MODELAGEM E ANÁLISE DO COMPORTAMENTO DO SERVIDOR RIO EM AMBIENTES REAIS E HETEROGÊNEOS

Ariadne Grillo Pacheco

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.

Prof.ª Rosa Maria Meri Leão, Dr.

Prof. Daniel Ratton Figueiredo, Ph.D.

Prof.ª Morganna Carmem Diniz, Dsc.

Prof. Aloysio de Castro Pinto Pedroza, Dr.

RIO DE JANEIRO, RJ - BRASIL

SETEMBRO DE 2007

GRILLO PACHECO, ARIADNE

Modelagem e Análise do Comportamento do Servidor RIO em Ambientes Reais e Heterogêneos [Rio de Janeiro] 2007

XVII, 88 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2007)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Vídeo sob Demanda
2. Transmissão de Mídia Contínua
3. Diversidade de Caminhos
4. Compartilhamento de Banda
5. Qualidade de Serviço
6. Balanceamento de Carga
7. Redes Gigabit

I. COPPE/UFRJ II. Título (Série)

Dedico este trabalho a minha família e namorado,
aos professores e amigos que participaram
de alguma forma da concretização do mesmo.

Agradecimentos

Primeiramente quero agradecer a minha família, pois sem ela nunca teria chegado até aqui. Sempre foram a base de tudo. Meus pais, que nas maiores dificuldades souberam colocar minhas necessidades acima de qualquer outra coisa. Melhores pais eu não poderia ter. Minha irmã, que sempre soube chamar a minha atenção nos momentos certos, sempre amiga e companheira.

Quero agradecer também ao meu namorado Ricardo que nunca me deixou desanimar quando as coisas ficavam difíceis e sempre soube me fazer sorrir, mesmo estando tão longe. Nem um oceano pelo meio conseguiu abalar todo carinho e atenção.

Aos professores Edmundo e Rosa, pela orientação e pela paciência, por terem entendido as dificuldades pelas quais passei e mesmo assim terem cobrado dedicação e feito com que o trabalho fosse concluído. Agradeço imensamente.

Ao pessoal da república por ter passado os maiores apertos junto comigo e mesmo assim conseguir não perder o bom humor. E aos vários amigos que fiz durante o mestrado.

Ao pessoal do LAND um muito obrigado por tudo, as dicas, o apoio quando alguma coisa saía errado, e mesmo os momentos de descontração. Sempre solícitos para tudo. Gostaria de agradecer imensamente ao Bernardo, Bene, Allyson, GD, Guto, Sadoc, Fernando, Hugo, os Adms do laboratório, principalmente o Fabrício,

e aos meninos do projeto Diverge, Bruno, Marcello e Vinicius. Um agradecimento especial ao Flávio, sem ele este trabalho não teria saído, sempre com as respostas certas para as dúvidas mais ilógicas.

Outro agradecimento especial vai para a Carol, por sempre estar disposta a ouvir a todos e sempre com um carinho especial com cada um. E quando chegamos cansados de uma noite longa aquele chazinho é tudo.

A Deus por tudo, por me dar forças, saúde e vontade para continuar meu caminho.

Aos professores da banca por aceitarem o convite prontamente.

Ao pessoal da Fiocruz, UFF e UFMG pela disponibilidade e boa vontade durante os experimentos.

Ao CNPq pela bolsa de estudos e a Coppetec pela continuação da mesma.

Por fim, a todos os amigos que contribuíram de alguma forma ou simplesmente me deram força para concluir o trabalho.

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

MODELAGEM E ANÁLISE DO COMPORTAMENTO DO SERVIDOR RIO EM AMBIENTES REAIS E HETEROGÊNEOS

Ariadne Grillo Pacheco

Setembro/2007

Orientadores: Rosa Maria Meri Leão
Edmundo de Souza e Silva

Programa: Engenharia de Sistemas e Computação

Fornecer um serviço de vídeo sob demanda na Internet é uma tarefa desafiadora. Vídeo sob demanda é uma tecnologia base para muitas aplicações importantes tais como educação a distância. O principal objetivo deste trabalho é modelar um serviço de vídeo sob demanda baseado no Sistema de Armazenamento Multimídia RIO. O Sistema de Armazenamento Multimídia RIO é atualmente utilizado pelo curso de educação a distância do CEDERJ. Vários experimentos foram realizados utilizando a ferramenta de modelagem Tangram-II. Métricas analisadas incluem o número de clientes atendidos pelo sistema para obter um dado nível de desempenho. Nós validamos o nosso modelo por comparar resultados obtidos em um ambiente real com os computados pelo modelo.

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

ANALYSIS AND MODELING OF RIO SERVER BEHAVIOUR IN A REALS AND HETEROGENEOUS ENVIRONMENT

Ariadne Grillo Pacheco

September/2007

Advisors: Rosa Maria Meri Leão
Edmundo de Souza e Silva

Department: Computer and System Engineering

The deployment of a high quality video-on-demand service over the Internet is a challenging task. Video on demand is a base technology for many important applications such as distance learning. The main objective of this work is to model a video on demand service based on the RIO Multimedia Storage System. The RIO Multimedia Storage System is currently being used by the CEDERJ distance learning course. Several experiments were performed using the Tangram-II modeling tool. Metrics analyzed include the number of clients served by the system in order to obtain a given level of performance. We validate our model by comparing results obtained in a real environment with those computed by the model.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

Buffer	:	Região de memória temporária utilizada para escrita e leitura de dados.
Canal	:	Meio através do qual trafegam os pacotes (*Link*).
FEC	:	Mecanismo de Correção de Erros (*Forward Error Correction*).
RIO	:	Operações de entrada e saída aleatórias (*Randomized I/O*).
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

Algo só é impossível até que alguém duvide e acabe provando o contrário. *Albert Einstein*

1.1 Motivação

O rápido aumento na popularidade dos arquivos de mídia contínua na Internet se deve ao crescimento da velocidade das redes de acesso à Internet e também ao desenvolvimento e aumento do número de novas aplicações multimídia, inclusive as online, como educação a distância, programas de rádio e TV, entre outros.

Três características chave das mídias contínuas são a **alta necessidade de banda**, as **restrições de tempo real na entrega dos dados** multimídia e a **possibilidade de acesso parcial ou interativo** a estas mídias. Com interativo, queremos dizer que o usuário é capaz de efetuar operações do tipo VCR, utilizadas em videocassetes, ações do tipo pausar, avançar, retroceder e pular para pontos específicos do objeto multimídia.

Hoje, a demanda por este tipo de aplicação teve um aumento significativo, fruto da necessidade de conciliar um grande número de atividades com maior eficácia e eficiência. A educação a distância, por exemplo, é uma forma que o aluno encontra para estudar, podendo se adequar ao seu próprio horário, já que os horários de aulas de cursos presenciais nem sempre são flexíveis ao seu cotidiano.

A ampla expansão e implantação destas aplicações multimídia distribuídas visa dar ênfase renovada às soluções propostas para assegurar o melhor desempenho possível ao transferir dados entre diferentes pontos, sem necessariamente impor o custo e a complexidade das soluções de QoS tradicionais [50]. Esta expansão se deve ao surgimento de redes com maior largura de banda e a redução do preço dos discos, o que viabiliza a projeção e a implantação de servidores multimídia com custo mais baixo. É neste contexto que se faz necessário o estudo e a análise de novas técnicas para verificar o atendimento da qualidade exigida por estas aplicações. A modelagem do sistema visa estudar cenários que ainda não são possíveis de serem estudados num ambiente real.

1.2 Contribuições e Objetivo

O objetivo deste trabalho é a modelagem de um serviço de vídeo sob demanda em uma rede de alta velocidade com o desenvolvimento de modelos nos quais são avaliadas diversas opções de arquitetura para fornecimento de um serviço VoD (*Video on Demand*) distribuído, comparando os resultados dos modelos com testes em ambiente real, visando a validação e parametrização dos mesmos. O foco é analisar e avaliar várias configurações do servidor multimídia RIO, desenvolvido no laboratório LAND (Laboratory for modeling, analisys and development of networks and computing systems), usando como rede de suporte uma rede *Gigabit*. O servidor RIO é atualmente usado no curso de ensino a distância de computação da Fundação CECIERJ(Centro de Ciências do Estado do Rio de Janeiro)/Consórcio CEDERJ (Centro de Ciências e Educação Superior a Distância do Estado do Rio de Janeiro).

A rede *Gigabit* usada, consiste em uma parceria entre a RNP - Rede Nacional de Ensino e Pesquisa [9] e o CPqD (Centro de Pesquisa e Desenvolvimento em Telecomunicações) [2], com recursos financeiros do FUNTTEL (Fundo para o Desenvolvimento Tecnológico das Telecomunicações) e apoio da FINEP (Financiadora de Estudos e Projetos) [6] e faz parte do ambiente usado pelo projeto DIVERGE - Subprojeto do Projeto GIGA [8].

O Projeto Giga consiste na implementação e uso de uma rede óptica experimental voltada para o desenvolvimento de tecnologias de rede óptica, aplicações e serviços de telecomunicação associados a tecnologia IP e banda larga. Também prevê a transferência de tecnologia a empresas brasileiras.

O Projeto DIVERGE, no qual se insere este trabalho, versa sobre transmissão de mídia contínua em tempo real, tendo como principal objetivo estudar problemas inerentes à distribuição de vídeo e voz, como teleconferência e distribuição de vídeo em larga escala, utilizando servidores multimídia com enfoque à aplicações em educação.

Este trabalho foi realizado com o envolvimento das seguintes instituições: Universidade Federal do Rio de Janeiro (UFRJ), Universidade Federal Fluminense (UFF), Fiocruz, onde o *switch* giga foi alocado dentro do Canal Saúde e UFMG, que não está conectada à Rede Giga, mas faz parte do projeto através da Internet. A Rede Giga pode ser considerada como uma VPN (*Virtual Private Network* ou Rede Virtual Privada) que interconecta *switches* giga da UFRJ, UFF e Fiocruz. Na Figura 1.1 pode ser vista a arquitetura da Rede Giga com as instituições participantes. Conectados a esta rede ficam o servidor de vídeo e as estações clientes.

Outro ponto importante é a análise quantitativa da inserção de nós de armazenamento em redes de alta velocidade e a análise qualitativa de nós de armazenamento em redes de baixa velocidade, como é o caso da Internet. Avaliaremos o quanto esta inserção pode ajudar no ganho em número de clientes e no desempenho geral do sistema.

As principais contribuições deste trabalho são:

- Modelo do serviço de VoD distribuído oferecido pelo servidor RIO para usuários seqüenciais;
- Modelo do serviço de VoD distribuído oferecido pelo servidor RIO para usuários interativos;
- Parametrização e validação dos modelos com testes em ambiente real.

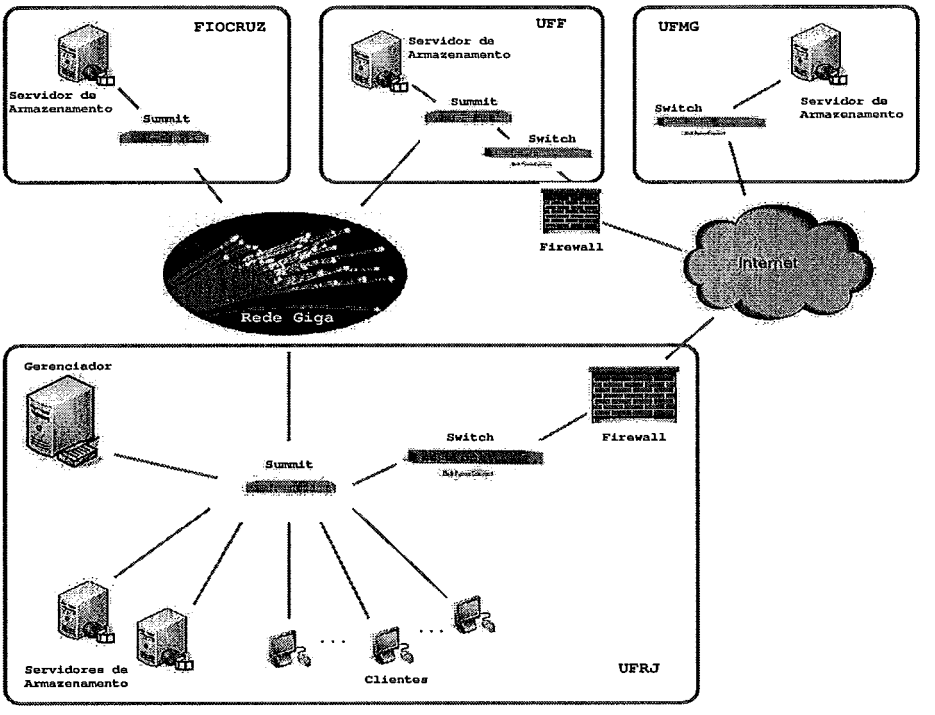


Figura 1.1: Arquitetura Completa da Rede

1.3 Organização da Dissertação

Esta dissertação está organizada como segue. No capítulo 2 fazemos uma revisão geral dos trabalhos relacionados, simplesmente citando as questões mais relevantes sobre aplicações multimídia. A intenção não é fornecer uma abordagem completa do assunto encontrada na literatura, pois os trabalhos desenvolvidos na área são inúmeros e só estamos interessados no que é mais relevante para este trabalho. No capítulo 3 descrevemos as características do servidor multimídia RIO e seus componentes, falando um pouco sobre cada um dos objetos presentes nesta arquitetura. No capítulo 4 apresentamos o ambiente de modelagem do trabalho, discorrendo sobre os modelos gerados para simulação através do uso da ferramenta Tangram-II, uma breve descrição da ferramenta e a metodologia utilizada para a realização dos testes feitos em ambiente real para validação dos modelos de simulação desenvolvidos. No capítulo 5 apresentamos os resultados obtidos nas simulações e nos testes em ambiente real. E finalmente no capítulo 6 apresentamos nossas conclusões e sugestões para trabalhos futuros.

Capítulo 2

Aplicações Multimídia

Que ninguém se engane: só se consegue a simplicidade através de muito trabalho. *Clarice Linspector*

Diversos tipos de aplicações multimídia vêm sendo desenvolvidos continuamente graças ao avanço na área tecnológica, que se expande para diversas áreas de conhecimento como, por exemplo, Medicina, Biologia, Engenharia e Educação. Estas aplicações integram diferentes mídias, tais como textos, gráficos, vídeo e áudio, que contêm os dados necessários para os serviços que, em conjunto com informação adicional, permitem a classificação e a apresentação destes dados.

Alguns dos fatores que influenciam a expansão de tais aplicações são: o aumento da capacidade de processamento, o aumento da capacidade de armazenamento, o surgimento de mídias digitais como o CD-ROM e o DVD, tudo isso aliado a custos mais baixos de equipamentos eletrônicos. No entanto, há cerca de 15 anos [7], as aplicações multimídia se limitavam a aplicações *stand alone*, a redes locais ou a CD-ROMs. Com o aumento da velocidade das redes de acesso à Internet, as aplicações multimídia de acesso remoto ganharam novo impulso e atingem um número de usuários cada vez maior.

2.1 Conceitos Básicos e Definições

Mídias contínuas sobre redes com a política de melhor esforço, como a Internet, constituem uma área desafiadora por apresentarem diversas questões que impactam a qualidade destas aplicações, como variações significativas no atraso, na vazão, na largura de banda e a perda de pacotes, devido a congestionamentos e às características heterogêneas da rede. É por este caráter desafiador e seu alto grau de importância que este tópico de pesquisa vem sendo amplamente explorado.

De acordo com [25], um sistema multimídia pode ser dividido em três componentes básicos: o servidor multimídia, as técnicas de compartilhamento de recursos para a transmissão de dados através da rede e os métodos para melhorar a utilização da banda da rede e dos *buffers*.

2.1.1 O Servidor Multimídia

O servidor multimídia é o componente chave de um sistema multimídia distribuído, onde seu desempenho, em termos de quantos usuários podem ser suportados simultaneamente, afeta o custo geral do sistema e pode ser decisivo para determinar sua viabilidade econômica, um importante parâmetro para o desenvolvimento do sistema, o que faz com o que o estudo deste tópico seja amplamente abordado [48, 42, 15].

Este componente é composto por um ou mais processadores, memória e um determinado número de discos rígidos, que são utilizados para o armazenamento dos objetos multimídia. Estes objetos são compostos de blocos que são consumidos pelos usuários. A forma mais comum de se implementar a transmissão de blocos é a CBR (*constant bit rate*), onde a taxa de transmissão disponibilizada para o usuário é fixa. Para suavizar a chegada variável dos blocos e evitar interrupções na exibição do objeto multimídia é necessário utilizar *buffers* no usuário. E para enviar os blocos para o usuário, o servidor primeiro precisa recuperá-los da memória principal, o que faz com que seja necessário ter um *buffer* do lado do servidor também.

O conjunto de setores do disco que o servidor envia para o usuário é aqui chamado de bloco de dados, onde cada bloco é armazenado no *buffer* do usuário e é então consumido por este. Enquanto um usuário decodifica e exibe um determinado bloco de dados, outros usuários podem ser servidos pelo sistema. Desta forma o servidor pode multiplexar a banda dos discos entre vários usuários, que são então servidos concorrentemente.

Quando um usuário faz uma requisição para um objeto e esta requisição é admitida pelo sistema, de acordo com seu controle de fluxo, o servidor começa a enviar os dados do objeto. O usuário tem que esperar um tempo determinado até que seu *buffer*, chamado de *playout buffer* esteja preenchido completamente, para só então começar a exibir o objeto. Este intervalo entre o pedido e a exibição do objeto é conhecido como *startup latency*, a latência inicial. A partir do momento que o bloco de dados é recebido e passa a ser consumido, o usuário faz a requisição do próximo bloco. Se um bloco não estiver presente no *buffer* no momento de ser consumido, ocorre o que chamamos de *hiccup*, quando há o congelamento da imagem por causa da interrupção do serviço.

Assumindo que o servidor possua múltiplos discos, os blocos dos objetos multimídia estão de alguma maneira distribuídos entre todos os discos do sistema, dependendo do tipo de armazenamento que o sistema utiliza. A forma mais direta e simples é copiar o objeto inteiro para o mesmo disco, mas isto acarretaria perda de desempenho quando o sistema estiver com alto acesso, onde vários usuários tentariam acessar sempre o mesmo disco, sobrecarregando o mesmo. Para contornar esta questão de desbalanceamento, estratégias mais sofisticadas para distribuir o bloco entre os múltiplos discos do sistema foram propostas. As técnicas mais conhecidas são *Data Striping*, onde os blocos são distribuídos continuamente ao longo dos discos de forma *Round Robin*, e Alocação Aleatória, onde os blocos são distribuídos de forma que tanto os discos quanto a posição que o bloco ocupa nestes discos são escolhidos de forma aleatória. Posteriormente neste capítulo, na seção 2.4, abordaremos em maiores detalhes estas técnicas.

Outras importantes questões devem ser levadas em consideração no projeto de

um sistema multimídia, como por exemplo, o mecanismo de reposição dos objetos, a escalabilidade dos discos e a tolerância a falhas do sistema em geral. A questão da reposição dos objetos multimídia vem a tona pelo fato destes objetos serem muito grandes, o que faz com que o sistema possa chegar em um ponto onde fique sobrecarregado, necessitando que haja uma substituição de objetos antigos por objetos novos. O que acaba levando à segunda questão, que é uma possível reconfiguração dos discos, pois a demanda por espaço pode ser maior do que o disponível. Outra questão crucial é com relação a falhas, a preocupação em manter a integridade e a acessibilidade dos dados do sistema, em especial dos discos. Esta questão é de suma importância porque a queda de um simples disco pode comprometer o funcionamento de todo o sistema, caso o mesmo não possua nenhum mecanismo de recuperação. Para contornar o problema podem ser usadas técnicas de redundância, que serão abordadas em maiores detalhes na seção 2.3.

2.1.2 Transmissão de Informação

Há várias questões relativas a desempenho para transmissão de fluxos contínuos que também devem ser levadas em consideração. Com relação à transmissão dos dados temos que, como a Internet não possui nenhum mecanismo de reserva de banda para transmissão de fluxos de tempo real, é necessário pensar em mecanismos para amenizar problemas como o atraso variável de chegada dos pacotes (*jitter*), causado por possíveis congestionamentos randômicos da rede. Fluxos de tempo real são muito sensíveis a estes atrasos, pois eles degradam severamente a qualidade da reprodução de mídias contínuas. Um mecanismo simples para reduzir o *jitter* é o uso do *playout buffer* no cliente, para que haja balanceamento nas variações na chegada dos pacotes.

Ao lado dos atrasos aleatórios presentes na rede, a perda de pacotes também pode degradar a qualidade do serviço. Um tipo de perda é quando pacotes são descartados devido ao congestionamento nos roteadores. Além disso, o atraso de um bloco pode ser tão grande, que ele não estará presente no receptor no momento de sua exibição, o que caracteriza também um cenário de perda.

A retransmissão é o método mais usual para fazer a recuperação da perda de um pacote. Porém, as características da rede e os requisitos de qualidade que a aplicação multimídia exige devem ser muito bem analisados. A retransmissão só é possível se há tempo suficiente para que o pacote seja pedido novamente e chegue antes do tempo de sua exibição, ou seja, o RTT (*Round Trip Time*) deve ser baixo o suficiente para que a retransmissão chegue a tempo [43].

Como outros mecanismos de recuperação de perda, podemos citar os baseados em redundância, os baseados na utilização dos pacotes que já foram recebidos pela aplicação e os baseados na diversidade de caminhos, que pode ter aumentada sua eficiência se usada em conjunto com os outros mecanismos.

2.1.3 Técnicas de Compartilhamento de Recursos

Em sistemas VoD (*Video on Demand*), a largura de banda do servidor é um dos principais fatores limitantes para o atendimento de grandes quantidades de clientes. Visando contornar esta limitação, várias técnicas foram propostas com o intuito de otimizar o acesso ao servidor. Antes de mais nada, convém salientar que os sistemas convencionais de VoD utilizam fluxos *unicast*, o que limita o número de clientes que podem ser atendidos, devido à grande exigência de banda para transmissão. Ou seja, os provedores de mídia convencionais servem cada usuário com um fluxo separado dos demais, o que faz com que os recursos do sistema, principalmente a banda da rede, possam se esgotar rapidamente.

Uma abordagem comum para lidar com este problema é fazer com que vários usuários compartilhem o mesmo fluxo e os mesmos *buffers*, técnica conhecida como compartilhamento de recursos, reduzindo assim a demanda por banda da rede, banda do disco e espaço de armazenamento [25]. A idéia é reduzir a demanda pela banda da rede, acesso ao disco e espaço de armazenamento, ao mesmo tempo em que o sistema fornece QoS para os usuários.

2.2 Arquiteturas para Aplicações Multimídia

De um modo simplificado, um sistema de armazenamento do tipo vídeo sob demanda pode ser visto como um problema de sistemas de arquivos distribuídos e, sendo assim, os usuários de um sistema de vídeo sob demanda têm duas necessidades básicas: localizar material relevante e recuperá-lo para posteriormente poder visualizá-lo. Nas subseções seguintes discorreremos sobre algumas das arquiteturas mais usadas para sistemas de distribuição de vídeo.

2.2.1 VoD - Video on Demand

Um sistema VoD consiste em um servidor de vídeo com material armazenado digitalmente em dispositivos de armazenamento de alta capacidade e uma rede de comunicação conectando os usuários ao servidor. Este sistema pode ser visto como um sistema cliente-servidor, onde os clientes constituem os usuários que acessam os vídeos armazenados no servidor, como ilustra a Figura 2.1.

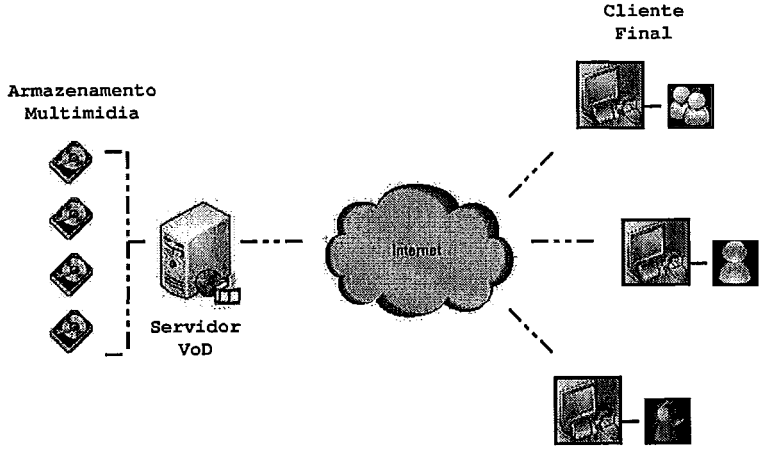


Figura 2.1: Arquitetura VoD

O serviço de Vídeo sob Demanda, que a partir de agora trataremos simplesmente por VoD, oferece aos clientes uma vasta gama de vídeos que eles podem escolher e assistir no momento em que acharem mais oportuno.

Nas propostas típicas para sistemas VoD, os usuários são servidos individual-

mente ao alocar e dedicar um canal de transmissão e um conjunto de recursos do sistema para cada usuário. A arquitetura VoD pode ser dividida em dois grandes componentes: o servidor e a interface com o usuário. O servidor fica responsável por controlar todo o funcionamento do sistema e toda a demanda de pedidos, por receber, processar, enviar os dados para os usuários e também por estabelecer, manter e modificar os fluxos usados para entregar os pedidos [10]. A interface com o cliente compreende um dispositivo que pode ter várias formas e tamanhos, desempenhando uma variedade de funções, como enviar, receber e processar as mensagens de controle enviadas para o cliente e também receber, decodificar e exibir o vídeo. Uma importante consideração na avaliação de um sistema VoD é o custo desta interface com o usuário, ou seja, se o preço do dispositivo é viável.

Um dos problemas de sistemas VoD é a centralização da carga, ou seja, um problema de escalabilidade, já que todos os pedidos têm que ser processados pelo mesmo servidor. Por exemplo, se possuirmos um canal com capacidade 1Gbps e cada vídeo precisar de uma banda de aproximadamente 1.5Mbps (um filme codificado em MPEG-2, com tela de 320x240 *pixels*, por exemplo), o canal seria capaz de servir pouco mais de 650 clientes, supondo que tenhamos 100% de utilização do canal, o que torna o sistema economicamente inviável, para uma quantidade tão baixa de clientes. Outra desvantagem é o sistema possuir um único ponto de falha, já que se o servidor falhar todo o sistema deixa de funcionar.

2.2.2 DVoD - Distributed Video on Demand

Um sistema de vídeo sob demanda distribuído, ou simplesmente DVoD, consiste de um conjunto de dispositivos de armazenamento conectados por redes WAN (*Wide Area Network*) ou MAN (*Metropolitan Area Network*) de alta velocidade, como mostra a Figura 2.2.

Sistemas DVoD surgiram para resolver o problema de escalabilidade citado anteriormente com relação a sistemas VoD, ou seja, o problema de capacidade limitada para os fluxos [17], já que desta forma o armazenamento passa a estar distribuído, se-

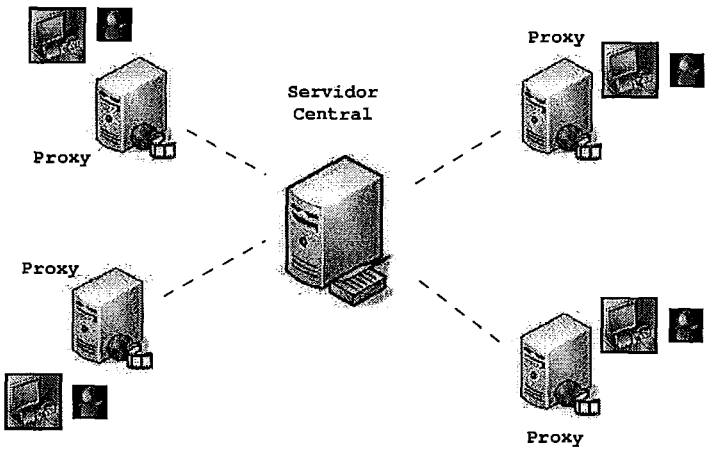


Figura 2.2: Sistema de Vídeo sob Demanda Distribuído

parando a parte de controle do conteúdo propriamente dito. As principais vantagens que esta arquitetura apresenta, conforme visto em [11], diz respeito à redução de tráfego na rede do servidor central (já que ele não terá que fazer a entrega de todos os blocos de dados pedidos), melhoria no tempo de resposta às requisições, diminuição do *jitter* e aumento da confiabilidade do sistema, já que se algum dos servidores de armazenamento falhar, existem maneiras de recuperação, como a replicação dos blocos, que será discutida em seções posteriores.

A seguir fazemos uma revisão das propostas na literatura para sistemas DVoD.

Proxies

Devido ao gargalo que o serviço VoD enfrenta para conectar os clientes ao servidor, por causa de seus altos requisitos de banda, foi necessário distribuir a carga do lado do servidor em vários servidores distribuídos, conhecidos por *proxies* [55, 56, 36]. Em [56], foi desenvolvido uma técnica chamada *Video Staging*, que tem como idéia principal fazer o pré-armazenamento (*prefetching*) de uma quantidade de dados e armazená-los *a priori* nos *proxies*. Em [55], a idéia é também armazenar o prefixo dos filmes, a fim de diminuir a latência inicial. Em [36], a idéia é que seja feita uma seleção não contígua de blocos intermediários, para auxiliar operações de VCR, como pausar, avançar e retroceder, por exemplo.

Independente da forma de armazenamento escolhida, pode-se obter um aumento significativo da capacidade total do sistema e da qualidade do serviço para o usuário com o uso de *proxies* [11]. Entretanto, o problema de termos um ponto único de falha ainda permanece, uma vez que é preciso recorrer ao servidor VoD quando um bloco requisitado por um cliente sendo servido não se encontra no *proxy*.

CDN - Content Distribution Network

As redes de distribuição de conteúdo, ou CDNs, geralmente fornecem escalabilidade ao distribuir diversos servidores ao longo da Internet próximos aos seus clientes. Estas soluções são baseadas na disposição de equipamento dedicado ao longo da rede.

A Figura 2.3 mostra um ambiente típico de uma CDN, onde os servidores replicados estão localizados ao longo das redes nas quais os usuários estão conectados. Em tal ambiente, o conteúdo multimídia baseado na requisição dos usuários é recuperado do servidor de origem e o usuário é servido pelo conteúdo que está no servidor mais próximo a ele.

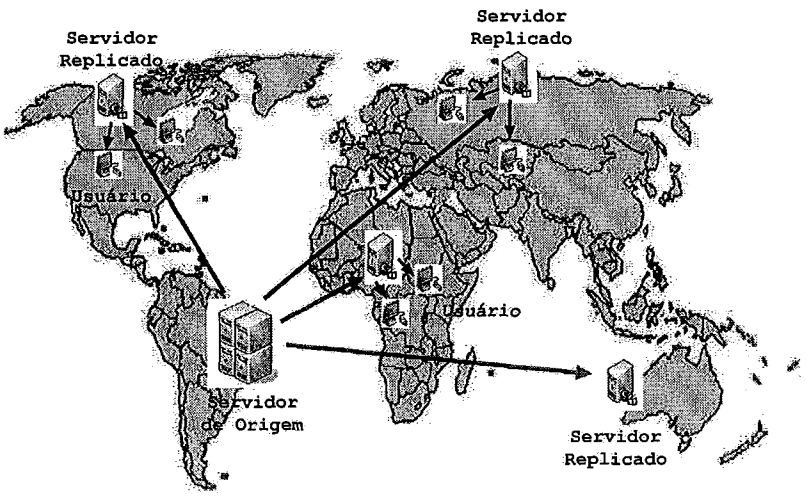


Figura 2.3: Arquitetura de uma CDN

O maior exemplo de tal solução é o operado por empresas como a Akamai [1] e a Digital Island [4], que possuem dezenas de milhares de servidores em funcionamento

ao redor do mundo. As CDNs foram anunciadas como a solução definitiva para distribuição de conteúdo multimídia na web, mas apesar do grande *marketing*, estudos comprovaram que o balanceamento de carga destas CDNs está longe do ótimo [33]. Por exemplo, uma requisição de um *browser* para um determinado item é roteado para uma boa réplica, onde boa quer dizer que este item é servido para o cliente mais rapidamente se comparado ao tempo que ele levaria para recuperar o mesmo do servidor de origem. Em resumo, o roteamento destas CDNs apenas evita que uma requisição seja direcionada para o pior servidor, ao invés de direcioná-lo para o melhor deles.

Em geral, informação estática sobre localizações geográficas e conectividade de rede não é suficiente para escolher uma boa réplica, ou seja, um bom servidor. Ao invés disto, uma CDN deve incorporar informação dinâmica sobre as condições da rede e das réplicas, para rotear as requisições de forma que seja feito um balanceamento da carga.

Uma CDN é efetivamente uma coleção de *caches* amplamente dispersas, com duas diferenças cruciais. A primeira é que as *caches* são potencialmente populadas proativamente, a fim de prever uma futura demanda [27]. Segundo, que as *caches* são coordenadas por um mecanismo que roteia as requisições do cliente para uma boa *cache*. Estas características que as diferenciam de um servidor VoD com Proxies.

2.2.3 Peer-to-Peer

Para resolver a limitação com relação ao número de servidores que podem ser adicionados ao sistema, que arquiteturas DVoD ainda não são capazes de resolver, faz-se necessária a utilização de outra técnica que possa adicionar capacidade ao sistema de acordo com o aumento do número de clientes.

A arquitetura Peer-to-Peer (P2P), mostrada na Figura 2.4, foi proposta em [31, 40, 51] para resolver o problema de escalabilidade de serviços VoD para requisições de um grande número de clientes. Uma importante vantagem de ambientes P2P é o fato dos nós (chamados *peers*) tomarem para si o custo dos recursos computacionais e de

armazenagem, de tal forma que a carga do servidor seja reduzida e a escalabilidade geral aumente. Porém, como o comportamento dos clientes é imprevisível em redes P2P, o mecanismo de recuperação de falhas se faz necessário de acordo com a partida dos clientes do sistema

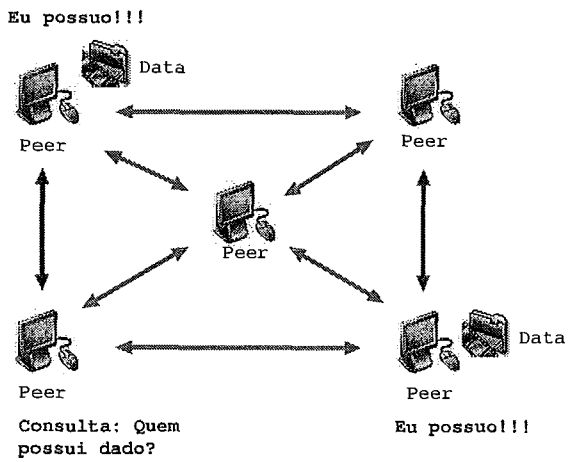


Figura 2.4: Arquitetura Peer-to-Peer

O melhor exemplo de uma arquitetura desse sistema colaborativo é o BitTorrent [16], um sistema que se tornou extremamente popular como uma maneira de distribuir conteúdos muito procurados. O BitTorrent divide o arquivo em pequenos blocos e assim que um nó baixa um desses blocos do servidor original ou de outro nó, este novo nó que possui o bloco passa a se comportar como um servidor para este bloco em particular, contribuindo assim com recursos para servir o bloco. Ao leitor interessado em uma descrição mais detalhada do BitTorrent veja [41]. Mas apesar do enorme potencial e popularidade deste tipo de esquema ele ainda sofre de um número de ineficiências que podem comprometer parte de seu desempenho geral. Tais ineficiências são mais sentidas entre grandes populações heterogêneas durante as chamadas *flash crowds*, que são um enorme e repentino aparecimento de tráfego que geralmente leva ao colapso o servidor afetado [30].

Porém, para sistemas de ensino à distância as arquiteturas P2P são pouco adequadas, devido à transiência dos nós e do conteúdo. Um sistema deste tipo necessita que seu conteúdo, ou seja, todas as aulas dos cursos, estejam sempre disponíveis com a mesma qualidade de serviço. Os sistemas P2P, pelo contrário, não oferecem ne-

nhum tipo de garantia com relação à disponibilidade em tempo integral do conteúdo específico. Portanto, pelo discorrido até agora, sistemas DVoD são os mais indicados para o objetivo deste trabalho.

No capítulo seguinte apresentamos o sistema DVoD que é utilizado neste trabalho.

2.3 Mecanismos de Replicação

A replicação de dados e serviços em diferentes computadores e em diferentes redes aumenta a disponilibidade do serviço, a tolerância a falhas e a qualidade de serviço (QoS) dos sistemas multimídia distribuídos [18]. Por exemplo, quando um usuário faz uma requisição de um dado que não está disponível no servidor momentaneamente, um objeto gerente no servidor faz o pedido de uma cópia do dado para o servidor de armazenamento mais próximo do usuário, de forma transparente. Assim, com a disponibilidade da réplica no servidor de armazenamento, a garantia que os usuários podem continuar sua apresentação em uma situação de falha é significativamente maior do que sem réplica, como visto em [38]. Naquele trabalho são discutidas algumas questões de projeto e implementação de um mecanismo de replicação para um sistema multimídia distribuído, que foi desenvolvido como uma infra-estrutura para compartilhar materiais técnicos de ensino entre grupos de conferência.

Em um mecanismo de replicação é importante identificar o servidor de armazenamento para o qual será feita a cópia, entre um número de servidores, de tal modo que os benefícios para os usuários sejam maximizados e o custo agregado com a manutenção do sistema seja minimizado tanto quanto possível [32].

Um considerável número de trabalhos foi feito no campo de replicação em servidores [19, 13, 35, 44, 54, 57]. Em [44] foi proposto um método para balanceamento de carga e replicação para grandes objetos. Este método se mostrou adequado para sistemas Grid, provedores *Data Warehouse* e provedores de *websites* dinâmicos. O método tenta distribuir a carga de requisição dos objetos igualmente nos servidores

de acordo com suas capacidades, para que a probabilidade de sobrecarga, e conseqüente falha, seja reduzida. Em [54] é proposta uma estratégia de replicação em *proxies* de forma transparente. Define-se uma função que tem como objetivo minimizar o custo total da transferência dos dados para um servidor de acordo com um dado padrão de tráfego, onde dados N servidores para replicação, estamos interessados em descobrir quantas réplicas devem ser criadas e em quais *proxies* replicá-las para obter desempenho ótimo. Em [57] é feito um estudo da replicação ótima em um *cluster* de servidores VoD, que disponibilizam serviços de alta qualidade e alta disponibilidade.

Para identificar os requisitos de replicação temos que analisar as características dos tipos de mídias que serão utilizadas pelas aplicações. Normalmente, define-se como replicação a criação de novas cópias do objeto. Porém, esta redundância pode se dar através de replicação parcial ou total dos objetos.

A replicação parcial é feita ao dividir-se um objeto em b blocos e replicar uma quantidade (escolhida de acordo com a necessidade) em tantos servidores quanto possíveis, ou seja, não será o objeto todo que será replicado, apenas uma parte dele. A eficiência desta técnica aumenta conforme aumenta-se o número de servidores, pois a probabilidade do objeto inteiro estar disponível entre todos os servidores aumenta. Assim, a técnica é mais penalizada à medida que o número de servidores disponíveis diminui. Em [34] é feita uma análise sobre replicação parcial dos blocos, onde há uma discussão em como decidir qual o tipo de replicação será usada, se a parcial ou a completa. E se a parcial for a escolhida, o número de blocos, em proporção, que devem ser replicados.

Com a replicação total temos que o objeto terá todos os seus blocos replicados, ou seja, o objeto será integralmente replicado, o que garante que o mesmo seja sempre recuperado, mesmo que um servidor de armazenamento falhe. A desvantagem desta técnica é o custo de armazenamento, pois a cada réplica o custo de armazenamento cresce com proporção de 100% o tamanho do objeto. Porém, a disponibilidade e a confiabilidade de que um objeto estará presente para o usuário aumentam.

Quando um servidor falha, a carga que até o momento vinha sendo suprida

por este deve ser absorvida pelo resto do sistema. A idéia por trás disto é que se esta carga for consistentemente distribuída por todos os servidores que compõem o sistema, evita-se que o sistema seja sobrecarregado e que a partir daí haja um desencadeamento de falhas e, conseqüentemente, a redução da qualidade de serviço.

Em resumo, a decisão de qual nível de replicação usar depende das características do sistema e da necessidade da aplicação utilizada.

2.4 Políticas de Armazenamento

Levando-se em conta que os objetos multimídia são muito grandes para serem armazenados na memória principal do sistema e sabendo-se que eles precisam ser recuperados do disco à medida que as requisições para os mesmos são encaminhadas, viu-se a necessidade de armazenar esses objetos em locais apropriados, como discos paralelos, que também são necessários para melhor aproveitar a banda do sistema, para melhorar a capacidade de armazenamento antecipado para os vários usuários e para melhorar o suporte a aplicações de alto desempenho. Sendo assim, os blocos dos vários objetos são distribuídos entre todos os discos que compõem o sistema. A forma que seria considerada mais simples seria a de armazenar todo o objeto no mesmo disco, o que, por um lado, facilitaria a manutenção do sistema, mas, por outro, acarretaria um sério problema de sobrecarga caso um determinado objeto seja altamente requisitado, como acontece no caso de um vídeo popular, por exemplo. Assim, um alto grau de desbalanceamento de carga pode ocorrer, limitando o número de usuários que podem ser servidos pelo sistema.

Várias são as formas de armazenamentos dos dados nos discos dos servidores multimídia, onde o método mais comumente utilizado é a divisão dos dados em blocos para que estes possam ser lidos, armazenados no *buffer* do servidor e a seguir enviados para que o usuário possa executá-lo [29, 39, 49].

Duas das técnicas mais conhecidas na literatura são descritas a seguir.

2.5 Data Striping

Esquema onde os objetos multimídia são divididos em blocos de dados de tamanhos fixos e distribuídos, de forma organizada e seqüencial, em múltiplos discos de forma *Round Robin*, onde os blocos consecutivos são armazenados em discos consecutivos, como ilustra a Figura 2.5.

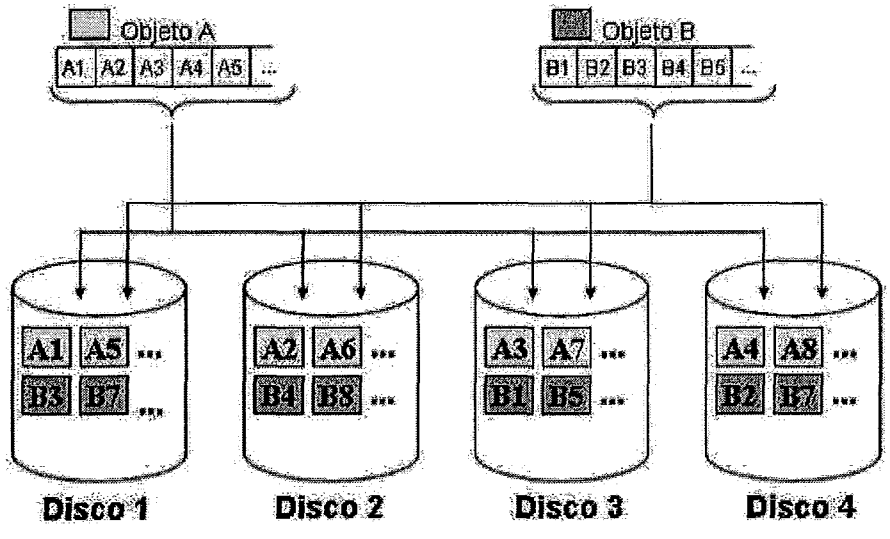


Figura 2.5: Data Striping

Sua abordagem tradicional é o servidor recuperar um bloco de dados em cada ciclo de tempo, para cada um dos fluxos ativos. Este bloco é então entregue e consumido pelo cliente no próximo ciclo, enquanto o servidor recupera o próximo bloco do disco. Esta divisão dos objetos feita de forma igualitária garante que todos os dispositivos serão utilizados igualmente a todo o tempo [15].

A técnica de *Data Striping* é geralmente projetada para *workloads* com determinadas características como, por exemplo, padrões de acesso seqüencial com requisitos CBR (*Constant Bit Rate*).

2.6 Alocação Aleatória

Neste tipo de política os dados são aleatoriamente distribuídos em discos também escolhidos aleatoriamente, como mostra a Figura 2.6.

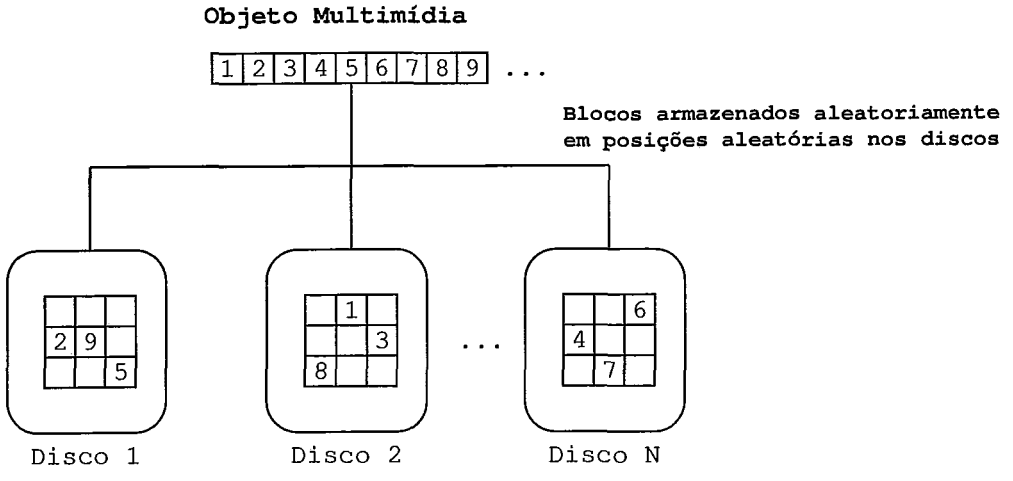


Figura 2.6: Alocação Aleatória dos Dados

A técnica de alocação aleatória tem a vantagem de mapear os padrões de acesso dos diferentes tipos de aplicações multimídia dentro do mesmo padrão de acesso aleatório na camada física, o que simplifica o problema de caracterização de tráfego para controle de admissão, já que ele não precisa se preocupar em como os diferentes padrões de acesso afetam a distribuição da carga entre os diferentes discos do sistema, por possuir esta característica randômica. Assim, é mais fácil suportar aplicações multimídias heterogêneas, ou seja, aplicações de diversos tipos e com características diferentes, no mesmo sistema de armazenamento, porque todas as aplicações geram a mesma distribuição aleatória das requisições para os discos do sistema [46].

Esta é a técnica usada no sistema RIO, que é utilizado neste trabalho e é descrito no capítulo seguinte.

Capítulo 3

Sistema Multimídia RIO

Logo que, numa inovação, nos mostram alguma coisa de antigo, ficamos sossegados. *Friedrich Nietzsche*

O objetivo deste capítulo é apresentar a arquitetura e as características do sistema multimídia RIO (*Randomized I/O*), que é utilizado neste trabalho, e uma rápida descrição de seus componentes. Nas próximas seções abordaremos os objetos da arquitetura do sistema.

3.1 Introdução ao Servidor RIO

O sistema RIO (Randomized I/O), que a partir de agora trataremos simplesmente por RIO, foi projetado para gerenciar um sistema de armazenamento de discos paralelos com suporte à entrega de dados em tempo real com garantias estatísticas de atraso em um ambiente multiusuário. Foi projetado para suportar cargas como mundos virtuais interativos 3D, visualizações interativas e outras, bem como a execução de áudio e vídeo. Foi desenvolvido, a princípio, como um sistema de armazenamento para o projeto *Virtual World Data Server* (VWDS) do Departamento de Ciência da Computação da UCLA (*University of California, Los Angeles*) [47]. O primeiro protótipo foi implementado em uma máquina SMP (máquinas multiprocessadas), que está operacional desde 1997 e foi utilizado para exibições

de vídeos MPEG, simulações em 3D e, posteriormente, aplicações para visualizações científicas em tempo real e realidade virtual para área médica. Em seguida foi implementada uma versão para um *cluster* de PCs desenvolvida em C++, onde disponibilizou-se um cliente de visualização de vídeos MPEG, chamado *riomtv*, e um cliente interpretador de comandos para administração do sistema, o *riosh*.

A partir do protótipo desenvolvido na UCLA, o Laboratório de Análise, Modelagem e Desenvolvimento de Redes e Sistemas de Computação (LAND) desenvolveu uma nova versão do servidor RIO, que inclui várias novas funcionalidades. Foi elaborado um novo visualizador de vídeos, o *RioMMClient*, que possui mais funcionalidades que o antigo *riomtv*. Este novo visualizador possui uma interface gráfica que propicia ao usuário interagir com o vídeo, podendo avançar, retroceder ou pausar, juntamente com o fato de possuir um módulo de sincronização com transparências, que é utilizado, por exemplo, em uma aula onde o professor exibe o vídeo acompanhado com as transparências que está utilizando para a explicação, como aparece à direita da Figura 3.1. Deste modo o usuário pode acompanhar as transparências que estão sendo exibidas pelo professor durante a aula e caso salte para algum ponto do vídeo a transparência acompanha o movimento.

Em [21] foram adicionadas funcionalidades como *buffers* no servidor para permitir a gerência dos dados que foram solicitados de tal forma que o *jitter*, causado pela leitura dos discos e transmissão pela rede, fosse minimizado e a adição de um novo controle de admissão para novos usuários do sistema, além de um módulo capaz de extrair diversas medidas de desempenho, como por exemplo, tempo médio de leitura de um bloco de dados nos discos.

No RIO, todos os tipos de mídias são tratados igualmente como um objeto e armazenados da mesma forma. Estes objetos são divididos em blocos de tamanhos iguais e armazenados aleatoriamente nos vários discos do sistema, como descreveremos na seção 3.3. O tamanho dos blocos, como vários outros parâmetros do sistema, é definido no arquivo de configuração do sistema.

A Figura 3.2 ilustra uma visão global da arquitetura do sistema. O RIO fornece o armazenamento de dados para múltiplos usuários que utilizam diferentes tipos de

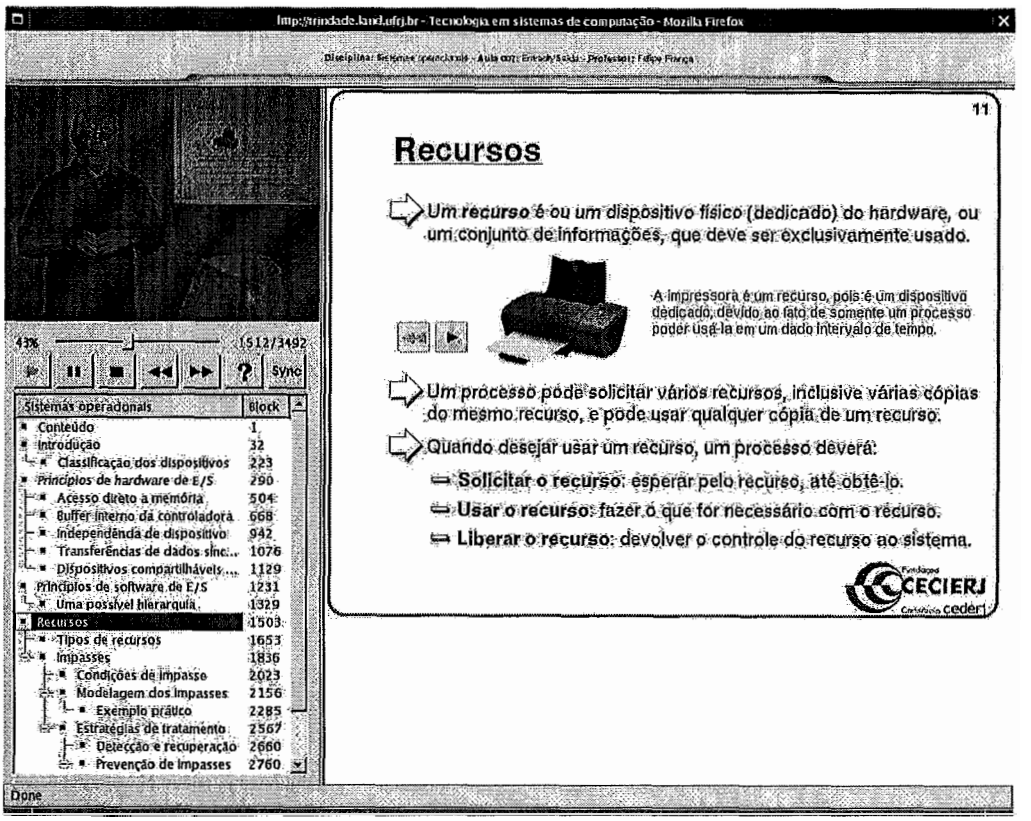


Figura 3.1: RioMMClient

aplicações multimídias.

Os clientes estão sempre enviando informações telemétricas para o servidor, que são usadas para determinar qual parte do objeto multimídia precisa ser enviado para o cliente em um futuro imediato. No caso de um vídeo, informação telemétrica consiste basicamente da posição temporal atual do vídeo e os comandos dos usuários para funcionalidades VCR, tal como pausar, avançar ou retroceder.

A Figura 3.3 mostra uma visão básica da arquitetura do sistema RIO. Nesta visão temos que o RIO é composto por um único nó servidor, que chamaremos também de nó gerenciador, nós de armazenamento e clientes. A comunicação entre os clientes e o servidor, incluindo os nós de armazenamento, é feita através dos protocolos TCP (*Transmission Control Protocol*), usado para troca de mensagens de controle, e UDP (*User Datagram Protocol*), usado para o envio dos blocos de dados do vídeo. Como pode ser observado, os blocos dos dados são enviados diretamente dos nós

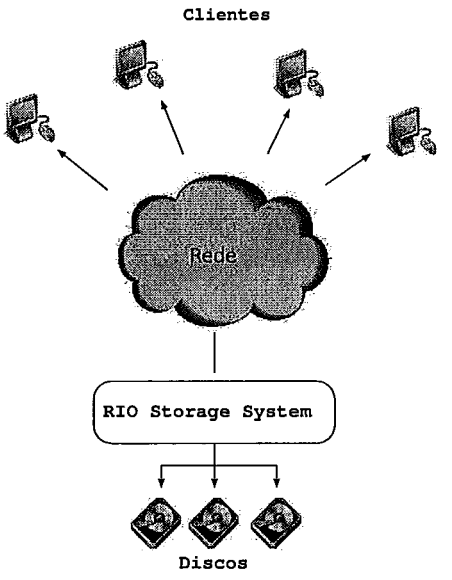


Figura 3.2: Arquitetura do Sistema multimídia

de armazenamento para os clientes. O nó gerenciador fica com o encargo de fazer a admissão dos novos clientes para o sistema, de controlar os blocos dos objetos, possuindo informação de onde cada um deles se encontra, ou seja, o gerenciamento dos metadados dos objetos, o recebimento das requisições para estes blocos, o estado de cada uma das filas dos nós de armazenamento e, conseqüentemente, a escolha de qual possui a menor fila para encaminhar os pedidos. Desta forma o servidor pode decidir, *a priori*, para qual nó enviar o seu pedido, fazendo assim o balanceamento da carga.

As características de tempo-real de mídias contínuas afetam o desenvolvimento de aplicações multimídias em diferentes níveis, como armazenamento dos dados, gerenciamento das informações, gerenciamento do processador e da memória das máquinas envolvidas no sistema, a comunicação dos dados, a apresentação dos dados, entre outros. Discorreremos brevemente sobre alguns deles.

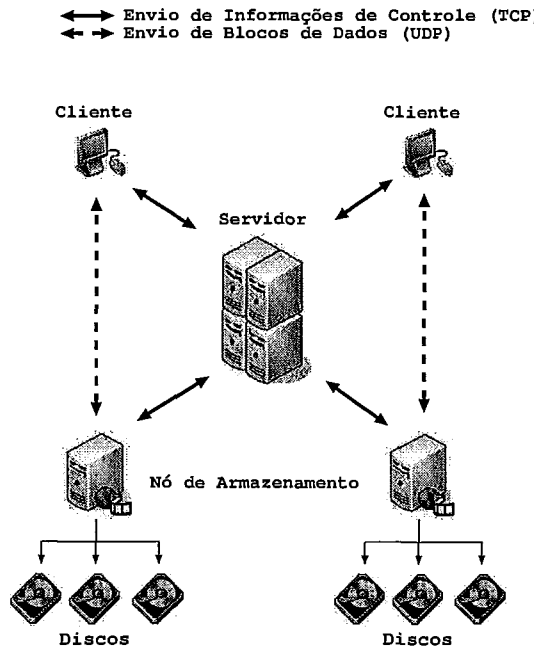


Figura 3.3: Arquitetura básica do RIO

3.2 Componentes do Sistema RIO

Nesta seção apresentamos os componentes da arquitetura do sistema RIO. Estes componentes podem ser visualizados na Figura 3.4, onde é apresentada uma detalhada descrição sobre a versão original do servidor RIO.

3.2.1 Nó Servidor - Gerenciador

O nó servidor (*Server Node*) é estruturado de forma a conter diretórios, arquivos de configuração do sistema, informações sobre os usuários e arquivos contendo objetos armazenados e discos que estão sendo utilizados.

É o nó servidor quem controla as informações de atendimento aos clientes e faz o gerenciamento global do sistema. O RIO usa os metadados para fazer o mapeamento de um bloco que foi requisitado por um cliente no nó de armazenamento que será responsável por servir este determinado bloco. Os metadados contêm informação sobre os objetos e os blocos do sistema, tais como tamanho do bloco, número de

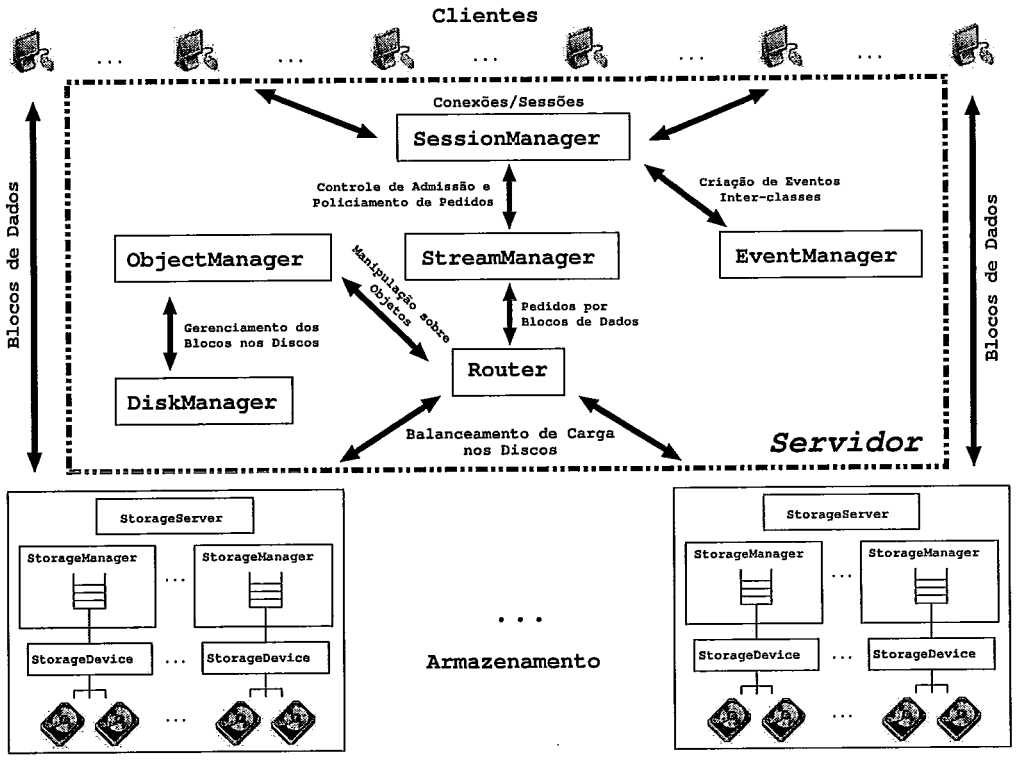


Figura 3.4: Componentes do Servidor RIO

blocos de dados de um objeto, número máximo de objetos armazenados e um ponteiro para o primeiro bloco de dados de cada região do espaço de armazenamento do sistema.

Gerenciador de Sessão (*SessionManager*)

O *SessionManager* é responsável por criar um ponto de conexão para que os clientes acessem o RIO. A cada nova conexão uma *thread* é disparada para atender as requisições do novo cliente, criando assim uma nova sessão para fazer o atendimento de todos os pedidos. Como pode ser visto na Figura 3.4, cada pedido do cliente pode executar um método do próprio *SessionManager*, do *StreamManager*, do *ObjectManager* ou do *Router*.

Gerenciador de Fluxos (*StreamManager*)

O *StreamManager* é responsável pelo gerenciamento dos fluxos que já foram abertos e pela abertura dos novos, que é quando ele executa o controle de admissão descrito na seção 3.4.

Os pedidos feitos por cada fluxo ativo no sistema são encaminhados para o objeto roteador para que este os distribua entre os servidores de armazenamento (*Storage-Servers*) para serem atendidos. Porém, nem todas as requisições têm garantias de atendimento no momento de sua chegada, devido ao policiamento de tráfego, descrito na seção 3.5. Todos os fluxos possuem uma fila para armazenar os pedidos que esperam ser atendidos, o que é feito assim que possível graças ao *StreamManager*, que verifica periodicamente os fluxos com pedidos pendentes e os envia ao *Router* priorizando o tráfego de tempo real.

Gerenciador de Objetos (*ObjectManager*)

O *ObjectManager* é responsável por gerenciar os metadados de todos os objetos presentes no servidor. É ele quem efetua as operações sobre os objetos, tais como criação, abertura, fechamento e exclusão.

Quando um objeto é criado, é o *ObjectManager* que aloca cada bloco e cada uma das réplicas. Quando um objeto é excluído, ele faz a desalocação dos respectivos blocos. Em ambos os casos ocorre a atualização dos metadados dos objetos e dos discos.

Roteador (*Router*)

O *Router* é o componente responsável por receber os pedidos de leitura/escrita dos fluxos, escolher para qual servidor de armazenamento enviar os pedidos e receber as mensagens de controle enviadas pelos servidores de armazenamento. É ele que envia o comando sobre qual nó de armazenamento vai servir qual fluxo.

Para executar esta tarefa, ele conta com duas filas em cada disco, como ilustra a

figura 3.5, uma para tráfego de tempo real (Fila RT - *Real Time Queue*) e uma para tráfego sem restrição de tempo (Fila NRT - *Non-Real Time Queue*). A política de atendimento de cada fila é FIFO e a fila com restrição de tempo tem prioridade no envio dos pedidos de dados para os discos.

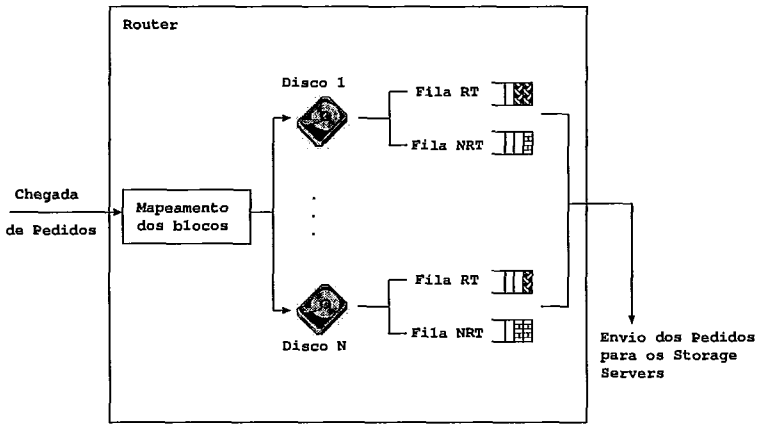


Figura 3.5: Componente Router

Assim que o *Router* recebe um pedido de leitura/escrita, ele faz o encaminhamento para o dispositivo que pode atender este pedido, o que foi determinado pelo *ObjectManager*. Se houver mais de um dispositivo com a cópia do bloco solicitado, o *Router* executa um algoritmo de balanceamento de carga, examinando cada uma das filas dos dispositivos e enviando o pedido para o que possuir a menor delas. Caso aconteça de as duas filas possuírem o mesmo tamanho é escolhido o bloco na ordem em que a replicação foi feita. Não custa salientar que a natureza das filas (RT ou NRT) é levada em consideração na hora da escolha da menor delas.

3.2.2 Nó de Armazenamento

Os nós de armazenamento (*Storage Nodes*), são os locais onde se encontram os discos do sistema, onde os blocos de dados se encontram fisicamente, para então serem usados pelo nó servidor, que é o gerenciador do sistema. O *StorageServer* faz o gerenciamento de cada um destes nós e fica responsável por receber os pedidos de leitura e escrita de um bloco que são enviados pelo nó servidor, para então fazer a autenticação do pedido e se este for válido, executá-lo, como mostra a Figura 3.6.

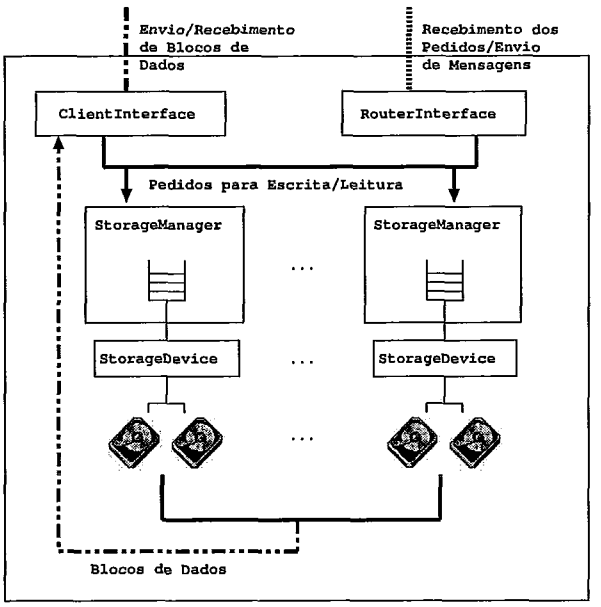


Figura 3.6: Componente StorageServer

A seguir apresentamos os componentes do *StorageServer*.

Interface com o Roteador (*RouterInterface*)

O *RouterInterface* é o responsável por fazer a troca de informações de controle (como, por exemplo, pedido/confirmação de envio de blocos) entre o nó de armazenamento e o nó servidor, através de uma conexão TCP. É este componente que recebe os pedidos enviados pelo servidor para transferência de blocos de dados localizados no *StorageServer*.

Dispositivo de Armazenamento (*StorageDevice*)

O *StorageDevice* é o componente responsável pela realização das operações de entrada e saída, que são realizadas através de chamadas do sistema operacional. Para cada disco do sistema, que pode ser um arquivo ou uma partição do disco rígido do computador, existe apenas uma instância do *StorageDevice*.

Gerenciador de Armazenamento (*StorageManager*)

Existe um *StorageManager* para gerenciar cada um dos *StorageDevice*. Ele é o responsável pelo escalonamento dos pedidos feitos ao disco que está sob seu gerenciamento. Desta forma é criada uma fila com os pedidos que precisam ser atendidos. Para cada pedido de leitura associa-se um *buffer* para o armazenamento do bloco solicitado.

Interface com o Cliente (*ClientInterface*)

Este componente é o responsável pelo envio e recebimento dos blocos de dados solicitados. É através dele que os blocos do vídeo são transmitidos pelo nó de armazenamento durante uma sessão do cliente e são recebidos durante a cópia do vídeo para o sistema.

Apesar dessas operações serem feitas via UDP, está implementado um controle da transmissão por parte do cliente, havendo retransmissão no caso de alguma perda. Para tal transmissão foi criado um protocolo, baseado no TCP, onde cada bloco a ser enviado é fragmentado e o controle do transmissor é feito através de algumas variáveis, como por exemplo, a quantidade de fragmentos que compõe o bloco, a quantidade de fragmentos já recebidos pelo receptor e o endereço do receptor, formado pelo IP, porta e identificação da requisição.

3.2.3 Clientes

Os clientes fazem o acesso ao servidor RIO (composto pelos servidores de armazenamento e pelo nó gerenciador) através de uma interface que estabelece o que pode ser solicitado ao servidor, a ordem dos pedidos e o formato das informações.

Os clientes disponíveis neste componente são o *riosh* e o *RioMMClient*. O primeiro faz a administração dos objetos armazenados no servidor. O segundo é o cliente utilizado para a visualização dos vídeos.

Cliente *riosh*

O *riosh* é o interpretador de comandos do servidor RIO. É através dele que são executadas as funções como cópia, exclusão, consulta do conteúdo dos objetos presentes no servidor, entre outras.

Cliente *RioMMClient*

É o cliente utilizado para a visualização dos vídeos. Possui funcionalidades VCR, como tocar (*play*), parar (*stop*), pausar (*pause*), avançar (*fast forward*), retroceder (*fast rewind*), saltar pelo índice ou mover a barra de progresso para a posição desejada no vídeo, além de um módulo de sincronização com transparências, como foi mostrado na Figura 3.1 da seção 3.1. Para exibição do vídeo recebido utiliza-se o MPlayer [28].

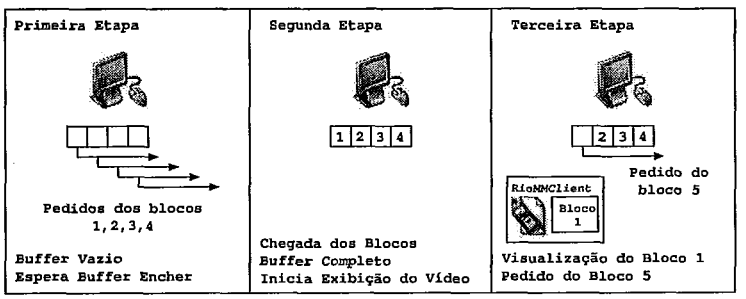


Figura 3.7: Funcionamento do Cliente RioMMClient

Como pode ser visto na Figura 3.7, o cliente solicita a princípio tantos blocos quantos forem necessários para encher o seu *playout buffer* (Blocos 1,2,3 e 4 da primeira etapa). Após o recebimento de todos os blocos (segunda etapa), o *buffer* encontra-se cheio e tem-se o início da exibição do vídeo. A partir deste ponto ele passa a solicitar um novo bloco a cada vez que um outro for consumido, até que o vídeo tenha fim (terceira etapa). Na figura, o bloco 1 é repassado para o *RioMMClient* para sua exibição e o bloco 5 é requisitado para o gerenciador do sistema RIO. Caso o bloco seja perdido, chegue atrasado ou incompleto, o cliente faz a recuperação dos fragmentos que chegaram corretamente, ou seja, agrupa todos os fragmentos pertencentes ao mesmo bloco, e os repassa ao MPlayer, já que ele não

precisa conter o bloco inteiro para exibi-lo.

A Figura 3.1 mostra a interface do cliente para visualização de vídeos.

3.3 Armazenamento dos Dados no Servidor

O RIO gerencia uma coleção de objetos multimídia que armazena uma quantidade arbitrária de dados. Cada um destes objetos é dividido em blocos de tamanho fixo, que são as unidades de alocação de espaço de armazenamento. Cada bloco de dados é armazenado em uma posição aleatória de um disco escolhido aleatoriamente, como descrito na seção 2.6.

Fornecer garantias de tempo-real em um sistema com alocação aleatória limitaria a carga a níveis muito baixos, subutilizando os recursos do sistema. Entretanto, o servidor dá garantias de que os pedidos são servidos dentro de um dado limite de atraso com probabilidade alta. Assume-se que a probabilidade de exceder o limite do atraso igual a 10^{-6} seria satisfatória para a maioria das aplicações [45].

3.4 Controle de Admissão

No momento em que novos fluxos de tempo real são abertos, o servidor RIO executa o controle de admissão através do *StreamManager*. Em [21] foi implementado um novo controle de admissão, onde o servidor faz uma simulação no momento da admissão de um novo cliente, usando as informações de cada um deles, como a carga atual e as medidas de desempenho do servidor (tempo de serviço dos discos e tempo de espera nas filas) obtidas em tempo real, para verificar se a QoS pode ser mantida para todos os clientes, incluindo o novo, durante toda sua execução. Apesar de ser um controle mais apurado, esta nova implementação tem um custo computacional mais alto que a originalmente proposta.

O controle de admissão original é baseado nas taxas solicitadas por cada cliente. Toda vez que o servidor é iniciado ele lê um arquivo de configuração do sistema,

onde consta a taxa total aceita pelo servidor e a taxa reservada para fluxos sem restrições de tempo.

Outra variável presente no arquivo de configuração do sistema é a taxa alocada, que é inicializada com o valor 0 e, de acordo com o que cada cliente solicita para abertura de um fluxo, esta variável é incrementada com este valor. Assim, para que um cliente seja admitido pelo servidor ele precisa enviar informações, tais como se o tráfego é de tempo real ou não, se a operação a ser realizada é de leitura ou escrita e a largura de banda solicitada.

Se o tipo de tráfego solicitado for de tempo real, o novo usuário é admitido se a seguinte condição for verdadeira:

(*Taxa Alocada* + *Taxa Solicitada*) <= (*Taxa Total* - *Taxa Reservada Sem Restrição de Tempo*).

Caso o novo usuário tenha sido admitido, a variável Taxa Alocada é atualizada de tal forma que seja acrescida da Taxa Solicitada, ou seja, *Taxa Alocada* = *Taxa Alocada* + *Taxa Solicitada.*

Este esquema não leva em consideração a variabilidade do tráfego e, por este motivo, o esquema de [21] é mais eficiente, pois não assume taxa constante.

3.5 Policiamento de Tráfego e Pedidos

Com o objetivo de evitar situações de congestionamentos e conseqüente atrasos indesejáveis e perdas dos pacotes de dados devido a rajada de pedidos gerados pelos clientes, o servidor implementa um algoritmo de policiamento de tráfego.

O mecanismo utilizado hoje no servidor foi implementado em [21], onde o policiamento de pedidos é feito pelo componente *StreamManager* através de uma fila *leaky bucket*, como mostra a Figura 3.8. A idéia é gerar fichas a uma determinada taxa e enviar os pacotes ou *bytes* apenas se uma ficha estiver disponível. Com isto temos dois parâmetros: a taxa de geração de fichas r e a capacidade de fichas F

que cada cliente pode armazenar. A quantidade de fichas vai sendo incrementada de acordo com a taxa de geração e tem como limite o valor F. Este valor é configurado na inicialização do servidor e a taxa de geração de fichas é configurada a partir da taxa solicitada pelo cliente. A quantidade máxima de pedidos que serão repassados em qualquer intervalo de tempo t é $rt + F$.

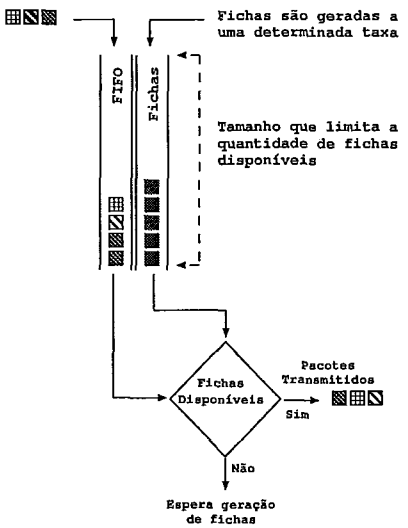


Figura 3.8: Fila Leaky Bucket

Para cada tipo de tráfego o policiamento é feito de forma diferente. Para tráfego de tempo real, o envio de cada pedido depende do cálculo do intervalo mínimo entre pedidos no momento de sua admissão, o que é feito pelo *StreamManager*. Para tráfego de tempo não real, o pedido é enviado para o *Router* quando o número de pedidos que foram enviados para este componente e não foram servidos for menor que um certo número. Este número é calculado pelo *StreamManager* durante a inicialização do sistema, de acordo com o número máximo de pedidos que cada fila dos discos do sistema pode conter.

3.6 Replicação no servidor RIO

Usar somente alocação aleatória dos dados pode causar flutuações estatísticas na carga gerada para um determinado disco e, conseqüentemente, desbalanceamentos,

ou seja, alguns discos podem ter cargas mais altas que outros durante curtos períodos de tempo. Isto causa o que nós chamamos de efeito cauda longa na distribuição do tempo de resposta. Para resolver este problema replica-se aleatoriamente uma fração ou a totalidade dos blocos de dados. Isto dá ao sistema flexibilidade no roteamento das requisições para os discos, melhorando o balanceamento de carga e causando uma grande redução na cauda da distribuição do tempo de resposta [37].

Além de melhorar o balanceamento de carga, a replicação aumenta a disponibilidade do serviço. Por exemplo, quando um aluno faz uma requisição do material de uma aula, a hipótese de que o material não estará disponível porque um dos nós de armazenamento está fora de operação não é uma situação desejada. Neste caso, o servidor direciona o pedido para outro servidor de armazenamento que contenha a cópia do que foi solicitado. O nó gerenciador controla essa transferência de forma transparente, sem qualquer intervenção por parte do usuário.

O balanceamento de carga implementado no RIO consiste de um algoritmo de roteamento simples, onde uma requisição de leitura de um bloco replicado é direcionada para o disco que tem a menor fila. O escalonamento utilizado por cada fila obedece a política FIFO.

Capítulo 4

Ambiente de Modelagem e Testes

Divide as dificuldades que tenhas de examinar em tantas partes quantas for possível, para uma melhor solução. *René Descartes*

Neste capítulo é apresentado o ambiente de modelagem usado para a confecção dos modelos gerados para simulação, que é a ferramenta Tangram-II. Descreveremos em particular o ambiente de modelagem e o gerador de tráfego, pois foram os módulos da ferramenta usados neste trabalho. Mostraremos também o ambiente utilizado para os testes que foram realizados para parametrização e validação dos modelos implementados.

4.1 A Ferramenta Tangram-II

O Tangram-II [24, 12], ilustrado na Figura 4.1, é um ambiente para experimentação e modelagem de sistemas computacionais e de comunicação, desenvolvido para pesquisa e com propósitos educacionais, que oferece uma interface baseada no paradigma de orientação a objetos e uma variedade de módulos adicionais úteis para experimentos em redes de computadores e ferramentas multimídia que auxiliam no processo de modelagem e trabalho colaborativo.

No paradigma usado na ferramenta, o sistema modelado é representado por uma coleção de objetos que interagem através de envio e recebimento de mensagens e

também através de eventos, que definem o comportamento de um objeto, juntamente com as ações e condições associadas a estes eventos. O estado interno de cada objeto é representado por um conjunto de variáveis.

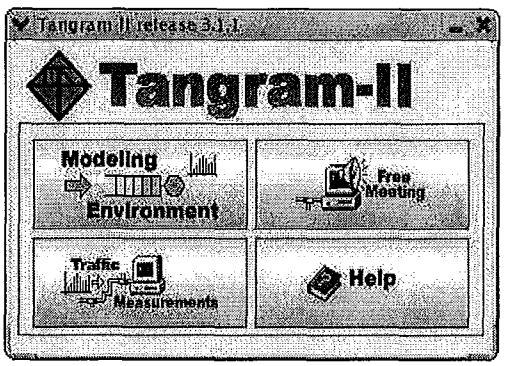


Figura 4.1: Ambiente de Modelagem

A Figura 4.2 mostra a Interface do Ambiente de Modelagem. Os ícones à esquerda da figura mostram as opções disponíveis para os usuários:

1. Módulo de Especificação do Modelo
2. Módulo de Geração de Modelo Matemático
3. Módulo para Solução Analítica do Modelo
4. Módulo de Geração de Medidas de Interesse
5. Módulo de Descritores de Tráfego
6. Módulo de Simulação

4.1.1 Modelagem

Na modelagem, os eventos são gerados espontaneamente por um objeto, desde que as condições especificadas na criação do objeto, que são expressões *booleanas* avaliadas de acordo com o estado atual do objeto, tenham sido satisfeitas. As mensagens são uma abstração usada para representar as interações entre os objetos,

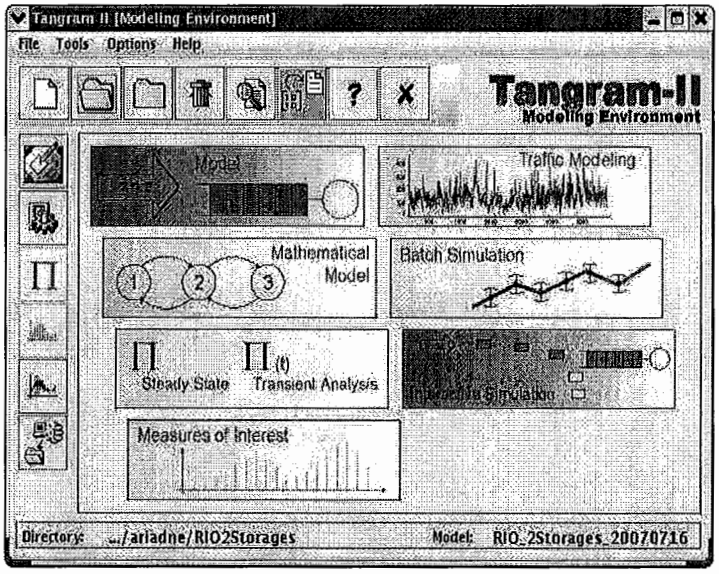


Figura 4.2: Ambiente de Modelagem

e sua entrega e recebimento são realizadas em tempo zero. Quando um evento é executado ou uma mensagem é recebida, um conjunto de ações especificadas pelo usuário no modelo, de acordo com suas necessidades, é realizado com uma dada probabilidade. Como resultado destas ações tomadas, o estado do objeto pode mudar e mensagens podem ser enviadas para outros objetos no modelo.

Para definir um modelo ou criar um novo objeto, a ferramenta fornece uma interface gráfica de domínio público TGIF (*Tangram Graphic Interface Facility*), desenvolvida pela UCLA [14], que é mostrada na Figura 4.3.

Na Figura 4.3, mostramos também um exemplo de atributos de um objeto. É neste ambiente que o comportamento do modelo é especificado. Para a criação de um novo modelo, utilizando a interface do TGIF, o usuário pode usar objetos pré-definidos que estão presentes na Biblioteca de Objetos da ferramenta.

Na simulação, as medidas de interesse são obtidas a partir de um atributo do objeto chamado recompensa. As recompensas podem ser de taxa ou de impulso.

- Recompensa de Taxa: As recompensas de taxa estão associadas com os estados do modelo. Por exemplo, se uma recompensa de taxa r_i está associada ao estado i, então o sistema ganha uma recompensa r_i por cada unidade de tempo

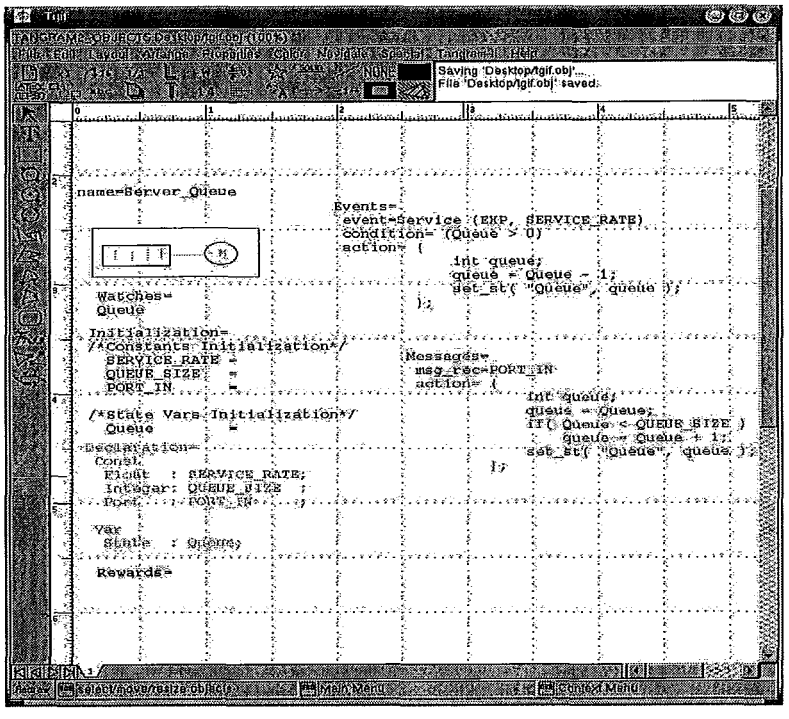


Figura 4.3: TGIF - Tangram Graphic Interface Facility

gasta no estado i. Supondo que estamos interessados em calcular a utilização de um objeto que representa uma fila no modelo, poderíamos definir uma recompensa que expressa esta medida de interesse, e que acumula o valor 1 a cada unidade de tempo que a fila contiver ao menos um pacote, ou seja, não esteja vazia, como mostrado na Figura 4.4.

```
Rewards=
rate_reward = utilization
condition= (Queue > 0)
value= 1.0;
```

Figura 4.4: Exemplo de Recompensa

- Recompensa de Impulso: As recompensas de impulso estão associadas com transições entre os estados do sistema. Se uma recompensa r_{ij} está associada com a transição do estado i para o estado j, então o sistema ganha uma recompensa r_{ij} cada vez que o sistema faz uma transição do estado i para o estado j. As recompensas de impulso podem ser usadas como um mecanismo de contagem, podendo ser associadas com o disparo de um evento ou com a

recepção de uma mensagem.

Estas recompensas são genéricas o bastante para permitir a definição de uma ampla quantidade de medidas de interesse. Elas são definidas com o atributo *Rewards* nos objetos.

4.1.2 Gerador de Tráfego

O Gerador de Tráfego do Tangram-II é uma ferramenta muito útil para descobrir as características da rede, sendo executada tanto na máquina de origem, quanto na máquina de destino. Esta ferramenta é capaz de estimar parâmetros entre a fonte e o destino, tais como *jitter*, capacidade do gargalo, atraso em um sentido (*one-way delay*), taxa de descarte de pacotes, entre outros. Ela é capaz de injetar pacotes na rede segundo um modelo especificado pelo usuário. Na Figura 4.5 mostramos a interface da ferramenta.

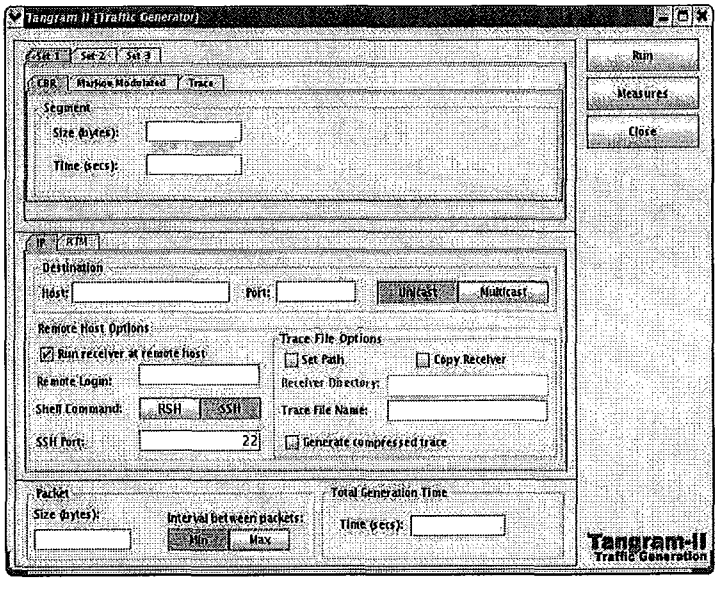


Figura 4.5: Interface do Gerador de Tráfego

Antes de utilizar o Gerador de Tráfego, o usuário deve definir que medidas de interesse deseja estimar, para escolher o método correto para geração de sondas (*probes*) [20]. Para tal, as seguintes especificações devem ser feitas na interface da ferramenta:

- Direção na geração das sondas: as sondas podem ser enviadas de uma fonte a um destino (*One-Way*), em dois sentidos onde dois *hosts* enviam sondas um ao outro (*Two One-Way*), ou as sondas são enviadas de um ponto da rede para outro, onde são recebidas e enviadas de volta ao *host* de origem, como funciona com a ferramenta *ping*, só que no nível de aplicação (*Round Trip Time*).

- Modelo de geração de sondas: esta geração pode ser feita com taxa constante de *bits* (CBR), modulada por uma cadeia de Markov (*Markov Modulated*), através da utilização de um arquivo de *traces* ou através de pares de pacotes (*Packet Pair*), onde o tráfego é gerado em um intervalo constante, como o CBR, mas ao invés de enviar apenas um pacote, dois são enviados.

- Características de geração de tráfego: tamanho L dos pacotes a serem transmitidos, tempo total T de geração, e número de *bytes* D por *frame*. Assim, o número de pacotes N gerados por *frame* depende do tamanho do *frame* e do tamanho do pacote, isto é $N = D/L$.

- Escolha do tráfego: se IP ou ATM.

  Estes são apenas alguns parâmetros que podem ser especificados. Vários outros parâmetros são mostrados no manual da ferramenta [23].

As medidas de interesse são obtidas a partir dos arquivos de *traces* gerados pela ferramenta. Neste trabalho, o Gerador de Tráfego foi utilizado em medições descritas no capítulo 5.

4.2 Ambiente de Testes

Nesta seção será apresentada uma descrição do ambiente disponível para os testes em ambiente real e a configuração que utilizamos.

4.2.1 Arquitetura da Rede

A Figura 4.6 mostra a arquitetura da rede Giga que se encontra disponível para testes. A seguir descreveremos os equipamentos do ambiente de testes.

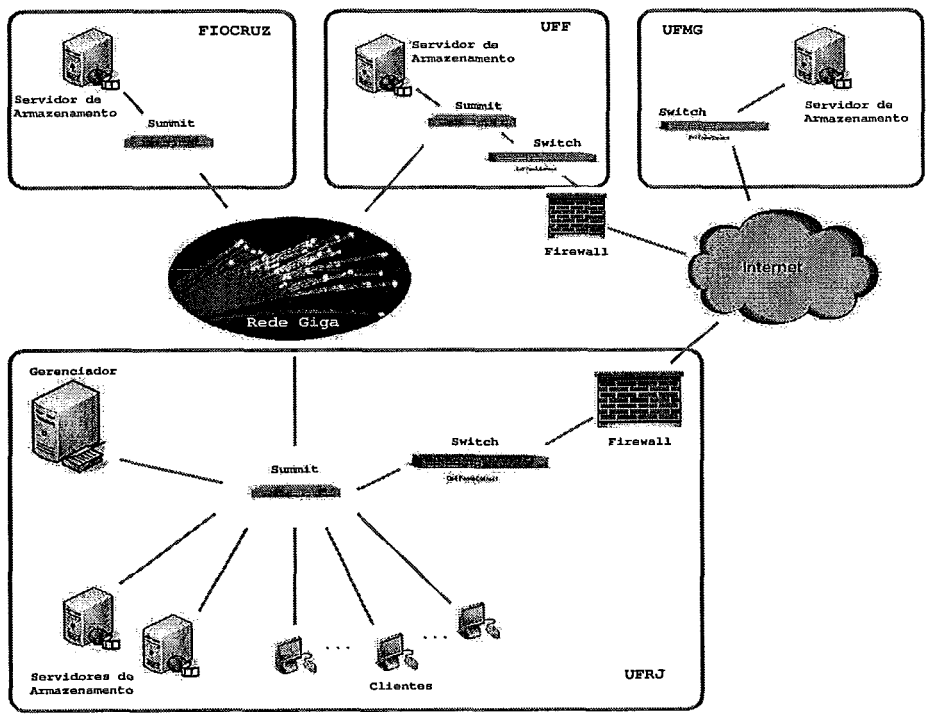


Figura 4.6: Arquitetura Completa da Rede

Os *switches* que são utilizados para interligar as instituições à rede Giga são do modelo *Summit* 200-24 produzido pela *Extreme Networks* [5]. Este modelo possui 24 portas *Ethernet* 10/100 MBits por segundo (chamadas portas 100) e 2 portas *GigaBit* por segundo (também chamadas portas 1000 - de 1000 MBits). Na Tabela 4.1, mostramos a relação do *hardware* utilizado para os servidores de armazenamento. Uma das máquinas que está localizada na UFRJ foi escolhida para rodar um servidor de armazenamento e o gerenciador do servidor simultaneamente. As máquinas possuem o sistema operacional *Scientific Linux* 4.3 ou superior instalado e todas possuem Processador Intel(R) Pentium(R) 4.

As máquinas onde os clientes foram disparados, através do emulador de clientes que está descrito a seguir, têm como configuração mínima: Processador Pentium IV 2 GHz, 1 GB de RAM, HD IDE 20 MB/s. Todas as máquinas clientes foram

Instituição	Processador	Memória	Disco	Placa de rede
UFRJ	3.60GHz	1 GB	(SATA-40GB)51 MB/s	1 Gbit/s
UFRJ	3.20GHz	1.25 GB	(IDE-80GB)52 MB/s	1 Gbit/s
FIOCRUZ	3.20GHz	512 MB	(SATA-160GB)53 MB/s	1 GBit/s
UFF	2.40GHz	512 MB	(IDE-80GB)28 MB/s	100 Mbit/s
UFMG	3.20GHz	2 GB	(SATA-80GB)55 MB/s	1GBit/s

Tabela 4.1: Relação do *Hardware* das Máquinas Utilizadas como Servidor de Armazenamento.

disparadas na UFRJ, onde estavam conectadas a uma das portas 1000 do *Summit* através de um *switch* Dell Powerconnect 2724 [3].

Na UFF e Fiocruz, as máquinas estão ligadas diretamente às portas 1000 do *Summit*, sendo que na UFF uma das portas 100 é utilizada para conexão ao *switch* do laboratório, o que permite acesso à Internet. Na Fiocruz, a máquina não possui acesso à Internet, estando completamente isolada na rede Giga. A segunda porta giga disponível em todos os *Summits* é utilizada para a conexão com a fibra óptica. Na UFMG, como o acesso é feito através da Internet, a máquina está ligada ao *switch* do laboratório.

As máquinas da UFF, Fiocruz e UFMG são utilizadas como servidores de armazenamento. Na UFRJ existem 19 máquinas, sendo duas reservadas para servidores de armazenamento e gerenciador do servidor (onde uma delas possui uma instância de cada um), e as outras para disparo de clientes.

A plataforma utilizada nos testes está representada na Figura 4.7. Como pode ser observado, dezenove máquinas foram utilizadas na UFRJ, sendo uma para o gerenciador e um nó de armazenamento, em conjunto, e as outras dezoito para a execução dos clientes. E também utilizamos uma máquina na Fiocruz para um nó de armazenamento.

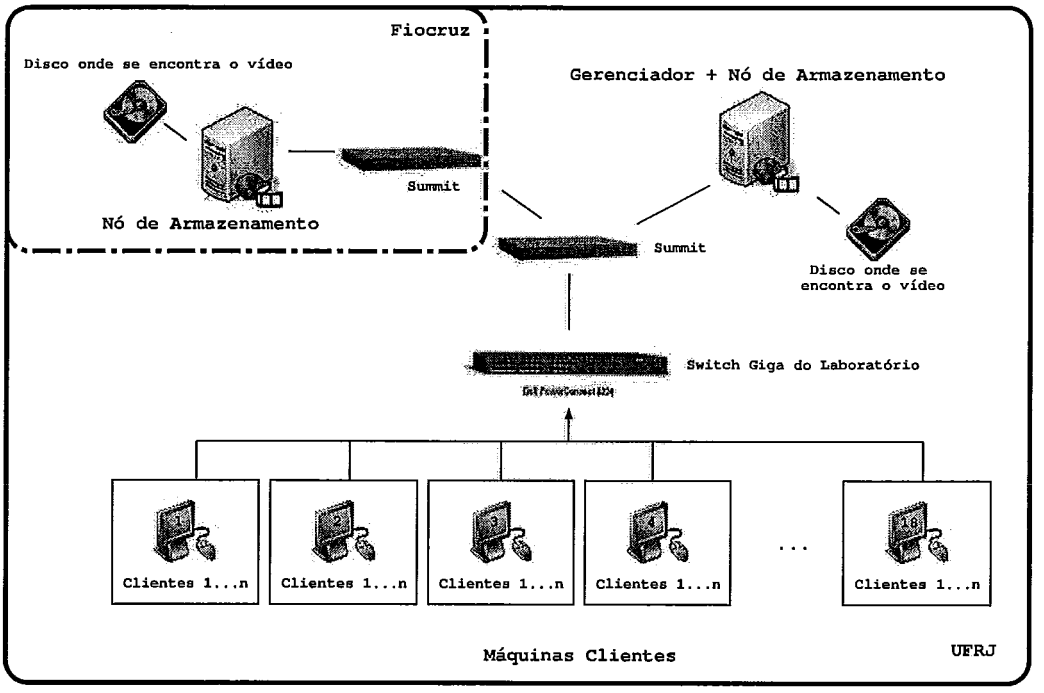


Figura 4.7: Plataforma de Testes

4.2.2 Emulador de Clientes

O emulador de clientes utilizado neste trabalho foi desenvolvido com o intuito de permitir experimentos mais abrangentes com o RIO. Ele possui os mesmos mecanismos de um cliente comum do sistema, com a única diferença de não haver a exibição do vídeo que é recebido. Desta forma, o custo computacional na decodificação do vídeo é eliminado e, assim, um número maior de emuladores pode ser disparado em cada máquina cliente.

Cada instância do emulador representa um cliente no sistema. Os *logs* utilizados como entrada para o emulador de clientes foram extraídos dos *logs* das ações dos usuários do curso de Tecnologia de Sistemas e Computação do consórcio CEDERJ, onde os alunos comparecem a um dos pólos de ensino do projeto de ensino à distância onde os cursos são oferecidos, como mostra a Figura 4.8. O estudante comparece a um dos pólos porque faz todo o curso de graduação através do regime semipresencial, com sistema de tutoria a distância (por telefone, Internet e plataforma - servidor RIO) e atendimento presencial.

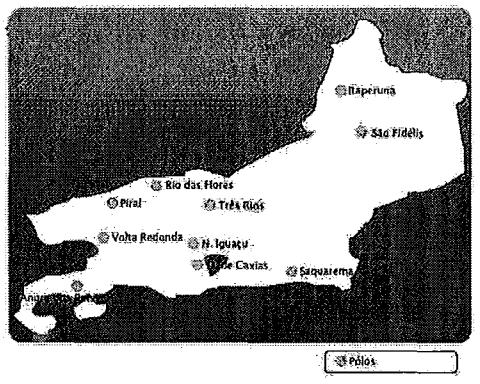


Figura 4.8: Pólos do CEDERJ

O *log* de entrada para o experimento seqüencial pode ser visto na Figura 4.9.

```
12 0   0 -1  ⟶ play inicial
12 0   5  0  ⟶ play (unpause)
12 900 2 -1  ⟶ quit
               ↓
             Bloco
           Ação
       Tempo (segs)
Número da Sessão
```

Figura 4.9: Log entrada do emulador para acesso seqüencial

Como pode ser observado na figura 4.9, o cliente não faz nenhum tipo de interação durante todo o tempo que permanece conectado. No instante 0 ele começa a assistir o vídeo e só sai do sistema quando o vídeo termina, no instante 900 segundos.

A Figura 4.10 mostra o *log* de entrada para emular um cliente interativo. Da mesma forma que no experimento seqüencial, o cliente inicia sua exibição no tempo 0 e sai do sistema em algum instante do tempo, tendo executado diversas ações durante a seção. As ações possíveis são: play, pause, jump, stop e quit.

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

- Tempo de requisição, recepção e exibição de cada bloco;
- Blocos perdidos;

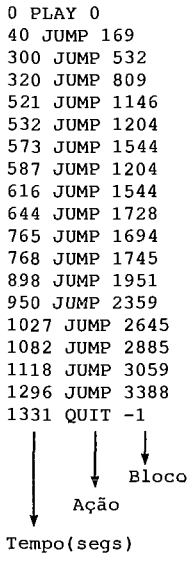


Figura 4.10: Log de entrada para o experimento interativo

- Quantidade de informação que foi requisitada ao servidor, recuperada da cache, recebida e exibida;

Os mecanismos de coleta de dados foram implementados em [53], e a Figura 4.11 apresenta a saída gerada, de onde são coletadas as estatísticas utilizadas nos resultados do trabalho. De agora em diante, o emulador será chamado simplesmente de cliente dentro deste trabalho.

4.2.3 Metodologia dos Experimentos em Ambiente Real

Para a realização dos experimentos deste trabalho, foi utilizada uma configuração distribuída para o servidor RIO. Em todos os experimentos o nó gerenciador do servidor encontra-se na UFRJ. Os nós de armazenamento foram colocados na UFRJ e na Fiocruz.

Para os dois cenários, sequencial e interativo, foram feitos experimentos com 1 e 2 nós de armazenamento com um número incremental de clientes. O número inicial de clientes foi 300 e cada incremento variou da ordem de 20 a 50 clientes. Ao final de cada experimento, as medidas desejadas foram coletadas e analisadas.

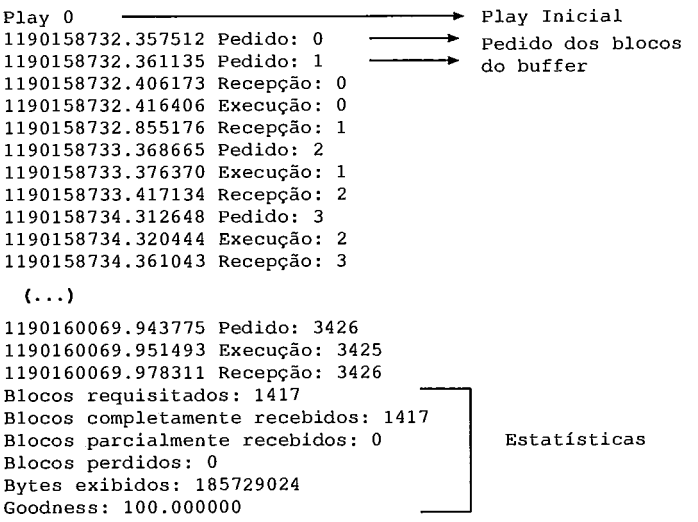


Figura 4.11: Formato do *log* de saída do emulador

Apesar de termos disponíveis para testes até 5 servidores de armazenamento, só utilizamos 2 deles devido à limitação no número de máquinas disponíveis para emular os clientes. Aumentar o número de servidores de armazenamento não influencionou nos resultados, já que o número de máquinas clientes permaneceu o mesmo.

Dispúnhamos de 18 máquinas para emular os clientes, sendo que cada uma delas possui capacidade de emular entre 20 e 40 clientes. Desta forma não foi possível emular uma população superior a 550 clientes.

O objetivo destes experimentos é verificar o número máximo de usuários que o nó de armazenamento suporta, para validação dos modelos de simulação.

Os *logs* de comportamento selecionados para o experimento sequencial possuem duração de 25 minutos. E os utilizados para o experimento interativo têm duração entre 20 e 45 minutos.

Nos testes realizados com 2 servidores de armazenamento foi possível atender esta população com uma qualidade excelente, portanto para avaliarmos o número de clientes que poderiam ser atendidos com 3 ou mais servidores de armazenamento teríamos que emular um número maior de clientes.

No próximo capítulo apresentamos os resultados obtidos tanto na simulação

quanto em ambiente real.

Carga dos Clientes

Para a geração da carga dos pedidos dos clientes utilizamos um emulador que dispara vários clientes inicializados em intervalos baseados em um processo de *Poisson*. A máquina onde cada cliente deve ser disparado é escolhida aleatoriamente e, após a espera do intervalo de chegada (através de uma chamada de sistema *usleep*), o cliente inicia sua execução na máquina escolhida.

Como dito anteriormente, cada máquina cliente suporta uma quantidade específica de clientes, assim não pudemos gerar a carga máxima que pretendíamos, pois as máquinas clientes se tornam o gargalo do sistema. Se o número de clientes por máquina exceder o máximo que ela suporta, o *Goodness* retornado, pela máquina específica, é baixo. Porém, sabemos que o sistema ainda não está saturado, e sim a máquina cliente.

Em seguida, o cliente simula a visualização de cada bloco de acordo com o tempo de consumo coletado anteriormente para o vídeo utilizado. Assim, após a simulação da exibição do bloco atual, o cliente envia o pedido do próximo bloco a ser armazenado em seu *playout buffer*.

Capítulo 5

Modelos e Resultados

Precisamos analisar o todo para depois, compreendermos as partes. *Aristóteles*

Neste capítulo são apresentados os modelos desenvolvidos usando a ferramenta Tangram-II, apresentada no capítulo anterior, e os resultados dos modelos e dos experimentos realizados em ambiente real.

5.1 Modelos

Dois modelos foram desenvolvidos: um para avaliar o sistema na presença de usuários sequenciais e outro para avaliar usuários interativos.

Através destes modelos é possível analisar e observar o comportamento de um cliente admitido no sistema e do Gerenciador, onde podemos visualizar, por exemplo, o comportamento do *buffer* do cliente no Gerenciador, o tamanho das filas nos discos e o *playout buffer* do cliente. Além disso, podemos verificar se a adição de um nó de armazenamento numa rede mais lenta, fora da rede Giga, impacta o sistema. Pretendemos responder, por exemplo, as seguintes questões: Até que ponto um nó de armazenamento fora da rede Giga aumenta a escalabilidade do serviço? Quantos clientes interativos podemos atender com um certo número de nós de armazenamento?

5.1.1 Primeiro Cenário: Modelo Sequencial

O primeiro modelo, representado na Figura 5.1, representa a configuração do servidor RIO no contexto do projeto DIVERGE, descrito anteriormente. Neste cenário, apresentamos o sistema sem replicação, ou seja, existe apenas uma cópia do vídeo no sistema.

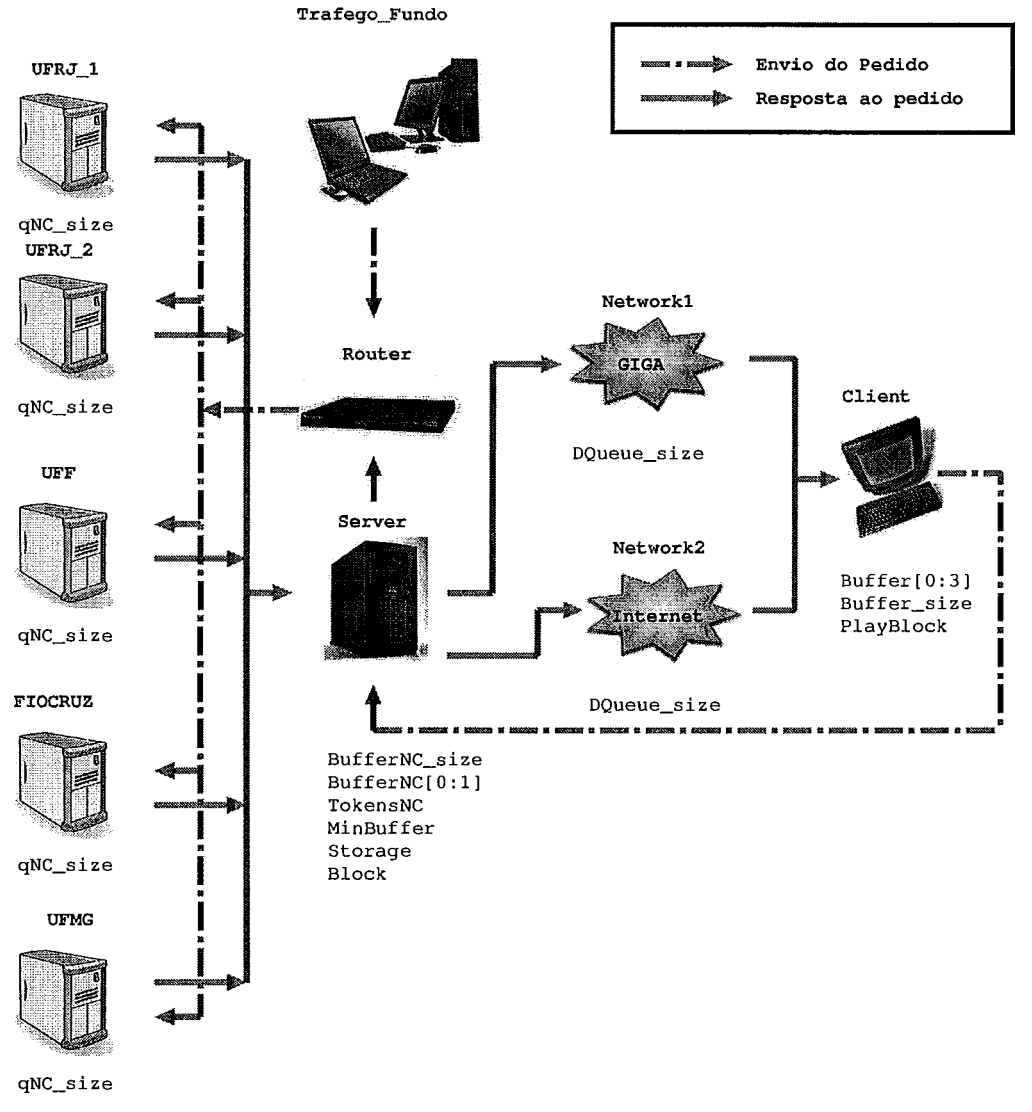


Figura 5.1: Modelo do Servidor com 5 Nós de Armazenamento, 4 na Rede Giga e 1 na UFMG

O modelo possui os seguintes objetos:

1. **Objeto Cliente**, que é o cliente a ser analisado;

2. **Objeto Tráfego de Fundo**, que representa a carga agregada gerada por todos os outros clientes que estão utilizando os recursos do servidor e da rede;

3. **Objeto Roteador**, que faz parte do servidor RIO e faz o escalonamento e encaminhamento dos pedidos que chegam ao objeto Nó Gerenciador;

4. **Objeto Nó Gerenciador**, que tem a função de controlar e gerenciar todos os *Storages* e blocos de dados;

5. **Objetos Nós de Armazenamento**, que representam cada um dos servidores de armazenamento das outras instituições envolvidas no projeto da Rede Giga;

6. **Objeto Rede Internet**, que representa a rede por onde passa o fluxo dos dados que vêm da UFMG;

7. **Objeto Rede Giga**, por onde passa o tráfego de todas as instituições que estão interligadas pela rede Giga.

Como estamos interessados em descobrir o impacto que um determinado cliente, ao ser admitido, sofre com o aumento do número de clientes no sistema, iremos analisar o comportamento de apenas um cliente, separadamente dos demais, para verificar a possibilidade de ocorrência de perda durante o envio dos blocos solicitados para a visualização do vídeo. Consideraremos perdido todo bloco que não estiver presente no *buffer* do cliente no momento em que deveria ser exibido. A escolha desta abstração se deve ao fato de ser inviável analisar o comportamento de uma quantidade tão grande de clientes, pois o tempo de simulação aumentaria consideravelmente, com todos os eventos gerados por estes clientes. Assim, agrupamos a carga dos demais clientes em um único objeto, que simula a requisição de pedidos ao sistema, da mesma forma como seria feito se todos os clientes estivessem separados. Esta carga agregada atua como o tráfego concorrente nos discos dos nós de armazenamento.

Modelamos também as duas redes por onde passam os blocos do vídeo, sendo que cada uma delas apresenta características inerentes ao seu real funcionamento: a rede

Giga, com atraso pequeno e a Internet, com um atraso maior, representando os pedidos que vêm da UFMG, por um *link* mais lento. O objetivo é analisar a quantidade de pacotes atrasados, que são considerados aqui como perdidos por simplicidade de terminologia, devidos aos retardos das redes e das filas nos servidores.

Não consideraremos as perdas entre a UFMG e UFRJ devido ao fato de representarmos no modelo a transmissão de um pacote de tamanho 128KB (tamanho do bloco do servidor RIO) e não os fragmentos de 1500 *bytes* que são enviados pela rede no ambiente real. Esta simplificação foi introduzida já que o custo computacional da simulação seria muito elevado se representássemos no modelo cada um dos fragmentos sendo transmitido pela rede. Como cada bloco tem 128KB, teríamos a transmissão de 90 fragmentos de 1500 *bytes* ao invés da transmissão de um bloco. Dado que simulamos a transmissão de um bloco de 128KB, uma perda deste bloco significaria, no ambiente real, uma perda de uma rajada de 90 fragmentos na rede.

Através de medições, constatamos que média de perda entre a UFMG e UFRJ é de 7%, porém a probabilidade de perda de uma rajada de 90 fragmentos neste *link* é nula. Portanto, não consideramos perda no objeto que representa a Internet no modelo. No resultado geral, teríamos uma perda, em média, de 7% dos fragmentos que fossem enviados pela UFMG, porém não perderíamos um bloco inteiro, que é o que estamos medindo.

O atraso nas redes foi obtido através de medições, em ambos os *links*, como será descrito na seção 5.2.

Do lado do servidor são modelados dois objetos. O servidor, também podendo ser chamado de gerenciador, representa a parte de controle do RIO, onde se encontra a tabela de pedidos e o *lookahead buffer* (um *buffer* que faz pré-armazenamento dos próximos blocos que, teoricamente, serão pedidos) de cada cliente. É ele que recebe os pedidos de blocos vindos dos clientes e é ele que controla o tamanho das filas nos discos dos *Storages* e faz o encaminhamento para o disco escolhido. Juntamente com o objeto Gerenciador, modelamos o objeto Roteador, que é a parte do servidor que faz o encaminhamento dos pedidos para os *Storages* de acordo com o tamanho das filas de cada um deles, quando há replicação, ou caso contrário, encaminha para

o *Storage* que possui o bloco solicitado. Quando não há replicação, este envio é feito de forma aleatória, da mesma forma como é feita a cópia dos blocos dos dados pelo servidor, quando um vídeo é copiado para o mesmo. O disco e a posição em que o bloco ficará armazenado serão escolhidos aleatoriamente.

A seguir fazemos uma descrição detalhada de cada um dos objetos supracitados.

- **Objeto Tráfego de Fundo**

  Este objeto representa o tráfego agregado de todos os clientes que estão fazendo solicitações de blocos ao servidor. O objetivo é gerar carga para as filas de requisições dos discos nos nós de armazenamento. A escolha desta abstração foi feita como forma de otimizar o tempo de simulação. Se milhares de clientes fossem modelados separadamente o tempo de simulação seria consideravelmente alto e, consequentemente, inviável. A escolha de para qual disco será enviado o pedido destes clientes é feita de forma aleatória, como acontece no servidor RIO. A distribuição utilizada para modelar o intervalo entre chegadas de pedidos é a exponencial, com parâmetro igual ao número de clientes que estão sendo representados vezes a taxa média de leitura de cada bloco do vídeo.

- **Objeto Rede Giga**

  O objeto Rede Giga é responsável por transmitir para o cliente os blocos de dados servidos pelos *Storages Servers* das instituições que compõem a rede Giga (UFRJ, UFF e FIOCRUZ). A rede Giga é representada por um servidor infinito (não existe fila), onde para cada mensagem recebida o evento de serviço é clonado, ou seja, um novo evento é escalonado pelo simulador, não sendo necessário esperar pelo término do evento de serviço escalonado anteriormente. A distribuição usada para representar o tempo de transmissão de um bloco foi a distribuição Gaussiana, pois, de acordo com [26], esta distribuição representa uma boa aproximação para modelar o atraso na rede. Mas além desta informação, fizemos experimentos para comprovar que a Gaussiana foi a distribuição que melhor representa esta variável aleatória, como pode ser visto na seção 5.2.

- **Objeto Rede Internet**

  O objeto Rede Internet é o objeto responsável por enviar os blocos de dados vindos do *Storage Server* da UFMG, pois esta é a única instituição que não faz parte da rede Giga. A média e variância utilizadas como parâmetro para a Gaussiana que representa o tempo de transmissão de um bloco foram medidas utilizando-se a ferramenta Tangram-II, com o módulo gerador de tráfego, como será explicado em detalhes na seção 5.2. Da mesma forma que o objeto rede Giga, neste objeto o atendimento dos pedidos é feito por um servidor infinito.

- **Objeto Gerenciador**

  O objeto Gerenciador é o responsável pelo encaminhamento das solicitações feitas pelos clientes. É este objeto que recebe os pedidos feitos pelos clientes e é ele quem controla todas as informações do sistema, ou seja, tem função de gerenciamento. Este objeto possui as seguintes variáveis de estado:

  - Um contador que indica quantas posições do *lookahead buffer* do cliente no Gerenciador estão preenchidas;
  - Um vetor que representa este *buffer*;
  - Um contador que indica quantos blocos o cliente está esperando que ainda não foram servidos;
  - Uma variável que indica qual o próximo bloco que será requisitado pelo cliente;

  A cada requisição feita pelo objeto Cliente (de acordo com a leitura do *trace* de vídeo), o Gerenciador faz a verificação para saber se este bloco que foi solicitado se encontra no *lookahead buffer*, ou seja, se já estava armazenado no Gerenciador. Se o bloco estiver neste *buffer*, o Gerenciador faz o envio imediato do bloco para o cliente, decrementa a variável que representa quantos blocos estão armazenados no Gerenciador e faz o pedido de um bloco para o *Storage Server* para preencher a posição deste *buffer* que ficou vazia. Se o bloco não estiver presente no *lookahead buffer* do Gerenciador, a variável que

representada a quantidade de blocos solicitados é incrementada e o Gerenciador envia um pedido deste bloco para os *Storages Servers*, através do objeto Roteador, para que algum deles envie o bloco diretamente ao cliente. O objeto Gerenciador é quem faz o encaminhamento deste bloco, mas só armazena blocos que foram solicitados para preenchimento do *lookahead buffer* após a liberação de uma posição.

- **Objeto Cliente**

  O objeto Novo Cliente representa a carga gerada pelo cliente que tem seu comportamento analisado e está assistindo um vídeo a partir de um *trace* gerado de uma aula real do curso oferecido pelo CEDERJ. A metodologia utilizada para a criação dos *traces* e os vídeos escolhidos são detalhados na próxima seção. A distribuição utilizada para representar os pedidos gerados pelo cliente é a distribuição FILE, cujo parâmetro é o arquivo de *trace* citado posteriormente, que contém os intervalos entre os pedidos de leitura do bloco do vídeo. A cada bloco lido, o cliente faz uma nova solicitação ao Gerenciador. Este objeto possui as seguintes variáveis de estado:

  - Um vetor que representa o *playout buffer* no cliente, onde são armazenados os blocos de dados que foram pedidos e foram recebidos do Gerenciador;
  - Uma variável que indica qual é o bloco que está sendo visualizado no momento;
  - Uma variável que indica quantas posições do *buffer* estão preenchidas;

- **Objetos *Storage Servers***

  Estes objetos representam os servidores de armazenamento do sistema. São eles que contêm os blocos do vídeo e que fazem o envio dos dados para os clientes, no caso do bloco não estar no *lookahead buffer* do Gerenciador. A distribuição Normal foi usada para representar o atraso no disco. Ela foi parametrizada de acordo com medições feitas nos discos das máquinas.

A Tabela 5.1 mostra os valores obtidos para média e variância do atraso em disco, através de medições usando o programa *hdparm*.

Storage	Média (ms)	Variância
UFRJ1	2.3809	$1.5117e^{-}3$
UFRJ2	2.6737	$2.8596e^{-}3$
UFF	2.4449	$1.7684e^{-}3$
Fiocruz	2.1929	$3.3166e^{-}4$
UFMG	2.3041	$8.8082e^{-}3$

Tabela 5.1: Atraso dos discos coletados com o *hdparm*

O objeto *Storage Server* possui uma variável de estado que representa o tamanho da fila de requisições que é incrementada toda vez que chega um pedido vindo do Gerenciador e é decrementada a cada evento de serviço.

Quando algum dos *Storages* recebe um pedido de envio de bloco, este objeto verifica se este bloco já havia sido pedido pelo cliente que está sendo analisado. Em caso afirmativo, é enviada uma mensagem de envio do bloco e a variável que indica a quantidade de blocos que ainda precisam ser enviados para o cliente analisado é decrementada. Caso contrário, o bloco é armazenado no *lookahead buffer* no nó gerenciador e o contador, presente neste nó, que representa a quantidade de blocos armazenados e que ainda precisam ser enviados, é incrementado.

A cada bloco servido, este objeto informa ao objeto Roteador (que faz parte do sistema do lado do servidor) para que o mesmo decremente sua tabela contendo o tamanho das filas de cada um dos nós de armazenamento. É através desta tabela que o objeto roteador sabe para qual *Storage* enviar seu pedido quando há mais de uma cópia do bloco no sistema. É onde ficam armazenadas as informações de onde se encontram determinados blocos. Desta forma, conseguimos um balanceamento de carga maior, de acordo com a quantidade de cópias de um determinado bloco no sistema.

Como estamos interessados em analisar somente o comportamento de um cliente em específico, quando um pedido feito pelo objeto tráfego de fundo é servido pelo

objeto *Storage Server*, ele é apenas descartado, não sendo enviado para nenhum outro objeto, e uma posição na fila do objeto *Storage Server* que serviu a requisição é liberada. Com os pedidos feitos pelos clientes agregados conseguimos preencher as filas nos *Storages* como se existissem vários clientes no sistema e o descarte destes pedidos depois de serem servidos não impacta as redes do sistema.

5.1.2 Segundo Cenário: Modelo Sequencial com Replicação

No segundo cenário, modelamos o sistema com replicação igual a 2, ou seja, o sistema possuirá duas cópias do vídeo.

A diferença principal deste modelo para o modelo do primeiro cenário é o objeto Roteador. Quando um pedido do cliente chega, o objeto sorteia aleatoriamente dois *Storage Servers* (já que estamos utilizando nível de replicação 2) entre todos que contêm as cópias do bloco. O pedido é então encaminhado para o que tiver a menor fila. Este número de *Storages* é passado para o modelo através de uma variável que representa o número de replicação usado.

Uma outra diferença importante é uma simplificação feita neste modelo de modo a diminuir o tempo de simulação. O objeto tráfego de fundo gera muitos eventos, pois representa o tráfego agregado de uma população da ordem de milhares de clientes. No modelo original os nós de armazenamento possuem uma fila única contendo as requisições dos clientes. Esta fila, contendo tanto pedidos que chegam do objeto tráfego de fundo quanto do cliente que está sendo analisado, é servida pelo mesmo evento de serviço. Com a simplificação introduzida no modelo, o evento de serviço dos discos foi dividido em dois: (a) um evento para representar o serviço dos pedidos do cliente analisado, com a média e a variância iguais as do modelo original, e (b) um evento para representar o serviço de um *batch* de pedidos de clientes do restante da população (pedidos gerados pelo objeto tráfego de fundo). Os parâmetros do evento que representa o serviço de um *batch* de pedidos de clientes, que são a média e variância da distribuição normal, foram calculados usando a propriedade de que uma soma de variáveis aleatórias com distribuição normal também é uma variável

aleatória com distribuição normal [52]. Desta forma, dado que servimos *batches* de pedidos, foi possível diminuir proporcionalmente a taxa de geração de pedidos do objeto tráfego de fundo, reduzindo o tempo de simulação.

A implementação no modelo foi feita da forma descrita a seguir. Os eventos de serviço estão condicionados ao valor do primeiro bloco da fila única no nó de armazenamento. A cada execução de um destes eventos é verificado se o valor do bloco é igual ou diferente de zero. Se for igual a zero, significa que o pedido foi feito pelo objeto tráfego de fundo, e então será servido pelo evento (b). Caso o valor do primeiro bloco da fila seja diferente de zero, sabemos que este bloco foi requisitado pelo cliente analisado e, portanto, ele é servido pelo evento (a).

5.1.3 Terceiro Cenário: Modelo Interativo

Nos cenários 1 e 2 consideramos usuários sequenciais. Neste cenário iremos considerar usuários interativos. A seguir descreveremos somente os objetos que foram modificados do modelo sequencial.

- **Objeto Cliente**

  O objeto cliente se comporta como ilustrado na Figura 5.2. A cada requisição de um determinado bloco, o cliente faz uma busca em seu *buffer*. Se o bloco estiver presente, o cliente consome este bloco e faz a requisição de um novo ao Gerenciador. Se o bloco não se encontra presente, o cliente esvazia seu *buffer* e faz o pedido ao Gerenciador dos blocos necessários para preencher as posições vazias e entra em um estado de espera para dar algum tempo dos blocos serem servidos pelos nós de armazenamento ou serem enviados pelo gerenciador se este já os possuir. O cliente aguarda um tempo específico (aqui estamos considerando um tempo de 2 segundos) para novamente fazer a verificação se o bloco requisitado já chegou. O pedido do bloco ao gerenciador só é feito se o mesmo não havia sido pedido, pois o cliente armazena em um vetor os pedidos já feitos, para não repetir o mesmo pedido, o que geraria carga desnecessária no sistema. Se depois de esperado o tempo necessário e

feita uma nova verificação o cliente constatar que o bloco ainda não chegou, ele incrementa uma recompensa de perda de bloco.

Este objeto possui as seguintes variáveis de estado complementares ao modelo sequencial:

- Um vetor com tamanho igual ao *buffer* do cliente, que representa o que já foi pedido pelo mesmo. Este vetor é consultado quando for feita uma verificação da presença do bloco requisitado. Se este não estiver presente, e o cliente constata que os blocos já foram pedidos, ele não os pede novamente;
- Uma variável que indica que o bloco não se encontra no *buffer* e o cliente precisa entrar em um estado de espera;
- Uma variável que indica quantas vezes o cliente verificou se o bloco estava presente no *buffer*, para incrementar a recompensa de perda;

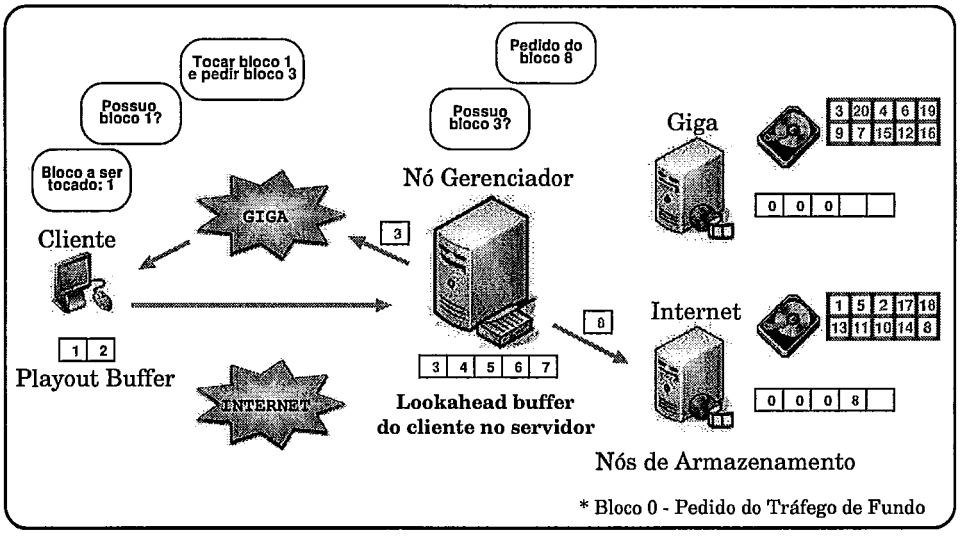


Figura 5.2: Comportamento do Cliente Interativo

O *playout buffer* do cliente já inicia completo, ou seja, quando a simulução inicia todos os *buffers* do sistema estão preenchidos, para evitar a latência inicial do pedido de todos os blocos, apenas por uma questão de simplicidade. O comportamento do cliente é lido de um *trace* que possui os números dos blocos que foram requisitados durante a visualização de um vídeo, como será

mostrado na seção 5.3. A partir do número do bloco obtido, o cliente verifica se o mesmo encontra-se em *buffer* para então exibir ou pedir este bloco.

- **Objeto Nó Gerenciador**

  O objeto Nó Gerenciador possui as mesmas características do anterior, só possuindo uma variável de estado que representa um vetor com o que já foi pedido para os nós de armazenamento, depois que o cliente fez alguma requisição. Este vetor é consultado quando o cliente faz um pedido que poderia já ter sido solicitado pelo Gerenciador aos nós de armazenamento. Por exemplo, supondo que o nó gerenciador possui um *buffer* de 5 posições, se o cliente pede o bloco 3 ao nó gerenciador e este não o possui, isto quer dizer que ele percorreu todo o *buffer* e não encontrou o bloco 3. Com isto, ele irá pedir os blocos 3, 4, 5, 6 e 7 para os nós de armazenamento. Se depois o cliente fizer o pedido do bloco 4 ao Gerenciador e este ainda não tiver chegado, o Gerenciador não pedirá o bloco 4 novamente, pois terá visto no vetor citado que o bloco já foi pedido e está sendo aguardado.

- **Objeto Tráfego de Fundo** A grande diferença deste objeto para o do modelo sequencial é com relação à carga gerada pela representação dos clientes agregados. Para uma aproximação mais realista, depois de verificar que o cliente interativo faz cerca de 30% mais pedidos ao servidor do que o usuário sequencial, aumentamos a carga do objeto tráfego de fundo nesta mesma proporção, significando um aumento relativo ao maior número de pedidos feitos por parte de um usuário interativo. Assim, os clientes agregados estariam se comportando, mesmo que de forma abstrata, como um usuário interativo, fazendo mais requisições e, consequentemente, gerando mais carga para os nós de armazenamento.

5.2 Medição do Atraso nas Redes

Visando a parametrização dos modelos propostos estimamos as características do atraso do caminho entre a UFMG e a rede Giga.

Para tal estimativa, foram realizadas medições com o Gerador de Tráfego do Tangram-II, nos dois sentidos, como mostra a Figura 5.3. A métrica analisada foi o retardo fim-a-fim entre as instituições, ou seja, o tempo que um pacote leva para sair de sua origem e chegar ao seu destino. Este atraso pode ser calculado tanto para uma via, quanto para duas. Neste trabalho utilizamos o atraso do caminho da UFMG para a UFRJ, que é a instiuição que possui todos os clientes.

Figura 5.3: Sentido da Medição do Atraso nas Redes

Um dos objetivos desta medição é obter a melhor distribuição que represente esta métrica e seus parâmetros. Com estes dados em mãos podemos parametrizar os respectivos objetos de acordo com dados reais.

Com a medição foi possível encontrar a melhor distribuição para o atraso da rede que representa o canal por onde passam os blocos que são servidos pelo nó de armazenamento que se encontra na UFMG, instituição que não faz parte da rede Giga. A distribuição escolhida foi a Gaussiana, que foi a que mais se ajustou aos resultados obtidos.

Na Tabela 5.2 mostramos a média e variância obtidas nas medições nas duas redes.

Rede	Média (ms)	Variância
Giga	3.948172000	276.324355987
Internet	31.4854913	1704.474214098

Tabela 5.2: Atrasos entre as Instituições

Esta medição foi realizada no período de férias escolares, ou seja, a média do atraso entre a UFRJ e a UFMG no presente momento é, provavelmente, maior do que na época em que foi feita a medição. O atraso na rede Giga não diferiu do

encontrado anteriormente, devido ao fato de a rede ainda não se encontrar com alta utilização. Porém, esta diferença entre os valores do atraso para a UFMG não onera nossos resultados, pois o atraso das redes não influencia tanto o resultado quanto se todos os pedidos de todos os clientes estivessem passando por ela. Assumindo que a rede não é o gargalo do nosso sistema.

5.3 Geração dos *Traces*

Para a geração do *trace* usado para simular o comportamento de usuários sequenciais utilizamos um programa implementado na linguagem C que extrai o tempo que um aplicativo real leva para consumir determinado bloco de um objeto. O programa faz a extração do tempo que a ferramenta Mplayer leva para tocar um bloco de 128KBytes. A partir do tempo de consumo de um bloco é gerado um *trace*, extraído de um vídeo de uma aula oferecida pelo CEDERJ, que contém o intervalo entre pedidos de blocos de clientes (o cliente do RIO solicita um novo bloco assim que um bloco do *buffer* é consumido). Este *trace* é utilizado pela distribuição FILE da ferramenta Tangram-II, para simular os pedidos feitos pelo novo cliente.

A Figura 5.4 mostra como funciona o algoritmo de extração dos tempos de consumo do vídeo, para a confecção do *trace* utilizado nas simulações.

```
Função extrai_duração_blocos( Nome_vídeo)
   criação do pipe;
   executa mplayer;
   se abre_arquivo( Nome_vídeo)
     bloco = 0;
     tempo_anterior = tempo_atual;
     enquanto não for final do arquivo
       lê um bloco de dados;
       escreve bloco no pipe;
       duração_bloco = tempo_atual - tempo_anterior;
       imprime no trace o valor duração_bloco;
       tempo_anterior = tempo_atual;
       bloco = bloco + 1;
     fim-enquanto
   fecha_arquivo();
   fim-se
fim-função
```

Figura 5.4: Algoritmo de extração do tempo de duração dos blocos dos objetos multimídia

O programa lê os blocos de dados do objeto e repassa para o aplicativo MPlayer

responsável pela reprodução da mídia. O mecanismo utilizado para a comunicação entre os processos é o *pipe*.

O *trace* utilizado no modelo interativo foi extraído de *logs* dos alunos do CEDERJ. Foram extraídos todos os blocos que possuem a ação *play* e passados pelo modelo através de um *plugin* desenvolvido para a ferramenta Tangram-II, que possibilita a manipulação de inteiros e funções. Não entraremos em detalhes com relação ao *plugin*, desenvolvido em outro trabalho do laboratório. A Figura 5.5 mostra o formato do *log* utilizado.

```
1578 ──► Número de linhas do arquivo
1    ──► Primeiro bloco a ser tocado
2    ──► Segundo bloco a ser tocado
3
4
5    ──► Interação (salto do bloco 5
10                  para o bloco 10)
256
257
258
312
318
.
.
.
3958 ──► Último bloco a ser tocado - linha 1578
```

Figura 5.5: Formato do log passado para o modelo interativo

O *plugin* carrega (em tempo zero de simulação) a sequência númerica passada através do número de linhas do arquivo de *log* mostrado, o que nos dá a quantidade de todos os blocos que serão passados para o modelo. Como pode ser observado na figura, ele irá carregar um arquivo contendo 1578 linhas de números inteiros, que representam o número dos blocos. O tempo de solicitação dos blocos não é passado pelo *log*, pois este é um parâmetro presente no modelo.

5.4 Resultados

Nesta seção apresentamos os resultados obtidos com os modelos de simulação descritos na seção 5.1. Experimentos foram realizados tanto por simulação quanto em ambiente real, de forma a verificar o número de clientes suportados pelo sistema, a escalabilidade do servidor, o ganho obtido com a adição de novos nós de armaze-

namento e com o uso de replicação, e o comportamento do sistema de acordo com o tipo de acesso do cliente, se sequencial ou interativo.

5.4.1 Objetivo e Medidas Obtidas nos Experimentos

O objetivo geral dos experimentos é analisar o desempenho do servidor RIO, quando submetido a cargas reais, em diferentes cenários. Aqui estamos analisando um ambiente heterogêneo, com o comportamento do cliente podendo ser sequencial ou interativo para o acesso ao conteúdo multimídia. Analisaremos também as questões supracitadas.

A métrica analisada, o *Goodness*, nos dá a fração de blocos exibidos pela quantidade de blocos que foram requisitados, ou seja, o percentual de *bytes* recebidos dentre todos que foram requisitados ao Gerenciador. O *Goodness* médio do sistema foi calculado através da análise conjunta do *Goodness* de cada cliente, dentre uma população de clientes, onde calculamos a média e variância da população inteira. Com esta métrica, conseguimos mensurar quanto o sistema suporta em número de clientes, assumindo que o mínimo que o cliente necessita para ter uma boa QoS é possuir um *Goodness* médio acima de 80%.

$$Goodness = \frac{BlocosExibidos}{BlocosRequisitados}$$

5.4.2 Primeiro Cenário: Modelo Sequencial sem Utilização de Replicação

O primeiro cenário é caracterizado pelo acesso sequencial ao vídeo, sem que haja replicação no sistema, ou seja, existe apenas uma cópia do vídeo distribuída entre todos os nós de armazenamento.

Cada simulação foi executada com 5 rodadas e intervalo de confiança de 95%, ou seja, este intervalo de confiança nos garante que temos um nível de confiança de 95% em que a média se encontra no intervalo observado.

O tempo de duração de cada simulação foi de 15 minutos, que é o tempo total do vídeo utilizado. Assim, a simulação representa o cliente assistindo a um vídeo do início ao fim. O vídeo utilizado nestas simulações possuem taxa de 1,25Mbps, em média.

Foram feitas simulações utilizando-se de 1 a 5 nós de armazenamento, pois esses valores levam em conta as limitações no equipamento disponível no ambiente real. Num primeiro momento, supomos que os discos seriam o gargalo do sistema e parametrizamos o modelo com os valores mostrados na tabela 5.1.

O incremento do número de clientes durante as simulações foi de 10 clientes por experimento, de acordo com o número de nós de armazenamento.

Resultados e Análise do Primeiro Cenário

No primeiro cenário esperamos obter ganhos de desempenho com a adição de novos nós de armazenamento. A inclusão de um nó de armazenamento em uma rede lenta (este nó também deve processar cerca de 20% de todo o tráfego) não deve prejudicar este cenário, devido ao fato do atraso maior de transmissão na rede não ser o gargalo do nosso sistema, mas sim o acesso ao disco.

Na Figura 5.6 mostramos o resultado da simulação com 1 nó de armazenamento na rede Giga e nas Figuras 5.7, 5.8 e 5.9 mostramos, respectivamente, o resultado da adição gradativa de nós de armazenamento na rede Giga.

O ganho obtido foi proporcional à capacidade do disco, pois o disco que possui maior variância possui fila maior. Com isto, a medida que o número de clientes aumenta, este disco passa a onerar mais o sistema.

O resultado apresentado na Figura 5.10 é o mais interessante neste cenário, pois através dele verificamos que a adição de um nó de armazenamento num caminho mais lento, sem nenhum tipo de balanceamento de carga, não impacta a capacidade geral do sistema. Com isto, a inclusão do novo nó na UFMG contribui com o aumento do número de clientes atendidos. Logo, para um determinado número de clientes e valor de *goodness*, é vantajoso adicionar um nó num *link* mais lento, ao

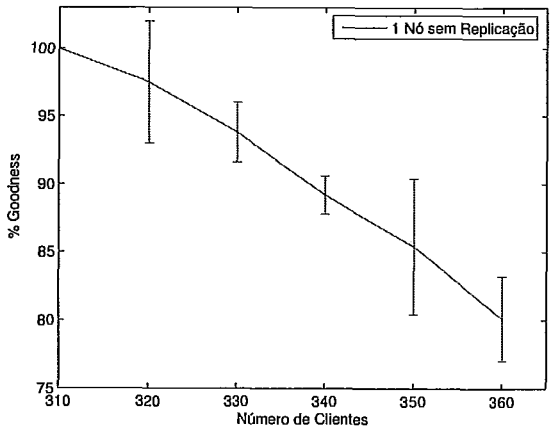


Figura 5.6: 1 Nó de Armazenamento na rede Giga sem replicação

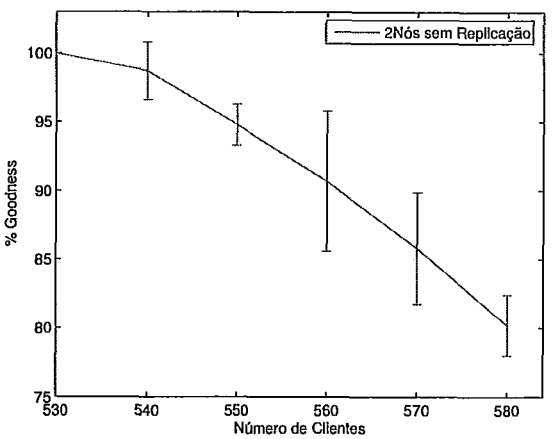


Figura 5.7: 2 Nós de Armazenamento na rede Giga sem replicação

invés de atender a população com 4 nós na rede Giga.

Conforme o esperado, a adição de novos nós de armazenamento possibilitou o atendimento de um número maior de clientes, sem perda de qualidade. A leve inclinação da curva apresentada na Figura 5.11 com a adição do quinto nó de armazenamento se deve ao fato do quinto nó ser a UFMG, que é a instituição que possui a maior variância no tempo de serviço, o que aumenta sua fila de pedidos.

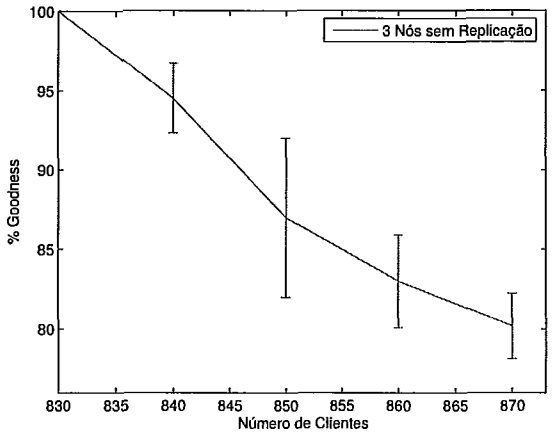


Figura 5.8: 3 Nós de Armazenamento na rede Giga sem replicação

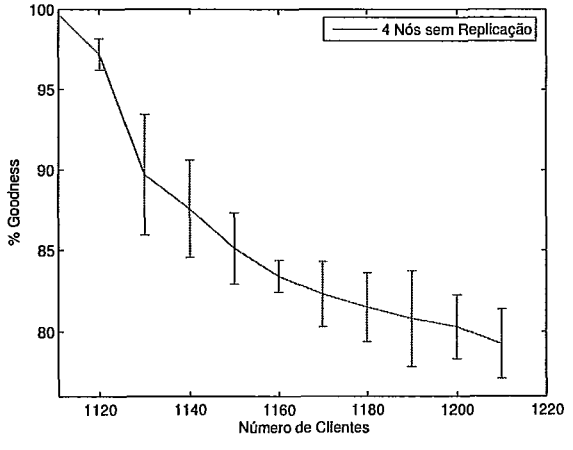


Figura 5.9: 4 Nós de Armazenamento na rede Giga sem replicação

5.4.3 Segundo Cenário: Modelo Sequencial com Utilização de Replicação

Neste cenário utilizamos a mesma configuração do sistema RIO do cenário anterior, porém com o uso de replicação. O número de replicações utilizado no modelo foi de 2. Lembrando que estamos utilizando replicação total, ou seja, o sistema possui duas cópias completas do vídeo. Vale lembrar também que a simulação com um nó de armazenamento não faz sentido neste cenário, já que não seria possível a utilização de replicação.

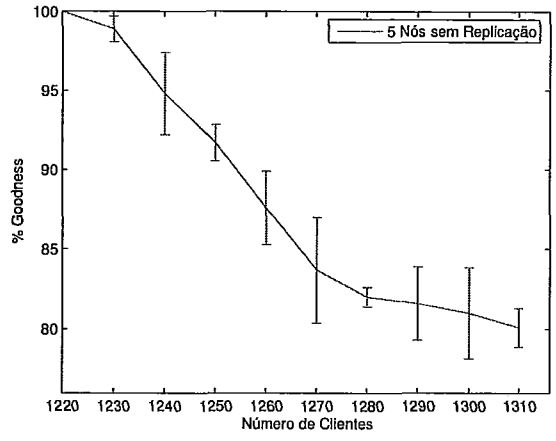


Figura 5.10: 5 Nós de Armazenamento - 4 na rede Giga e 1 na UFMG sem replicação

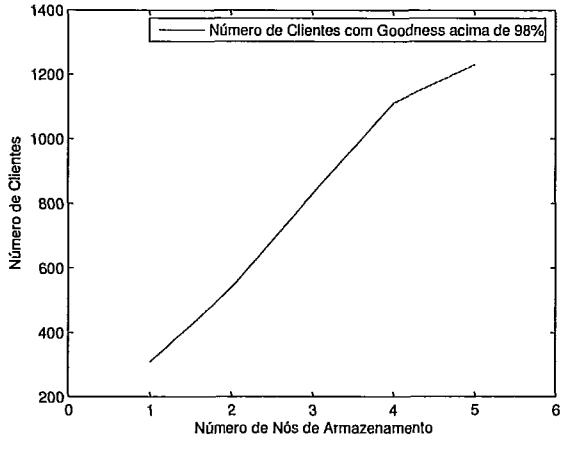
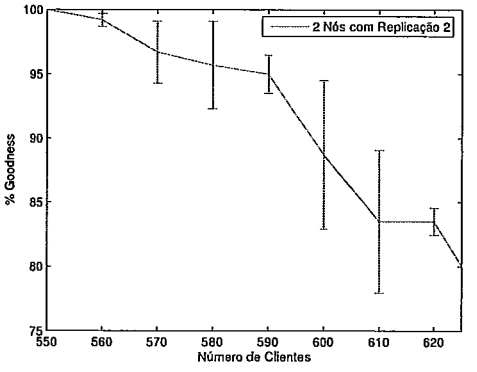
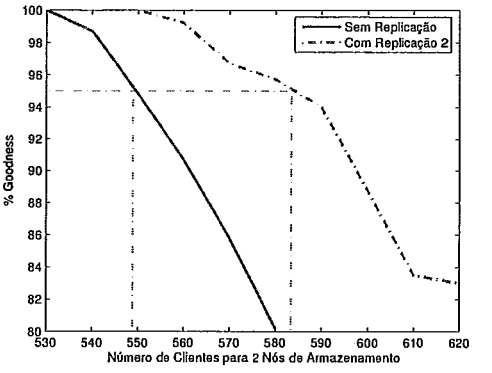


Figura 5.11: Número de Clientes x Nós de Armazenamento sem Utilização de Replicação

Na Figura 5.12, que apresenta o resultado das simulações com 2 nós de armazenamento com o uso de replicação e a comparação com a curva sem o uso de replicação, podemos verificar que o ganho com relação ao cenário sem o uso de replicação não é muito grande, porém é vantajoso, já que a confiabilidade do sistema aumenta, por possuirmos duas cópias do vídeo no sistema. O ganho obtido é de aproximadamente 6% para *goodness* acima de 95%. Este ganho é relativo ao fato de os dois nós possuírem características similares, ou seja, com ou sem o uso de replicação a carga média direcionada para cada um dos nós é próxima nos dois cenários.



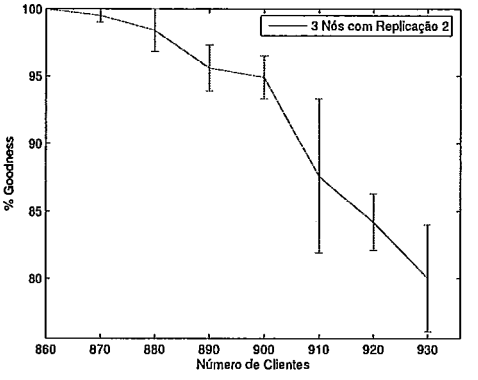
(a) 2 Nós com Replicação.



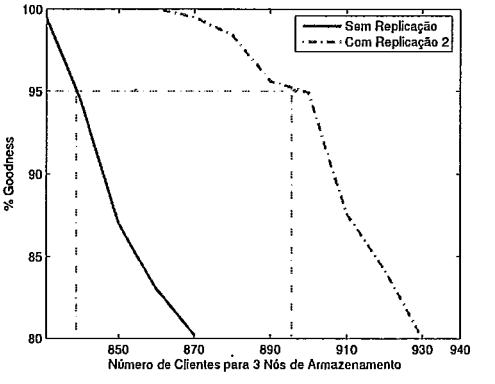
(b) 2 Nós Com e Sem Replicação.

Figura 5.12: Comparação do ganho com o uso de replicação 2 para 2 nós de armazenamento.

Já com a adição de um novo nó de armazenamento, como mostra a Figura 5.13, percebemos um ganho aproximado de 7% em termos do número de clientes atendidos para valores de *goodness* acima de 95%.



(a) 3 Nós com Replicação.

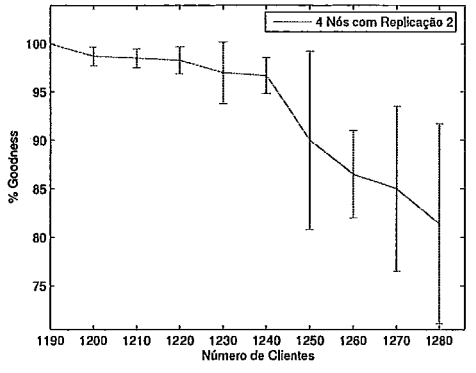


(b) 3 Nós Com e Sem Replicação.

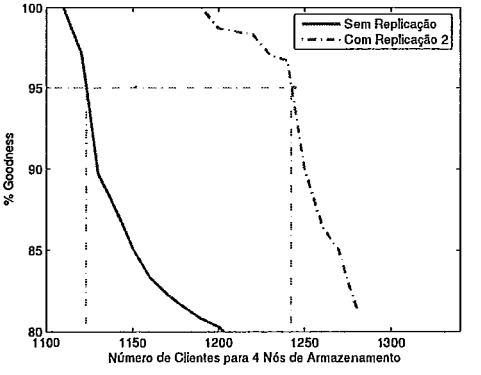
Figura 5.13: Comparação do ganho com o uso de replicação 2 para 3 nós de armazenamento.

Na Figura 5.14, temos o gráfico que representa a quantidade de clientes atendidos com 4 nós de armazenamento na rede Giga, utilizando-se replicação. Para esta configuração o ganho obtido foi em torno de 11% para *goodness* acima de 95%.

Conforme esperado, notamos que o aumento do número de nós de armazenamento na rede Giga com replicação propiciou um aumento no número de clientes atendidos em até 11%. O motivo para este aumento é que o impacto da variabilidade do tamanho das filas dos nós de armazenamento diminui devido ao balanceamento de carga, e com isto é possível atender mais clientes para um dado valor de *goodness*.



(a) 4 Nós com Replicação.

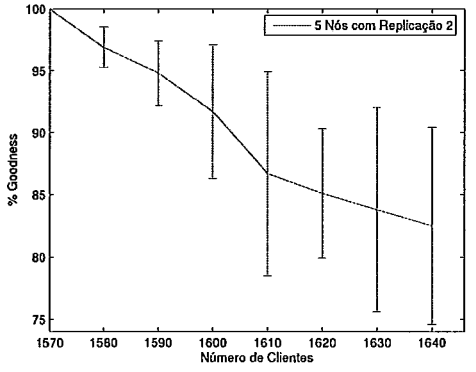


(b) 4 Nós Com e Sem Replicação.

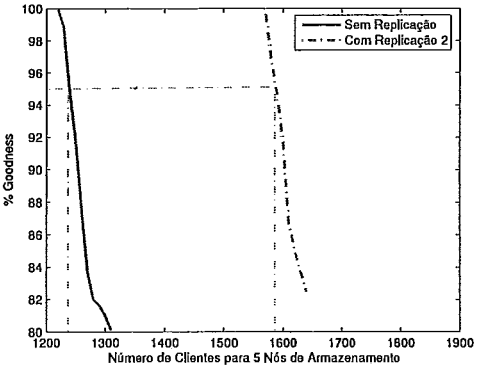
Figura 5.14: Comparação do ganho com o uso de replicação 2 para 4 nós de armazenamento.

Na Figura 5.15, temos a representação da quantidade de clientes que é possível conseguir com 5 nós de armazenamento, 4 na rede Giga e 1 na UFMG, também utilizando-se replicação. Constatamos aqui uma maior diferença no ganho com o uso de replicação, cerca de 27%. Esta maior diferença se deve ao fato do nó da UFMG possuir o maior valor de variância no tempo de acesso a disco. Com isto, a medida que aumentamos o número de clientes, no caso sem replicação estamos enviando cerca de 20% dos pedidos para a UFMG, que passa a onerar um pouco mais o sistema. Já com a utilização de replicação, a UFMG passa a servir de forma mais eficiente, já que só irá receber pedidos quando sua fila estiver com tamanho satisfatório.

Na Figura 5.16, mostramos a comparação do número de clientes com o aumento gradativo do número de servidores de armazenamento. Conforme o esperado, a adição de novos nós de armazenamento possibilitou o atendimento de um número



(a) 5 Nós com Replicação.



(b) 5 Nós Com e Sem Replicação.

Figura 5.15: Comparação do ganho com o uso de replicação 2 para 5 nós de armazenamento.

maior de clientes, sem perda de qualidade. Até mesmo a adição de um nó na UFMG, enviando dados através da Internet, contribuiu para um aumento no número de clientes atendidos.

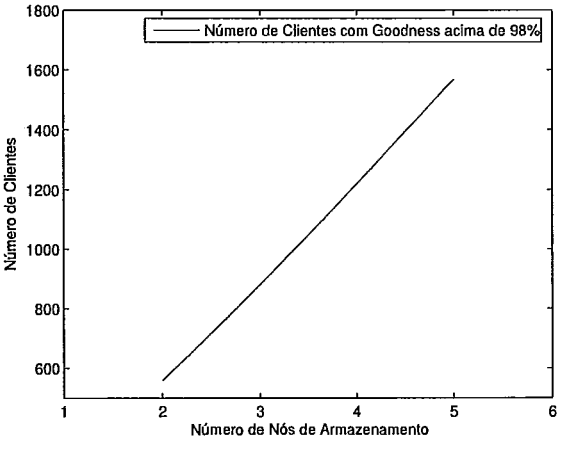


Figura 5.16: Número de Clientes x Nós de Armazenamento - Com replicação

A medida que aumentamos o número de clientes, os nós de armazenamento ficam mais sujeitos a rajadas de pedidos. Por este motivo, a adição de novos nós com replicação é vantajosa, pois a carga é distribuída entre os nós, o que diminui a variabilidade das filas e aumenta a capacidade do sistema.

Como podemos observar, o ganho ao acrescentar um nó de armazenamento na UFMG (em um *link* mais lento) sem o uso de replicação não foi tão grande quanto acrescentar um nó na UFMG com o uso de replicação, porém um ganho razoável pode ser observado com a adição deste nó.

Como visto anteriormente, no cenário sem replicação 20% dos pedidos são direcionados para o disco da UFMG (independente do tamanho da fila deste disco) e o atraso destes pedidos certamente é maior do que os atendidos pelos nós na rede Giga, além de eles passarem por um enlace de baixa velocidade até chegar ao cliente, o que pode causar mais atrasos no cliente, apesar de este não ser o gargalo do nosso sistema. Já no cenário com replicação, os pedidos são atendidos pelo nó de armazenamento com menor fila. Como a fila do nó de armazenamento da UFMG é maior que a dos outros nós, menos pedidos serão enviados para este nó do que no cenário sem replicação.

5.4.4 Terceiro Cenário: Modelo Interativo

No terceiro cenário temos um modelo representando um cliente com acesso interativo no sistema. O comportamento do cliente foi representado através de um *log* extraído das aulas do curso oferecido pelo CEDERJ.

O vídeo utilizado tanto na simulação quanto nos testes reais possui taxa de 1.25Mbps. A justificativa para a escolha desta taxa é que uma boa parte dos vídeos oferecidos pelo CEDERJ possui taxa de 1.25Mbps.

O *log* utilizado para alimentação do modelo possui nível médio de interatividade, ou seja, o usuário executa algumas interações, mas também permanece assistindo ao vídeo em sequência. O vídeo possui um pouco mais de 1900 blocos e o número de ações interativas é próximo a 100.

A Figura 5.17 mostra o resultado da simulação do modelo interativo para um nó de armazenamento e as Figuras 5.18, 5.19, 5.18 e 5.21 mostram a configuração para 2, 3, 4 e 5 nós de armazenamento, respectivamente.

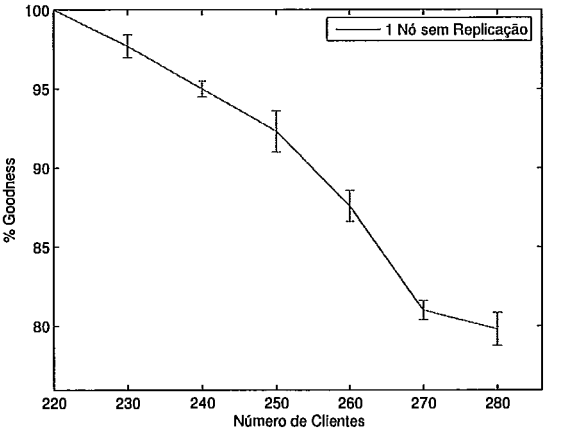


Figura 5.17: Resultado do Modelo Interativo com 1 Nó de Armazenamento sem Replicação

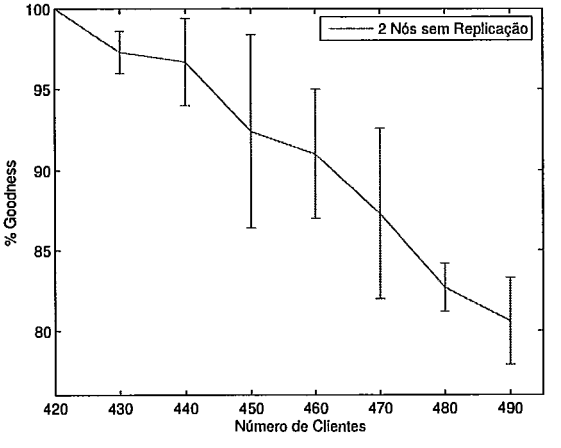


Figura 5.18: Resultado do Modelo Interativo com 2 Nós de Armazenamento sem Replicação

A partir da observação dos gráficos para 4 e 5 nós de armazenamento é possível verificar que a adição de um nó na UFMG sem o uso de replicação trouxe ganhos ao sistema conforme o cenário para usuários sequenciais.

Podemos verificar que este cenário não apresentou uma piora significativa com relação ao cenário sequencial em termos do número de clientes atendidos para um dado valor de *goodness*. Porém, a comparação dos dois modelos é injusta, pois neste modelo o cliente interativo está mais tolerante a falhas, ou seja, no momento de

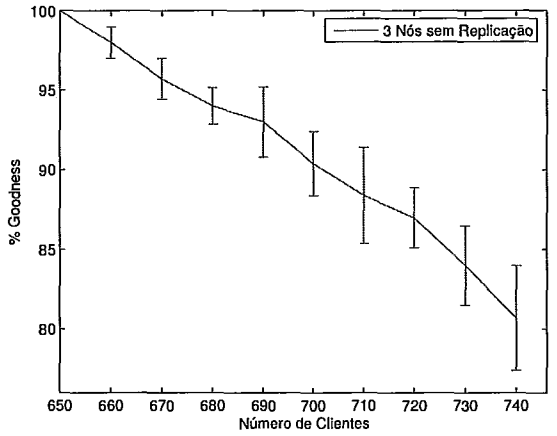


Figura 5.19: Resultado do Modelo Interativo com 3 Nós de Armazenamento sem Replicação

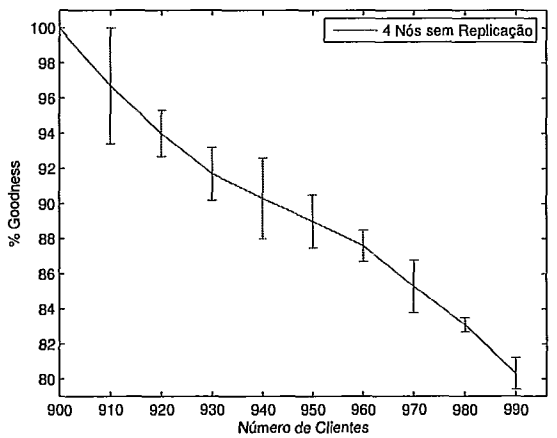


Figura 5.20: Resultado do Modelo Interativo com 4 Nós de Armazenamento sem Replicação

exibir um bloco, ele aguarda um tempo específico para que o nó gerenciador possa enviar o bloco, diferentemente do modelo do cliente sequencial onde se o bloco não está presente no momento de sua exibição, ele o considera como perdido. O cliente interativo foi modelado desta forma baseado na versão mais atual do cliente do servidor RIO.

Outro fator que deve ser levado em consideração é a característica não interativa do objeto tráfego de fundo, já que este se comporta quase da mesma forma que

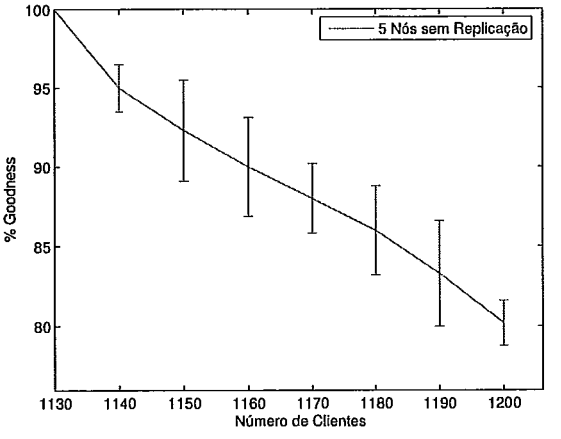


Figura 5.21: Resultado do Modelo Interativo com 5 Nós de Armazenamento sem Replicação - 4 na Rede Giga e 1 na UFMG

no modelo sequencial. O aprimoramento deste objeto foi sugerido como trabalho futuro, porém para uma aproximação mais realista, aumentamos a taxa da geração de pedido deste objeto em 30%, como foi dito na seção 5.1.3.

A Figura 5.22 apresenta de forma resumida o aumento gradativo no número de clientes para as configurações dos números de Nós de Armazenamento, variando de 1 a 5, sem a utilização de replicação.

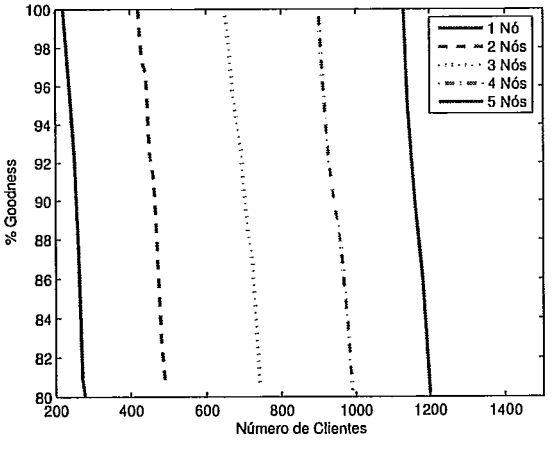


Figura 5.22: Resultados para 1,2,3,4 e 5 Nós de Armazenamento sem Replicação

5.4.5 Resultados em Ambiente Real

A tabela 5.3 mostra um resumo dos resultados obtidos em ambiente real para acesso sequencial, e a Tabela 5.4 mostra um resumo dos resultados obtidos em ambiente real para acesso interativo. Na primeira coluna temos o número de nós de armazenamento utilizados nos experimentos. Na segunda coluna temos o tipo de acesso, se este é sequencial ou interativo. Na terceira coluna apresentamos o número total de clientes disparados e na quarta coluna o número de usuários que conseguiram se conectar ao servidor e conseguiram chegar ao final do experimento ainda conectados ao servidor. Na quinta e sexta colunas mostramos o *Goodness* médio e a variância, respectivamente, obtidos de todos os clientes conectados.

N. *Storage*	Tipo	N. Clientes	Conectados	*Goodness* médio	Variância
1	Seq.	400	391	99.5882	0.683861
1	Seq.	550	507	97.6028	44.1013
2	Seq.	450	450	99.8167	0.207359
2	Seq.	500	492	99.2907	0.210666
2	Seq.	550	502	98.9734	15.4524

Tabela 5.3: Resultados em Ambiente Real com Clientes Assistindo a um Vídeo de Forma Seqüencial

N. *Storage*	Tipo	N. Clientes	Conectados	*Goodness* médio	Variância
1	Int.	390	390	99.959	0.0548678
1	Int.	420	418	99.9618	0.0368158

Tabela 5.4: Resultados em Ambiente Real com Clientes Assistindo a um Vídeo de Forma Interativa

Podemos verificar uma variação em termos do número de clientes conectados. Esta variação pode estar relacionada com a utilização do laboratório no momento dos experimentos.

Comparando os resultados dos experimentos com os resultados do modelo temos

que:

- Para o modelo sequencial foram atendidos 391 clientes com 99% de *goodness* médio e 507 clientes com 97% de *goodness* médio com um nó de armazenamento. Como o número obtido na simulação foi menor do que o obtido em ambiente real resolvemos fazer uma adaptação do modelo, considerando a placa de rede o gargalo do sistema e não mais o disco.

- Já com dois nós, com o cliente sequencial com 99% de *goodness* médio foram atendidos 502 clientes. Este resultado pode estar relacionado com a restrição no número de máquinas para emular os clientes.

- Para o modelo interativo temos um resultado bem próximo ao obtido em ambiente real.

5.4.6 Reparametrização do Modelo Sequencial

Devido ao fato de termos conseguido mais clientes no ambiente real do que no modelo de simulação, como foi mostrado na seção 5.4.5, fizemos uma variação na distribuição do tempo de serviço dos nós de armazenamento do modelo sequencial, para mostrar que o modelo é facilmente parametrizável, apesar de que a idéia de um modelo de simulação seja capturar as tendências do sistema real.

Agora assumimos que o disco da máquina não é mais o gargalo do sistema. Como não nos aprofundamos nas limitações do sistema operacional, supomos um novo gargalo, a placa de rede das máquinas onde se encontram os nós de armazenamento. Medições foram feitas em [22] utilizando máquinas no mesmo laboratório em que fizemos nossos experimentos. Naquele trabalho, foram realizados experimentos onde foram verificadas as taxas de envio das placas de rede. Como estamos interessados em um ambiente com capacidade giga, utilizaremos os valores encontrados para as máquinas conectadas à porta *Gigabit* do *Switch* disponível naquele experimento. Duas máquinas foram utilizadas e as taxas encontradas foram de 648Mbps e 638Mbps. Utilizaremos estes valores para parametrizar o modelo. Como não havia

mais nenhum tipo de informação acerca da medição como, por exemplo, a variância desta capacidade, utilizaremos esta taxa em uma distribuição exponencial para representar o intervalo do tempo de serviço dos blocos.

De posse desta nova configuração realizamos novas simulações com 1 e 2 nós de armazenamento. A idéia inicial era tentar validar este modelo com o número de usuários para dois nós, porém devido à limitação do número de máquinas clientes não foi possível realizar o experimento real com um número grande de clientes. Como dito anteriormente, a quantidade de máquinas disponíveis para emular os clientes não foi suficiente para alcançar um número tão grande de clientes.

A Figura 5.23 mostra o gráfico que representa o número de clientes versus o *goodness* médio, supondo-se que a placa de rede da máquina onde se encontra o nó de armazenamento é o gargalo do sistema.

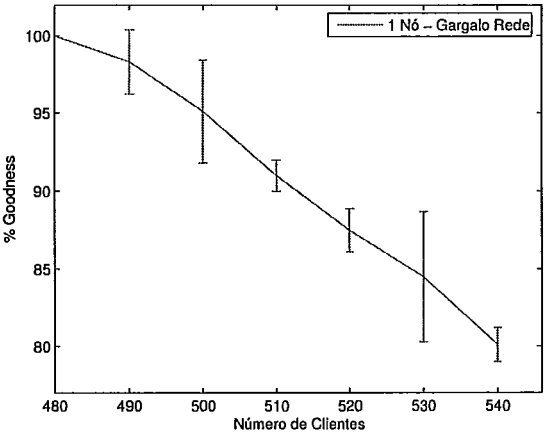


Figura 5.23: 1 Nó de Armazenamento na Rede Giga supondo que o gargalo do sistema é a placa de rede

Como pode ser verificado, conseguimos um ganho considerável no número de clientes com esta nova hipótese. Alguns fatores podem ter influenciado para que no ambiente real o número de clientes obtidos tenha sido mais alto. Uma medida que pode influenciar neste tempo de serviço é o padrão de acesso no disco. Este padrão de acesso pode influenciar devido ao fato dos clientes estarem acessando o mesmo vídeo. Se o intervalo entre chegadas for pequeno e os clientes requisitarem os

mesmos blocos do vídeo, é possível que os pedidos de um segundo cliente não sejam recuperados do disco e sim de uma *cache* do sistema operacional, que possui os dados pedidos recentemente. Como o acesso à *cache* é mais rápido do que o acesso ao disco, o tempo de serviço pode ser diminuído e com isso consegue-se atender um número maior de clientes. Desta forma o gargalo do sistema poderia passar a ser a placa de rede dos nós de armazenamento e não mais o disco dos mesmos. Experimentos neste sentido podem ser considerados como trabalhos futuros, pelo fato das máquinas disponíveis não estarem dedicadas exclusivamente aos experimentos realizados.

Na Figura 5.24 mostramos os resultados obtidos com 2 nós de armazenamento com esta nova parametrização.

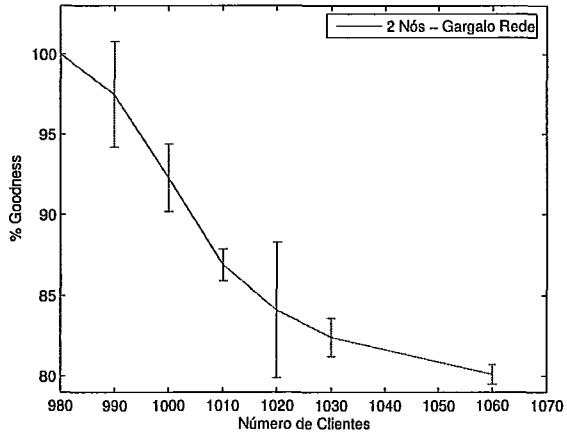


Figura 5.24: 2 Nós de Armazenamento na Rede Giga supondo que o gargalo do sistema é a placa de rede

Como era esperado, a capacidade do sistema aumentou com a adição do novo nó de armazenamento, como ocorreu em todos os casos anteriores, mostrando assim que o RIO é escalável em qualquer cenário.

Capítulo 6

Conclusões e Trabalhos Futuros

Se todo o ano fosse festa, divertir-se seria mais aborrecedor que trabalhar. *Shakespeare*

Devido a sua popularidade, a Internet se tornou o principal meio para disseminação de aplicações focadas em educação a distância. Muitos estudos têm sido realizados sobre a distribuição de conteúdo multimídia na Internet, mas apenas uma pequena parte desses estudos foca na educação a distância. Esse trabalho faz o estudo de um servidor multimídia usado para uma aplicação de educação a distância em uma rede de alta velocidade.

Como visto no capítulo 2, diversas soluções foram propostas para distribuição de conteúdo multimídia. Porém, um servidor de vídeo sob demanda distribuído (DVoD) é uma das melhores opções para o ensino a distância, devido a sua escalabilidade e ao baixo custo para a adição de novos nós de armazenamento.

Neste estudo avaliamos o servidor RIO através de modelos de simulação e experimentos em ambiente real. O objetivo deste trabalho foi a modelagem de um serviço de vídeo sob demanda em uma rede Giga, com a elaboração de modelos nos quais são avaliadas opções de arquitetura para fornecimento de um serviço VoD (*Video on Demand*) distribuído, comparando os resultados dos modelos com testes em ambiente real, visando a validação e parametrização dos mesmos.

Foram elaborados modelos para estudo de três cenários: usuários sequenciais

acessando o sistema RIO sem replicação, usuários sequenciais acessando o sistema RIO com replicação e usuários interativos acessando o sistema RIO sem replicação.

Em cada um dos cenários variamos o número de nós de armazenamento para avaliarmos a escalabilidade do sistema. Estudamos também o impacto de um nó de armazenamento localizado fora da rede Giga.

Como principais contribuições deste trabalho temos a elaboração de modelos para o sistema de vídeo sob demanda RIO. Os modelos são facilmente parametrizáveis e foram validados através de comparação com ambiente real. Uma das principais vantagens do uso de modelos é a sua simplicidade com relação a realização de testes em ambiente real.

Como principais conclusões a partir dos resultados temos que o servidor RIO é escalável, pois obtemos ganhos significativos com a adição de novos nós de armazenamento. A adição de um nó de armazenamento na UFMG através de um caminho mais lento tanto com a utilização de replicação quanto sem pode aumentar a escalabilidade do sistema, para uma população grande de clientes.

Por fim, foi possível constatar que os resultados obtidos com a experimentação real foram próximos aos obtidos com os modelos de simulação, o que significa que o modelo capturou de maneira satisfatória o comportamento dos clientes e poderá ser utilizado em trabalhos futuros.

6.1 Trabalhos Futuros

Um trabalho que pode ser feito é a experimentação com um número maior de clientes e nós de armazenamento, utilizando as outras instiuições para gerar carga de clientes, de forma que todos os pedidos gerados não estejam concentrados em um único ambiente.

Um outro trabalho seria a simulação do sistema com diversos níveis de replicação, para que seja possível avaliar de maneira mais abrangente aspectos tais como o balanceamento de carga e os benefícios do aumento do nível de replicação. Se-

ria interessante avaliar também se a replicação parcial do conteúdo poderia trazer alguma vantagem sobre a replicação total.

Com relação ao modelo elaborado para usuários interativos, pretendemos aprimorar o objeto que representa o tráfego agregado dos clientes do sistema.

Referências Bibliográficas

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

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

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

[4] Digital Island. *URL: http://www.digitalisland.com.*

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

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

[7] NEAD - Núcleo de Educação a Distância. URL `http://200.206.4.4/html/ead/historico.html`.

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

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

[10] ALMEROTH, K., E AMMAR, M. The use of multicast delivery to provide a scalable and interactivevideo-on-demand service. *Selected Areas in Communications, IEEE Journal on 14*, 6 (1996), 1110–1122.

[11] BARNETT, S., E ANIDO, G. A Cost Comparison of Distributed and Centralized Approaches to Video-on-Demand. *Selected Areas in Communications, IEEE Journal on 14*, 6 (1996), 1173–1183.

[12] CARMO, R. M. L. R., DE CARVALHO, L. R., DE SOUZA E SILVA, E., DINIZ, M. C., E MUNTZ, R. R. Tangram-ii: A performability modeling environment tool. In *Proceedings of the 9th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools* (London, UK, 1997), Springer-Verlag, pp. 6–18.

[13] CARTER, R., E CROVELLA, M. Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks. *IEEE INFOCOM 97* (1997).

[14] CHENG, W. The TANGRAM graphical interface facility (TGIF) manual.

[15] CHERVENAK, A. L., PATTERSON, D. A., E KATZ, R. H. Choosing the best storage system for video service. In *MULTIMEDIA '95: Proceedings of the third ACM international conference on Multimedia* (New York, NY, USA, 1995), ACM Press, pp. 109–119.

[16] COHEN, B. Incentives Build Robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer Systems 6* (2003).

[17] CORES, F., RIPOLL, A., QAZZAZ, B., SUPPI, R., YANG, X., HERNANDEZ, P., E LUQUE, E. Exploiting Traffic Balancing and Multicast Efficiency in Distributed Video-on-Demand Architectures. *LECTURE NOTES IN COMPUTER SCIENCE* (2003), 859–869.

[18] COULOURIS, G., DOLLIMORE, J., E KINDBERG, T. *Distributed Systems: Concepts and Design.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2000.

[19] CROVELLA, M., E CARTER, R. Dynamic Server Selection In The Internet. *Proceedings of the Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS '95)* (1995), 158–162.

[20] DE ARAGÃO ROCHA, A. A. Medições Ativas na Internet: Algoritmos Baseados em Retardo Fim-a-Fim e Experimentos. Tese de Mestrado, 2003.

[21] DE QUEVEDO CARDOZO, A. Mecanismos para garantir qualidade de serviço de aplicações de vídeos sob demanda. Tese de Mestrado, UFRJ/COPPE-PESC, 2002.

[22] DE SOUZA E SILVA, E., E DUARTE, F. P. Proposta de testes de equipamentos da huawei. Relatório técnico, LAND/UFRJ, 2006.

[23] DE SOUZA E SILVA, E., E LEÃO, R. M. M. Tangram-II User's Manual. Relatório técnico, tech. rep., Universidade Federal do Rio de Janeiro, Oct. 2000. http://www.land.ufrj.br.

[24] DE SOUZA E SILVA, E. A., DA SILVA, A. P. C., DE A. ROCHA, A. A., LEÃO, R. M. M., DUARTE, F. P., SILVEIRA, F., JAIME, G. D. G., E MUNTZ, R. R. Modeling, analysis, measurement and experimentation with the Tangram-II integrated environment. In *VALUETOOLS* (2006), p. 7.

[25] E SILVA, E., LEÃO, R., RIBEIRO-NETO, B., E CAMPOS, S. Performance Issues of Multimedia Applications. *Performance Evaluation of Complex Systems: Techniques and Tools, Performance* (2002), 374–404.

[26] ELTETO, T., E MOLNAR, S. On the distribution of round-trip delays in TCP/IP networks. *Conference on Local Computer Networks, LCN'99.* (1999), 172–181.

[27] GADDE, S., CHASE, J., E RABINOVICH, M. Web caching and content distribution: a view from the interior. *Computer Communications 24*, 2 (2001), 222–231.

[28] GEREOY, A., E OTHERS. MPlayer-Movie player for linux.

[29] GHANDEHARIZADEH, S., ZIMMERMANN, R., SHI, W., REJAIE, R., IERARDI, D. J., E LI, T.-W. Mitra: A scalable continuous media server. In *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, H. Jin, T. Cortes, e R. Buyya, Eds. IEEE Computer Society Press and Wiley, New York, NY, 2001, pp. 595–613.

[30] GKANTSIDIS, C., RODRIGUEZ, P., E OTHERS. Network coding for large scale content distribution. *Proceedings of IEEE Infocom* (2005).

[31] GUO, Y., SUH, K., KUROSE, J., E TOWSLEY, D. A Peer-to-Peer on-Demand Streaming Service and Its Performance Evaluation. *International Conference on Multimedia and Expo, ICME'03. 2* (2003).

[32] IYER, N., E CANDAN, K. Server Replication in Interactive, Push-Based Data Delivery Networks, 2004.

[33] JOHNSON, K. L., CARR, J. F., DAY, M. S., E KAASHOEK, M. F. The measured performance of content distribution networks. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop* (2000).

[34] LIN, W., CHIU, D., E LEE, Y. Erasure code replication revisited. *Fourth International Conference on Peer-to-Peer Computing.* (2004), 90–97.

[35] MAYER, C., CANDAN, K., E SANGAM, V. Effects of User Request Patterns on a Multimedia Delivery System. *Multimedia Tools and Applications 24*, 3 (2004), 233–251.

[36] MIAO, Z., E ORTEGA, A. Proxy caching for efficient video services over the Internet. *Proc. Packet Video 99*.

[37] MITZENMACHER, M. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems 12*, 10 (2001), 1094–1104.

[38] ON, G., SCHMITT, J., LIEPERT, M., E STEINMETZ, R. Replication with QoS Support for a Distributed Multimedia System. *Proceedings of the 27th EUROMICRO Conference (Workshop on Multimedia and Telecommunications), Warszaw, Poland*, 0–7695.

[39] OZDEN, B., RASTOGI, R., E SILBERSCHATZ, A. Disk striping in video server environments, 1996.

[40] PADMANABHAN, V., WANG, H., CHOU, P., E SRIPANIDKULCHAI, K. Distributing streaming media content using cooperative networking. *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video* (2002), 177–186.

[41] POUWELSE, J., GARBACKI, P., EPEMA, D., E SIPS, H. The bittorrent p2p file-sharing system: Measurements and analysis. *International Workshop on Peer-to-Peer Systems (IPTPS)* (2005).

[42] RANGAN, P., VIN, H., E RAMANATHAN, S. Designing an On-Demand Multimedia Service. *Communications Magazine, IEEE 30*, 7 (1992), 56–64.

[43] RIBEIRO, B. F. M. Diversidade de caminhos para aplicações em tempo real. Tese de Mestrado, UFRJ/COPPE-PESC, Rio de Janeiro, RJ, Brasil, 2003.

[44] SANGAM, V., MAYER, C., E CANDAN, K. Fairly Redistributing Failed Server Load in a Distributed System. *Workshop on Reliable and Secure Middleware* (2003), 871–884.

[45] SANTOS, J., E MUNTZ, R. Performance analysis of the RIO multimedia storage system with heterogeneous disk configurations. *Proceedings of the sixth ACM international conference on Multimedia* (1998), 303–308.

[46] SANTOS, J. R., MUNTZ, R. R., E RIBEIRO-NETO, B. Comparing random data allocation and data striping in multimedia servers. In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (New York, NY, USA, 2000), ACM Press, pp. 44–55.

[47] SANTOS, J. R. G. *Rio: a universal multimedia storage system based on random data allocation and block replication.* PhD thesis, Los Angeles, CA, USA, 1998.

[48] SHENOY, P., GOYAL, P., E VIN, H. Issues in multimedia server design. *ACM Computing Surveys (CSUR) 27*, 4 (1995), 636–639.

[49] SHENOY, P., E VIN, H. M. Efficient striping techniques for multimedia file servers. Relatório Técnico CS-TR-96-27, 1, 1998.

[50] 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, 2003.

[51] TRAN, D. A., HUA, K., E DO, T. A peer-to-peer architecture for media streaming.

[52] TRIVEDI, K. Probability and Statistics with Reliability, Queuing, and Computer Science Applications, 1982.

[53] VALLE, M. R. Um Estudo do Servidor RIO na Rede Giga Usando Carga Real de uma Aplicação de Ensino a Distância. Tese de Mestrado, 2007.

[54] VIEW, T. Optimal replica placement on transparent replication proxies for-read/write data. *Performance, Computing, and Communications Conference, 2002. 21st IEEE International* (2002), 103–110.

[55] WANG, B., SEN, S., ADLER, M., E TOWSLEY, D. Optimal proxy cache allocation for efficient streaming media distribution. *Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. INFOCOM'02. 3* (2002).

[56] WANG, Y., ZHANG, Z., DU, D., E SU, D. A Network-Conscious Approach to End-to-End Video Delivery Over Wide Area Networks Using Proxy Servers. *Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. INFOCOM'98. 2.*

[57] ZHOU, X., E XU, C. Optimal video replication and placement on a cluster of video-on-demand servers. *Parallel Processing, 2002. Proceedings. International Conference on* (2002), 547–555.