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.

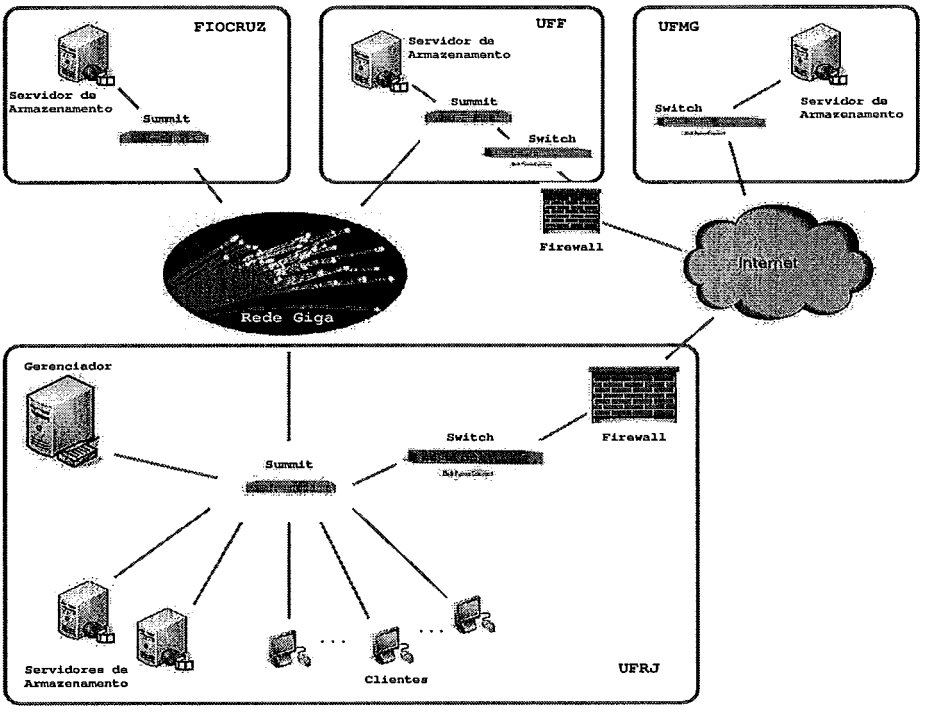

Figura 1.1: Arquitetura Completa da Rede

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.

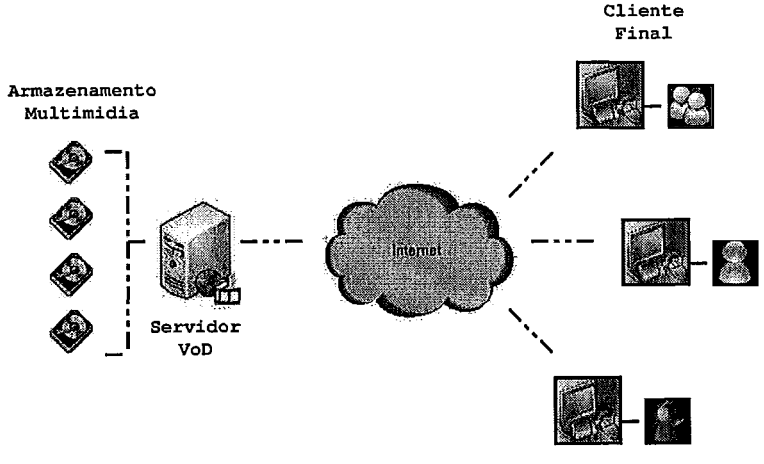

Figura 2.1: Arquitetura VoD

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-

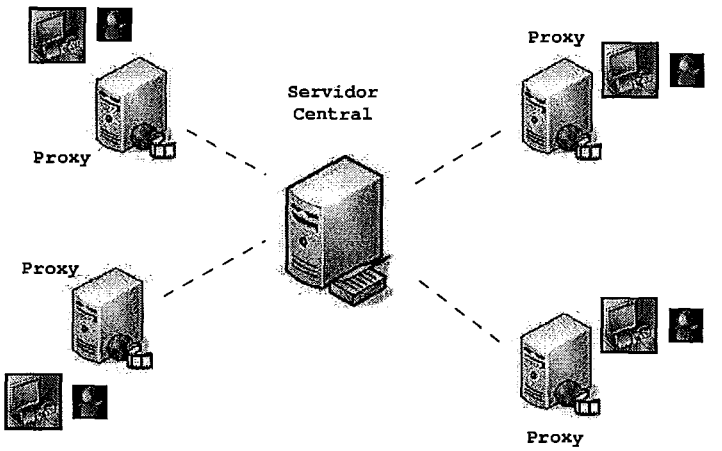

Figura 2.2: Sistema de Vídeo sob Demanda Distribuído

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.

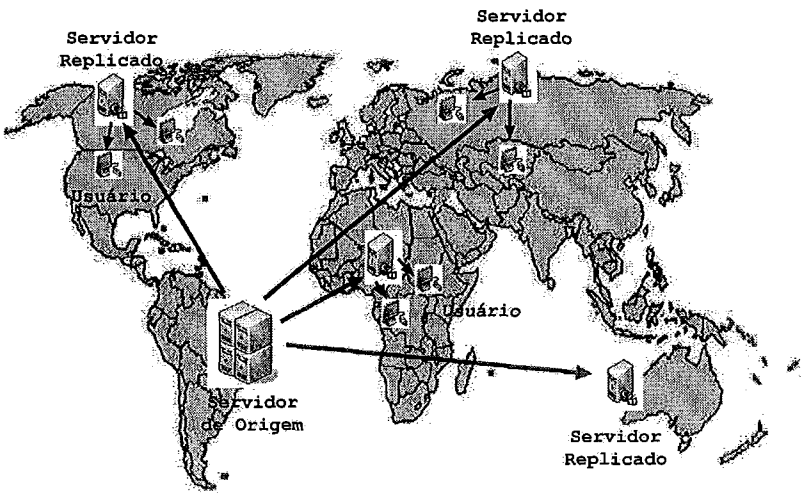

Figura 2.3: Arquitetura de uma CDN

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

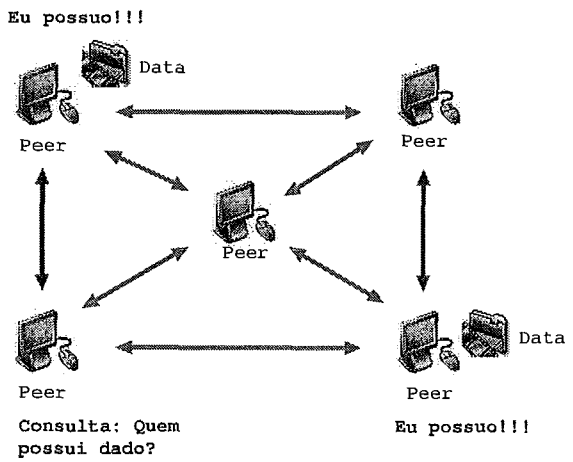

Figura 2.4: Arquitetura Peer-to-Peer

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.

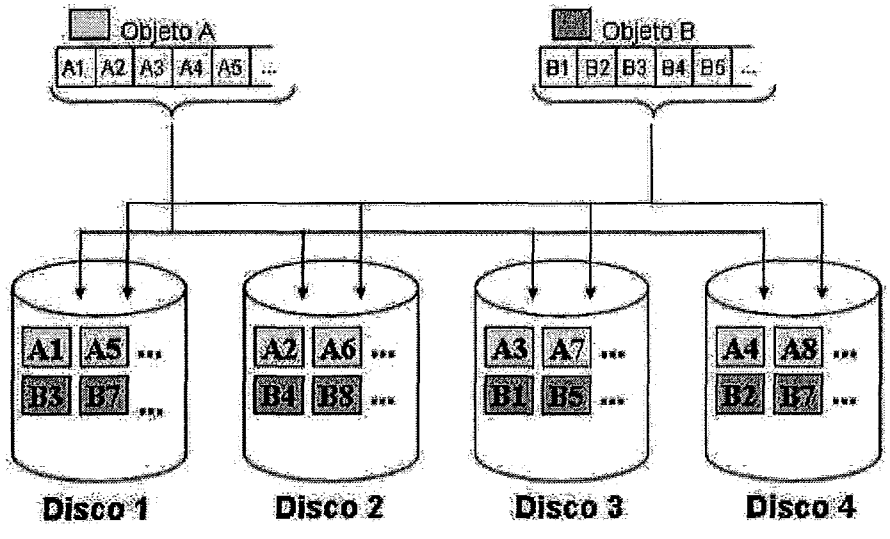

Figura 2.5: Data Striping

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.

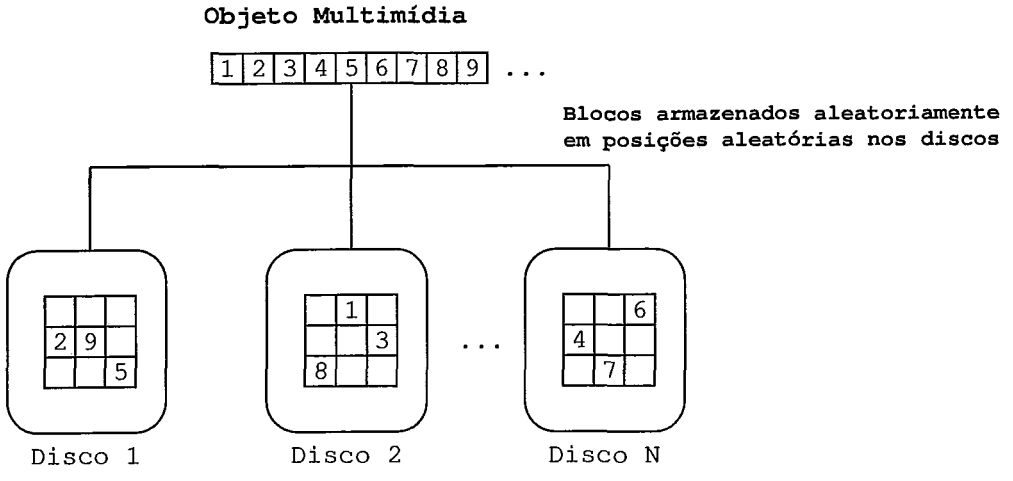

Figura 2.6: Alocação Aleatória dos Dados

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

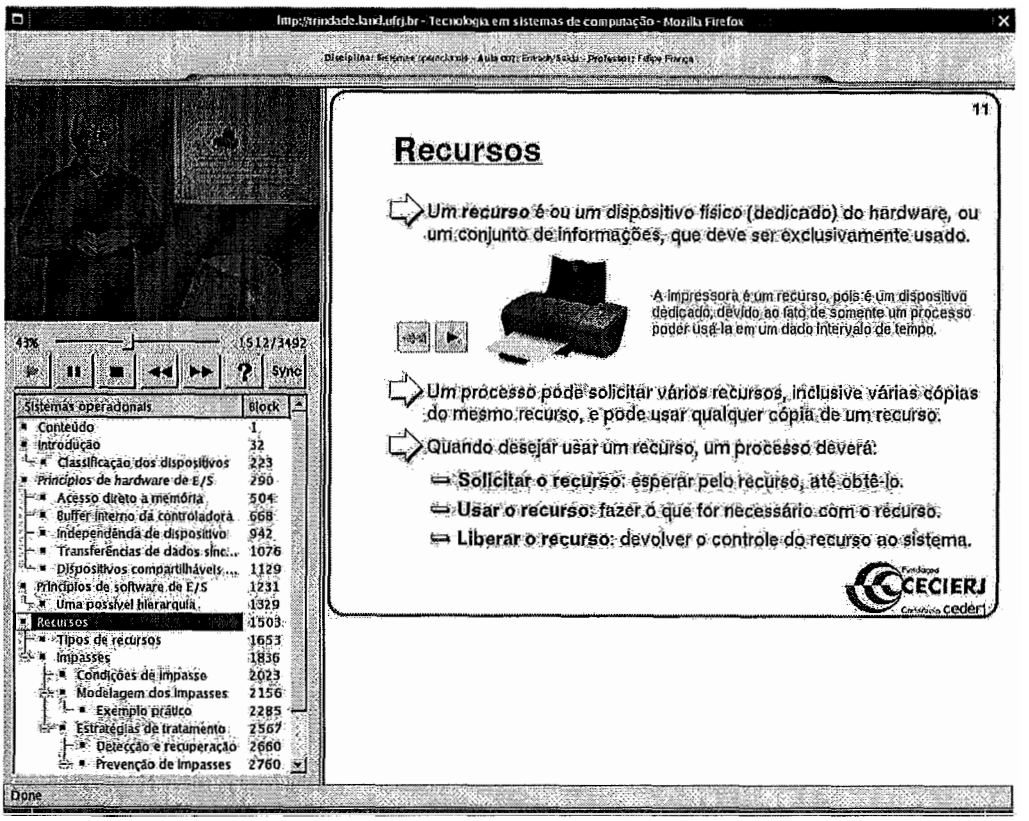

Figura 3.1: RioMMClient

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

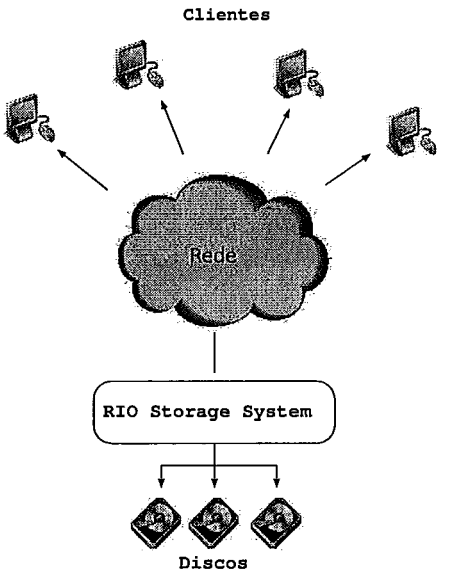

Figura 3.2: Arquitetura do Sistema multimídia

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.

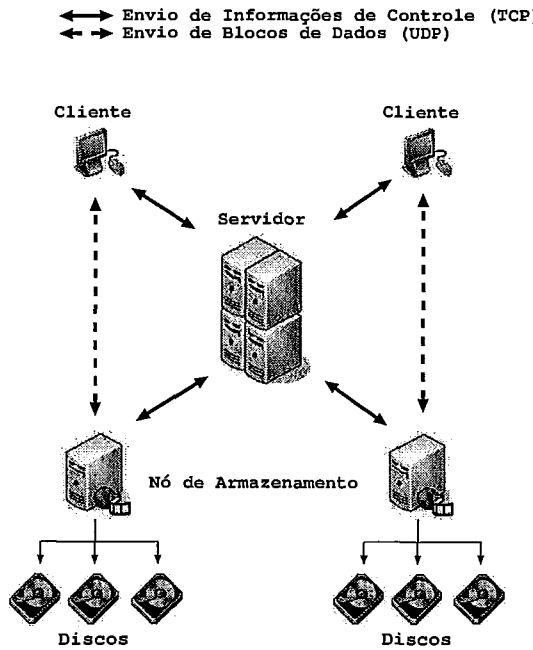

Figura 3.3: Arquitetura básica do RIO

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

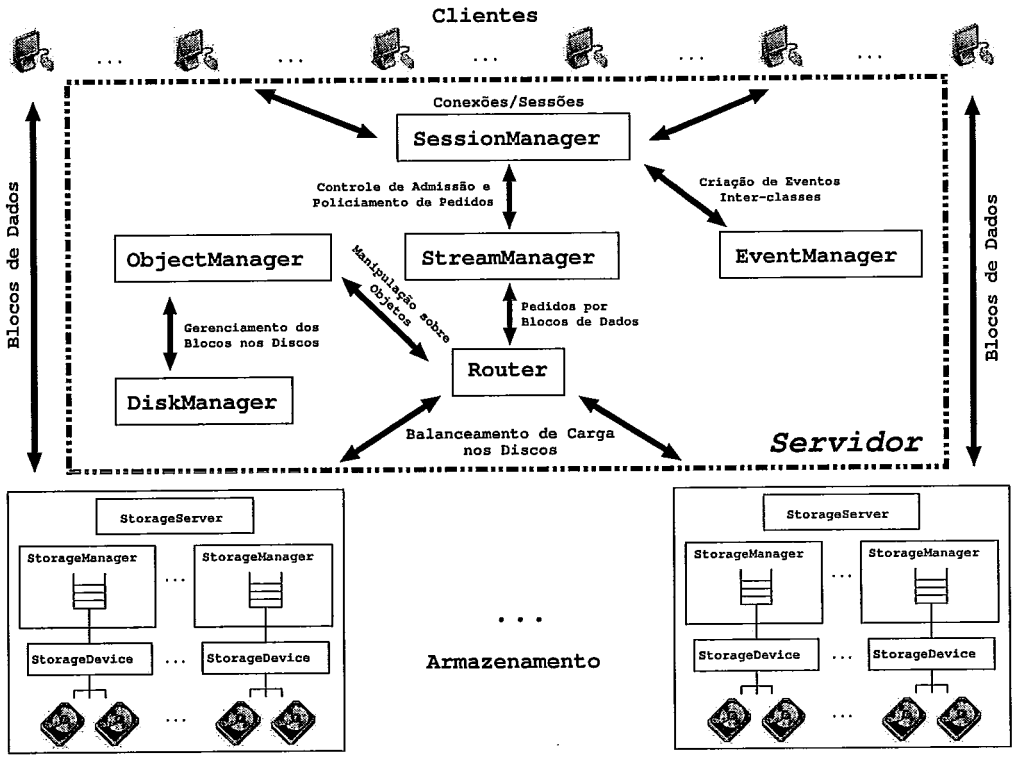

Figura 3.4: Componentes do Servidor RIO

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.

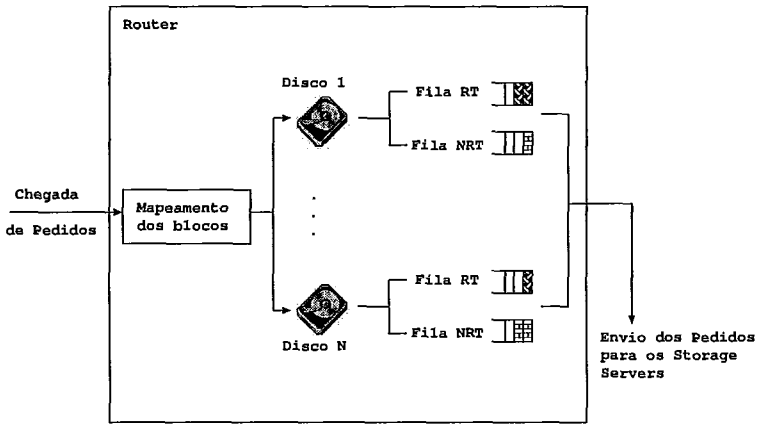

Figura 3.5: Componente Router

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.

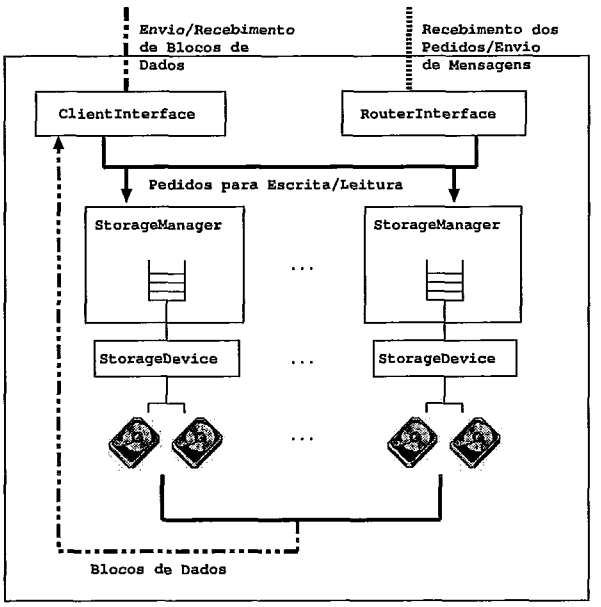

Figura 3.6: Componente StorageServer

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].

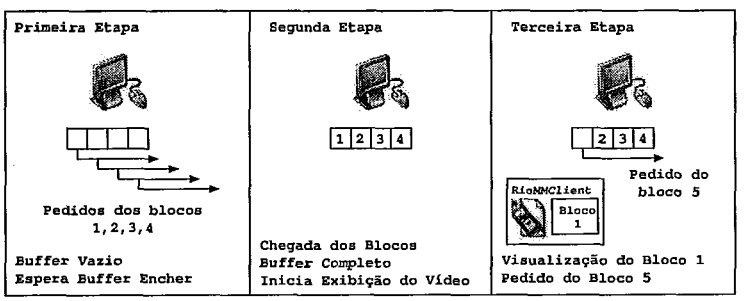

Figura 3.7: Funcionamento do Cliente RioMMClient

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 $rt + F$.

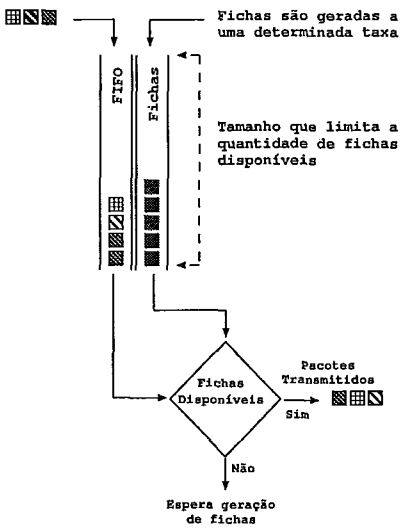

Figura 3.8: Fila Leaky Bucket

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,