	C *cache*	S *cache*	C *cache*	S *cache*	C *cache*
25	0,089	0,091	0,186	0,220	0,354	0,380	0,337	0,351	0,370	0,414
50	0,059	0,059	0,149	0,148	0,223	0,229	0,249	0,250	0,309	0,284
75	0,062	0,065	0,121	0,124	0,178	0,187	0,205	0,213	0,257	0,242
100	0,068	0,075	0,111	0,117	0,152	0,160	0,202	0,215	0,191	0,180
125	X	X	0,121	0,128	0,144	0,154	0,173	0,190	0,220	0,217
150	X	X	0,102	0,107	0,129	0,136	0,158	0,167	0,188	0,194
175	X	X	0,094	0,093	0,116	0,125	0,145	0,153	0,168	0,174
200	X	X	0,097	0,101	0,109	0,114	0,137	0,149	0,165	0,165
225	X	X	X	X	0,109	0,114	0,128	0,133	0,147	0,148
250	X	X	X	X	0,103	0,107	0,124	0,134	0,144	0,146
275	X	X	X	X	0,113	0,115	0,133	0,137	0,142	0,147
300	X	X	X	X	0,109	0,113	0,132	0,130	0,140	0,146
325	X	X	x	X	X	X	0,122	0,126	0,146	0,149
350	X	X	X	X	X	X	0,127	0,125	0,153	0,159

Server diferente de onde a cópia original reside. A escolha da localização da segunda cópia é feita de maneira aleatória pelo servidor. Serão feitos experimentos com 2 e 3 *Storage Servers* e durante os experimentos, os clientes não estarão utilizando *cache*.

5.4.2 Descrição dos Resultados Esperados

Na configuração utilizada durante esses experimentos, vão existir duas cópias de cada bloco armazenada em *storage servers* diferentes. Dessa maneira, será possível ao servidor escolher o *storage server* com a menor fila para servir a uma requisição de bloco. Com isso, espera-se diminuir o desbalanceamento a curto prazo. Isso poderá ser constatado com a observação de uma variabilidade menor entre as amostras coletadas do lado servidor e conseqüentemente com a melhoria da qualidade percebida pelos clientes.

5.4.3 Resultados e Análise

A Figura 5.19 exibe o gráfico onde o eixo Y representa o *goodness* médio obtido e o eixo X representa número de clientes. Cada curva representa uma configuração do servidor e o intervalo de confiança é de 95%.

A Figura 5.20 mostra o *goodness* médio obtido para os cenários com e sem replicação. O eixo Y representa o *goodness médio* e o eixo X a quantidade de clientes. Foram colocadas as curvas relativas aos experimentos com 2 e 3 *storages* com e sem replicação e sem utilização de *cache*.

Apesar de estarem dentro do intervalo de confiança, é possível notar que na média, o *goodness* percebido pelos clientes aumenta sensivelmente quando se replica o conteúdo. Essa melhora não é maior porque o gargalo do sistema é a interface de rede dos clientes, porém ela existe. Para entender melhor essa melhoria, uma breve análise no que está ocorrendo do lado do servidor é necessária.

A Tabela 5.7 exibe a média e o desvio padrão das amostras coletadas durante os experimentos. Da mesma maneira que nas seções anteriores, só foram contabilizadas

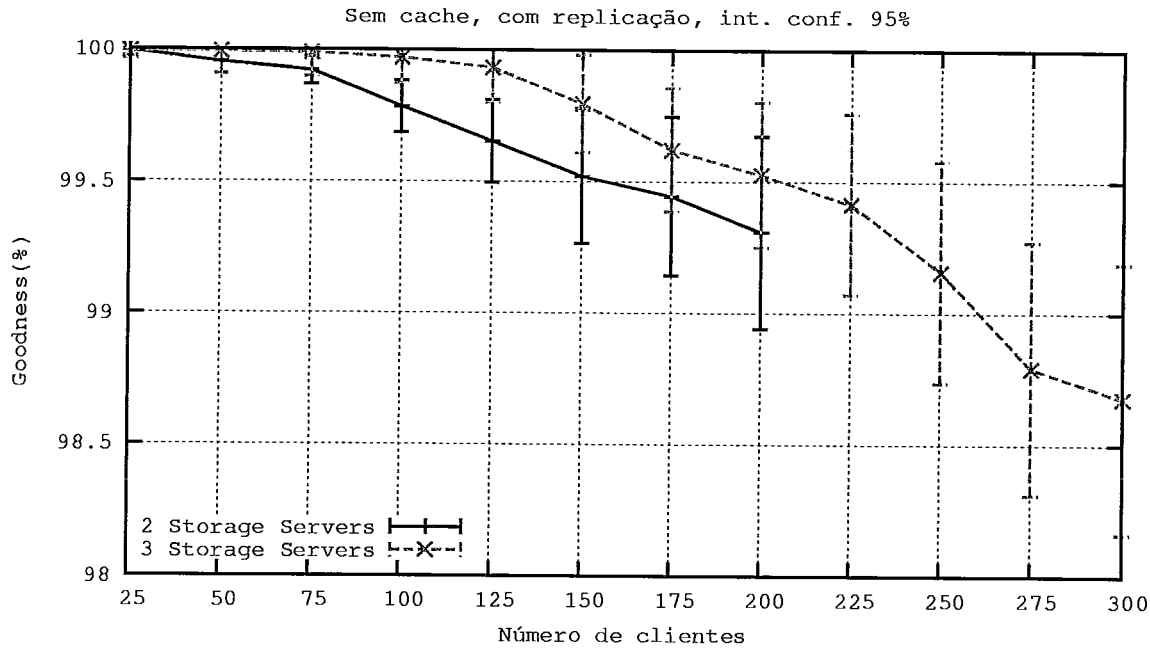

Figura 5.19: Gráfico que exibe o *goodness* médio percebido pelos clientes com replicação de conteúdo no servidor, experimentos com 2 e 3 *storage servers* respectivamente.

amostras coletadas entre 300 e 1800 segundos como garantia de que todos os clientes estavam conectados ao servidor.

Tabela 5.7: Média e Desvio Padrão nas taxas geradas pelo Storage Server com a replicação de conteúdo, MBits/Seg. Amostras coletadas entre 300 e 1800 segundos de duração do experimento.

	2 Storages		3 Storages	
Clientes	Média	D.Pad.	Média	D.Pad.
25	13,872	2,236	9,874	2,431
50	28,993	3,285	18,101	3,050
75	39,284	3,935	27,193	4,203
100	53,323	5,062	36,398	5,057
125	65,355	5,705	41,166	5,271
150	81,605	7,055	51,161	5,899
175	94,710	7,493	62,750	6,330
200	105,430	8,818	71,083	6,972
225	X	X	77,890	7,526
250	X	X	87,883	8,065
275	X	X	96,279	8,970
300	X	X	105,410	9,746

Os gráficos da Figura 5.21 mostram uma comparação entre experimentos com 2

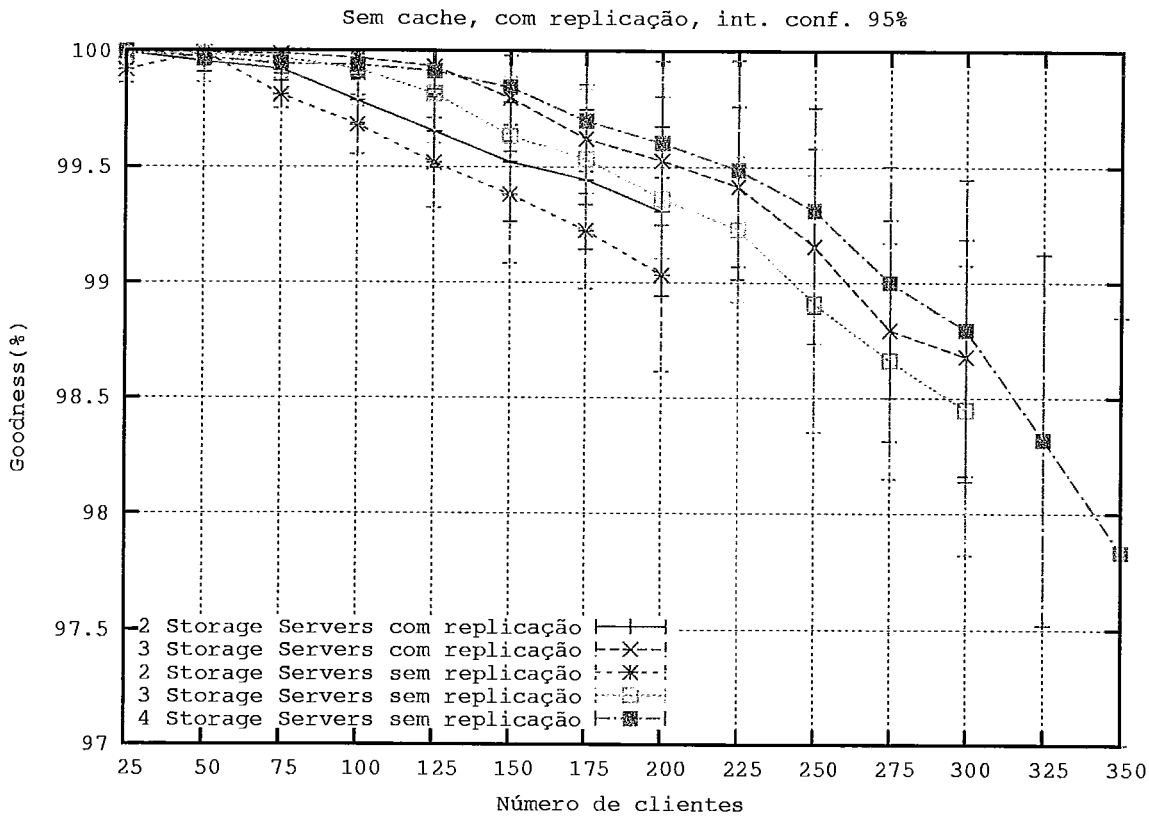

Figura 5.20: Comparação entre o *goodness* médio experimentado pelos clientes com e sem a replicação de conteúdo no servidor. Experimentos com 2 e 3 *storage servers* sem a utilização de *cache* nos clientes.

e 3 *storage servers* respectivamente. Nos gráficos, o eixo Y representa o coeficiente de variação e o eixo X o número de clientes.

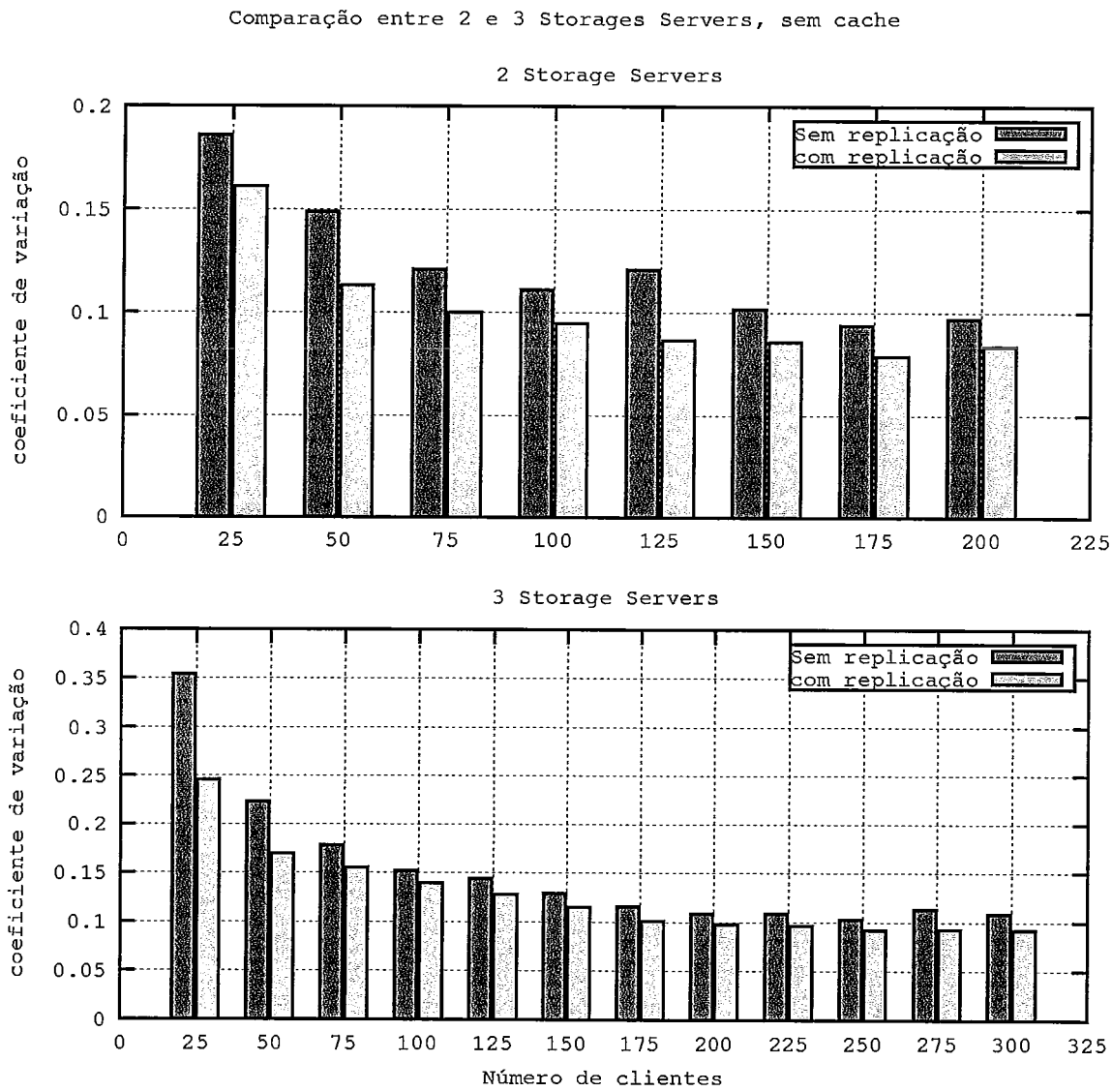

Figura 5.21: Comparação entre o coeficiente de variação observado nos experimentos com e sem replicação de conteúdo. Experimentos com 2 e 3 *storage servers* sem a utilização de *cache* nos clientes

É possível perceber que os experimentos com replicação apresentaram um resultado melhor no que diz respeito a variabilidade do tráfego gerado pelo servidor. Isso significa que, conforme descrito na literatura, o balanceamento de carga é melhor quando se utiliza replicação. Para atender ao público de ensino a distância, a utilização de replicação de conteúdo pode ser uma boa opção para melhorar a qualidade conteúdo transmitido.

5.5 Quarto Experimento: RIO submetido a uma carga sintética

5.5.1 Descrição e Objetivos

A característica determinante deste experimento diz respeito a utilização de uma carga sintética, ao invés da carga coletada diretamente das sessões dos alunos do CEDERJ. Essa carga foi gerada a partir de um modelo HMM hierárquico, que foi desenvolvido em um trabalho paralelo a esse [55]. Para o treinamento do modelo, foram utilizados os *logs* coletados das sessões dos alunos do CEDERJ.

Para manter a semelhança entre os experimentos, após a geração dos *logs* sintéticos, somente os que possuíam duração entre 30 minutos e 1 hora foram escolhidos. Estes foram então traduzidos para o formato de entrada do emulador de clientes.

O objetivo desse experimento é verificar se os resultados obtidos com a carga sintética são semelhantes aos obtidos com a carga real. A avaliação da acurácia do modelo em um ambiente real é uma etapa importante para a validação do modelo.

5.5.2 Descrição dos Resultados Esperados

Com a realização deste experimento espera-se obter resultados semelhantes aos obtidos no experimento com carga real. Dessa maneira o modelo poderá ser validado e a carga gerada por este poderá ser utilizada em experimentos futuros.

5.5.3 Resultados e Análise

No gráfico da Figura 5.22 é possível ver a relação entre o *goodness* médio e o número de clientes. De maneira semelhante a observada com os experimentos com carga real, é possível notar que a adição de novos *storage servers* reflete em uma melhoria do *goodness* obtido pelos clientes.

Nos gráficos da Figura 5.23 é ilustrada a comparação entre o *goodness* médio

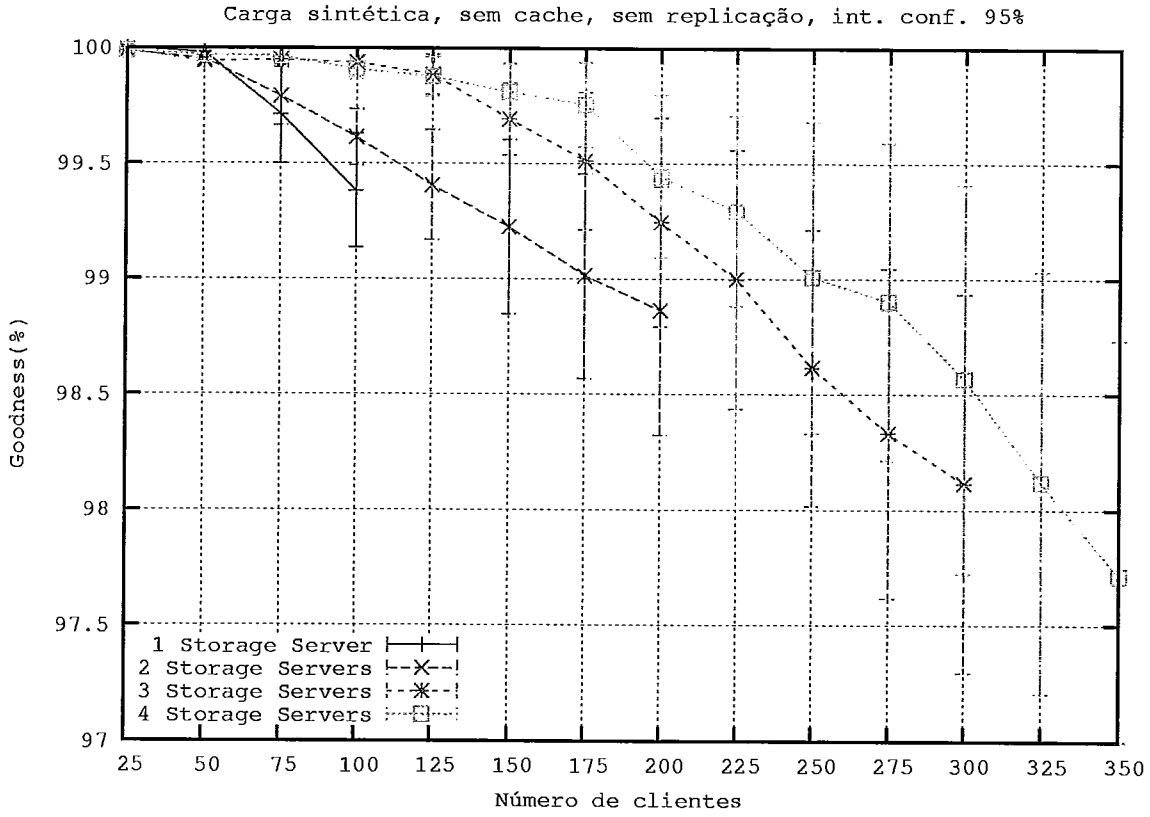

Figura 5.22: Gráfico do número de clientes X *Goodness*. 1 a 4 Clientes, sem utilização de *cache*, sem replicação e com utilização de carga sintética.

obtido com os experimentos com carga real e sintética, sem a utilização de *cache*. Os *goodness* obtido com a carga sintética é ligeiramente inferior. Apesar disso, essa diferença é muito pequena, ainda mais quando se leva em conta o intervalo de confiança de 95%, a maior diferença entre os valores encontrados foi de cerca de 0,3% de *goodness* no cenário onde existem 350 clientes e uma configuração com 3 *storage servers*.

Avaliando a carga imposta aos *storage servers*, a Tabela 5.8 exibe os valores da média e desvio padrão obtidos com as amostras coletadas do *storage server*. Pode-se notar que os valores exibidos nessa tabela são bem próximos dos obtidos com os experimentos que utilizam carga real.

Tabela 5.8: Média e Desvio Padrão das taxas geradas pelo Storage Server quando submetido a uma carga sintética. Amostras coletadas entre 300 e 1800 segundos de duração do experimento.

	1 Storage		2 Storages		3 Storages		4 Storages	
Clientes	Média	D.Pad.	Média	D.Pad.	Média	D.Pad.	Média	D.Pad.
25	26,512	2,371	14,102	2,382	8,853	2,741	6,875	2,546
50	53,783	3,263	27,003	4,396	17,809	3,856	13,443	3,678
75	76,914	4,635	38,819	5,245	27,437	5,170	20,549	4,110
100	104,713	7,377	51,582	5,730	37,292	5,875	27,540	5,369
125	X	X	65,132	7,923	43,150	6,459	33,335	5,638
150	X	X	77,991	8,269	54,332	6,991	38,381	6,100
175	X	X	94,541	8,625	61,943	7,471	44,835	6,770
200	X	X	107,504	10,447	69,894	7,863	53,598	7,302
225	X	X	X	X	78,661	8,620	60,787	7,919
250	X	X	X	X	88,160	9,296	68,603	8,451
275	X	X	X	X	94,533	10,503	73,609	9,657
300	X	X	X	X	108,156	11,379	76,396	10,258
325	X	X	X	X	X	X	86,472	10,735
350	X	X	X	X	X	X	91,841	11,459

Para ilustrar melhor as informações contidas na tabela, os gráficos da Figura 5.24 apresentam uma comparação entre o coeficiente de variação obtido nos experimentos com carga real e carga sintética respectivamente. Nesses gráficos, não é possível notar uma diferença muito grande nos resultados obtidos, a maior diferença observada foi de aproximadamente 0,4% no cenário com 25 clientes e 4 *storage servers*.

A partir dos resultados obtidos, é possível constatar que os experimentos utili-

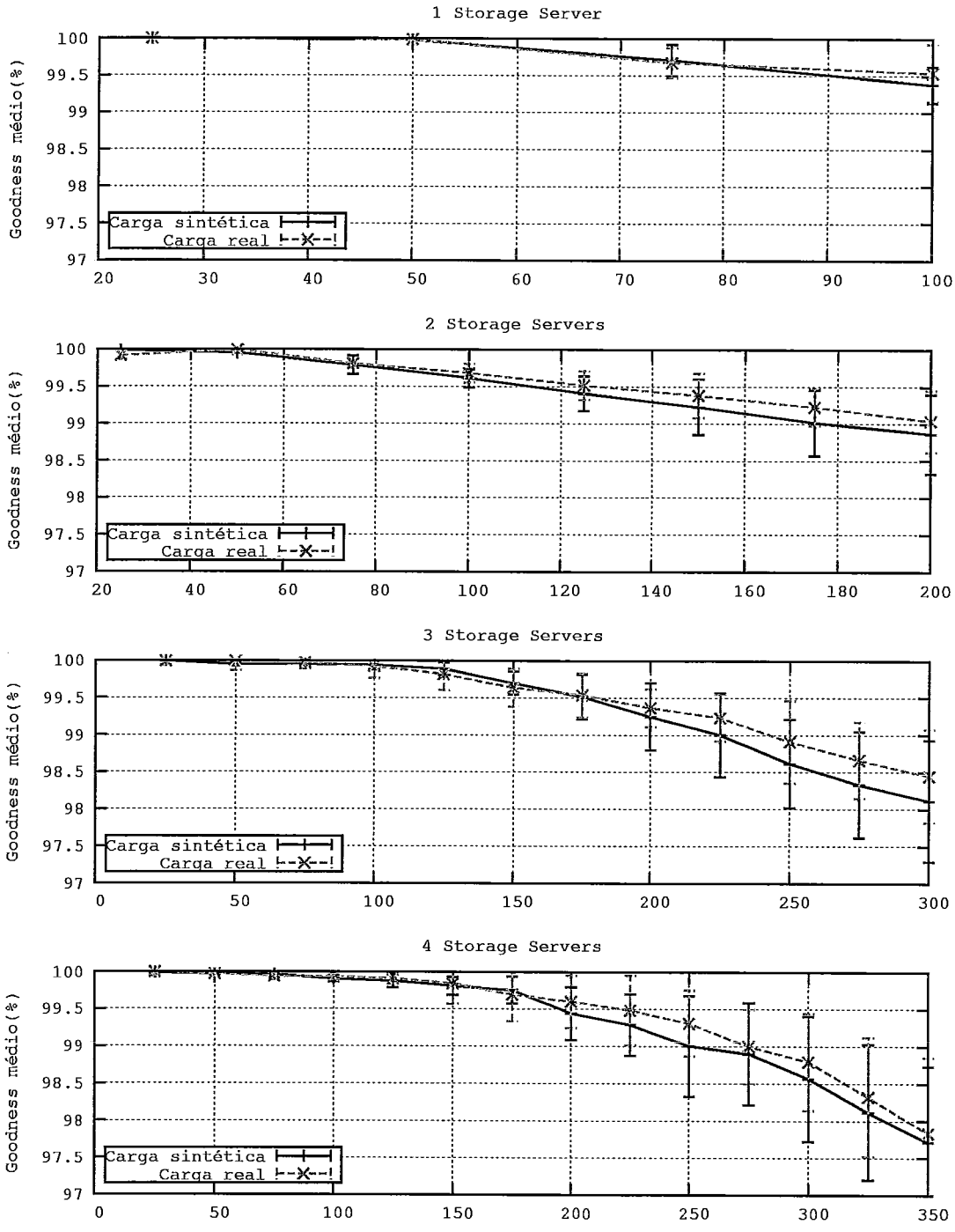

Figura 5.23: Comparação do *goodness* observado entre os experimentos que utilizam carga real e carga sintética.

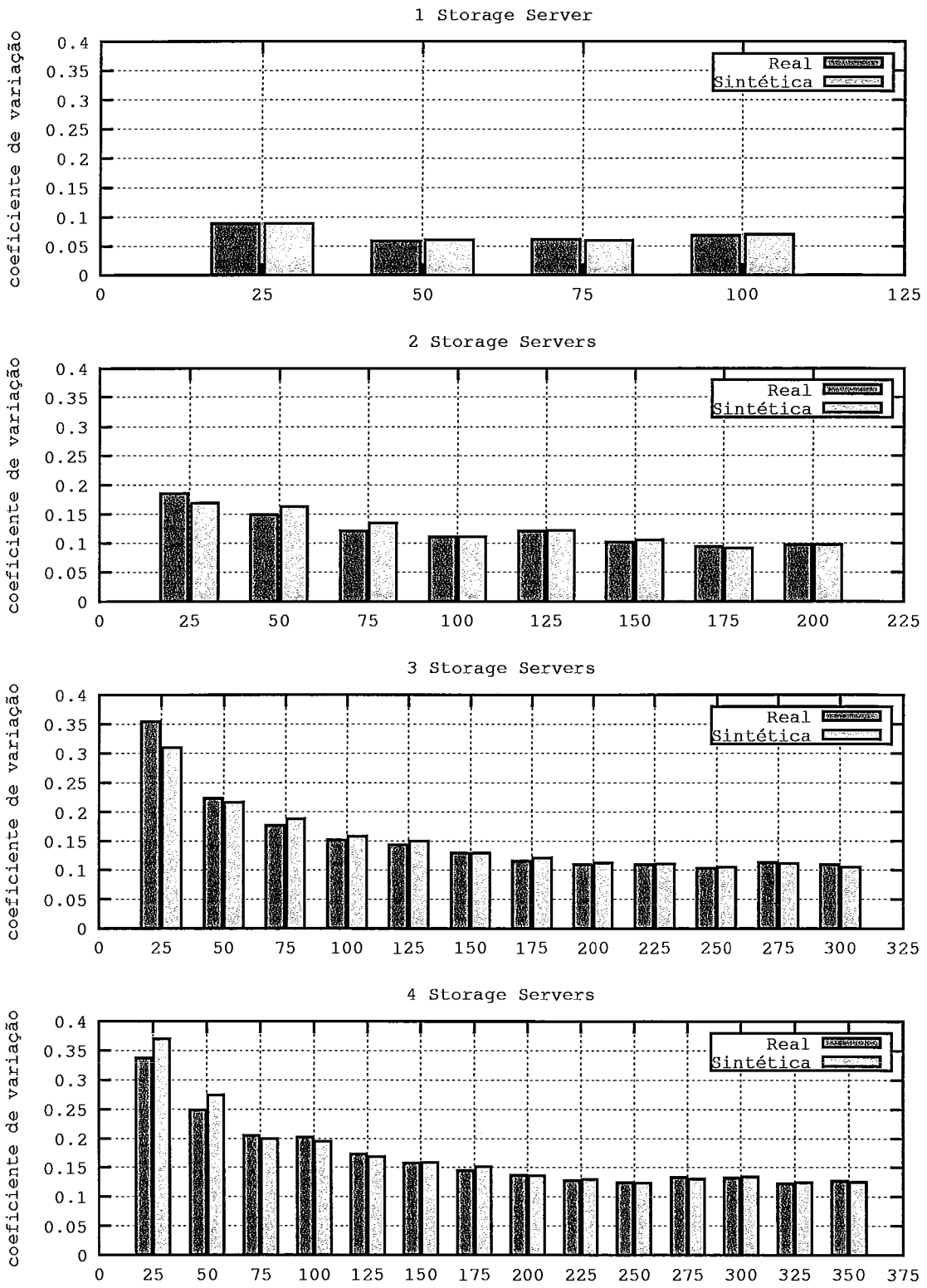

Figura 5.24: Comparação entre os coeficientes de variação obtidos nos experimentos com carga real e carga sintética.

zando a carga sintética gerada pelo modelo HMM hierárquico apresentam resultados muito próximos dos que utilizaram uma carga real. Isso indica que o modelo foi capaz de capturar as características de interatividade dos *logs* reais de maneira satisfatória. Esse modelo poderá ser utilizado a partir de então para gerar carga sintética.

Uma das vantagens do modelo é a possibilidade de emular o número de clientes que forem necessários. Não havendo mais a limitação imposta pelo número de *logs* reais disponíveis e ainda realizar estudos com parâmetros diferentes que emulariam outros comportamentos de usuários.

Capítulo 6

Conclusões e Trabalhos Futuros

A demanda por aplicações como vídeo e áudio sob demanda na Internet vêm crescendo continuamente. Para a transmissão desse tipo de conteúdo, é necessário observar restrições muito rígidas com relação ao tempo de entrega dos blocos de um objeto multimídia. Esse fato não foi previsto na concepção e nos primeiros estágios da Internet. Sendo assim, a Internet não pode garantir que essas restrições sejam atendidas de maneira satisfatória. Tornou-se necessário então prover essas garantias na camada de aplicação, o que levou ao surgimento de várias propostas de servidores visando dar suporte a distribuição de conteúdo multimídia.

A partir de então, a Internet vem se tornando o principal veículo para disseminação da educação a distância. Por isso faz-se necessário o desenvolvimento e aprimoramento de ferramentas que suportem essa disseminação. Muito tem sido estudado sobre a distribuição de conteúdo multimídia na Internet, mas apenas uma pequena fração desses estudos tem o foco na educação a distância, com testes em um ambiente real. Esse trabalho vem contribuir nessa área, por meio do estudo de um servidor multimídia através de experimentos realizados em uma rede de alta velocidade submetido a uma carga de usuários interativos.

Para permitir a realização dos experimentos apresentados nessa dissertação, foi montada toda uma infraestrutura utilizando a Rede Giga e ligando as seguintes instituições: UFRJ, UFF, Fiocruz e UFMG. Utilizamos o servidor Multimídia RIO,

desenvolvido no LAND, distribuído entre todas as instituições participantes. Observamos o comportamento do servidor quando submetido a uma carga real de usuários interativos, coletada durante as sessões geradas pelos alunos do CEDERJ. Essa carga apresenta características de interatividade que ainda não haviam sido experimentadas com o RIO. Foram observados também os ganhos relativos a utilização de *cache* no cliente e replicação de conteúdo. Foram avaliadas a escalabilidade do servidor bem como o impacto de escolhas de projeto tais como utilização de *cache*, replicação de conteúdo e o posicionamento de um *storage server* em uma rede de características inferiores a Rede Giga. Também foi possível avaliar a acurácia de um modelo de caraga sintética desenvolvido em um trabalho paralelo a este.

As principais conclusões deste trabalho foram:

- Foi mostrado que o servidor é escalável ao se verificar ganhos no *goodness* quase linear com a adição de novos *storage servers*.
- A adição de um *storage server* em uma rede com características inferiores as demais é vantajosa a partir do momento em que a carga poupada com a adição de um novo *storage server* é suficiente para compensar as perdas na rede inferior.
- Através da captura do tráfego gerado por um *storage server*, foi possível verificar que a técnica de alocação aleatória de blocos é eficaz para promover um bom balanceamento de carga entre os *storage servers*.
- Um aumento na variabilidade da taxa de requisições tem um impacto negativo na qualidade do serviço prestado ao cliente visto que isso implica em uma latência maior no atendimento dos pedidos.
- Apesar da economia de banda obtida com a utilização de *cache*, não foi possível verificar ganhos com relação a variabilidade do tráfego gerado pelo *storage server*.
- A replicação de conteúdo é efetiva para diminuir os efeitos da variabilidade na taxa de requisições promovendo uma melhoria do *goodness* obtido pelos

clientes.

- Os resultados obtidos com a carga gerada pelo modelo utilizado foram muito próximos dos obtidos com uma carga real. Sendo assim, foi confirmada a acurácia do modelo, que poderá ser utilizado para a geração de carga em experimentos futuros.

6.1 Trabalhos Futuros

Um trabalho futuro que pode ser feito é a realização de experimentos com um número maior de clientes e *storage servers*. Seria interessante que esses clientes estivessem distribuídos em outras localidades de maneira a distribuir o tráfego de forma mais homogênea, alguns desses clientes poderiam acessar o servidor através da rege giga ou através da Internet. Esse tipo de experimento pode ser facilmente realizado pois é possível emular uma grande quantidade de clientes usando o modelo de geração de carga sintética. Para isso será necessário a disponibilidade maior de equipamentos, não só na UFRJ mas em todas as instituições participantes.

Com experimentos em uma escala maior, seria possível avaliar de maneira mais abrangente aspectos tais como o balanceamento de carga e os benefícios da replicação. Seria interessante avaliar se a replicação de apenas uma fração do conteúdo poderia trazer ganhos satisfatórios.

Referências Bibliográficas

[1] Akamai. URL http://www.akamai.com/.

[2] BitTorrent. URL http://www.bittorrent.com.

[3] CEDERJ: Centro de EDucação de Ensino superior a distância do estado do Rio de Janeiro. URL http://www.cederj.rj.gov.br.

[4] CPqD - Centro de Pesquisa e Desenvolvimento em Telecomunicações. URL http://www.cpqd.com.br/.

[5] Dell Homepage. URL http://www.dell.com/.

[6] Extreme Networks. URL http://www.extremenetworks.com/.

[7] FINEP - Financiadora de Estudos e Projetos. URL http://www.finep.gov.br/.

[8] Gnutella. URL http://www.gnutella.com.

[9] Kazaa. URL http://www.kazaa.com.

[10] LAND: Laboratório de ANálise, modelagem e Desenvolvimento de redes e sistemas de computação. URL http://www.land.ufrj.br/.

[11] Napster. URL http://www.napster.com.

[12] NeTraMet - a Network Traffic Flow Measurement Tool. URL http://www.caida.org/tools/measurement/netramet/.

[13] Projeto Giga. URL http://www.projetogiga.org.br/.

[14] RFC 2721 - RTFM: Applicability Statement. URL `http://www.faqs.org/ftp/rfc/pdf/rfc2721.txt.pdf`.

[15] RNP - Rede Nacional de Ensino e Pesquisa. URL `http://www.rnp.br/`.

[16] Skype. URL `http://www.skype.com/`.

[17] YouTube. URL `http://www.youtube.com/`.

[18] ALMEIDA, J. M., EAGER, D. L., FERRIS, M. C., E VERNON, M. K. Provisioning Content Distribution Networks for Streaming Media. In *Proc. of Infocom* (junho 2002).

[19] BARNETT, S. A., E ANIDO, G. J. A cost comparison of distributed and centralized approaches to video-on-demand. *IEEE Journal on Selected Areas in Communications 14* (agosto 1996), 1173–1183.

[20] BENSON, S., GOLUBCHIK, L., E MUNTZ, R. M. Fault Tolerant Design of Multimedia Servers. In *Proc. of SIGMOD* (1995), pp. 364–375.

[21] BOTELHO, V. M. Experimentos com o servidor rio em um ambiente distribuído e heterogêneo. *Tese de Mestrado, Universidade Federal do Rio de Janeiro* (2006).

[22] CHAN, S. H. G., E TOBAGI, F. Distributed servers architecture for networked video sevices. In *Proc. of IEEE/ACM Transactions on Networking Vol 9, Num 2* (2001), pp. 125–136.

[23] CHEN, S., WANG, H., SHEN, B., WEE., S., E ZHANG, X. Segment-based Proxy Caching for Internet Streaming Media Delivery. *IEEE Multimedia Magazine 12* (Setembro 2005).

[24] CHOE, Y. R., E PAI, V. S. Achieving Reliable Parallel Performance in a VoD Storage Server Using Randomization and Replication. In *Proceedings of IPDPS* (2007), pp. 1–10.

[25] DE QUEVEDO CARDOZO, A. Mecanismos para garantir qualidade de serviço de aplicações de vídeos sob demanda. *Tese de Mestrado, Universidade Federal do Rio de Janeiro* (2004).

[26] DE SOUZA E SILVA, E., LEÃO, R. M. M., RIBEIRO-NETO, B., E CAMPOS, S. Performance Issues of Multimedia Applications. *Lecture Notes in Computer Science (LNCS 2459)* (2002), 374–405.

[27] DO, T., HUA, K., E TANTAOUI, M. P2vod: Providing fault tolerant video-on-demand streaming in peer-to-peer environment. *In IEEE Communications Society* (2004).

[28] FABBROCINO, F., SANTOS, J. R., E MUNTZ, R. An implicitly scalable real-time multimedia storage server. Relatório Técnico 980009, 1998.

[29] GADDE, S., CHASE, J., E RABINOVICH, M. Web caching and content distribution: A view from the interior. In *Proc. of the 5th International Web Caching and Content Delivery Workshop* (Maio 2000).

[30] GAO, L., ZHANG, Z.-L., E TOWSLEY, D. Proxy-Assisted Techniques for Delivering Continuous Multimedia Streams. In *Proc. IEEE/ACM Transactions on Networking Vol 11, Num 6* (dezembro 2003).

[31] GEREOFFY, A. MPlayer - Movie Player for linux. URL `http://www.mplayerhq.hu/homepage/`.

[32] GOLUBCHIK, L., LUI, J. C. S., TUNG, T. F., CHOE, A. L. H., LEE, W. J., FRANCESCHINIS, G., E ANGLANO, C. Multipath continuous media streaming: What are the benefits. *Performance Evaluation 49* (2002), 429–449.

[33] GOMES, L. Will All of Us Get Our 15 Minutes On a YouTube Video?, Agosto 2006. URL `http://online.wsj.com/public/us`.

[34] GORZA, M. L. Um novo mecanismo de compartilhamento de recursos para transmissão de vídeo com alta interatividade e experimentos. *Tese de Mestrado, Universidade Federal Fluminense* (2004).

[35] GOU, Y., SEN, S., , E TOWNSLEY, D. Prefix caching assisted periodic broadcast for streaming popular videos. In *Proc. of ICC (International Conference on Communication)* (abril 2002).

[36] GUO, Y., SUH, K., KUROSE, J., E TOWSLEY, D. P2cast: Peer-to-peer patching scheme for VoD service. In *Proc. of the 12th WWW Conference* (Maio 2003).

[37] GUSTAFSSON, E., E KARLSSON, G. A literature survey of traffic dispersion. *IEEE Network Magazine 11* (1997), 28–36.

[38] JOHNSON, K., CARR, J., DAY, M., E KAASHOEK, M. The Measured Performance of Content Distribution Networks. In *Proc. of the 5th International Web Caching and Content Delivery Workshop* (Maio 2000).

[39] KANGASHARJU, J., ROSS, K. W., E ROBERTS, J. W. Performance Evaluation of Redirection Schemes in Content Distribution Networks. In *Proc. of the 5th International Web Caching and Content Delivery Workshop* (Maio 2000).

[40] KUROSE, J., E ROSS, K. *Computer Networking : A Top-Down Approach Featuring the Internet*, 3rd ed. Addison Wesley, Boston, MA, 2004.

[41] LIANG, Y. J., APOSTOLOPOULOS, J. G., E GIROD, B. Analysis of packet loss for compressed video: Does burst-length matter? In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing* (2003).

[42] MA, H., E SHIN, K. G. Multicast video-on-demand services. *ACM SIGCOMM Computer Communication 32* (2002), 31–34.

[43] NETTO, B. C. M. *Patching* interativo: Um novo método de compartilhamento de recursos para transmissão de vídeo com alta interatividade. *Tese de Mestrado, Universidade Federal do Rio de Janeiro* (2004).

[44] RIBEIRO, B., E SILVA, E., E TOWSLEY, D. On the Efficiency of Path Diversity for Continuous Media Applications. Relatório técnico, UMass CMPSCI Technical Report 05-19. 2005.

[45] SANTOS, D. A. S., BORGES, A., N., B. R., E CAMPOS, S. V. A. Performance analysis and optimization of a distributed video on demand service. *IEEE Int. Symposium on Performance Analysis of Systems and Software1/6/2006* (2003), 110–121.

[46] SANTOS, J., E MUNTZ, R. Design of the RIO (Randomized I/O) Storage Server. *UCLA CSD Tech Rp., Jun* (1997).

[47] SANTOS, J., E MUNTZ, R. Performance analysis of the RIO multimedia storage system with heterogeneous disk configurations. In *Proc. of the 6th ACM international conference on Multimedia* (1998), pp. 303–308.

[48] SANTOS, J. R., E MUNTZ, R. Comparing random data allocation and data striping in multimedia servers. *ACM Sigmetrics 3* (2000), 44–55.

[49] SAROIU, S., GUMMADI, K. P., DUNN, R. J., GRIBBLE, S. D., E LEVY, H. M. An Analysis of Internet Content Delivery Systems. In *Proc. of OSDI* (Dezembro 2002).

[50] SHAIKH, A., TEWARI, R., E AGRAWAL, M. On the effectiveness of DNS-based server selection. In *Proc. of the IEEE Infocom conference* (Abril 2001).

[51] SOHRABY, K., E SIDI, M. On the Performance of Bursty and Modulated Sources Subject to Leaky Bucket Rate-Based Access Control Schemes. In *Proc. of IEEE Transactions on Comunications Vol 42, Num 3* (março 1994).

[52] TAO, S., XU, K., XU, Y., FEI, T., GAO, L., GUERIN, R., KUROSE, J., TOWSLEY, D., E ZHANG, Z. Exploring the Performance Benefits of End-to-End Path Switching. In *Proc. of IEEE ICNP* (2004).

[53] TRAN, D. A., HUA, K. A., E DO, T. T. A peer-to-peer architecture for media streaming. *IEEE JSAC Special Issue on Advances in Overlay Networks* (2004), 121–133.

[54] V. PADMANABHAN AND H. WANG AND P. CHOU AND K. SRIPANIDKULCHAI. Distributing Streaming Media Content Using Cooperative Networking. In *Proc. of ACM NOSSDAV* (2004).

[55] VIELMOND, C., LEÃO, R. M. M., E DE SOUZA E SILVA, E. Um Modelo HMM Hierárquico Para Usuários Interativos Acessando um Servidor Multimídia. *XXV Simpósio Brasileiro de Redes de Computadores (SBRC 2007).* (2007).

[56] WANG, B., SEN, S., ADLER, M., E TOWSLEY, D. Optimal proxy cache allocation for efficient streaming media distribution. In *Proc. of IEEE Infocom* (junho 2002).

[57] Y. BIRK. Random raids with selective exploitation of redundancy for high performance video servers. In *Proc. of NOSSDAV* (Maio 1997).