Mecanismos para Garantir Qualidade de Serviço de Aplicações de Vídeo sob Demanda

por

Adriane de Quevedo Cardozo

UFRJ

Tese submetida para a obtenção do título de

Mestre em Ciências em Engenharia de Sistemas e Computação

ao Programa de Pós-Graduação de Engenharia de Sistemas e Computação

da COPPE/UFRJ

por

Adriane de Quevedo Cardozo

Outubro 2002

MECANISMOS PARA GARANTIR QUALIDADE DE SERVIÇO DE APLICAÇÕES DE VÍDEO SOB DEMANDA

Adriane de Quevedo Cardozo

TESE 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 de Souza e Silva, Ph.D.

Profa. Rosa Maria Meri Leão, Dr.

Prof. Berthier Ribeiro-Neto, Ph.D.

Prof. Sérgio Vale Aguiar Campos, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

OUTUBRO DE 2002

CARDOZO, ADRIANE DE QUEVEDO

Mecanismos para Garantir Qualidade de Serviço de Aplicações de Vídeo sob Demanda [Rio de Janeiro] 2002

XVI, 123 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2002)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Servidores Multimídia
2. Controle de Admissão
3. Gerenciamento de *Buffers*
4. Qualidade de Serviço
5. Medições

I. COPPE/UFRJ   II. Título (Série)

Dedico este trabalho aos meus familiares,
em especial aos meus pais Narciso e Nilta.

Agradecimentos

Gostaria de agradecer aos meus familiares, principalmente aos meus pais e a minha irmã Letícia por tudo, pelo amor, pelo apoio, pela compreensão, por tudo que me ensinaram. Sou eternamente grata a vocês.

À Deus por ter me dado saúde e força para superar as dificuldades.

Aos meus orientadores, Edmundo e Rosa, pela oportunidade, confiança e colaboração para a realização deste trabalho.

Gostaria de agradecer ao Kelvin, que sempre esteve ao meu lado, pela sua amizade, ajuda e incentivo durante todo o curso. E também a sua mãe, Lizarb, que nos ajudou muito.

Ao pessoal do LAND, que me acompanhou direta ou indiretamente, em especial aos amigos Ana Paula, Magnos, Sidney, Ratton, Raniery, Bruno, Allyson, Fernando e Carol e ao Flávio e Bernardo que dedicaram horas me ajudando nas simulações e em algumas discussões nesta reta final. Ao Prof. Richard Muntz e José Renato que cederam o código do servidor RIO para a realização deste trabalho.

Finalmente, gostaria de agradecer a algumas pessoas que também me incentivaram e apoiaram: Rossane, Chiquinho, Juliana, Lilian, Ricardo, Marluce, Rodrigo, Paulo André, César, Augusto, entre outros.

Ao CNPq pelo financiamento da bolsa de estudos e à Lucent e ao CPqD pela extensão da minha bolsa.

Muito obrigada a todos.

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

MECANISMOS PARA GARANTIR QUALIDADE DE SERVIÇO DE APLICAÇÕES DE VÍDEO SOB DEMANDA

Adriane de Quevedo Cardozo

Outubro/2002

Orientadores: Edmundo A. de Souza e Silva
Rosa Maria Meri Leão

Programa: Engenharia de Sistemas e Computação

O avanço e o desenvolvimento de novas tecnologias viabilizou o uso de novas aplicações multimídia, denominadas aplicações de mídia contínua, na Internet. Devido à natureza contínua de seus dados e à alta interatividade, tais aplicações impõem limites na busca e na entrega de seus dados, pois perdas e atrasos podem prejudicar a qualidade da apresentação. Neste trabalho são abordados os problemas da rede e os aspectos de servidores multimídia que comprometem a qualidade de serviço (QoS) oferecida às aplicações de mídia contínua, em especial, às aplicações de vídeo sob demanda. Foi realizado um estudo aprofundado do servidor RIO, com o objetivo de melhorar o serviço oferecido aos usuários e torná-lo um protótipo operacional. Um conjunto de mecanismos foi elaborado e implementado, incluindo o gerenciamento de *buffers* no servidor e no cliente, que visa ao aumento do número de clientes atendidos, e um controle de admissão, que visa à garantia de QoS. Através do controle fino do sistema e do ambiente de medição criado, é possível o estudo e a coleta de diversas medidas, que podem auxiliar na configuração do sistema. Vários experimentos mostraram o desempenho dos mecanismos implementados. Como parte deste trabalho, também foi desenvolvido um modelo do servidor, utilizando a ferramenta Tangram-II, onde várias medidas foram obtidas através de simulações.

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

MECHANISMS TO PROVIDE QUALITY OF SERVICE TO VIDEO-ON-DEMAND APPLICATIONS

Adriane de Quevedo Cardozo

October/2002

Advisors: Edmundo A. de Souza e Silva
Rosa Maria Meri Leão
Department: Computer and System Engineering

The fast development of new technologies has made technically possible the creation of an infrastructure capable of supporting new multimedia applications (called continuous media applications) using the Internet. Due to the continuous nature of data and the high interactivity, such applications impose deadlines in the retrieval and delivery of the media. In this work we discuss the main technical issues involved in the design and implementation of such applications, focusing on video-on-demand applications. A detailed study of a multimedia server, the RIO Multimedia Storage Server, was done aimed at improving the QoS provided to the users and developing an operational prototype. Some mechanisms were proposed and implemented, including an admission control to provide QoS and a buffer management on the server and client side to increase the number of admitted clients. We also developed an environment to collect and compute several statistics to aid the configuration and analysis of the system. Several experiments were performed to evaluate the performance of the multimedia server. A multimedia server model was developed using the Tangram-II tool which allow the computation of several measures of interest.

Palavras-chave

1. Servidores Multimídia
2. Controle de Admissão
3. Gerenciamento de *Buffers*
4. Qualidade de Serviço
5. Medições

Glossário

QoS :	Qualidade de Serviço (*Quality of Service*);
CBR :	Taxa Constante de Bits (*Constant Bit Rate*);
VBR :	Taxa Variável de Bits (*Variable Bit Rate*);
FIFO:	O primeiro a chegar é o primeiro a sair (*First In First Out*);
SCAN:	Política de escalonamento na qual os pedidos são ordenados de acordo com o posicionamento da cabeça de leitura/escrita do disco e com a localização dos dados;
RT:	Tempo real (*Real Time*);
NRT:	Não é de tempo real (*Non Real Time*);
RIO:	Operações de entrada e saída aleatórias (*Randomized I/O*);
RTT:	Tempo de ida e volta entre um emissor e um receptor (*Round Trip Time*);

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o avanço e o desenvolvimento de novas tecnologias foi possível a criação de uma infra-estrutura, utilizando a Internet, capaz de suportar novas aplicações multimídia, também denominadas aplicações de mídia contínua, incluindo ensino à distância, visualização de vídeos, aplicações de voz, videoconferência, ambientes virtuais, entre outras. Tais aplicações possuem requisitos diferentes das aplicações tradicionais, como conteúdo Web (texto e imagem), transferência de arquivos e correio eletrônico.

As aplicações de mídia contínua consistem de uma sequência de dados (geralmente, amostras de áudio ou quadros de vídeo) que devem ser apresentados em um determinado intervalo de tempo. Devido à natureza contínua de seus dados e à alta interatividade, essas aplicações impõem limites na busca e na entrega de informações, pois perdas e atrasos podem causar interrupções que comprometem a qualidade da apresentação.

Esse tipo de aplicação traz diversos desafios para o projeto de tais sistemas. Primeiro, porque elas usam uma rede que ainda não suporta a qualidade de serviço (QoS) desejada. Embora existam várias propostas para estender a arquitetura da Internet de forma a minimizar esse problema, por vários anos ainda não se terá implementada a infra-estrutura necessária a prover a QoS adequada às diferentes aplicações multimídia. Segundo, porque o servidor multimídia precisa possuir grande espaço de armazenamento e ser capaz de eficientemente manipular uma grande quantidade de informação, que envolve a recuperação dos dados armazenados nos discos e a transmissão destes dados pela rede, obedecendo requisitos estreitos de tempo e garantindo a QoS para todos os clientes acessando o servidor.

Esse novo tipo de aplicação deverá se tornar comum na Internet devido a diversos fatores, entre eles, a disponibilidade de redes com maior largura de banda e a redução do preço do disco que viabiliza o projeto e a implementação de servidores de baixo custo. Os usuários poderão, por exemplo, participar de cursos de ensino à distância, fazer ligações telefônicas utilizando a Internet e videoconferência com um grupo de pessoas.

Tudo isso faz com que seja necessário o estudo e o desenvolvimento de novas técnicas, que podem atuar tanto na rede quanto na aplicação, para oferecer a qualidade de serviço exigida por essas novas aplicações.

1.1 Objetivos e Contribuições

Um dos objetivos deste trabalho é o estudo do desenvolvimento de sistemas multimídia. Para isto são abordados os problemas da rede e os aspectos do projeto de servidores multimídia que comprometem a qualidade de serviço oferecida às aplicações de mídia contínua. Neste trabalho também é feito o estudo aprofundado de um servidor multimídia, servidor RIO [46], inicialmente desenvolvido na UCLA (*University of California Los Angeles*), para entender os problemas envolvidos, tentar melhorar o serviço oferecido aos usuários e torná-lo um protótipo operacional. Atualmente o servidor RIO é utilizado em um projeto de ensino à distância do CEDERJ (Centro de Ensino Superior a Distância do Estado do Rio de Janeiro) [8].

É feita a proposta e a implementação de um conjunto de mecanismos que inclui o gerenciamento de *buffers* no servidor e no cliente, para compensar as flutuações da carga dos discos e da rede, visando a atender um maior número de usuários, e um novo controle de admissão estatístico, para garantir a qualidade de serviço aos clientes do sistema.

Uma análise das informações dos vídeos armazenados é feita em tempo real para prever o escalonamento dos pedidos de cada cliente e, em conjunto com as informações de desempenho do servidor (extraídas em tempo real), prever a carga do sistema. A cada novo pedido de admissão, é feita a simulação de todo o sistema, incluindo o novo cliente, desde a chegada das requisições no servidor até o recebimento dos dados em cada um dos clientes. O novo cliente é admitido se a qualidade de serviço puder ser garantida para todos os clientes do sistema.

Através deste controle fino e da infra-estrutura de medição criada, é possível o estudo e a coleta de diversas medidas, como por exemplo, o tempo de espera na fila e o tempo de serviço dos discos utilizados, o tempo de geração da lista de requisições para cada cliente, o tempo de geração da lista de eventos utilizada no processo de admissão, o tempo gasto para a execução do teste de admissão (simulação), o tamanho do *playout buffer* de todos os clientes e o RTT entre o servidor e cada um dos clientes. Com isto, vários experimentos podem ser realizados de forma a ajustar tanto os parâmetros do servidor quanto os parâmetros dos clientes para alcançar os níveis de qualidade de serviço desejados.

Como parte do trabalho, foi criado um modelo do servidor RIO com o uso da ferramenta Tangram-II [28]. Através de simulações, várias medidas podem ser obtidas que auxiliam na configuração do protótipo e entendimento do sistema.

1.2 Organização do texto

No capítulo 2 são abordados os problemas e as soluções encontradas para o uso da Internet por aplicações multimídia. É feita uma breve descrição de protocolos e propostas para extensão da arquitetura da Internet.

No capítulo 3 são estudados os aspectos envolvidos no projeto de servidores multimídia, entre eles, políticas de acesso aos dados, organização dos dados, escalonamento dos discos, controle de admissão de usuários e técnicas para compartilhamento de recursos.

No capítulo 4 é detalhado o servidor RIO utilizado na implementação da nova proposta. Neste capítulo é descrito o funcionamento do servidor, bem como todos os componentes de sua arquitetura.

No capítulo 5 é descrito o modelo de controle de admissão, o gerenciamento de *buffers* proposto e a infra-estrutura criada. Ainda neste capítulo são descritos os detalhes de implementação e as diferenças entre a versão original do servidor RIO e a versão implementada durante este trabalho.

No capítulo 6 são apresentados os testes realizados e os resultados obtidos do protótipo. Além disso, é apresentado o modelo de simulação do servidor criado utilizando a ferramenta Tangram-II e os resultados obtidos pelas simulações.

No capítulo 7 são mostradas as aplicações do servidor RIO. E finalmente no capítulo 8 são apresentadas as conclusões desta dissertação e sugestões para trabalhos futuros.

Capítulo 2

Aplicações de Mídia Contínua na Internet

Vários fatores viabilizaram a transmissão de mídia contínua na Internet, tais como o aumento da largura de banda disponível, o desenvolvimento de algoritmos mais eficientes para compressão de dados e o avanço na capacidade de processamento dos computadores.

Há alguns anos atrás, ainda não era possível a transmissão de vídeo em tempo real na Internet, pois a largura de banda disponível era insuficiente para este tipo de aplicação. Devido à baixa capacidade da época, não era viável a manipulação de vídeo com boa qualidade. Além disso, era necessário salvar todo o conteúdo do vídeo localmente para depois começar sua exibição, o que é inadequado, tanto pelo tempo de espera para a transmissão do arquivo, quanto pela baixa qualidade do vídeo gerado.

Atualmente, com o uso de algoritmos mais sofisticados de compressão de mídia contínua é possível reduzir o espaço de armazenamento e a largura de banda necessária para a transmissão desse tipo de mídia. Além disso, com o desenvolvimento dos protocolos de *Streaming*, o usuário pode começar a apresentação do vídeo/áudio enquanto o restante do arquivo ainda está sendo transmitido.

Existem diversos padrões para compressão de mídia contínua, como MPEG [38], H.261 e H.263+ [12] para compressão de vídeo, e GSM,G.729, G.723.3 e MP3 para compressão de áudio [26]. O tráfego gerado por essas técnicas tem taxa variável (VBR) e os dados precisam ser entregues ao cliente dentro de limites de tempo para que seja possível a sua decodificação e exibição sem haver descontinuidade. Devido ao tipo de serviço oferecido pela Internet (*best effort*) e às características de tempo real das aplicações de mídia contínua, existem vários problemas que precisam ser solucionados para a transmissão deste tipo de conteúdo.

2.1 Classes de Aplicações de Mídia Contínua

De acordo com [26], as novas aplicações multimídias podem ser divididas em três classes:

- **Transmissão de áudio e vídeo pré-armazenados**

  Neste tipo de aplicação, o conteúdo já foi gravado, codificado e armazenado no servidor. Os clientes se conectam no servidor e escolhem o conteúdo que desejam visualizar. Dependendo do tipo de serviço oferecido, o usuário pode interagir através de diversos comandos, entre eles, retroceder, avançar e pausar.

  Nesta classe pode-se dizer que, além das aplicações de mídia contínua, como vídeo e áudio sob demanda, estão as aplicações de visualização interativa. Estas aplicações permitem o usuário navegar por um banco de dados 3D ou 4D em um ambiente virtual. Alguns exemplos deste tipo de aplicação são o simulador urbano desenvolvido na UCLA [20, 37] e jogos interativos.

- **Transmissão de áudio e vídeo ao vivo**

  Este tipo de aplicação é similar à transmissão de televisão e rádio. Os dados não estão armazenados no servidor. Eles são capturados através de equipamentos específicos e transmitidos pela rede, como por exemplo, programas de rádios na Internet.

- **Transmissão de áudio e vídeo interativos em tempo real**

  Este tipo de aplicação permite a comunicação entre os usuários em tempo real, tais como as aplicações de voz e de videoconferência. O tempo de resposta para este tipo de aplicação é mais rigoroso (da ordem de milisegundos).

Neste trabalho é abordado o primeiro tipo de aplicação: transmissão de áudio e vídeo sob demanda. Embora o acesso a este tipo de aplicação possa ser feito através de um servidor Web, esta arquitetura não é recomendada, pois o servidor Web não foi projetado para oferecer suporte para aplicações de mídia contínua. A maior parte das aplicações de vídeo/áudio sob demanda é disponibilizada através de servidores multimídia, que foram desenvolvidos especialmente para fornecer a garantia dos requisitos exigidos por estas aplicações.

A arquitetura básica deste tipo de serviço envolve três componentes: o cliente, a rede e o servidor multimídia, conforme ilustrado na figura 2.1. Os protocolos para comunicação entre o cliente e o servidor podem ser padrões abertos, que estão descritos na seção 2.3, ou protocolos proprietários.

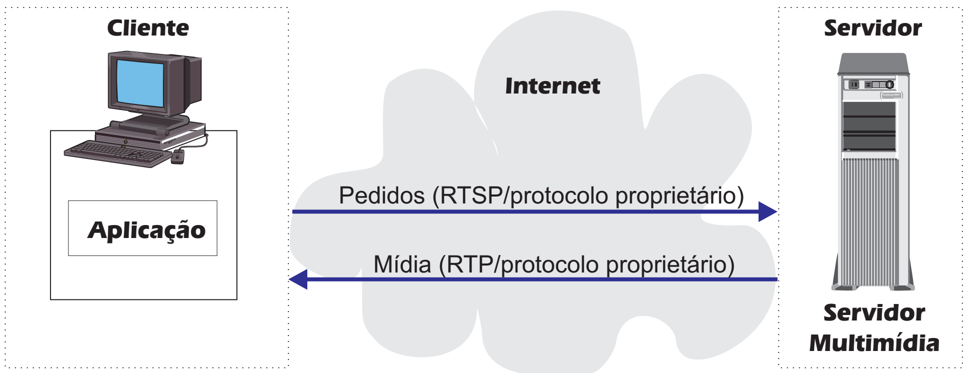

Figura 2.1: Arquitetura de acesso às aplicações de vídeo/áudio sob demanda

2.2 Problemas e algumas soluções encontradas

Embora o enfoque deste trabalho seja a transmissão de áudio e vídeo pré-armazenados em um servidor multimídia, os problemas descritos nesta seção afetam todos os tipos de aplicações de mídia contínua. Dependendo do tipo de conteúdo, várias alternativas podem ser adotadas para compensar cada um dos problemas.

A Internet é uma rede baseada na comutação de pacotes. Quando um computador deseja enviar um pacote para outro computador, este pacote é transmitido por uma série de enlaces. A transmissão é do tipo *store-and-forward* com multiplexação estatística dos pacotes, sem reserva prévia de banda, portanto retardos aleatórios ou mesmo descartes podem ocorrer nos roteadores que conectam os enlaces.

Como pode ser observado na tabela 2.1 [26], algumas aplicações são sensíveis aos atrasos e perdas que podem ocorrer na entrega de seus dados. O nível aceitável de perdas e atrasos varia para cada tipo de aplicação.

Para as aplicações tradicionais, tais como transferência de arquivos, correio eletrônico e conteúdo Web, o mais importante é a integridade dos dados, ou seja, não podem haver perdas, mesmo que o tempo para a sua execução seja grande, o que pode dificultar o seu uso, mas não impossibilitá-lo. No entanto, as aplicações de mídia contínua são sensíveis aos atrasos encontrados na rede, pois precisam manter uma certa taxa de consumo durante a sua execução, mas tolerantes a perdas, que podem ocasionar pequenas falhas/pausas durante a sua exibição.

Aplicação	**Tolerância a perdas**	**Largura de banda (mínima)**	**Restrições de Tempo**	**Protocolo**
Transferência de arquivos	Não	-	Não	TCP
Correio Eletrônico	Não	-	Não	TCP
Contéudo Web	Não	-	Não	TCP
Vídeo sob demanda	Sim	Áudio:(alguns)Kbps - 1Mbps Vídeo:10Kbps - 6Mbps	Sim (segundos)	UDP e/ou TCP
Áudio/Vídeo em tempo real	Sim	Idem à anterior	Sim (milisegundos)	UDP e/ou TCP

Tabela 2.1: Exemplos de Aplicações na Internet e seus requisitos de serviço

A Internet oferece dois protocolos de transporte para o desenvolvimento de aplicações: TCP (*Transmission Control Protocol*) e UDP (*User Datagram Protocol*). O protocolo TCP oferece um serviço orientado a conexões e confiável, mas sem garantias de atrasos e taxas de envio. Já o protocolo UDP não é orientado a conexões e não fornece garantias de entrega e nem de atrasos, entretanto permite que a aplicação envie os dados sem qualquer controle de taxa.

O uso de cada um dos protocolos depende das necessidades de cada aplicação, que podem ser classificadas pela tolerância a perdas, largura de banda e restrições de tempo. A tabela 2.1 apresenta alguns exemplos de aplicações, classificando cada uma de acordo com os critérios citados acima e o(s) protocolo(s) de transporte utilizado(s) [26].

O protocolo UDP é mais utilizado para o desenvolvimento de aplicações de mídia contínua, já que estas são, em certo nível, tolerantes a perdas e necessitam manter uma taxa mínima de exibição. Entretanto, o protocolo TCP também é utilizado para aplicações multimídia, mas na maioria das vezes apenas para o transporte de comandos e informações de controle. O protocolo TCP, além de fornecer serviço orientado à conexão e transporte confiável, possui um controle de congestionamento que pode degradar o desempenho destas aplicações multimídia, pois ele limita a taxa de transmissão da aplicação quando a rede está congestionada. Outra limitação do TCP é a impossibilidade de seu uso para transmissões *multicast*. Com *multicast*, o servidor multimídia pode transmitir o conteúdo para diversos clientes ao mesmo tempo, economizando banda da rede.

Como visto anteriormente, cada pacote, TCP ou UDP, passa por diversos roteadores até chegar no seu destino. Dependendo do congestionamento da rede, este pacote pode sofrer atrasos ou ser descartado, conforme ilusta a Figura 2.2. Suponha que t_{proc_i} seja o tempo de processamento no roteador i, t_{fila_i}, o tempo de espera na fila para transmissão no roteador i, t_{trans_i}, o tempo para transmissão do pacote no roteador i e $t_{prop_{ij}}$, o tempo de propagação no enlace entre o roteador i e o roteador j. Como pode ser observado na Figura 2.2, o tempo gasto para enviar um pacote de um roteador para outro é

$$t_{r_i-ao-r_j} = t_{proc_i} + t_{fila_i} + t_{trans_i} + t_{prop_{ij}} \tag{2.1}$$

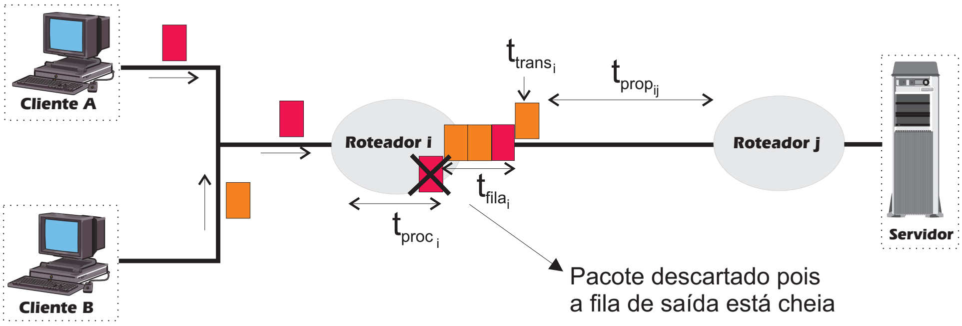

Figura 2.2: Atrasos e perdas na Internet

Sendo assim, o tempo gasto entre o envio e o recebimento deste pacote é a soma dos tempos entre todos os roteadores durante a sua transmissão, incluindo o emissor. Os tempos no emissor são similares aos tempos em um roteador. Suponha $R-1$ roteadores entre o emissor e o receptor, o tempo total entre envio e recepção de cada pacote é

$$\Sigma_{i=1}^{R} t_{r_i-ao-r_{i+1}} \tag{2.2}$$

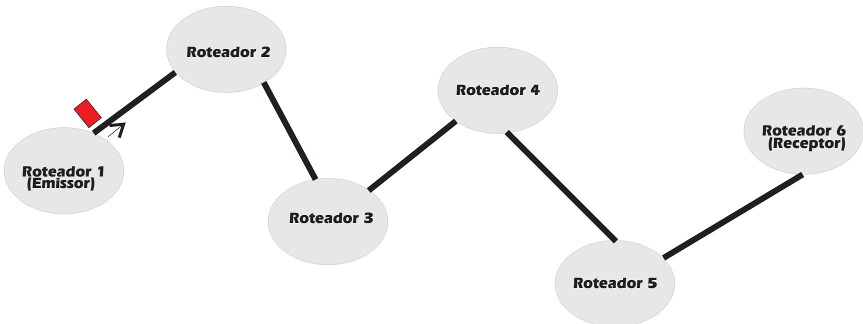

Figura 2.3: Exemplo do tempo de envio de um pacote entre o emissor e o receptor

Considere a situação descrita na Figura 2.3. O tempo de envio do pacote, entre o emissor e o receptor (utilizando a equação 2.2) é

$$\Sigma_{i=1}^{5} t_{r_i-ao-r_{i+1}} = t_{r_1-ao-r_2} + t_{r_2-ao-r_3} + t_{r_3-ao-r_4} + t_{r_4-ao-r_5} + t_{r_5-ao-r_6}$$

Como os atrasos introduzidos na rede são aleatórios, dependendo do tamanho das filas nos roteadores, o tempo entre o envio e a recepção pode variar muito de um pacote para outro. Esta variação dos tempos de chegada entre pacotes de um mesmo fluxo é chamada de *Jitter*.

Várias soluções foram propostas para compensar os atrasos e perdas devido ao tipo de serviço oferecido pela Internet. A seguir são apresentadas algumas destas soluções.

Perdas de pacotes

As aplicações multimídia são tolerantes a pequenas perdas, contudo no caso de perdas em rajadas é necessário o uso de mecanismos para que a QoS seja mantida. Uma das soluções é o uso do protocolo TCP para transmissão dos dados. Entretanto, a cada pacote perdido é solicitada a sua retransmissão, o que aumenta o atraso no recebimento deste pacote. Além disso, devido ao mecanismo de controle de congestionamento do TCP, a taxa de envio é reduzida, o que pode impossibitar o uso da aplicação. Outras soluções podem ser a implementação de um mecanismo de retransmissão utilizando o protocolo UDP ou a utilização de outros mecanismos para recuperação de pacotes, como *Forward Error Correction* e *Interleaving*. O mecanismo de *Forward Error Correction* adiciona informações redundantes para que seja possível a recuperação de pacotes sem a sua retransmissão e o *Interleaving* reordena os pacotes antes do envio de forma que o impacto das perdas seja reduzido [26].

Atrasos e *jitter*

Para compensar os atrasos introduzidos na rede e o *jitter*, a técnica mais utilizada é o atraso no início da exibição (*playout delay*) do vídeo/áudio. O cliente armazena a mídia em um *buffer* (*playout buffer*) e só começa a exibir depois que este *buffer* estiver cheio (Figura 2.4). O tamanho do *buffer* deve ser grande o suficiente para que todos os pacotes sejam recebidos antes de seu tempo para exibição. Caso contrário, ocorrerão interrupções durante a apresentação. A escolha deste tamanho pode ser fixa durante toda a execução ou variável de acordo com o comportamento da rede. Entretanto, quanto maior o tamanho do *buffer*, maior será a latência inicial, que é o tempo gasto entre o pedido do cliente e o início da visualização do objeto.

Vários trabalhos, entre eles [42] e [10], têm considerado o uso de algoritmos para reduzir a variabilidade da taxa de transmissão (*smoothing algorithms*). Estes

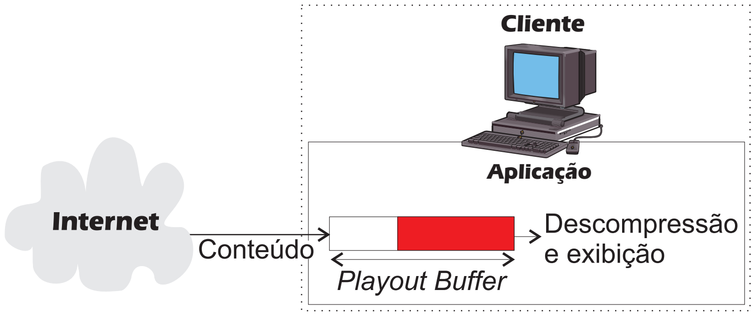

Figura 2.4: *Playout buffer* no cliente

algoritmos exploram o *buffer* do cliente para determinar a taxa de transmissão a ser utilizada. Em [42] é proposto um algoritmo (*optimal smoothing algorithm*) para computar o escalonamento para a transmissão dos dados (dado o vídeo e o tamanho do *buffer* do cliente), e é feito um estudo sobre o impacto dos requisitos necessários para cada fluxo durante a sua transmissão. Em [10] é proposto um esquema que através da monitoração da largura de banda disponível na rede, do *buffer* do cliente e da informação *a priori* do vídeo armazenado, tenta minimizar as flutuações da taxa e manter constante a qualidade durante toda a apresentação.

Outras soluções que podem ser adotadas para compensar os problemas na entrega dos dados são: alterar a qualidade do vídeo, mudar a taxa de exibição ou priorizar as partes mais importantes, de acordo com os recursos disponíveis na rede.

Todas as técnicas comentadas nesta seção são implementadas na camada de aplicação. Na seção 2.4 são apresentadas algumas propostas para modificação da arquitetura de rede de forma a garantir a QoS para as aplicações de mídia contínua.

2.3 Protocolos de domínio público para aplicações multimídia

Nesta seção são apresentados os protocolos RTSP, RTP e RTCP desenvolvidos para aplicações multimídia na Internet [26]. No entanto, vários desenvolvedores implementam protocolos proprietários, tanto para o encapsulamento de pacotes para transmissão, como para interação com o usuário.

***Real Time Streaming Protocol* (RTSP)**

RTSP, definido na RFC (*Request For Comments*) 2326, é um protocolo que permite o controle da transmissão do conteúdo multimídia. Ele define as mensagens entre o cliente e o servidor para oferecer ao usuário os mesmos recursos de um videocassete, tais como avançar e retroceder. Este protocolo não define o padrão de compressão dos dados, nem como os pacotes devem ser encapsulados para transmissão e nem como o cliente deve utilizar o seu *buffer* para compensar as variações dos atrasos da rede (*jitter*).

***Real Time Protocol* (RTP)**

RTP, definido na RFC 1889, é um protocolo que indica como o conteúdo multimídia deve ser encapsulado para transmissão na rede. Ele define os campos necessários de cada pacote, incluindo o número de sequência, tipo de mídia, entre outros. Para utilizar este protocolo, o desenvolvedor deve incorporar o código para o encapsulamento e para a extração de pacotes RTP, tanto no servidor, quanto no cliente.

***Real Time Control Protocol* (RTCP)**

RTCP, também definido na RFC 1889, é um protocolo para coleta de estatísticas que pode ser utilizado junto com o RTP. Os pacotes RTCP são enviados periodicamente contendo estatísticas, tais como número de pacotes recebidos, número de pacotes perdidos e *jitter*.

2.4 Propostas para extensão da arquitetura da Internet

Conforme descrito na seção 2.2, a Internet não fornece QoS para nenhum tipo de tráfego, incluindo áudio e vídeo. Mesmo com o uso dos mecanismos citados para compensar os atrasos e perdas na rede, ainda não é possível a garantia da qualidade de serviço das aplicações de mídia contínua, caso a rede fique congestionada.

Existem diversas opiniões sobre o que deve ser modificado na arquitetura da Internet. Alguns pesquisadores argumentam que aumentando a largura de banda (juntamente com o uso de mecanismos de *cache* para informações já armazenadas) e possibitando o uso de *multicast* na rede é o suficiente para garantir a QoS exigida por estas aplicações. Entretanto, outros dizem que o aumento da largura de banda pode ser muito custoso e não resolverá o problema, pois novas aplicações com maiores requisitos de banda surgirão, como por exemplo, vídeo com alta definição. De outro lado, alguns pesquisadores acreditam que é necessário estender a arquitetura da Internet para que seja possível a reserva de recursos de acordo com a necessidade de cada aplicação. O que implica em grandes mudanças, pois é necessário [26]:

- A criação de diferentes classes de serviço, para que cada roteador possa diferenciar pacotes de diferentes classes de tráfego.

- Um protocolo para reserva de banda e *buffers*, permitindo que cada aplicação declare a QoS desejada e as características do seu tráfego e faça a reserva em todos os roteadores entre o transmissor e o receptor.

- A mudança na política de escalonamento das filas nos roteadores, para garantir as reservas efetuadas. Atualmente, a política de escalonamento utilizada nas filas de transmissão de cada roteador é FIFO (*First-In-First-Out*), ou seja, os pacotes são transmitidos de acordo com a ordem de chegada. Várias políticas de escalonamento de filas podem ser utilizadas para fornecer atendimento diferenciado de forma a garantiar a QoS desejada, entre elas, *Priority Queuing*, *Round Robin* e WFQ (*Weighted Fair Queuing*).

- O uso de um mecanismo de policiamento de tráfego que serve para regular a taxa de transmissão da aplicação, de acordo com a especificação declarada no momento da reserva dos recursos. Entre os critérios para policiamento estão: a taxa média, a taxa de pico e o número máximo de pacotes que podem ser transmitidos em um intervalo de tempo.

- Um controle de admissão para determinar se existem recursos disponíveis para que novas aplicações possam ser admitidas.

Atualmente, existem duas propostas para extensão da Internet [26, 6], arquitetura IntServ (*Integrated Services Internet Architecture*) e a arquitetura DiffServ

(*Differentiated Services Internet Architecture*), ambas desenvolvidas pela IETF (*Internet Engineering Task Force*).

IntServ visa a fornecer garantia de QoS para aplicações individuais. As principais características são: reserva de recursos e controle de admissão, através da caracterização de tráfego e especificação dos requisitos de QoS. O protocolo de sinalização escolhido para a reserva de recursos é o RSVP (*Resource ReServation Protocol*). Cada roteador, ao receber a caracterização de tráfego, a QoS desejada e o tipo de serviço solicitado, executa o controle de admissão que admite ou não a nova aplicação de acordo com os recursos disponíveis. A arquitetura IntServ define duas classes de serviço: *guaranteed* QoS *service* e *controlled-load service*.

Para superar alguns problemas da arquitetura IntServ, tais como escalabilidade, pois cada roteador precisa manter o estado de cada fluxo, foi desenvolvida a arquitetura DiffServ (ainda em seu estágio inicial de seu desenvolvimento). Na arquitetura DiffServ é fornecido um conjunto de classes com diferentes níveis de serviço. Esta proposta fica entre os dois extremos (arquitetura IntServ e arquitetura atual), pois necessita de algumas alterações na arquitetura atual para sua implantação. A idéia é introduzir um número de classes, associar cada pacote a uma determinada classe, fornecer diferentes níveis de serviço nas filas dos roteadores (o que implica em implantar políticas de escalonamento baseadas na classe do pacote) e cobrar dos usuários de acordo com a classe escolhida para a transmissão de seus pacotes. Nesta arquitetura, os roteadores não mantêm o estado de cada fluxo (*stateless*).

As propostas das arquiteturas IntServ e DiffServ especificam a estrutura e o comportamento de seus componentes de uma forma geral, deixando em aberto detalhes específicos de funcionamento, como por exemplo, disciplina de atendimento a ser utilizada e algoritmo para controle de admissão.

Existem vários trabalhos na literatura, entre eles, [25], [23], [51], [52], [7], [47] e [40], com propostas de métodos de caracterização de tráfego e algoritmos para controle de admissão que podem ser utilizados com diferentes algoritmos de escalonamento, como *Earliest Deadline First* (EDF), *Static Priority* (SP), FIFO e *Generalized Processor Sharing* (GPS). Outros trabalhos, como [5] e [24], comparam diversos algoritmos de controle de admissão propostos na literatura através de simulações e experimentos. Em [6] é feito um estudo e análise de desempenho (através de simulações) de controle de admissão fim-a-fim, e em [3] são criados modelos analíticos para avaliar o desempenho de mecanismos EMBAC (*End-to-end Measurement Based connection Admission Control*).

Capítulo 3

Características de Servidores Multimídia

Devido às características de tempo real e à alta interatividade das aplicações de mídia contínua, os tempos para a entrega dos dados destas aplicações devem ser mantidos dentro de certos limites para que a QoS da aplicação não seja prejudicada. Tais tempos de retardo são inseridos durante a transmissão na rede, na máquina do cliente e no servidor multimídia.

No capítulo 2 foram apresentados os problemas e algumas soluções propostas para compensar a variabilidade e as perdas durante a transmissão de mídia contínua na Internet. Neste capítulo são abordados vários aspectos que influenciam no projeto de servidores multimídia no que diz respeito ao oferecimento da qualidade de serviço exigida por estas aplicações.

3.1 Arquitetura geral

A figura 3.1 apresenta a arquitetura geral de um servidor multimídia. Conforme visto no capítulo 2, o acesso às aplicações de áudio/video sob demanda é feito através de servidores multimídia utilizando uma rede de alta velocidade para a transmissão dos dados. O servidor multimídia é o componente responsável pela leitura dos dados armazenados nos discos do sistema, pela transmissão destes dados para todos os clientes conectados e pelo armazenamento de novos conteúdos (objetos). Para superar as limitações de *hardware* (espaço de armazenamento e largura de banda), são utilizados vários discos e servidores em paralelo, possibilitando assim, o aumento do número de clientes atendidos simultaneamente.

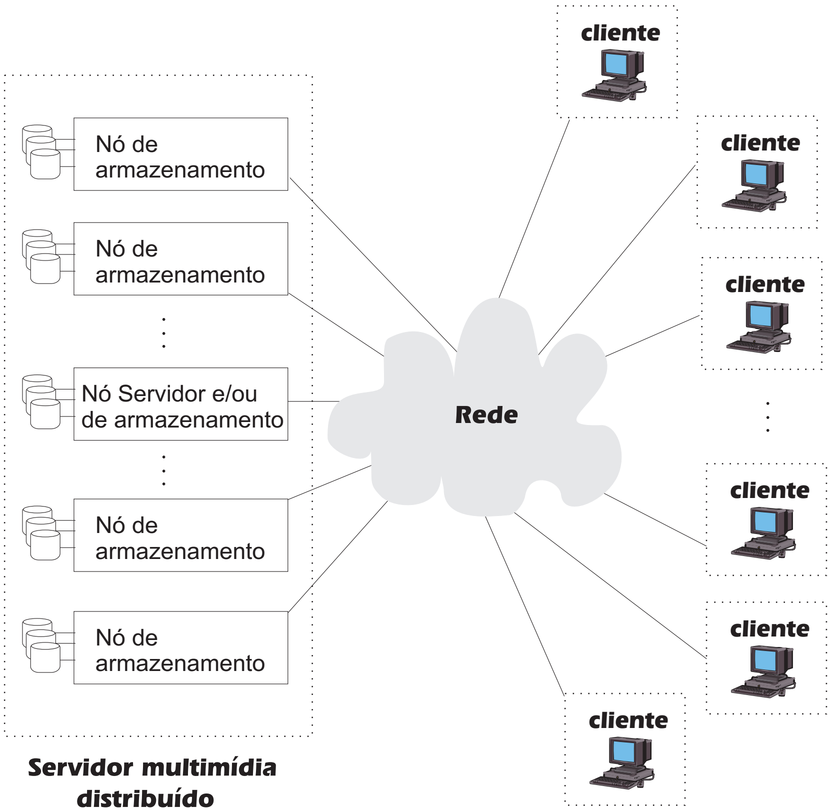

Figura 3.1: Arquitetura de um sistema multimídia

Conforme visto na capítulo 2, antes do armazenamento e transmissão de um objeto, geralmente áudio e vídeo, são utilizadas técnicas de compressão que exploram a redundância nos dados de forma a reduzir o espaço necessário para o armazenamento e a largura de banda para sua transmissão. A técnica mais popular para o armazenamento dos dados é a divisão do objeto (arquivo comprimido) em vários blocos e a distribuição destes blocos nos discos que compõem o sistema, de acordo com uma política de organização dos dados. Quando um cliente solicita a exibição de um objeto, cada bloco de dados que compõe este objeto é lido pelo servidor e armazenado em um *buffer* para posterior envio ao cliente, conforme ilustrado na Figura 3.2.

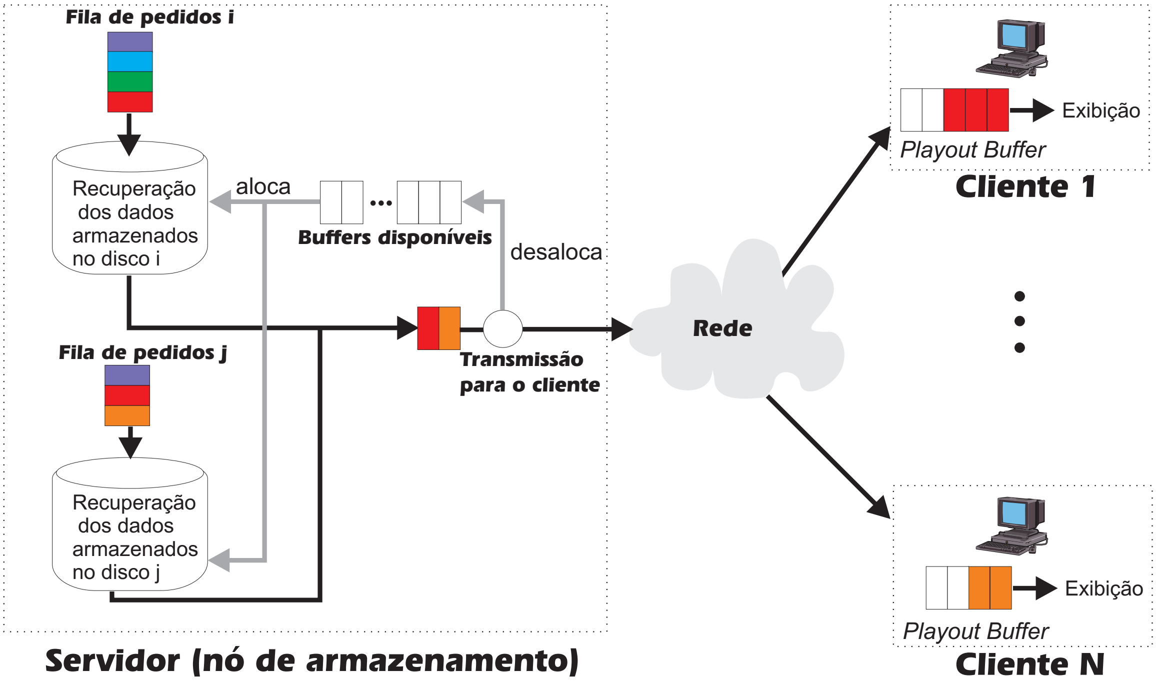

Figura 3.2: Fluxo dos dados no servidor multimídia

O modelo de serviço oferecido pelo servidor multimídia pode ser *client-pull* ou *server-push* [29]. No modelo *client-pull*, o cliente envia os pedidos dos blocos do objeto selecionado para o servidor. Ao receber cada pedido, o servidor faz a leitura dos dados e envia para o cliente. No modelo *server-push*, o servidor é responsável pelo escalonamento dos pedidos e pela transmissão dos dados. Para isto, o servidor precisa saber antecipadamente os dados que cada cliente consumirá para poder escalonar os blocos a serem enviados.

Cada cliente decodifica e exibe o conteúdo recebido do servidor a uma taxa pré-estabelecida (durante a compressão do objeto). Os dados que chegam antes de seu tempo para exibição são armazenados no *buffer* do cliente. O desafio para o servidor é manter dados suficientes nos *buffers* de todos os clientes de forma que nenhuma execução seja interrompida.

O desempenho do servidor é limitado pela taxa de transmissão da rede ou pela taxa de recuperação dos dados nos discos. Antes de admitir um novo cliente, o servidor deve utilizar um algoritmo de controle de admissão para verificar se o novo cliente pode ser admitido sem violar os requisitos de QoS de todos os clientes do sistema.

Vários fatores influenciam no projeto de servidores multimídia, como por exemplo: o modo de transmissão a ser utilizado (*unicast* ou *multicast*) e o gerenciamento de *buffers* do servidor que determina se os *buffers* utilizados pelo servidor para a leitura/escrita dos dados nos discos (ilustrados na Figura 3.2) devem ser alocados dinamicamente (como em [30]) ou estaticamente no momento da inicialização do sistema (como em [29]).

Nas seções a seguir são abordados os seguintes aspectos: políticas de acesso, organização dos dados, algoritmos de escalonamento dos discos, algoritmos de controle de admissão de usuários e técnicas para compartilhamento de recursos de comunicação.

3.2 Políticas de Acesso

Um objeto codificado, como por exemplo um vídeo, é composto de uma seqüência de quadros que devem ser apresentados a uma certa taxa para que a noção de movimento seja mantida, por exemplo, 30 quadros por segundo. Devido às técnicas de compressão, a quantidade de bits de cada quadro varia para permitir que a qualidade do vídeo seja mantida constante durante toda a visualização. Como a taxa de consumo de quadros é constante e o tamanho de cada quadro é variável, a taxa de consumo de dados (bits/seg) no cliente é variável (VBR).

O servidor multimídia pode enviar os dados para o cliente a uma taxa variável (VBR) ou constante (CBR). No envio utilizando VBR, o servidor pode enviar blocos de dados com tamanho constante em intervalos de tempo variáveis ou blocos de dados com tamanho variável em intervalos de tempo constantes. No envio utilizando CBR, como o servidor transmite um bloco de dados com tamanho constante em intervalos de tempo constante e a taxa de consumo de dados no cliente é variável, é necessário um *buffer* muito maior para compensar a variação na taxa de chegada dos quadros e, assim, evitar interrupções durante a visualização do conteúdo.

A quantidade de dados a ser enviada e o tempo de envio dependem do padrão de acesso de cada cliente. Em [41] são descritas duas políticas de acesso aos dados, *Constant Data Length* (CDL) e *Constant Time Length* (CTL). Com a política CDL, a quantidade de dados a ser enviada para cada cliente é fixa, mas os intervalos de envio são variáveis. Com CTL, a quantidade de dados a ser enviada para cada cliente é variável, mas os intervalos de envio são constantes.

Para evitar fragmentação do espaço de armazenamento e da largura de banda dos discos, a maior parte dos servidores armazena os objetos dividindo-os em blocos de mesmo tamanho. O tamanho de cada bloco deve ser grande o suficiente para amortizar os tempos de *seek* e de rotação durante o tempo de acesso ao disco. Tais tempos são detalhados na seção 3.4.

3.3 Organização dos Dados

A organização dos dados adotada pelo servidor multimídia define como os objetos devem ser armazenados nos discos do sistema. Uma solução simples poderia ser o armazenamento de todos os blocos de um mesmo objeto em um único disco. Entretanto, dependendo da popularidade de cada objeto, isto poderia causar desbalanceamento de carga entre os discos. Neste caso, um novo cliente que desejasse assistir um objeto armazenado no disco sobrecarregado não seria admitido mesmo se tivesse largura de banda disponível nos demais discos, causando a subutilização do sistema. Devido a estes problemas, a estratégia mais utilizada é a divisão dos blocos de um mesmo objeto entre todos os discos do sistema. Desta forma, os pedidos de leitura de todos os clientes são distribuídos, compartilhando a largura de banda de todos os discos [17, 46].

A maioria dos servidores multimídia propostos na literatura são projetados para apenas um tipo de aplicação, tipicamente vídeo sob demanda. Com isso, tais sistemas levam em conta o padrão de acesso destas aplicações para organizar os dados nos discos.

Entre as várias propostas para organização de dados em servidores multimídia que exploram o padrão de acesso, a mais comum é baseada na técnica de *Striping*, na qual cada objeto é dividido em blocos de dados de tamanho constantes, denominados de *stripe units*, e estes são distribuídos em todos os discos utilizando a política *round robin*, de modo que blocos consecutivos de um mesmo objeto são armazenados em discos consecutivos [36, 45, 46, 17]. Geralmente, com o uso de *Striping*, o servidor opera em ciclos de duração fixa com todos os discos sincronizados.

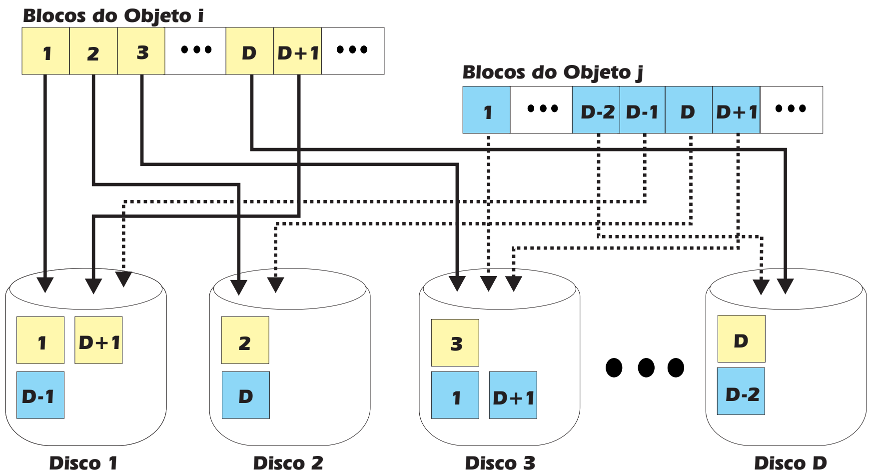

Figura 3.3: *Data Striping*

A Figura 3.3 ilustra o método de *Striping* mais utilizado, *Coarse-Grained layout* [45]. O esquema básico deste método considera que todos os clientes acessam os dados de forma seqüencial e com a mesma taxa constante de dados. A cada ciclo é feita a leitura de um bloco de dados de um dos discos para cada cliente. Se um cliente é servido no ciclo i pelo disco j, então no ciclo $i+1$, este cliente será servidor pelo disco $j+1$. Se j é o último disco, então o próximo disco volta a ser o primeiro, ou seja, o próximo disco é (j módulo D) + 1, onde D é o número de discos do sistema. Desta forma, a cada ciclo, a carga de um disco (pedidos de leitura dos clientes) é movida para o próximo disco, evitando a sobrecarga de qualquer disco. Neste esquema são necessários apenas dois *buffers* para cada cliente, um para o bloco que está sendo lido no ciclo corrente e outro para o bloco lido no ciclo anterior que está sendo transmitido para o cliente. Um novo cliente pode ser admitido pelo servidor enquanto existir pelo menos um disco com largura de banda disponível para o atendimento deste novo candidato. Como a carga de cada disco é movida de um disco para outro, o servidor precisa esperar até que a banda disponível circule até o disco que contém o primeiro bloco de dados do objeto requisitado pelo novo cliente para então iniciar o seu atendimento.

Existem alternativas que podem ser utilizadas para o suporte de clientes com requisitos de banda diferentes, entre elas, *Staggered Striping layout* [19], e outras para o suporte de fluxos VBR utilizando *data striping*.

Na prática, existem diversos fatores que podem dificultar a previsão do padrão de acesso e com isso dificultar a organização dos dados. Entre eles: as técnicas de codificação que, para manter a qualidade constante, geram fluxos com taxa variável, introduzindo variabilidade temporal no padrão de acesso; funcionalidades do tipo videocassete, tais como avançar e retroceder que alteram a sequencialidade de acesso; utilização de múltiplas resoluções; além do surgimento de novas aplicações multimídia, tais como ambientes virtuais 3D interativos em que o padrão de acesso é muito menos previsível do que nas aplicações convencionais [44].

Para contornar essas dificuldades, foi proposta a técnica de *Random Data Allocation* [2, 46], na qual cada objeto é divido em blocos de tamanho constante e para cada bloco a ser armazenado, é escolhido um disco e uma posição aleatoriamente, conforme ilustrado na Figura 3.4. Assim, todos os padrões de acesso de diferentes aplicações são mapeados no mesmo padrão aleatório no nível físico, facilitando o suporte a diversos tipos de aplicações. Entretanto, apenas com alocação aleatória, podem ocorrer flutuações na carga dos discos, fazendo com que o sistema possa se tornar desbalanceado durante alguns intervalos de tempo. Para resolver este problema, é utilizada a replicação dos blocos e um algoritmo que escolhe o disco com a menor fila para o atendimento do pedido [36, 46].

Com alocação aleatória, não há a necessidade de sincronização entre os discos do sistema. Cada pedido recebido é redirecionado para o disco com a menor fila que contém uma cópia do bloco requisitado. Todos os pedidos de um determinado disco são atendidos independentemente dos demais discos. O uso de replicação de dados, além de ajudar no balanceamento de carga, serve para aumentar a confiabilidade do sistema, pois possibilita que a execução dos clientes não seja interrompida na presença de falhas [45, 44, 37, 36]. Comparações entre *Striping* e *Random Data*

Allocation são apresentadas em [44, 45].

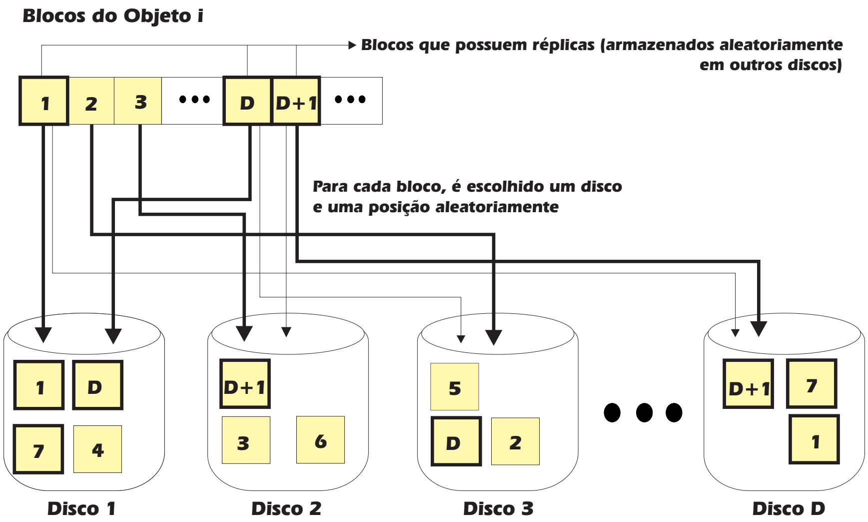

Figura 3.4: *Random Data Allocation*

Tanto com a técnica de *Striping*, quanto com a alocação aleatória, o servidor mantém uma estrutura de dados para fazer o mapeamento de cada bloco lógico requisitado para o(s) bloco(s) físico(s), de todos os objetos armazenados no sistema.

3.4 Escalonamento dos Pedidos

A política de escalonamento determina a ordem em que os pedidos serão enviados para cada disco do sistema. Os principais componentes do tempo de serviço de um pedido de leitura/escrita são: o tempo de *seek*, o tempo de rotação e o tempo de transferência dos dados. Para acessar os dados armazenados em uma determinada posição no disco, é necessário o reposicionamento da cabeça de leitura/escrita sobre esta área. O tempo gasto para posicionar a cabeça de leitura/escrita do cilindro atual até o cilindro desejado é denominado de tempo de *seek*, e o tempo gasto até que a posição desejada (no cilindro) seja posicionada sob a cabeça de leitura/escrita é denominado de tempo de rotação. Como pode ser observado, o tempo de serviço de um pedido depende do tempo de *seek* e de rotação, que por sua vez, dependem da localização atual da cabeça de leitura/escrita e da localização dos dados solicitados no disco.

Várias políticas foram desenvolvidas para otimizar o tempo de serviço, reduzindo o tempo de *seek* e de rotação, e conseqüentemente aumentando a vazão do sistema. As técnicas mais utilizadas para a redução do tempo de *seek* são *Shortest Seek First* (SSF), SCAN (na qual os pedidos são ordenados de acordo com a sua posição nos cilindros do disco, desta forma a cabeça de leitura/escrita move sempre na mesma direção, do cilindro mais interno para o mais externo ou vice-versa) e *bidirectional* SCAN (na qual, a cada SCAN, a cabeça de leitura/escrita alterna a direção de movimento). Algumas técnicas que tentam minimizar o tempo de *seek* e de rotação são *Shortest Time First* (STF), *Grouped Shortest Time First* (GSTF) e *Weighted Shortest Time First* (WSTF) [46, 17].

No entanto, para servidores de mídia contínua, o algoritmo de escalonamento do disco deve garantir as restrições de tempo para todos os clientes admitidos, além de otimizar o tempo de serviço. Por este motivo, são necessárias algumas modificações para o uso desses algoritmos em sistemas de tempo real.

A maioria dos servidores multimídia processa os pedidos em ciclos de duração fixa. A cada ciclo é lido um bloco de dados para cada cliente do sistema. De acordo com a duração de um ciclo e o tamanho do bloco de dados utilizado, existe um número máximo de blocos que pode ser lido dos discos dentro deste intervalo de tempo, de forma a garantir a QoS oferecida [13, 46]. A garantia de qualidade de serviço pode ser determinística (considerando o pior caso) ou estatística. Devido à variabilidade dos tempos de leitura, alguns discos podem acabar o serviço antes dos demais, ficando ociosos até o final do ciclo, e assim reduzindo o desempenho do sistema. Diversos algoritmos têm sido estudados para o atendimento baseado em ciclos. Os mais utilizados são SCAN e *bidirectional* SCAN, explicados anteriormente.

Outros algoritmos adotados para o atendimento dos pedidos em servidores multimídia são *Earliest Deadline First* (EDF), no qual os pedidos são atendidos de acordo com o tempo que o bloco tem que ser enviado para o cliente (*deadline*), *Static Priority* (SP), no qual os pedidos são atendidos de acordo com a sua prioridade, e *First Come First Serve* (FCFS), onde os pedidos são atendidos de acordo com a ordem de chegada. Em ambos os métodos, nem o tempo de rotação e nem o tempo de *seek* são minimizados. Entretanto, como tais algoritmos não são baseados em ciclos, para cada disco, o serviço do próximo pedido pode ser iniciado assim que o atendimento do pedido corrente seja finalizado.

Em [46] são utilizados dois algoritmos de escalonamento, RTSCAN (realtime SCAN) e FCFS. O RTSCAN é uma variação do algoritmo *bidirectional* SCAN, no qual o número de pedidos atendidos em um ciclo é limitado, de forma a garantir um limite de tempo para o atendimento. Porém, em várias simulações foi utilizado o algoritmo FCFS (limitando o tamanho do ciclo em um único pedido), pois ele minimiza as variações do tempo de serviço de cada pedido, embora o tempo médio de serviço possa ser maior do que com o uso de outros algoritmos, como por exemplo SCAN. A reordenação dos pedidos, de acordo com a localização no disco, aumenta a variação do tempo de serviço, pois pedidos que chegaram mais tarde (ou com *deadline* maior) podem ser atendidos antes de pedidos que chegaram primeiro (ou com *deadline* menor). Quanto maior o tamanho do ciclo, maior o número de pedidos que podem ser reordenados de forma a reduzir o tempo de *seek*. Entretanto, o limite de tempo oferecido pode ser maior, devido ao aumento da variação dos tempos de serviço, ou menor, devido à redução do tempo médio obtida pela redução do tempo de *seek*.

Algumas combinações de algoritmos de escalonamento podem ser utilizadas, como SCAN-EDF, no qual os pedidos são atendidos de acordo com o seu *deadline* e, entre os pedidos com mesmo *deadline*, os blocos são ordenados usando o algoritmo SCAN. Em [13] são apresentados diversos algoritmos para o atendimento de pedidos com e sem restrições de tempo dentro de um ciclo, entre eles, *Non-Greedy* FCFS (NG-FCFS), *Non-Greedy Gated* (NG-Gated), *Partially Greedy* FCFS, *Partially Greedy Gated*, *Greedy* FCFS e *Greedy Gated* (G-Gated), juntamente com modelos analíticos para o estudo do desempenho destes algoritmos. Em todos eles, os pedidos de tempo real são ordenados de acordo com o algoritmo SCAN.

Outros trabalhos estudam as políticas de escalonamento, juntamente com as políticas de organização de dados e de acesso, entre eles [45] e [41]. Em [45] é feita a comparação entre *Data Striping*, utilizando o algoritmo SCAN e o *Fixed Order* (no qual, em todos os ciclos, os pedidos são atendidos sempre na mesma ordem), com alocação aleatória, utilizando o algoritmo FCFS. Em [41] são estudadas as duas políticas de acesso, CTL e CDL (descritas na seção 3.2), e comparados dois algoritmos de escalonamento, um baseado em ciclos (juntamente com CTL) e o outro baseado em *deadlines* (juntamente com CDL). Além disto, são propostos dois algoritmos de escalonamento baseados em *deadlines*, *Workahead CTL-Round* e *Lazy*-EDF.

3.5 Controle de Admissão de Usuários

Como visto no capítulo 2, a técnica mais utilizada para compensar as variações da rede é o uso de um *buffer* (*playout buffer*) no cliente. Desta forma, este *buffer* precisa atingir um certo limite (armazenar uma certa quantidade de blocos de dados) antes de começar a exibição do conteúdo. Para que o usuário não perceba alterações, o tempo de resposta de um pedido do cliente, ou seja, o tempo entre o pedido de um bloco de dados até o recebimento deste, tem que ser no máximo o tempo de consumo de todos os blocos no *buffer* do cliente, conforme ilustrado na Figura 3.5. Quanto maior o tamanho do *playout buffer*, maior poderá ser a variabilidade nos tempos de atendimento do servidor e maior será a latência inicial. Entretanto, como observado na seção 2.2, algumas aplicações são tolerantes a pequenas falhas durante a sua execução, como por exemplo a perda de alguns quadros de um filme ou amostras de áudio.

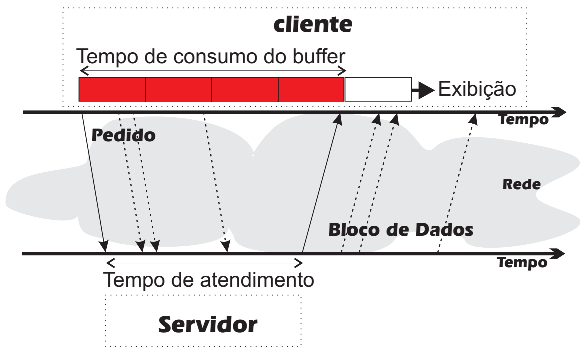

Figura 3.5: Tempo de resposta do servidor

Devido às variações nas taxas de consumo de cada cliente, vários pedidos podem chegar no servidor em um curto intervalo de tempo (rajadas). Além disso, pode acontecer de vários pedidos de vários clientes chegarem ao mesmo tempo. Como os recursos utilizados pelo servidor multimídia são limitados pela taxa de recuperação de dados nos discos e pela taxa de transmissão da rede, para garantir a qualidade de serviço exigida pelas aplicações de mídia contínua, o servidor deve utilizar um controle de admissão para decidir se um novo cliente pode ser admitido sem violar os requisitos dos clientes já admitidos.

Vários algoritmos para controle de admissão foram propostos na literatura. O objetivo do controle de admissão é utilizar de forma eficiente os recursos do sistema (atendendo ao maior número de clientes possível), e ainda garantir a QoS para todos os clientes.

Os algoritmos de admissão podem ser classificados de acordo com o nível de qualidade de serviço oferecido. Na literatura são definidos três níveis de QoS [17, 16]:

- Determinístico: oferece a garantia de que todos os dados serão entregues dentro do intervalo de tempo exigido.
- Estatístico: oferece garantia estatística para a entrega dos dados, ou seja, não é 100% garantida a entrega de todos os dados, podendo ocorrer flutuações na QoS. Neste nível de serviço, o servidor considera o comportamento estatístico do sistema durante a admissão de novos usuários.
- *Best Effort*: sem garantias. Onde o servidor tenta o possível, atendendo primeiro os pedidos com maior prioridade para depois atender os pedidos dos clientes deste tipo. Geralmente este nível de QoS está associado aos pedidos que não possuem restrições de tempo.

Outros trabalhos, como [22], [50] e [21], classificam os algoritmos de controle de admissão em duas categorias: os algoritmos baseados em parâmetros (*Parameter-based approach*) e os algoritmos baseados em medidas do comportamento atual do sistema (*Measurement-based approach*, também chamados de preditivos), sendo que os algoritmos baseados em parâmetros são subdivididos em determinísticos e estatísticos.

A desvantagem dos algoritmos determinísticos em relação aos algoritmos estatísticos e aos algoritmos baseados em medidas é que considerando os piores casos para alocação dos recursos (por exemplo, alocação baseada na taxa de pico), de modo a garantir a entrega dos dados, o sistema é subutilizado, pois existem variações de taxas, tanto nos tempos de serviço para leitura dos dados, quanto nas taxas de consumos dos clientes (devido às técnicas de compressão utilizadas - citadas no capítulo 2), que podem ser exploradas para atender um maior número de clientes simultaneamente.

A utilização dos recursos do servidor depende da escolha do algoritmo de controle de admissão. As principais diferenças entre cada um, além do nível de QoS oferecido, são o método de caracterização do tráfego de cada cliente e o teste de admissão.

O objetivo principal do método de caracterização do tráfego é descrever o tráfego do novo usuário o mais aproximado possível para que o algoritmo de controle de admissão possa efetuar a reserva mais precisa dos recursos necessários para cada sessão. Alguns exemplos de caracterização do tráfego são: taxa média do objeto, taxa de pico, histogramas, envelopes de tráfego e a distribuição dos quadros de um filme.

Através da caracterização do tráfego e das informação dos recursos disponíveis, é realizado o teste de admissão que verifica a possibilidade de atendimento para este novo usuário. Um exemplo de controle de admissão simples é a verificação se a soma da taxa já alocada para os clientes admitidos com a taxa média ou de pico do novo tráfego não ultrapassa a capacidade do servidor. Outros algoritmos de controle de admissão mais complexos realizam, por exemplo, uma simulação para verificar se a carga atual juntamente com a carga do novo usuário pode ser atendida sem violar as garantias de serviço [39, 21, 33].

Vários trabalhos abordam as limitações (dos discos e da rede) separadamente ou em conjunto, como em [32]. Na seção 2.4 foram citados alguns trabalhos que abordam controle de admissão para a rede. Para servidores multimídia, a maior parte dos trabalhos aborda apenas as limitações dos discos, como em [11], [50], [49], [33], [22], entre outros.

O controle de admissão também depende do modo de operação do servidor, como por exemplo, orientado a ciclos e não orientado a ciclos. A maioria dos servidores multimídia faz o atendimento dos pedidos baseado em ciclos de duração constante. A técnica tradicional adotada é a leitura de um bloco de dados para cada cliente em cada ciclo de atendimento. Dado que a duração de um ciclo é T, este esquema garante que o cliente recebe os dados a uma taxa $R_s = B/T$, onde B é o tamanho de um bloco de dados. A seqüência de dados enviada para um cliente é chamada de fluxo (*stream*). Para um ciclo de duração T, existe um número máximo de blocos que pode ser lido de cada disco. Este número máximo limita o número de clientes que podem ser admitidos pelo servidor. Desta forma o controle de admissão deve admitir ou não o novo cliente, baseado na carga atual do sistema. Para o cálculo do número máximo de blocos atendidos em um ciclo, pode-se considerar os piores casos no tempo de serviço (de modo a oferecer garantias absolutas de tempo real) ou considerar o comportamento estatístico do sistema.

No trabalho [44] é feita a comparação entre alocação aleatória (sendo que o atendimento dos pedidos não é orientado a ciclos) e *data striping* (na qual o atendimento dos pedidos é baseado em ciclos), assumindo que os blocos são enviados com taxa CBR. Para as duas estratégias de alocação, ambas detalhadas na seção 3.3, o número máximo de pedidos por disco é calculado considerando garantias estatísticas. Para cada estratégia, é computado o número máximo de clientes por disco tal que a probabilidade de não conseguir atender um pedido a tempo (P_{miss}) seja $P_{miss} \leq \epsilon$, que é a probabilidade usada para calcular o desempenho dos dois sistemas (por exemplo, 10^{-6}). Para calcular o número máximo de pedidos por disco com *data striping* foi utilizado um modelo detalhado do disco e experimentos para extrair os tempos de serviço. Para avaliar o desempenho do RIO foi construído um simulador, além de experimentos realizados com os discos utilizados pelo protótipo. Logo após, foram calculados o número máximo de clientes, para alocação aleatória e *data striping*, considerando a taxa de consumo dos clientes (característica da aplicação) e o limite de tempo para atendimento (*delay bound* - D_B - que depende da quantidade de *buffers* alocada para cada cliente).

A estratégia de leitura de um bloco para cada cliente durante um ciclo funciona bem para fluxos CBR. Entretanto para o atendimento de fluxos VBR, a carga muda de um ciclo para outro, sendo necessário considerar as variações na carga dos discos para fornecer a qualidade de serviço desejada e melhor utilização dos recursos.

A seguir são descritos alguns dos algoritmos propostos na literatura. Tais algoritmos são para servidores que efetuam o atendimento dos pedidos baseado em ciclos. As principais diferenças são a forma pela qual é estimado o tempo de serviço dos discos e a carga gerada pelos clientes do sistema.

Em [11] é feito um estudo comparativo de algoritmos de controle de admissão, considerando os três níveis de QoS: determinístico, estatístico e *best-effort*. Tais algoritmos são descritos em termos de como o tempo de serviço dos discos é estimado. A cada chegada de um novo usuário, o controle de admissão tenta prever o tempo total de serviço para todos os pedidos e admite este novo cliente se o tempo de serviço previsto é menor que a duração do ciclo. O tempo de serviço de um pedido j leva em conta o tempo para posicionar a cabeça de leitura/escrita a partir da localização do pedido j - 1 até a localização do pedido j, o tempo de rotação e o tempo de transferência dos dados. Os algoritmos determinísticos consideram os piores casos para a estimativa do tempo de serviço, enquanto que os algoritmos estatísticos utilizam medidas extraídas anteriormente do sistema, podendo utilizar os piores casos ou a média destes valores obtidos. Através das implementações, foram comparados os desempenhos dos algoritmos estudados de acordo com o número de usuários admitidos sem causar ciclos cujo tempo de atendimento fosse maior que o limite pré-estabelecido. A diferença entre os algoritmos determinísticos e os estatísticos ficou entre 10% e 40%.

Outras propostas de algoritmos de controle de admissão para servidores multimídia são [49] e [50]. Em [49] o comportamento futuro do sistema é previsto a partir do comportamento observado, ou seja, no momento da admissão de um novo cliente, o algoritmo de controle usa as medidas da carga atual (o tempo médio gasto na leitura de um bloco e seu desvio padrão observado durante um intervalo de tempo) para prever o tempo de serviço nos próximos ciclos, juntamente com a quantidade de blocos a ser lida, em cada ciclo, pelo novo cliente. Considere o servidor multimídia servindo n clientes, cada um requisitando um vídeo diferente (*video strand*), $S_1, S_2, \ldots, S_n$, com taxas (bytes/sec), $R_1, R_2, \ldots, R_n$, respectivamente. Seja $k_1, k_2, \ldots, kn$, o número de blocos a ser lido para cada cliente $S_1, S_2, \ldots, S_n$ durante cada ciclo. Se C é o número de cilindros do disco e o tempo de *seek* para mover a cabeça de leitura/escrita de um cilindro c_1 para um cilindro c_2 é $a + b \times |c_1 - c_2|$, onde a e b são constantes (parâmetros do tempo de *seek*), o limite superior do tempo total de *seek* de um ciclo é $a \times \sum_{i=1}^{n} k_i + b \times C$ e, sendo l_{rot}^{max} o tempo máximo de rotação, o tempo total de rotação durante um ciclo é limitado por $l_{rot}^{max} \times \sum_{i=1}^{n} k_i$. Desta forma, o tempo total de serviço para cada ciclo é limitado por:

$$\tau = b \times C + (a + l_{rot}^{max}) \times \sum_{i=1}^{n} k_i \tag{3.1}$$

Para garantir os requisitos de QoS, dado que M é o tamanho do bloco de dados utilizado e R é a duração de um ciclo, sendo $R = \min_{i \in [1,n]} \left(\frac{k_i \times M}{R_i}\right)$, o critério de admissão é $\tau \leq R$.

Este algoritmo pode, dependendo do tipo de serviço solicitado pelo cliente, fornecer garantias determinísticas (considerando os piores casos) ou garantias estatísticas (considerando o tempo médio de serviço observado e a fração de blocos que deve ser garantida para cada fluxo).

Considere n_d e n_p o número de clientes com garantias determinísticas e preditivas, respectivamente ($n = n_d + n_p$); e P_i a fração de blocos do cliente S_i que deve ser recuperada de modo a satisfazer os requisitos de QoS do cliente. Para os clientes com garantias determinísticas, P_i é 1, enquanto que para os clientes com garantias preditivas, P_i é menor que 1. Seja K o número de blocos lidos em um ciclo, sendo $K \leq \sum_{i=1}^{n} k_i$. O tempo médio gasto na leitura de um bloco é dado por $\eta = \tau / K$. Como este controle de admissão é baseado na suposição de que o tempo gasto na leitura de um bloco de dados não muda significativamente mesmo após a admissão de um novo usuário, são mantidos os valores de η durante os W ciclos mais recentes para prever o comportamento futuro. Se η_{avg} e σ representam a média e o desvio padrão de η durante W ciclos, o tempo previsto ($\hat{\eta}$) para os próximos ciclos pode ser calculado como $\hat{\eta} = \eta_{avg} + \epsilon \times \sigma$, onde ϵ é uma constante determinada empiricamente.

Quando o próximo cliente $n+1$ faz a requisição pelo vídeo S_{n+1}, com os parâmetros taxa R_{n+1} e k_{n+1} blocos a serem lidos durante cada ciclo, é realizado o teste de admissão de acordo com a garantia de serviço solicitada.

- Caso o cliente solicite garantias determinísticas, o servidor deve verificar se as garantias tanto para os n_d clientes, quanto para os n_p clientes, podem ser mantidas. Desta forma um novo usuário é admitido se as equações 3.2 e 3.3 são satisfeitas. Para verificar as garantias determiníticas é utilizada a equação

$$\tau = b \times C + (a + l_{rot}^{max}) \times \sum_{i=1}^{n_d+1} k_i \leq R \tag{3.2}$$

  Para verificar as garantias preditivas é utilizada a equação

$$\hat{\eta} \times (\sum_{i=1}^{n+1} k_i * P_i) \leq R \tag{3.3}$$

- Caso o cliente solicite garantias preditivas, o servidor deve verificar apenas se a equação 3.3 é satisfeita.

Em [50] é proposto um algoritmo de admissão que fornece garantia estatística para cada cliente, explorando a variação dos tempos de serviço do disco e a variação na taxa de consumo de cada cliente (número de blocos por ciclo). Esta última é obtida através da distribuição do tamanho dos quadros (*frames*) de cada filme.

Considere $f_1, f_2, \ldots, f_n$ o número de *frames* a serem lidos por ciclo para cada cliente $S_1, S_2, \ldots, S_n$, com taxas (*frames*/sec), $R_1, R_2, \ldots, R_n$, respectivamente. Seja q a probabilidade de *overflow* ($P(\tau > R) = q$), R a duração de um ciclo ($R = \min_{i \in [1,n]} f_i/R_i$), P_i a fração de *frames* do cliente S_i que deve ser recuperada de modo a satisfazer os requisitos de QoS e F_0 o número de *frames* que são garantidos de serem lidos durante os ciclos onde $\tau > R$. Os requisitos de serviço de um cliente são satisfeitos se

$$q \times F_0 + (1-q) \times \sum_{i=1}^{n} f_i >= \sum_{i=1}^{n} f_i * P_i \tag{3.4}$$

Para calcular q é utilizada a equação 3.5, onde τ_k representa o tempo para acessar k blocos (obtido através da distribuição do tempo de serviço), B_i representa o número de blocos que contém f_i *frames* do vídeo S_i, B é a variável aleatória que representa o número de blocos a serem lidos em um ciclo ($B = \sum_{i=1}^{n} B_i$), e k_{min} e k_{max} representam seus valores mínimo e máximo ($k_{min} = \sum_{i=1}^{n} B_i^{min}$; $k_{max} = \sum_{i=1}^{n} B_i^{max}$).

$$q = P(\tau > R) = \sum_{k=k_{min}}^{k_{max}} P(\tau_k > R)P(B = k) \tag{3.5}$$

O valor de F_0 é dependente do número de blocos a serem lidos e da relação entre o tamanho do bloco de dados e o tamanho dos *frames*. O tempo total de serviço para cada ciclo pode ser calculado de acordo com a equação 3.1. Já que $\tau \leq R$, o número de blocos, k_d, que são garantidos de serem lidos é limitado por

$$k_d \leq \frac{R - b \times C}{a + l_{rot}^{max}} \tag{3.6}$$

Assumindo que $f(S_i)$ representa o número mínimo de *frames* que podem estar contidos em um bloco de S_i, o limite inferior do número de *frames* acessados durante um ciclo onde $\tau > R$ é dado por

$$F_0 = k_d \times \min_{i \in [1,n]} f(S_i) \tag{3.7}$$

Quando um novo cliente solicita a abertura de um vídeo S_{n+1}, através dos parâmetros: média e desvio padrão da variável aleatória B_{n+1}, B_{n+1}^{min} e B_{n+1}^{max} e $f(S_{n+1})$, são calculados os valores de q e F_0. O novo cliente só é admitido se a condição abaixo for satisfeita:

$$q \times F_0 + (1 - q) \times \sum_{i=1}^{n+1} f_i >= \sum_{i=1}^{n+1} f_i * P_i \tag{3.8}$$

Para evitar que o tempo de serviço de todos os blocos exceda a duração de um ciclo ($\tau > R$), [49] e [50] utilizam um algoritmo que tenta reduzir a carga de um ciclo, descartando a quantidade de dados necessária, de modo que a perda é distribuída entre todos os clientes com garantias estatísticas.

Em [22] o processo de controle de admissão é dividido em duas partes: uma *off-line* e outra *on-line*. No processo *off-line*, são extraídas medidas de desempenho do servidor e informações dos vídeos armazenados, como por exemplo, número de blocos que podem ser lidos em um ciclo e média de blocos por ciclo de um vídeo, respectivamente. Nesta fase é gerado um gráfico da probabilidade de *overflow* em um ciclo de acordo com o número de blocos requisitados (através de simulações) e é obtida a demanda de blocos por ciclo e a probabilidade da demanda de blocos de acordo com o número de clientes admitidos. No processo *on-line*, primeiro, é configurada a QoS a ser oferecida para os clientes do sistema, como por exemplo, QoS maior que 90% (o que significa que a quantidade de ciclos cujo tempo de atendimento exceda o limite pré-estabelecido é limitado a 10% da quantidade total de ciclos). Para uma dada probabilidade de *overflow*, existe um número de blocos que podem ser lidos em um ciclo (L) que corresponde a taxa de *overflow* escolhida. De acordo com o número de blocos escolhido, é configurado o número inicial de clientes que podem ser admitidos (M_n, cuja demanda de blocos não exceda o valor determinado). Não é feito nenhum teste de admissão para os usuários que chegam dentro deste número inicial de clientes. Mas, a partir do instante que este número é ultrapassado, um novo cliente é admitido se a equação $L \geq M_{n+1} + E_n$ for satisfeita, onde L é o número máximo de blocos que podem ser lidos em um ciclo de acordo com a QoS oferecida, M_{n+1} é a média de blocos para $n + 1$ clientes e E_n é um parâmetro usado para controlar a utilização do sistema e as variações de taxas. Os parâmetros L e M_n são calculados no processo *off-line*. Logo após, é verificada a taxa de *overflow* com o novo cliente admitido. Caso o valor calculado neste segundo teste seja menor que a QoS oferecida, o novo cliente não é admitido. Após os testes realizados, o parâmetro E_n é ajustado de acordo com os resultados obtidos.

Além de [50, 22], outros trabalhos também exploram a informação *a priori* do vídeo armazenado para alocação dos recursos no servidor, ao invés da taxa média ou de pico do vídeo, entre eles, [39], [33] e [21].

Em [39] é proposto um mecanismo para fornecer serviço determinístico para fluxos VBR servidos pelos discos do sistema. Durante o pedido de admissão, para cada objeto (filme) existe um *trace* (*demand trace*) que descreve os pedidos de I/O (número de blocos a serem lidos em cada ciclo durante a sua execução). O servidor mantém o pior tempo de serviço em cada ciclo para cada disco do sistema (*load trace*). Antes do novo cliente ser admitido, o *trace* do cliente é combinado com a carga dos discos para verificar se algum dos discos excede a capacidade (duração de um ciclo). Caso o novo cliente não possa ser admitido no ciclo atual, a política de escalonamento de fluxos procura um ciclo no qual o novo fluxo possa ser escalonado. O novo fluxo é admitido se existir um k tal que $load[i][j+k] + serv\_time(demand[j]) \leq round\ time$ para todo j, sendo i o disco que contém os dados para o ciclo j, $load[i][j]$ a carga do disco i no ciclo j e $demand[j]$ o número de blocos a serem lidos do disco no ciclo j. A função $serv\_time()$ estima o pior tempo de serviço necessário para ler um dado número de blocos do disco de acordo com a carga atual.

O método de caracterização utilizado por [21] é parecido com o do [39]. Para cada objeto, a duração total de um vídeo é divida em intervalos constantes para a extração dos requisitos de banda para cada intervalo. Neste trabalho são propostos dois algoritmos para fluxos VBR, *Future-Max* (FM) e *Interval Estimation* (IE). Dada a largura de banda disponível do disco e um novo fluxo a ser admitido, o algoritmo FM utiliza o próximo maior requisito de largura de banda do vídeo para alocação dos recursos, ou seja, este requisito é atualizado à medida que o ciclo associado a este requisito de banda é alcançado, desta forma é escolhido o próximo maior requisito de banda a partir deste ciclo. Já o algoritmo IE utiliza uma combinação da maior largura de banda necessária e da largura média dos intervalos.

Em [33] são estudados e comparados quatro algoritmos para controle de admissão (para os discos) considerando fluxos VBR: *Simple Maximum*, *Instantaneous Maximum*, *Average* e *vbrSim*, juntamente com um algoritmo denominado *Optimal*, que possui o completo conhecimento do futuro para executar as suas decisões de admissão. Dois recursos do servidor são medidos: a largura de banda para leitura dos discos e a quantidade de *buffers* disponível. A largura de banda é definida por um número fixo de blocos que podem ser lidos dentro de um intervalo de tempo. A quantidade de blocos cuja leitura é garantida dentro de um intervalo de tempo é chamada de *minRead*. Este valor é obtido através de um programa que determina o número máximo de blocos que podem ser lidos considerando as piores condições. Para cada novo cliente é criada uma lista de pedidos de blocos do vídeo selecionado. Cada elemento desta lista indica o número de blocos a ser lido em um ciclo (como em [39]).

A diferença entre os algoritmos apresentados é no teste de admissão e no modo de caracterização utilizado. O algoritmo *Simple Maximum* utiliza a quantidade máxima de blocos a ser lida em qualquer ciclo como parâmetro para caracterização do tráfego do novo cliente. Neste algoritmo, um novo cliente é admitido se a soma de seu valor máximo com o valor agregado dos demais usuários não ultrapassar *minRead*. Já para o algoritmo *Instantaneous Maximum* é mantida uma lista dos pedidos agregados (de todos os clientes admitidos) para cada ciclo. Neste algoritmo, quando um novo cliente faz a solicitação de abertura de um vídeo, é adicionada a sua lista de pedidos na lista agregada. Este novo cliente é admitido se não houver *overflow* (soma dos blocos a serem lidos no ciclo não ultrapassar *minRead*) em nenhum dos ciclos.

Um dos problemas dos algoritmos *Simple Maximum* e *Instantaneous Maximum* é que eles não levam em conta a possibilidade da leitura antecipada dos dados nos ciclos cuja quantidade de blocos é menor que *minRead*, o que pode reduzir as taxas de pico dos fluxos. Desta forma, o terceiro algoritmo denominado de *Average* utiliza a média da quantidade de blocos por ciclo para alocação dos recursos. Neste algoritmo o novo cliente é admitido se a soma da sua média de blocos com a média agregada for menor que *minRead*. Para o algoritmo denominado de *vbrSim*, é feita uma simulação do atendimento utilizando a lista de pedidos agregados (assim como no algoritmo *Instantaneous Maximum*). No entanto, este algoritmo assume que serão lidos *minRead* blocos em cada ciclo de acordo com a quantidade de *buffers* disponíveis. Desta forma se o valor escalonado para um determinado ciclo é menor que *minRead*, a banda restante é utilizada para a leitura antecipada dos dados de ciclos seguintes. Para a admissão de um novo usuário, o algoritmo utiliza todas as informações das leituras já efetuadas para verificar se o novo cliente pode ser admitido sem causar *overflow* em algum ciclo.

Outros trabalhos estudam algoritmos de controle de admissão para servidores de mídia contínua que exploram mecanismos de *caching* para redução da carga nos discos, como em [27].

Como pode ser observado, cada controle de admissão reserva os recursos para cada cliente, de acordo com a caracterização do tráfego no momento da admissão. Para que um cliente não exceda a sua parcela alocada, interferindo nos demais clientes, é necessário um mecanismo de modo a regular o seu tráfego. O mecanismo mais utilizado para policiamento de tráfego é o *Leaky Bucket*, descrito na seção 5.2.

A proposta e os detalhes do controle de admissão estatístico baseado em medidas e simulação desenvolvido nesta tese são apresentados no capítulo 5.

3.6 Técnicas para compartilhamento de recursos

Devido à alta taxa de transferência dos discos atuais, atualmente o gargalo na entrega dos dados para as aplicações de mídia contínua é principalmente na transmissão do conteúdo pela rede até os clientes, e não no servidor multimídia.

Nos servidores multimídia convencionais, o envio dos dados é feito separadamente para cada cliente, o que faz com que os recursos do sistema, particularmente a banda de transmissão pela rede, sejam esgotados rapidamente.

Um técnica para contornar este problema é permitir o compartilhamento do fluxo de dados. Isto é possível através de mecanismos, denominados de *resource sharing techniques*, que permitem aos clientes o compartilhamento dos dados e dos *buffers*. O objetivo é reduzir o tráfego na rede, requisitos de largura de banda de transferência dos discos e de memória [14]. Sendo assim, vários trabalhos começaram a abordar o desenvolvimento de algoritmos para compartilhamento da largura de banda, quando um grande número de clientes está acessando o servidor multimídia.

Os mecanismos propostos na literatura podem ser divididos em duas categorias: *client request oriented* e *periodic broadcast*. A primeira é baseada na transmissão de um fluxo de dados, pelo servidor, em resposta aos pedidos efetuados pelos clientes, como por exemplo, *batching*, *stream tapping*, *patching*, *controlled multicast*, *Piggybacking* e *Bandwidth skimming*. Já a segunda categoria é baseada na transmissão periódica de um fluxo de dados, como por exemplo, *Pyramid Broadcasting*, *Permutation-based Pyramid Broadcasting* e *Skyscraper broadcast* [14].

Além destas técnicas de compartilhamento, é possível a utilização de *proxies* para reduzir o tráfego da rede (melhorando o desempenho para todas as aplicações). Já que não existe suporte para *multicast* em toda a rede, existem propostas para o uso de esquemas que utilizam transmissão *unicast* entre o servidor e os *proxies*, e o uso de técnicas de compartilhamento de banda através de *multicast* entre os *proxies* e os clientes. Com o uso de *proxies* também é possível reduzir o tempo de resposta dos pedidos de um cliente, caso os dados solicitados já estejam armazenados no *proxy*.

Capítulo 4

RIO Multimedia Storage Server: Versão Original

Nesta tese é feita a proposta e a implementação de um conjunto de mecanismos para análise de desempenho do servidor multimídia em tempo real de forma a garantir a QoS para todos os clientes do sistema. O trabalho é realizado utilizando-se do servidor RIO [46]. Para que seja possível entender o restante desta tese é importante que alguns detalhes sobre o servidor RIO sejam explicados.

Os principais motivos para a utilização do servidor RIO foram a flexibilidade da alocação aleatória para armazenamento e acesso aos objetos, a exploração de uma nova e promissora arquitetura e os projetos em conjunto com UCLA e UFMG (Universidade Federal de Minas Gerais), além do servidor suportar o armazenamento de diversos tipos de objetos. Esta última facilidade é essencial para a aplicação alvo deste trabalho: ensino à distância, em particular os projetos CEDERJ [8] e MAPPED [34].

O objetivo deste capítulo é fornecer uma visão geral sobre o funcionamento do servidor RIO (*Randomized I/O Multimedia Storage Server*) na sua versão original e suas principais características. Para isto é apresentada a sua arquitetura e cada um de seus componentes.

Vários artigos foram publicados sobre o servidor RIO, entre eles [36, 37, 43, 15, 44]. Em [36] é feita a descrição do projeto e implementação do servidor RIO. Em [37] são discutidas questões de tolerância a falhas. O trabalho descrito em [43] estuda o desempenho do servidor com configurações contendo discos heterogêneos, [15] descreve a arquitetura escalável do servidor e em [44] é comparado o desempenho do RIO com as técnicas tradicionais de *striping*.

4.1 Visão geral

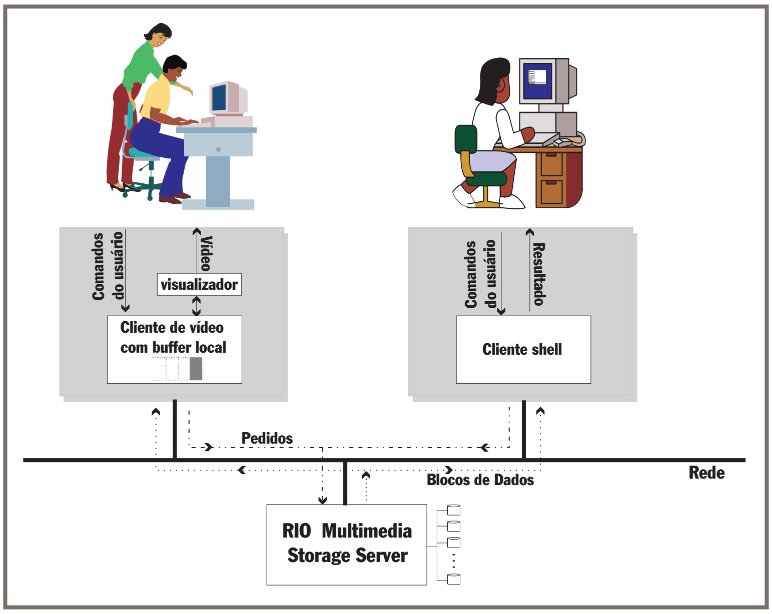

Figura 4.1: Visão geral do servidor RIO com dois clientes

O servidor RIO é um sistema de armazenamento paralelo multimídia universal, baseado em alocação aleatória e replicação de blocos. Por ser multimídia universal, ele suporta diversos tipos de objetos, como por exemplo: vídeo, áudio, texto e imagens, e é capaz de gerenciar múltiplas aplicações concorrentes, com ou sem restrições de tempo [46]. Aplicações tais como visualização de texto e de imagens, são exemplos onde não há restrição de tempo. Aplicações de tempo real podem ser exemplificadas como vídeo sob demanda e aplicações 3D interativas.

No servidor RIO, todos os tipos de mídia são chamados de objetos e são tratados da mesma forma. Cada objeto é divido em blocos de mesmo tamanho e estes são armazenados utilizando alocação aleatória, descrita na seção 3.3. O tamanho de um bloco de dados é um parâmetro do sistema selecionado durante a configuração do servidor.

O RIO foi projetado e desenvolvido no laboratório de multimídia da UCLA [15], sendo que o primeiro protótipo foi implementado para a máquina SUN E4000 e utilizado para visualização de vídeos MPEG e aplicações de simulação 3D. Em seguida foi disponibilizada uma versão para um agrupamento (*cluster*) de PCs. A versão utilizada neste trabalho é a 0.6.15, implementada em C++ para o sistema operacional LINUX. Nesta versão os clientes disponíveis são o **riomtv**, para visualização de vídeo, e o **riosh**, que é um interpretador de comandos para executar determinadas ações no servidor, como por exemplo, armazenamento, remoção e listagem de objetos no servidor.

Uma visão geral do servidor RIO é apresentada na Figura 4.1, que mostra dois clientes conectados no servidor através de uma rede local, por onde são transmitidos os pedidos e os blocos de dados entre o servidor e os clientes.

De uma maneira geral, a arquitetura do servidor RIO (ilustrada na Figura 4.2) é composta pelo nó servidor, pelos nós de armazenamento e pelos clientes. Todas as informações de controle (comandos e mensagens) entre os componentes são transmitidas utilizando o protocolo TCP, enquanto que a transmissão de dados (objetos solicitados) é feita utilizando o protocolo UDP.

Nó servidor

O nó servidor possui uma estrutura composta de diretórios, informações dos usuários, arquivos de configuração, arquivos dos objetos armazenados e arquivos dos discos utilizados.

Cada arquivo que representa um objeto contém informações de seus atributos e o mapeamento dos seus blocos lógicos nos blocos físicos localizados nos discos do servidor. E cada arquivo que representa um disco do sistema contém o mapa de blocos livres/alocados/reservados do respectivo disco. Essas informações sobre os objetos armazenados e os discos são denominadas **metadados**.

O nó servidor (*server node*) é a máquina responsável pelo gerenciamento do sistema e atendimento dos clientes. Ele controla os pedidos efetuados pelos clientes, faz o mapeamento dos blocos requisitados utilizando os metadados fornecidos pelo sistema de arquivos, e envia cada requisição para o nó de armazenamento responsável pelo processamento (nos casos de leitura/escrita de blocos).

Nós de armazenamento

Os nós de armazenamento (*storage nodes*) são as máquinas que possuem discos utilizados pelo servidor. Cada nó é gerenciado por um *StorageServer* que recebe pedidos para leitura ou escrita de um bloco repassados pelo nó servidor, faz autenticação do pedido, e se for válido, executa-o.

Clientes

Os clientes acessam o servidor RIO, através de uma interface (biblioteca) que estabelece o que pode ser solicitado ao servidor, a ordem dos pedidos e o formato das requisições e das respostas do servidor.

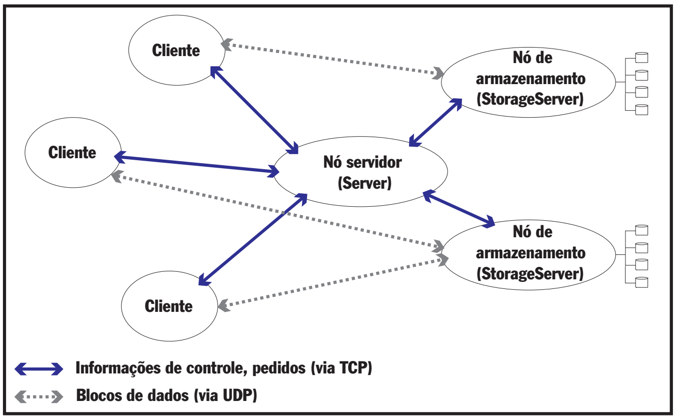

Figura 4.2: Arquitetura básica do RIO

Durante a inicialização do nó servidor, é criada uma conexão TCP com cada nó de armazenamento, conforme ilustra a Figura 4.2. Estas conexões são utilizadas pelo servidor para repassar os comandos dos clientes aos nós de armazenamento, como requisições de envio de blocos ou pedidos de armazenamento de blocos.

Cada cliente ao acessar o servidor RIO, estabelece uma conexão TCP com o nó servidor. Através da conexão criada entre o nó servidor e o cliente, é aberta uma sessão que possibilita a comunicação entre estes dois componentes. Por meio desta sessão, vários comandos podem ser executados, como por exemplo listar, apagar, renomear, criar objetos ou diretórios no servidor, e abrir um fluxo (*stream*), necessário para a transmissão de um objeto. Um fluxo pode ser, de tempo real ou não, de escrita ou de leitura. No momento da finalização da sessão, todos os recursos e objetos que estão sendo utilizados por esta são liberados e fechados.

A escolha do tipo de tráfego utilizado na abertura de um fluxo depende do tipo de tarefa. Por exemplo, se o cliente for copiar um objeto para o servidor, o fluxo a ser aberto não precisa ser de tempo real, pois para a cópia não existem restrições de tempo. Porém, se o cliente for visualizar um vídeo, o fluxo deverá ser de tempo real, pois existem restrições temporais para o envio dos blocos que o servidor deve garantir de forma que a QoS necessária possa ser satisfeita.

Todos os objetos transmitidos pelo fluxo aberto estão sujeitos ao policiamento e às garantias de qualidade de serviço oferecidas pelo servidor.

4.2 Componentes da arquitetura do servidor RIO

Nesta seção cada um dos componentes citados na Figura 4.2 é expandido e detalhado. Os principais componentes da arquitetura do servidor podem ser visualizados na Figura 4.3.

O protótipo do RIO é implementado utilizando múltiplas *threads*. O mecanismo de comunicação é baseado em estruturas de memória compartilhadas, através de filas (de acordo com a política FIFO), utilizando semáforos para a sincronização.

4.2.1 Nó servidor

Conforme descrito na seção 4.1, na arquitetura do servidor RIO existe um único nó responsável pelo gerenciamento do servidor RIO, denominado nó servidor ou simplesmente *Server*. Os principais componentes do *Server* são: *SessionManager*, *ObjectManager*, *StreamManager* e *Router*.

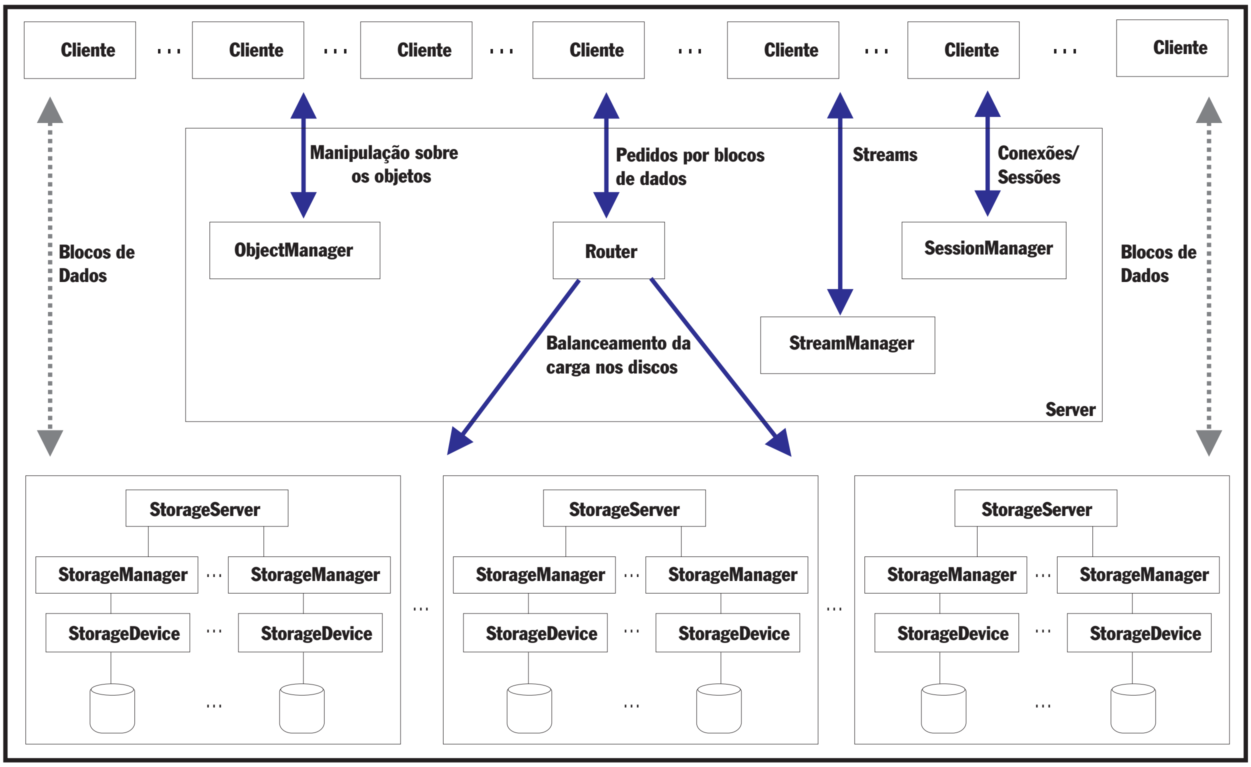

Figura 4.3: Componentes da Arquitetura do servidor RIO

Gerenciador de Sessão (*SessionManager*)

O objetivo do *SessionManager* é fornecer um ponto de conexão para os clientes acessarem o RIO. A cada nova conexão, ele dispara uma *thread* para atender o novo cliente. Esta *thread* cria uma nova sessão e faz o atendimento de todos os pedidos efetuados por este cliente. Cada pedido pode executar um método do próprio *SessionManager*, do *ObjectManager*, do *StreamManager* ou do *Router*, conforme mostra a Figura 4.3.

Gerenciador de Objetos (*ObjectManager*)

O *ObjectManager* gerencia os metadados de todos os objetos armazenados no servidor. É através dele que operações sobre os objetos, tais como criação, abertura, fechamento e exclusão, são efetuadas.

Quando algum objeto é criado, o *ObjectManager* aloca cada bloco e sua(s) réplica(s). Quando um objeto é excluído, é ele quem desaloca os respectivos blocos. Em ambos os casos, são atualizados os metados dos objetos e dos discos.

Gerenciador de Fluxos (*StreamManager*)

O *StreamManager* é responsável pelo gerenciamento dos fluxos já abertos e pela abertura de novos fluxos. Durante a abertura dos novos fluxos, o *StreamManager* executa o controle de admissão, descrito na seção 4.3.

Os pedidos de cada fluxo ativo no sistema são enviados para o roteador de forma a serem atendidos. Devido à política de atendimento e ao policiamento do tráfego, descritos na seção 4.4, nem todas as requisições podem ser processadas no momento da sua chegada. Cada fluxo possui uma fila onde são armazenados aqueles pedidos que precisam esperar para serem atendidos, como está representado na Figura 4.4.

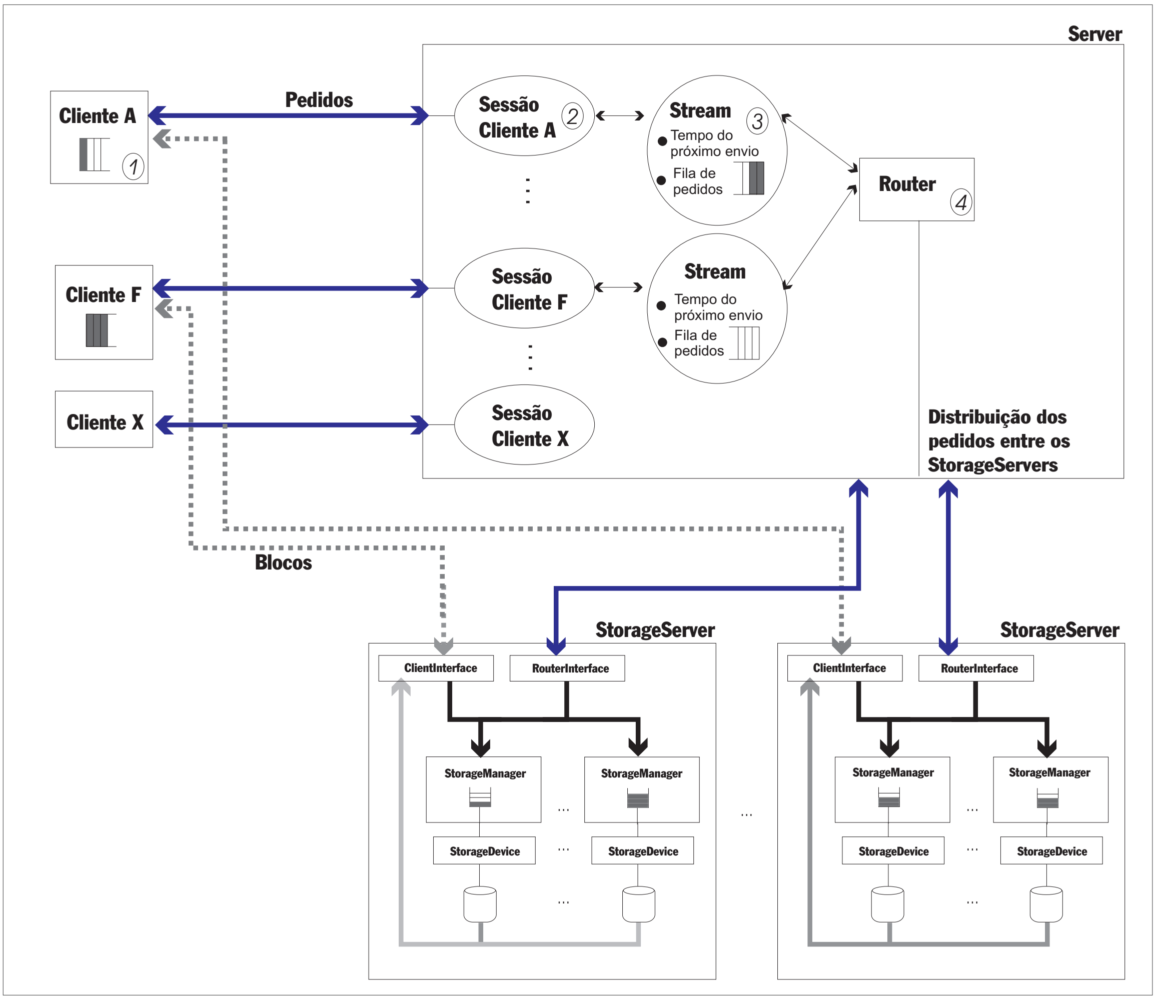

Figura 4.4: Sessões e *Streams* no servidor RIO

Para que os pedidos sejam atendidos assim que possível, o *StreamManager* verifica periodicamente os fluxos com pedidos pendentes, fazendo com que os pedidos sejam enviados ao *Router*, sempre começando pelos fluxos de tempo real.

Roteador (*Router*)

O componente *Router* é responsável por:

- receber os pedidos de leitura/escrita gerados pelos fluxos;
- escolher o(s) dispositivo(s) para atender os pedidos;
- enviar os comandos para os nós de armazenamento responsáveis e
- receber as mensagens de controle enviadas pelos nós de armazenamento.

Para a execução de sua tarefa, o *Router*, ilustrado na Figura 4.5, possui, para cada disco, uma fila para os pedidos de tempo real (Fila RT - *Real Time Queue*) e outra para os pedidos sem restrição de tempo (Fila NRT - *Non-Real Time Queue*). A política de atendimento de cada fila é FIFO e o envio dos pedidos para os discos é feito sempre começando pela fila de tempo real.

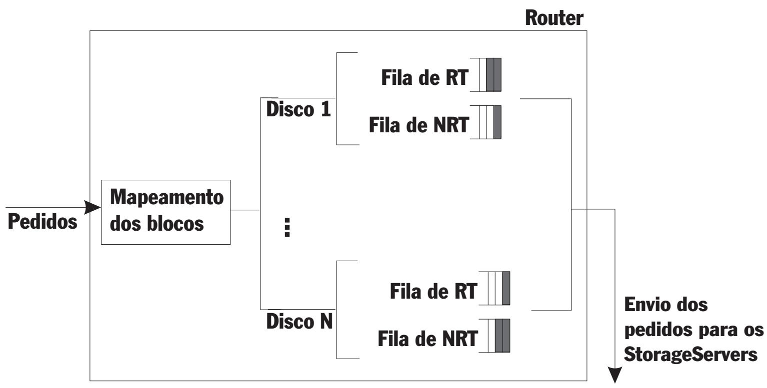

Figura 4.5: Componente *Router*

À medida que o *Router* recebe os pedidos enviados pelos fluxos, é feito o mapeamento do bloco requisitado, através do componente *ObjectManager*. Caso o pedido seja de leitura, a requisição é direcionada para o disco que contém o bloco solicitado e que possui a menor fila. No caso da escrita, é necessária a atualização do bloco e de suas réplicas, desta forma são adicionadas requisições em todas as filas dos discos envolvidos no armazenamento deste bloco.

4.2.2 Nós de armazenamento

Cada *StorageServer* é responsável pelo gerenciamento de um nó de armazenamento. Os componentes do *StorageServer*, ilustrado na Figura 4.6, são: *RouterInterface*, *StorageManager*, *StorageDevice* e *ClientInterface*.

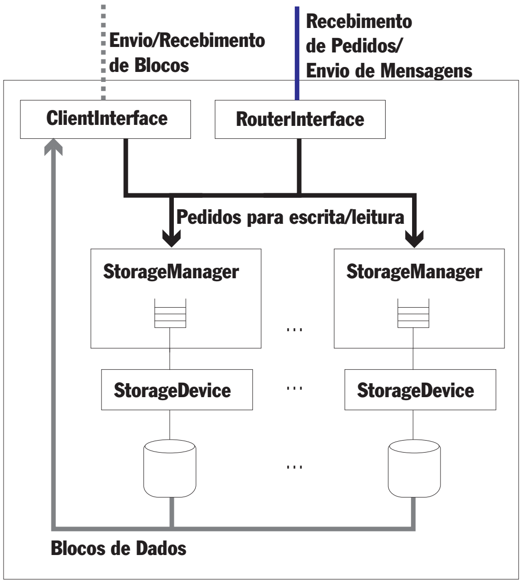

Figura 4.6: Componente *StorageServer*

Interface com o Roteador (*RouterInterface*)

Este componente recebe os pedidos enviados pelo servidor para transferência de blocos de dados localizados no *StorageServer*. Além disso, é responsável pelo envio das mensagens de controle ao nó servidor. Como descrito na seção 4.1, para as informações de controle, tanto o recebimento dos pedidos, quanto o envio das mensagens é feito utilizando uma conexão TCP.

No caso de leitura, o pedido é repassado imediatamente para o *StorageManager* responsável pelo atendimento. Caso contrário, o pedido é enviado ao *ClientInterface*.

Dispositivo de Armazenamento (*StorageDevice*)

Para cada disco do sistema existe apenas uma instância do *StorageDevice* (responsável pela realização das operações de E/S). Tais operações são realizadas utilizando as chamadas do sistema operacional.

Gerenciador de Armazenamento (*StorageManager*)

Cada *StorageManager* gerencia um *StorageDevice* e faz o escalonamento dos pedidos por blocos armazenados neste dispositivo. Para isto, cada *StorageManager* possui uma fila dos pedidos a serem atendidos, conforme mostra a Figura 4.6. Para cada pedido é associado um *buffer* para o armazenamento do bloco de dados solicitado.

Os discos no servidor RIO não são sincronizados, ou seja, os pedidos de cada disco são atendidos independentemente dos demais discos do sistema. Nesta versão, os pedidos na fila de cada *StorageManager* são atendidos utilizando o algoritmo de escalonamento FIFO. O uso do algoritmo FIFO foi escolhido pois esta política minimiza a variância do tempo de serviço do pedido, embora a média do tempo de serviço possa ser maior se comparado com outras políticas de escalonamento (como por exemplo, SCAN), conforme descrito na seção 3.4 [44, 37].

No caso do pedido ser de leitura, após a realização da operação, o *StorageManager* solicita ao *ClientInterface* o envio do bloco para o cliente. Para os pedidos de escrita, o componente *ClientInterface*, após o recebimento do bloco enviado pelo cliente, solicita ao *StorageManager* o armazenamento do respectivo bloco.

Interface com o cliente (*ClientInterface*)

O componente *ClientInterface* faz o envio/recebimento dos blocos de dados a serem transmitidos/armazenados. Para a transmissão dos blocos de dados via UDP, foi criado um protocolo específico. Cada bloco a ser enviado é dividido em vários pacotes. O transmissor controla o envio do bloco através de algumas variáveis, como por exemplo, a quantidade de pacotes que compõe o bloco, o endereço do receptor (IP, porta e identificação da requisição) e a quantidade de pacotes já recebida pelo receptor.

4.2.3 Clientes

Os únicos clientes disponíveis nesta versão são o riosh e o riomtv.

Cliente riosh

O **riosh** é o interpretador de comandos do servidor RIO. Cada usuário possui um nome de usuário e uma senha que serve para sua identificação. Através do riosh, o usuário pode acessar sua conta criada no servidor e executar várias ações, tais como listar objetos, criar diretórios e copiar objetos.

Cliente riomtv

O **riomtv** é utilizado para visualização de vídeo através do aplicativo *mtvp* [35]. Este cliente não possui uma interface para executar comandos, tais como avançar, voltar ou pausar. Sendo assim, após o início da exibição do vídeo, este é mostrado até o seu término.

Para compensar o *jitter*, este cliente possui um *playout buffer* que é preenchido antes do começo da visualização e utilizado para amazenar os blocos antes do momento de sua visualização.

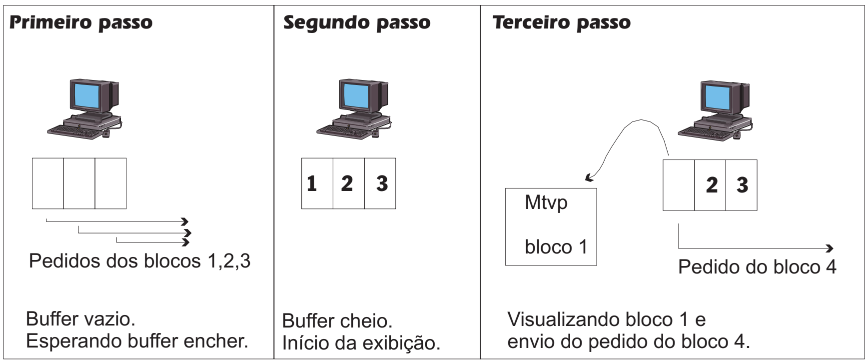

Figura 4.7: Funcionamento do Cliente riomtv

Na Figura 4.7, pode-se observar que o cliente solicita tantos blocos de acordo com o tamanho do seu *buffer* (blocos 1, 2 e 3 - primeiro passo). Após o recebimento de todos os blocos (segundo passo), o *buffer* encontra-se cheio e tem-se o início da visualização do objeto (terceiro passo). Conforme pode ser observado, o bloco 1 é repassado para o *mtvp* para sua exibição e o bloco 4 é solicitado ao servidor. Resumindo, o cliente envia bloco por bloco, que estão previamente armazenados no seu *buffer*, ao *mtvp* no momento em que se faz necessária a sua exibição. A cada posição do *buffer* liberada, é solicitado um novo bloco ao servidor.

4.3 Controle de Admissão de Usuários

O controle de admissão utilizado pelo servidor RIO é executado pelo *StreamManager* no momento da abertura de novos fluxos de tempo real.

Na configuração do sistema, é indicada a largura de banda total disponível (bytes/segundo) no servidor ($Taxa\_Total$) e a largura de banda reservada para fluxos que **não** são de tempo real ($Taxa\_Reservada\_NRT$). A reserva de banda garante que os discos terão tempo para atender os pedidos sem restrições temporais. Estes pedidos serão atendidos quando não existir mais nenhum pedido de tempo real na fila de espera do disco.

Durante a inicialização do servidor, a variável $Taxa\_Alocada$ é inicializada com o valor 0. Na abertura de um novo fluxo, de acordo com os parâmetros informados pelo cliente, o controle de admissão verifica se é possível o atendimento deste novo cliente. Os parâmetros informados por cada cliente são o tipo de tráfego (de tempo real ou não), a direção do tráfego (escrita ou leitura) e a largura de banda solicitada ($Taxa\_Solicitada$). Se o tipo de tráfego solicitado for de tempo real, o novo usuário é admitido se o resultado da equação abaixo for verdadeiro.

$$(Taxa\_Alocada + Taxa\_Solicitada) <= (Taxa\_Total - Taxa\_Reservada\_NRT)$$

Caso o novo cliente seja aceito, a variável $Taxa\_Alocada$ é atualizada da seguinte forma $Taxa\_Alocada = Taxa\_Alocada + Taxa\_Solicitada$. Através da taxa solicitada e do tamanho do bloco de dados utilizado pelo RIO, o *StreamManager* calcula o intervalo entre os pedidos deste novo fluxo, que será utilizado para o policiamento do tráfego, descrito na seção 4.4.

4.4 Atendimento dos Pedidos e Policiamento do Tráfego

O atendimento dos pedidos no servidor RIO prioriza àqueles originados pelos fluxos de tempo real em relação aos pedidos sem restrições temporais, conforme observado na seção 4.2.1.

Cada componente fluxo (item 3 da Figura 4.4) faz o policiamento e o envio de seus pedidos para o componente *Router*. Cada pedido pode ser inserido na fila de espera ou enviado imediatamente para o *Router*. Para cada tipo de tráfego, o procedimento para policiamento é diferente.

Para os fluxos que não são de tempo real, o pedido é enviado para o *Router* se o número de pedidos ativos não concluídos (já enviados para o componente *Router*) for menor que um número máximo. Este número é calculado pelo *StreamManager*, durante a inicialização do sistema, de acordo com o número máximo de pedidos que cada fila dos discos pode conter.

Para os fluxos que são de tempo real, o envio de cada pedido depende do cálculo do intervalo mínimo entre pedidos, executado pelo *StreamManager*, no momento de sua admissão. O procedimento implementado pelo RIO é baseado no envio de no máximo um pedido para o *Router* a cada intervalo entre pedidos.

Considere, por exemplo, um cliente que solicitou uma largura de banda de 1.5Mbits/segundo para a exibição de um vídeo. Dado que o $Tamanho\_Bloco$ é igual a $128KB$, o intervalo entre os pedidos deste cliente será:

$$
\begin{aligned}
Intervalo\_entre\_pedidos &= \frac{Tamanho\_Bloco}{Taxa\_Solicitada} \\
&= \frac{128KB}{192KB/seg} = 0.6667seg
\end{aligned}
$$

Ou seja, cada fluxo possui o tempo mínimo para o envio do próximo pedido que é igual ao intervalo acima. Caso o próximo pedido ocorra antes do intervalo pré-calculado, este pedido fica na fila de espera até que a condição seja satisfeita. A cada pedido enviado para o *Router*, o tempo do próximo envio é atualizado sendo igual ao tempo atual mais $Intervalo\_entre\_pedidos$.

No caso do cliente pausar a exibição de um vídeo, o número de pedidos que teriam sido enviados durante o tempo que o cliente ficou parado não é acumulado. Então, quando o cliente retorna a exibição do vídeo, ele continua recebendo apenas a taxa requisitada na abertura do fluxo.

Capítulo 5

Proposta de Gerenciamento de *Buffers* e Controle de Admissão de Usuários

Neste capítulo é apresentada a proposta e a implementação de um conjunto de mecanismos que visam à garantia de QoS para todos os clientes admitidos por um servidor multimídia e ao aumento no número de clientes admitidos. Isto inclui um controle de admissão estatístico, juntamente com o gerenciamento de *buffers* no servidor e no cliente, para compensar as flutuações da carga dos discos e da rede, respectivamente.

O objetivo principal do trabalho é manter uma infra-estrutura, com um controle detalhado do sistema, para estudo e coleta de medidas do servidor multimídia, mesmo que o desempenho seja reduzido se comparado ao uso de técnicas mais simples. Através do controle mais fino, o servidor pode ser ajustado de acordo com os níveis de qualidade de serviço desejados.

Nas seções a seguir são explicados os mecanismos propostos e a sua implementação dentro do servidor RIO, descrito no capítulo 4.

5.1 Novos mecanismos

A idéia principal desta proposta é utilizar as informações *a priori* dos vídeos armazenados para obter o escalonamento dos pedidos de cada cliente de acordo com o vídeo selecionado. Embora as aplicações de vídeo/áudio sob demanda possuam características de tempo real, o acesso típico é seqüencial. Desta forma é possível prever o seu comportamento, como por exemplo, a ordem dos pedidos e o momento de suas requisições.

Na literatura já existem algoritmos que exploram esta idéia, conforme observado na seção 3.5. As principais diferenças da proposta deste trabalho são:

- O controle de admissão não considera o atendimento dos pedidos baseado em ciclos para os cálculos de admissão, como na maioria dos trabalhos, pois o servidor RIO não faz o atendimento em ciclos.
- Existe um *buffer* no servidor para cada cliente, onde são armazenados blocos, não apenas para o envio imediato ao cliente (blocos já requisitados), mas também para o armazenamento dos blocos cuja leitura é feita antes de sua solicitação por parte do cliente (mecanismo de *prefetching*), utilizando a técnica *Fixed Lookahead*. Com o uso desta técnica, o servidor lê uma quantidade fixa de blocos antes das requisições, para cada cliente, durante toda a exibição do vídeo. Diferentemente de alguns trabalhos que fazem o *prefetching* dos blocos, de acordo com a quantidade de *buffers* disponíveis, de forma a evitar os ciclos em que o tempo de atendimento dos pedidos excede a duração pré-estabelecida, ou seja, alguns blocos são lidos nos ciclos anteriores quando a carga é menor do que a máxima permitida.
- Além das informações *a priori* dos vídeos requisitados, o controle de admissão é baseado em medidas de desempenho do servidor calculadas em tempo real, como por exemplo, tempo de serviço dos discos e tempo de espera nas filas. Estas medidas são utilizadas na simulação realizada durante o processo de admissão para prever o comportamento do sistema.

- Tanto as medidas de desempenho do servidor, quanto as medidas da rede, são consideradas no algoritmo de controle de admissão.

5.1.1 Idéia Básica

Como já abordado anteriormente, as aplicações de mídia contínua são compostas por uma seqüência de dados (geralmente por quadros de vídeo ou amostras de áudio) que devem ser apresentados em um determinado intervalo de tempo. Por exemplo, um vídeo deve ser apresentado a uma certa taxa para que a noção de movimento seja mantida (como por exemplo, 30 quadros por segundo). Conforme descrito na seção 3.2, devido às técnicas de compressão, o tamanho de cada quadro varia para permitir que a qualidade do vídeo seja constante durante toda a apresentação.

O método mais utilizado para o armazenamento de objetos em servidores multimídia é a divisão de tais objetos em vários blocos de dados. Durante a execução do cliente, os blocos que compõem o objeto requisitado são lidos pelo servidor e enviados para o cliente.

Supondo que o padrão de acesso típico das aplicações de vídeo/áudio sob demanda é seqüencial, é possível utilizar as informações *a priori* dos vídeos armazenados para prever o comportamento de cada cliente durante a sua execução. Através de ferramentas, pode-se extrair informações de um vídeo, como por exemplo a taxa de quadros utilizada e o tamanho de cada quadro. Dado o tamanho do bloco de dados utilizado, é possível saber a quantidade de quadros armazenados em um bloco e, conseqüentemente, o tempo de consumo (duração) deste bloco de dados. Desta forma, de acordo com o tamanho do *buffer* de cada cliente (número de blocos), é possível fazer a previsão do tempo de requisição de cada bloco do objeto selecionado desde que o cliente acesse a mídia de forma seqüencial e ignorando os retardos aleatórios na rede. Comentários sobre padrões de acesso não seqüenciais são apresentados na seção 5.1.4.

A Figura 5.1 ilustra a lista de previsão dos pedidos de dois clientes. É importante lembrar que um cliente só começa a exibição do conteúdo depois que o seu *playout*

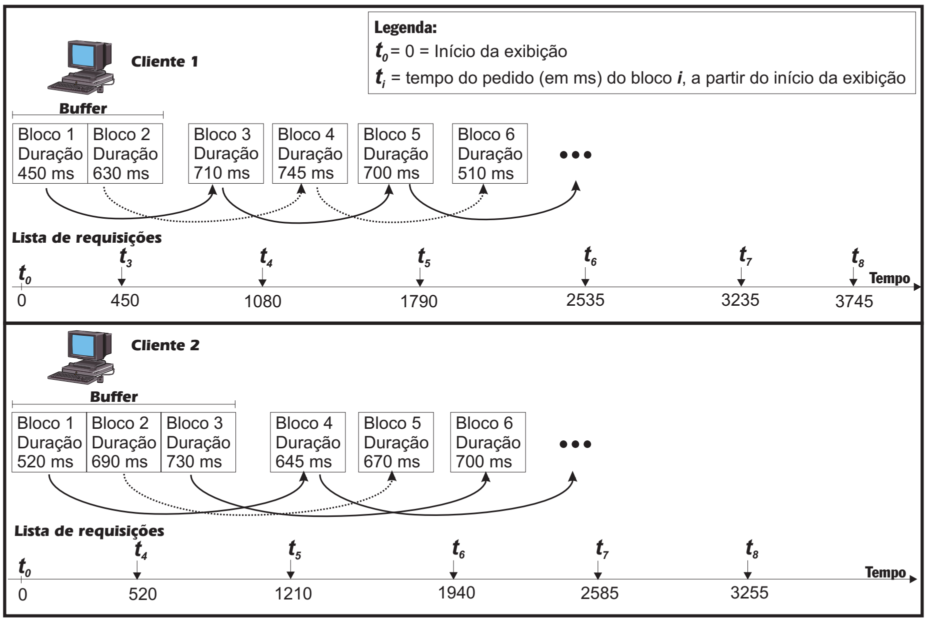

Figura 5.1: Exemplo com a previsão da lista de pedidos para dois clientes

buffer estiver cheio. Como pode ser observado, o cliente 1 possui um *playout buffer* com duas posições, sendo os tempos de consumo (d_{bloco_i}, duração do bloco i) para o vídeo 1: 450ms e 630ms, dos blocos 1 e 2, respectivamente. O tempo de requisição do bloco 3 para o cliente 1 será em $t_3 = 450ms$ (logo após d_{bloco_1} quando a posição do *buffer* esvazia); o tempo de requisição do bloco 4 será em $t_4 = 1080ms$ ($d_{bloco_1} + d_{bloco_2}$), e assim sucessivamente. Para o cliente 2, que possui um *buffer* com três posições, os tempos de consumo do vídeo 2 são 520ms, 690 ms e 730ms, dos blocos 1, 2 e 3, respectivamente. O tempo de requisição do bloco 4 será $t_4 = 520ms$, o tempo de requisição do bloco 5 será $t_5 = 1210ms$, o tempo de requisição do bloco 6 será $t_6 = 1940ms$, e assim sucessivamente.

O cálculo dos tempos das requisições até este momento levam em consideração apenas os tempos dos pedidos efetuados do lado do cliente. No entanto, vários outros tempos precisam ser considerados, como por exemplo o tempo do envio do pedido até o servidor, que depende dos retardos aleatórios na rede (explicado na seção 2.2), o tempo de espera na fila do disco, o tempo de serviço para leitura do bloco de dados e o tempo de transmissão deste bloco de dados. Na Figura 5.2 são ilustrados os tempos envolvidos no atendimento de um pedido, desde o momento da requisição de um bloco até o recebimento deste. Como observado na seção 3.5, quanto maior o *buffer* do cliente, maior poderá ser a variabilidade destes tempos. Entretanto, quanto maior o *buffer*, maior será a quantidade de memória necessária na máquina do cliente e maior será a latência inicial.

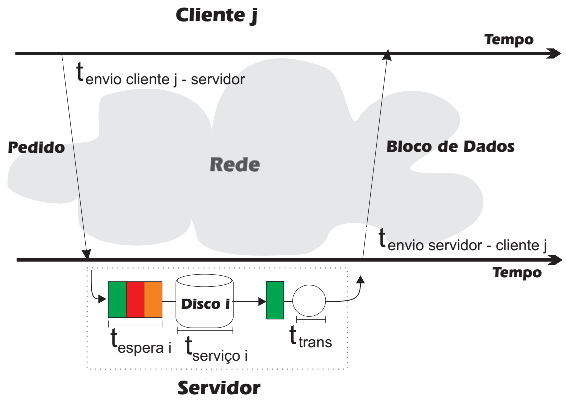

Figura 5.2: Tempos envolvidos no atendimento de um pedido

Para compensar as flutuações na carga dos discos, neste trabalho é proposto um gerenciamento de *buffers* no servidor que faz o *prefetching* dos dados de cada cliente de acordo com o seu comportamento. Para isto, a cada cliente é associado um *buffer* no servidor, onde são armazenados os próximos blocos a serem transmitidos. A cada pedido efetuado por um cliente, o bloco já armazenado no *buffer* é enviado e em seguida é feita a requisição para a leitura do próximo bloco a ser armazenado neste *buffer*.

O exemplo ilustrado na Figura 5.3 apresenta a lista com a previsão dos pedidos a serem efetuados pelo cliente 1 da Figura 5.1 e a lista com a previsão dos pedidos, deste mesmo cliente, a ser requisitada para o disco no servidor, utilizando o gerenciamento de *buffers*. Na Figura 5.3 o tempo de envio entre o cliente e o servidor foi considerado igual a zero, porém em um caso real os pedidos enviados pelo cliente sofrem retardos aleatórios na rede e por isso são deslocados. Suponha que cada cliente tem um *buffer* com três posições no servidor e que um cliente só começa a exibição do conteúdo depois que todos os *buffers* (tanto no servidor, quanto no cliente) estiverem cheios. A quantidade total de *buffers* de cada cliente é a soma do tamanho do *playout buffer* no cliente com o tamanho do *buffer* no servidor.

Juntamente com o gerenciamento de *buffers*, o controle de admissão proposto faz

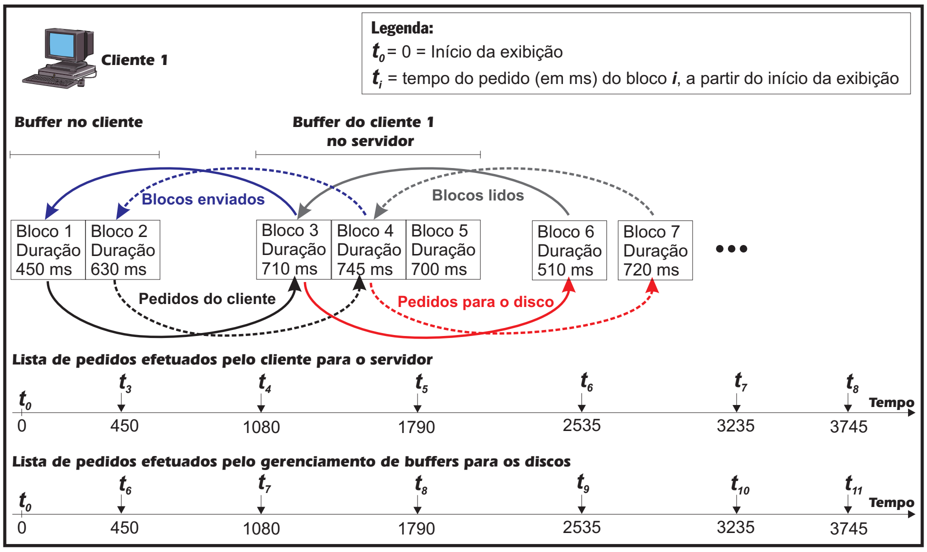

Figura 5.3: O Gerenciamento de *Buffers*

uma simulação no momento de admissão do novo cliente, de acordo com a carga atual (utilizando as listas de previsão de pedidos de cada cliente, entre outras informações) e as medidas de desempenho do servidor obtidas em tempo real, de modo a verificar se a QoS pode ser mantida para todos os clientes (incluindo o novo) durante toda a sua execução. Conforme visto na seção 3.1, um objetivo para o servidor multimídia é manter dados suficientes nos *buffers* de todos os clientes de forma que nenhuma execução seja interrompida ou que pelo menos o número de falhas durante a sua execução seja o menor possível para que não haja deterioração da QoS vista pela aplicação.

Os parâmetros de qualidade de serviço analisados são o número máximo de vezes que a exibição do vídeo pode ser congelada (*hiccup*) devido ao esvaziamento do *playout buffer* do cliente e o intervalo máximo que este *buffer* pode ficar vazio.

A Figura 5.4 ilustra o sistema simulado pelo controle de admissão juntamente com o gerenciamento de *buffers*. Durante a simulação são processados três tipos de eventos: chegada de um pedido (evento tipo 1), serviço de um pedido (leitura de um bloco - evento tipo 2) e chegada de um bloco de dados em um cliente (evento tipo 3). Ao final da simulação, é possível verificar se os requisitos de QoS podem ser garantidos para cada um dos clientes com a admissão do novo candidato. Os detalhes sobre os algoritmos de controle de admissão e gerenciamento de *buffers* são explicados nas seções a seguir.

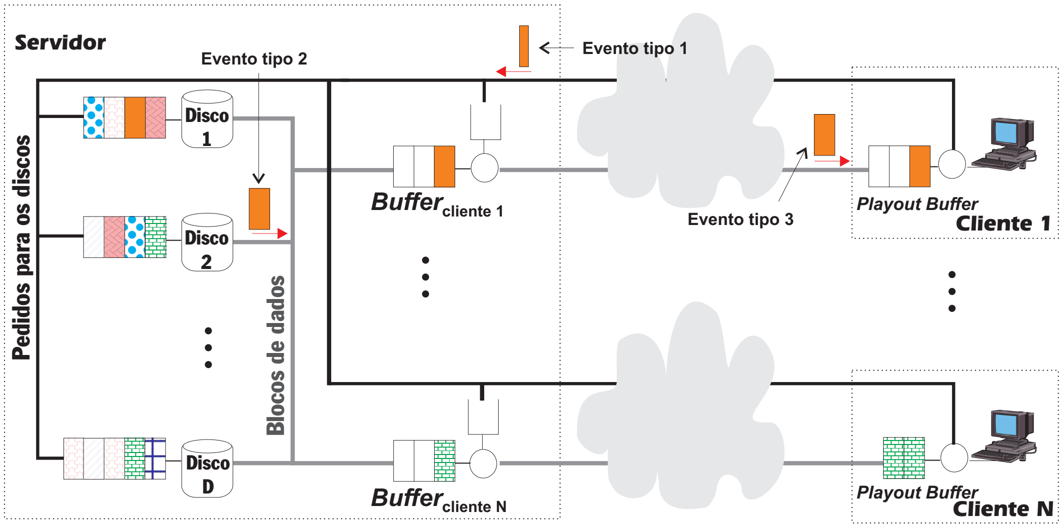

Figura 5.4: Visão do sistema simulado pelo Controle de Admissão

5.1.2 Extração da duração de cada bloco de dados de um objeto

Para extrair os tempos de consumo (duração de cada bloco) de um objeto, foi implementado um programa em C, que lê todos os blocos de dados deste objeto (do mesmo tamanho utilizado pelo servidor RIO) e repassa para o aplicativo responsável pela exibição da mídia (neste trabalho é usado o aplicativo *mtvp*). O mecanismo utilizado para a comunicação entre os processos é o *pipe*, conforme ilustra a Figura 5.5. O algoritmo criado para a extração dos tempos de cada objeto está descrito em português estruturado na Figura 5.6.

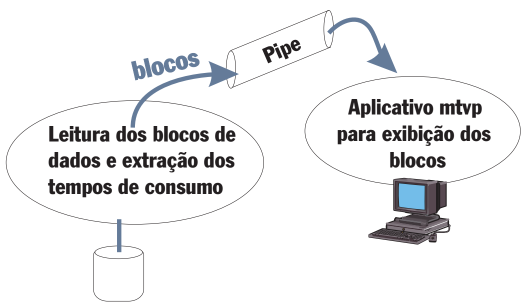

Figura 5.5: Processo de extração dos tempos de consumo de um objeto

```
Função extrai_duração_dos_blocos( Nome_do_Vídeo)
    criação do pipe;
    executa mtvp;
    se abre_arquivo( Nome_do_Vídeo )
      bloco = 0;
      tanterior = tcorrente;
      enquanto não for final do arquivo
        lê um bloco de dados;
        escreve bloco no pipe;
        dbloco = tcorrente - tanterior;
        imprime no arquivo de saída o valor dbloco;
        tanterior  = tcorrente;
        bloco = bloco + 1;
      fim-enquanto
    fecha_arquivo();
    fim-se
fim-função
```

Figura 5.6: Algoritmo para extração da duração dos blocos de dados de um objeto

Com a utilização do programa implementado, é criado um arquivo para cada um dos vídeos armazenados no servidor, contendo a duração de cada um de seus blocos. O número de blocos que compõe um vídeo depende do tamanho do vídeo e do tamanho do bloco de dados utilizado pelo servidor. O tempo de execução do programa é igual a duração total do vídeo selecionado. Para cada objeto, as informações são extraídas apenas uma vez e armazenadas no servidor multimídia.

5.1.3 Geração da lista de requisições para um cliente

A lista de requisições de cada cliente é criada no momento de admissão, de acordo com: (a) o vídeo selecionado, (b) o protocolo de comunicação (que indica a ordem dos passos executados entre o servidor e o cliente) e (c) os tempos envolvidos durante este processo. Cada item da lista de requisições identifica as informações de um bloco de dados do vídeo selecionado, como por exemplo, o tempo de requisição do bloco (que indica o tempo que o pedido deste bloco deve ser repassado ao disco responsável pela sua leitura), entre outros atributos (descritos na Figura 5.7). Esta lista é utilizada pelo controle de admissão descrito na seção 5.1.5 e pelo gerenciamento de *buffers*, de forma a identificar, durante a execução do cliente, quais os próximos blocos a serem transmitidos e quais os próximos blocos a serem lidos dos discos.

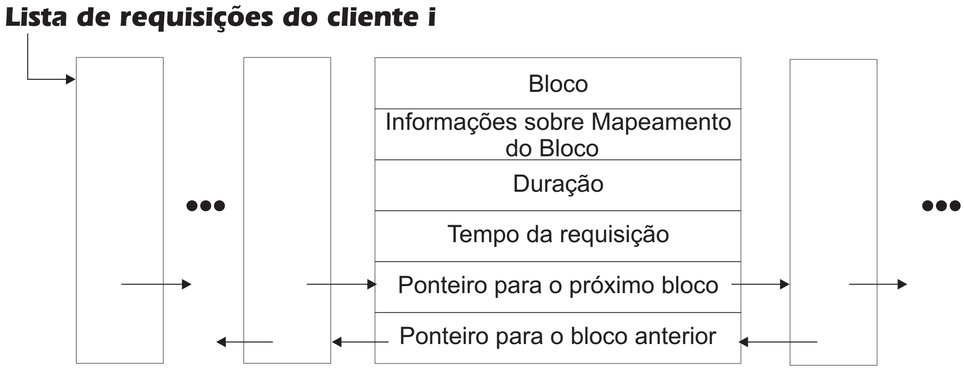

Figura 5.7: Estrutura de dados da lista de requisições

Seja:

- BC_j o tamanho (número de blocos) do *buffer* no cliente j;
- BS o tamanho (número de blocos) do *buffer* de cada cliente no servidor;
- RTT_j o *Round Trip Time* entre o servidor e o cliente j;
- t_{adm} a variável aleatória que representa o tempo para admissão de um novo cliente, incluindo todos os tempos envolvidos na admissão, tais como tempo gasto na geração da lista de requisições e na simulação, sendo $\overline{t_{adm}}$ sua média;
- $t_{serv\_disco_i}$ a variável aleatória que representa o tempo de serviço do disco i, que depende do tempo de *seek*, rotação e da taxa de transferência de dados do disco i, sendo $\overline{t_{serv\_disco_i}}$ a sua média;
- $t_{resp\_disco_i}$ a variável aleatória que representa o tempo de resposta do disco i, incluindo a chamada do sistema operacional para leitura do bloco de dados e o tempo de serviço do disco i, sendo $\overline{t_{resp\_disco_i}}$ a sua média. O tempo de resposta do disco também inclui o tempo de espera na fila de pedidos do dispositivo gerenciada pelo sistema operacional;
- $t_{fila\_disco_i}$ a variável aleatória que representa o tempo na fila de espera do disco i. Como descrito na seção 4.2.1, o componente *Router* possui uma fila para cada disco onde são armazenados os pedidos antes de serem enviados para o nó de armazenamento que contém o disco responsável pelos atendimentos.

$t_{fila\_disco_i}$ depende do tamanho da fila no momento da requisição e de cada disco, sendo $\overline{t_{fila\_disco_i}}$ a sua média;

- $t_{atend\_disco_i}$ a variável aleatória que representa o tempo de atendimento do disco i, incluindo o tempo de espera na fila do disco i no componente *Router* e o tempo de resposta do disco i, sendo $\overline{t_{atend\_disco_i}} = \overline{t_{fila\_disco_i}} + \overline{t_{resp\_disco_i}}$ a sua média;

- t_{trans} o tempo de transmissão na rede de um bloco de dados, que depende da taxa de transmissão (meio físico) do servidor;

- $t_{buffer\_servidor}$ o tempo médio para encher o *buffer* de um cliente no servidor, que depende dos tempos de leitura dos discos (para todos os blocos a serem lidos) e da quantidade de blocos para cada cliente (BS);

- $t_{buffer\_cliente_j}$ o tempo para encher o *buffer* no cliente j, que depende do tamanho do *buffer* deste cliente (BC_j) e dos tempos de leitura dos discos e t_{trans}. O valor de $t_{buffer\_cliente_j}$ é o maior valor entre o tempo até finalizar a leitura de BC_j blocos e o tempo até finalizar a transmissão de BC_j blocos.

Com o conhecimento de todos os tempos citados anteriormente, dos tempos de consumo de cada bloco de dados do objeto solicitado e do protocolo de comunicação é possível a criação da lista de requisições dos blocos para cada cliente admitido (descrita na Figura 5.7).

Considere o protocolo de comunicação (versão modificada neste trabalho) e os passos executados pelo servidor RIO descritos na Figura 5.8. Suponha que cada cliente possua um *buffer* no servidor com três posições e que o tamanho do *buffer* no cliente seja de duas posições. A partir da requisição de um objeto, considerando este momento como o tempo inicial, o tempo do pedido de cada bloco de dados a ser submetido para o disco (t_i o tempo da requisição do bloco i) é calculado da seguinte maneira: após a simulação efetuada pelo controle de admissão (t_{adm}), são feitas as requisições para o preenchimento do *buffer* no servidor. Sendo assim, o tempo de requisição dos blocos 1, 2 e 3 é o mesmo, $t_1 = t_2 = t_3 = t_{adm}$. As próximas requisições são feitas após RTT_j, quando o cliente solicita os blocos para o

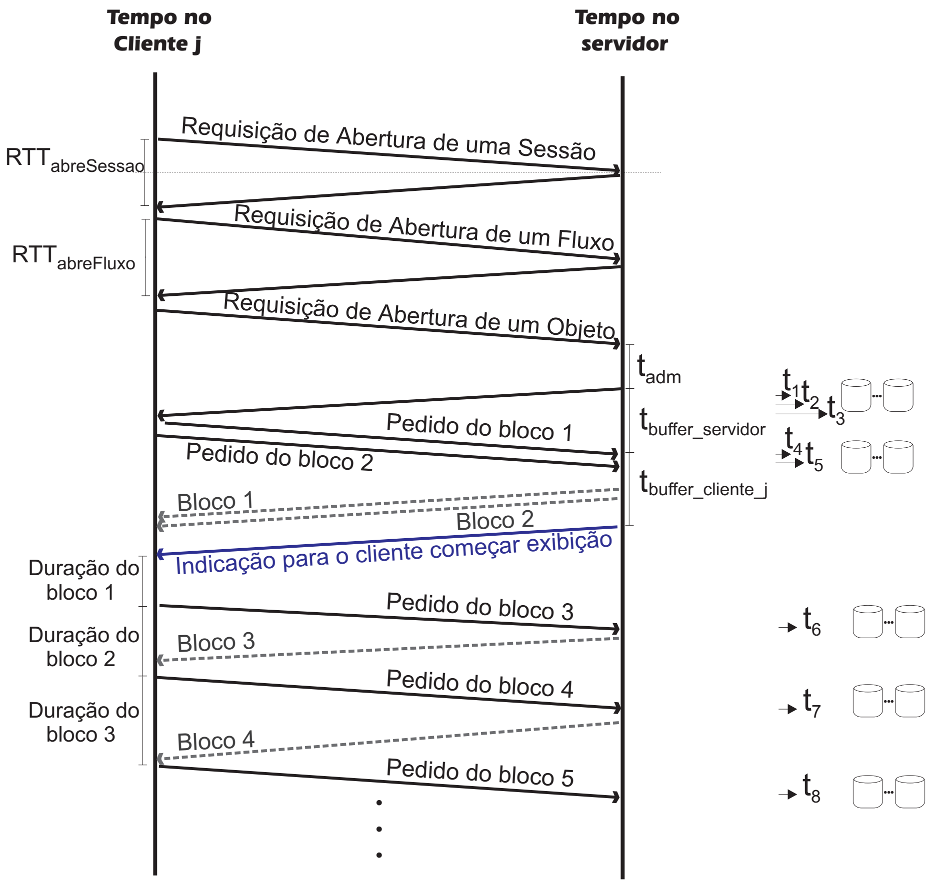

Figura 5.8: Protocolo de Comunicação utilizado pelo servidor RIO

preenchimento do seu *buffer*. A cada pedido efetuado, é enviado o bloco de dados correspondente já armazenado no *buffer* no servidor e é feita a requisição pelo próximo bloco. Desta forma, os tempos de requisição dos blocos 4 e 5 são idênticos, sendo $t_4 = t_5 = t_{adm} + RTT_j$. Após o preenchimento de todos os *buffers*, o usuário começa a exibição do conteúdo. Neste caso, o pedido pelo bloco 6 será em $t_6 = t_{adm} + max(t_{buffer\_servidor}, RTT_j) + t_{buffer\_cliente_i} + RTT_j + d_{bloco_1}$, o pedido do bloco 7 será em $t_7 = t_{adm} + max(t_{buffer\_servidor}, RTT_j) + t_{buffer\_cliente_i} + RTT_j + d_{bloco_1} + d_{bloco_2}$ e assim por diante até o último bloco de dados do objeto. Estes passos estão descritos no algoritmo apresentado na Figura 5.9. Os detalhes referentes à mudança no padrão de acesso, tais como operações de avançar e retroceder, são tratados na seção 5.1.4.

Parâmetros e medidas

Para a implementação dos mecanismos propostos foram necessárias várias alterações e inserções de novas características, tanto no servidor, quanto nos nós de armazenamento e na interface de comunicação entre o servidor e o cliente. Nesta seção são explicados como os parâmetros e os tempos envolvidos, utilizados para a geração da lista de requisições e pela simulação no momento de admissão, são obtidos.

A interface de comunicação foi modificada de forma que, durante os passos de abertura de uma sessão e de abertura de um fluxo, são coletados pelo cliente os tempos desde o envio até o recebimento das respostas do servidor, chamados de $RTT_{abreSessao}$ e $RTT_{abreFluxo}$, respectivamente, conforme ilustra a Figura 5.8. No momento da requisição do objeto selecionado pelo usuário, o cliente informa ao servidor o nome do objeto, o tamanho do seu *buffer* e o RTT (calculado através da média aritmética dos valores coletados).

Do lado do servidor, os parâmetros podem ser divididos em dois grupos. No primeiro grupo encontram-se os parâmetros que são configurados durante a inicialização do sistema, entre eles:

- A quantidade de *buffers* para cada cliente do lado do servidor (parâmetro de configuração *NumberOfBuffersForEachClient*). Atualmente esta quantidade é fixa durante toda a execução de um cliente e igual para todos os clientes.
- O tempo de transmissão de um bloco de dados (t_{trans}) que é obtido através dos parâmetros de configuração *NetworkRate* e *BlockSize* ($t_{trans} = \frac{BlockSize}{NetworkRate}$).
- Os parâmetros de qualidade de serviço, utilizados pelo controle de admissão, *NumberOfRequestsWhenEmptyBuffer* - que indica a proporção de blocos do vídeo que pode ocasionar o congelamento da exibição (número permitido de *hiccups*, como por exemplo, 10% dos blocos) e o intervalo máximo que o *playout buffer* no cliente pode ficar vazio - *MaxIntervalEmpty*.

No segundo grupo encontram-se os parâmetros obtidos através de medidas efetuadas, durante o funcionamento do sistema, pelos componentes do servidor. Para o cálculo dos tempos de espera nas filas dos discos, o componente *Router* mantém uma estrutura de dados em memória, utilizando uma matriz, onde para cada disco é armazenado o tempo médio de espera ($\overline{t_{fila\_disco_i}}$), de acordo com o tamanho da fila no momento da chegada do pedido. Ou seja, a cada inserção de um pedido na fila de um disco selecionado ($disco_i$), são armazenados dois valores: o tamanho atual da fila ($fila_i$) e o tempo corrente ($t_{inicial} = t_{corrente}$). No momento que este pedido é repassado ao *StorageServer* responsável, é calculado: (1) o tempo de espera deste pedido ($amostra$), através da diferença do tempo corrente e do tempo armazenado ($amostra = t_{corrente} - t_{inicial}$), e (2) o tempo médio de espera para atendimento neste disco. Esta média é calculada pela equação 5.1, chamada de média que se move exponenciamente (EWMA - *Exponential Weighted Moving Average*) [26]:

$$tempo\_medio = (1 - \tau) \times tempo\_medio + \tau \times amostra \tag{5.1}$$

Onde τ é um parâmetro que indica a combinação dos pesos do tempo médio calculado anteriormente e do valor da nova amostra. Este parâmetro é configurado durante a inicialização do sistema (por exemplo, $\tau = 0,125$). Desta forma, o tempo médio de espera do disco é atualizado, de acordo com o tamanho da fila armazenado inicialmente ($t_{medio\_fila}[disco_i][fila_i] = (1 - \tau) \times t_{medio\_fila}[disco_i][fila_i] + \tau \times amostra$).

Para o cálculo dos tempos de serviço e de resposta dos discos, cada componente *StorageManager* (responsável pelo gerenciamento dos pedidos de um disco), mantém duas estruturas de dados, utilizando dois vetores, um para armazenar os tempos médios de serviço ($\overline{t_{serv\_disco_i}}$) e o outro para armazenar os tempos médios de resposta ($\overline{t_{resp\_disco_i}}$), de acordo com o número de *threads* ativas durante a requisição do bloco para o disco.

Na versão original do RIO, a política de escalonamento de pedidos utilizada, descrita na seção 4.2.2, era FIFO. Entretanto, para minimizar o tempo de *seek* durante o atendimento de um pedido pelo disco, o componente *StorageManager* foi modificado de forma que são disparadas várias *threads* simultaneamente, cada uma responsável pela leitura/escrita de um bloco de dados. Tal modificação foi implementada pois o sistema operacional utilizado (Linux) faz um ordenamento na lista de requisições de E/S antes de submeter para a controladora do disco [1, 4].

O número máximo de *threads* em paralelo é um parâmetro configurável indicado na inicialização do componente *StorageServer*. Enquanto o número de *threads* ativas for menor que o valor máximo e existir um pedido a ser executado, o componente *StorageManager* dispara uma nova *thread* para o atendimento deste pedido e incrementa a variável que indica a quantidade de *threads* ativas. Neste momento, é armazenado o valor desta variável ($nThreads$) e o tempo corrente ($t_{inicial} = t_{corrente}$). Ao final da leitura/escrita do bloco são executadas quatro operações:

1. o valor da variável que indica a quantidade de *threads* ativas é decrementado;
2. é calculado o tempo de resposta, $amostraResp = t_{corrente} - t_{inicial}$, e o tempo de serviço que é obtido através dos tempos entre as saída de *threads*, $amostraServ = t_{corrente} - t_{saida\_thread\_anterior}$;
3. é armazenado o tempo de saída da *thread* atual ($t_{saida\_thread\_anterior} = t_{corrente}$);
4. são atualizados os tempos médios de serviço e resposta, utilizando a equação 5.1, de acordo com o número de *threads* armazenado inicialmente, $t_{medio\_serv}[nThreads] = (1 - \tau) \times t_{medio\_serv}[nThreads] + \tau \times amostraServ$ e $t_{medio\_resp}[nThreads] = (1 - \tau) \times t_{medio\_resp}[nThreads] + \tau \times amostraResp$.

Cabe salientar que este tempo de resposta calculado leva em consideração os tempos do sistema operacional, incluindo as chamadas para reposicionamento do descritor de arquivo e para leitura do bloco de dados, e o tempo gasto no gerenciamento com os demais processos ativos na máquina.

Outras medidas obtidas em tempo real são: o tempo de espera de cada cliente (tempo entre a chegada de um cliente até o início do processo de admissão deste cliente) e de cada etapa do processo de admissão (t_{adm}), que são (1) criação da lista de pedidos do novo cliente, (2) criação da lista agregada (utilizada na simulação) e (3) simulação da lista de eventos. Para cada medida é calculado o tempo médio e a variância. O cálculo do tempo de espera também é efetuado de acordo com o número de clientes na fila de espera para o processo de admissão e para as opções (2) e (3) também são calculados os tempos médios de acordo com o número de clientes simulados durante a admissão.

O elemento responsável pela criação da lista de requisições de cada cliente é o componente fluxo. A criação ocorre no momento da solicitação de abertura de um objeto (ilustrado na Figura 5.8), de acordo com o algoritmo descrito em português estruturado na Figura 5.9.

Após a criação da lista de requisições, o componente *StreamManager* executa o controle de admissão, descrito mais adiante. Caso o usuário seja admitido, este cliente é inserido na lista de clientes admitidos mantida pelo *StreamManager*, conforme exemplo ilustrado na Figura 5.11. A estrutura de dados utilizada para a criação da lista de clientes admitidos é descrita na Figura 5.10.

5.1.4 Gerenciamento de *Buffers*

Como pode ser observado na Figura 5.12, cada componente fluxo (item 2) armazena vários atributos de um único cliente (no exemplo, Cliente 1), entre eles, o tempo de ocorrência do último pedido efetuado pelo cliente; o RTT deste cliente (RTT_1); informações dos *buffers*, tanto no servidor (quantidade de blocos que já estão no *buffer*), quanto no cliente (quantidade de blocos que devem ser enviados para o cliente indicando a quantidade de posições vazias no *playout buffer*); um ponteiro para a lista de previsão de pedidos (lista de requisições); um vetor que contém informações dos blocos armazenados no *buffer* do servidor ($Buffer_{cliente_1}$) e uma fila de pedidos, onde são armazenados os pedidos de leitura do cliente que devem esperar para serem submetidos para os discos (de acordo com o policiamento de tráfego descrito na seção 5.2). Além disto, cada componente fluxo mantém um ponteiro para a estrutura de dados mantida pelo *StreamManager*, onde são definidos, entre outros atributos, dois ponteiros, um para o próximo bloco de dados a ser requisitado pelo cliente e o outro para o próximo bloco a ser requisitado para o disco (ilustrados na Figura 5.10), denominados de *NextBlockToBeRequested* e *NextBlockToBeFetched*, respectivamente.

```
Função cria_lista_requisições( RTT, BC, Nome_Arquivo_com_tempos)
  tbuffer_servidor = tbuffer_cliente = tempo = 0;
  TamBuffer = BC + BS;
  atualiza_tempo_de_servico_dos_discos( );
  cria vetor Tempo_Requisição[ Tambuffer ];
  se abre_arquivo( Nome_Arquivo_com_tempos )
    enquanto não for final do arquivo
      lê duração do bloco;
      aloca estrutura de dados para bloco corrente (bcorrente);
      faz mapeamento do bloco e atribui valores na estrutura de dados
      bcorrente->Bloco   = bloco;
      bcorrente->Duração = duração;
      se ( bloco >= Tambuffer )
        bcorrente->trequisição = Tempo_Requisição[módulo(bloco,TamBuffer)] + tempo;
        tempo = bcorrente->trequisição;
      fim-se
      senão
        se ( bloco < BS )
          bcorrente->trequisição = tadm;
          atualiza_tbuffer_servidor( tserv_disco(bcorrente->disco) );
        fim-se
        senão
          bcorrente->trequisição = tadm + RTT;
          atualiza_tleitura_BC( tserv_disco(bcorrente->disco) );
        fim-senão
        se ( bloco + 1 == Tambuffer )
          atualiza_tenviar_BC( ttrans, tbuffer_servidor, tleitura_BC );
          atualiza_tbuffer_cliente( tleitura_BC, tenviar_BC );
          tempo = tadm + max( tbuffer_servidor, RTT ) + tbuffer_cliente + RTT;
        fim-se
      fim-senão
      Tempo_Requisição[módulo(bloco,TamBuffer)] = duração;
      arruma ponteiros para o bloco anterior e próximo bloco;
      bloco = bloco + 1;
    fim-enquanto
    fecha_arquivo( );
  fim-se
  atualiza_tempo_médio_geração_da_lista_de_requisições( );
  retorna início_da_lista;
fim-função
```

Figura 5.9: Algoritmo para criação da lista de requisições de um cliente

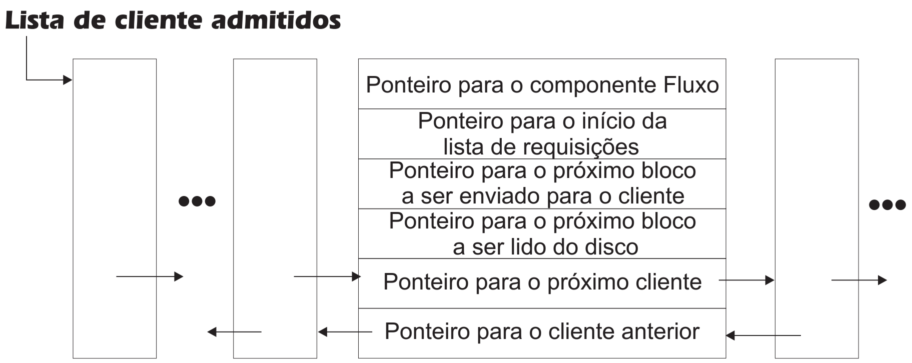

Figura 5.10: Estrutura de dados da lista de clientes admitidos

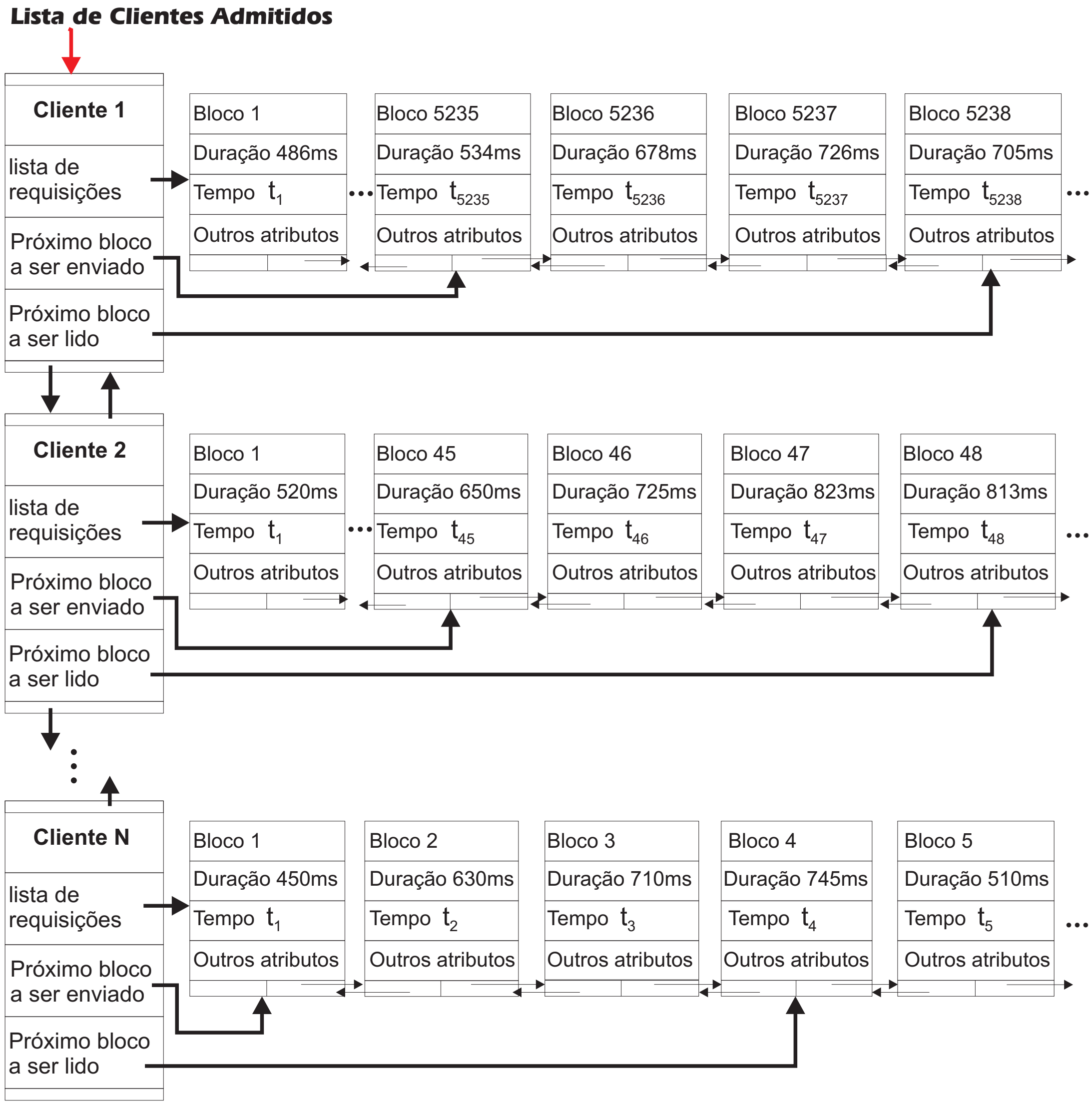

Figura 5.11: Lista de clientes admitidos mantida pelo componente *StreamManager*

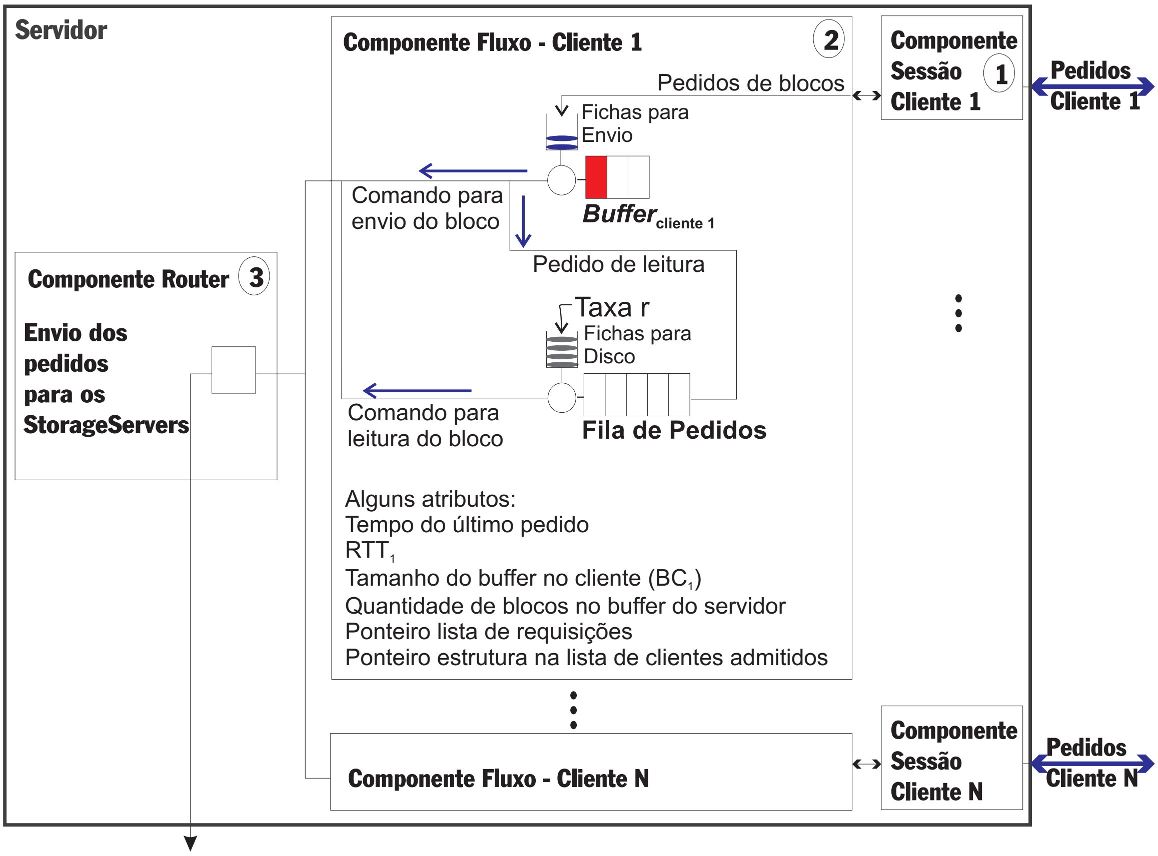

Figura 5.12: Visão de alguns componente da arquitetura do servidor RIO

De uma forma geral, a cada pedido solicitado por um cliente, o componente fluxo verifica se o bloco requisitado já está no *buffer* ou se já foi feita a sua requisição, mas a operação de leitura ainda não foi concluída. Caso uma destas opções seja verdadeira, o *StorageServer* responsável é notificado para que seja efetuada a transmissão deste bloco; caso contrário (quando o cliente altera o padrão de acesso), é feita a requisição para a leitura e o envio imediato do bloco solicitado para o cliente. Logo após, são feitas as atualizações nos dois ponteiros (*NextBlockToBeRequested* e *NextBlockToBeFetched*) e no *buffer* deste cliente, que durante o acesso seqüencial implica na leitura do próximo bloco a ser armazenado na posição livre, indicado pelo ponteiro *NextBlockToBeFetched*, conforme ilustra o caso 1 no exemplo descrito na Figura 5.13.

A cada bloco requisitado, o valor da variável que indica a quantidade de blocos a ser enviada é incrementada e a cada bloco enviado, o valor desta variável é decrementado. O mesmo ocorre com a variável que indica a quantidade de blocos armazenados no *buffer* no servidor. A cada bloco enviado para o cliente, o valor desta variável é decrementado e a cada bloco cuja leitura é concluída, o valor desta variável é incrementado. Com isto, para cada cliente, a qualquer momento, podem ser obtidas as seguintes informações: a quantidade de posições vazias, tanto no *playout buffer* do cliente, quanto no *buffer* do servidor; o próximo bloco a ser requisitado pelo cliente e o próximo bloco a ser lido do disco, e o tempo do próximo pedido, através do tempo corrente no servidor, do tempo do último pedido e dos tempos previstos na lista de requisições.

No entanto, operações do tipo videocassete (por exemplo, avançar e pausar), alteram o padrão de acesso, e conseqüentemente, a ordem dos pedidos. Neste caso, primeiro são atualizados os dois ponteiros (*NextBlockToBeRequested* e *NextBlockToBeFetched*) e, em seguida, os blocos já armazenados no *buffer*. Neste momento são verificados quais blocos ainda se encontram dentro da janela de transmissão (próximos blocos a serem requisitados que devem ser mantidos no *buffer*) e quais blocos devem ser substituídos, conforme será ilustrado a seguir.

Considere o exemplo ilustrado na Figura 5.13. Suponha que o *buffer* no servidor seja de três posições para cada cliente. De acordo com a situação inicial descrita nesta figura, os blocos 101, 102 e 103 estão armazenados no *buffer* no servidor. O caso 1 ilustra os ponteiros atualizados após a requisição do bloco 101 (acesso seqüencial). Neste caso os blocos 102 e 103 são mantidos no *buffer* e o bloco 101 é enviado e substituído pelo bloco 104. O caso 2 ilustra a alteração do padrão do acesso, quando o cliente requisita o bloco 150 (procedimento avançar), fazendo com que todos os blocos no *buffer* do servidor sejam substituídos pelos blocos 151, 152 e 153. O procedimento retroceder é ilustrado no caso 3, quando o cliente requisita o bloco 99. Neste caso os blocos 101 e 102 são mantidos e o bloco 103 é substituído pelo bloco 100.

Conforme visto no exemplo, quando ocorre a alteração no padrão de acesso, algumas ou todas as posições, tanto do *buffer* no cliente, quanto do *buffer* localizado no servidor são esvaziadas e são feitas as requisições para preenchimento destes *buffers*, de acordo com a situação atual do cliente. Como o servidor RIO utiliza alocação

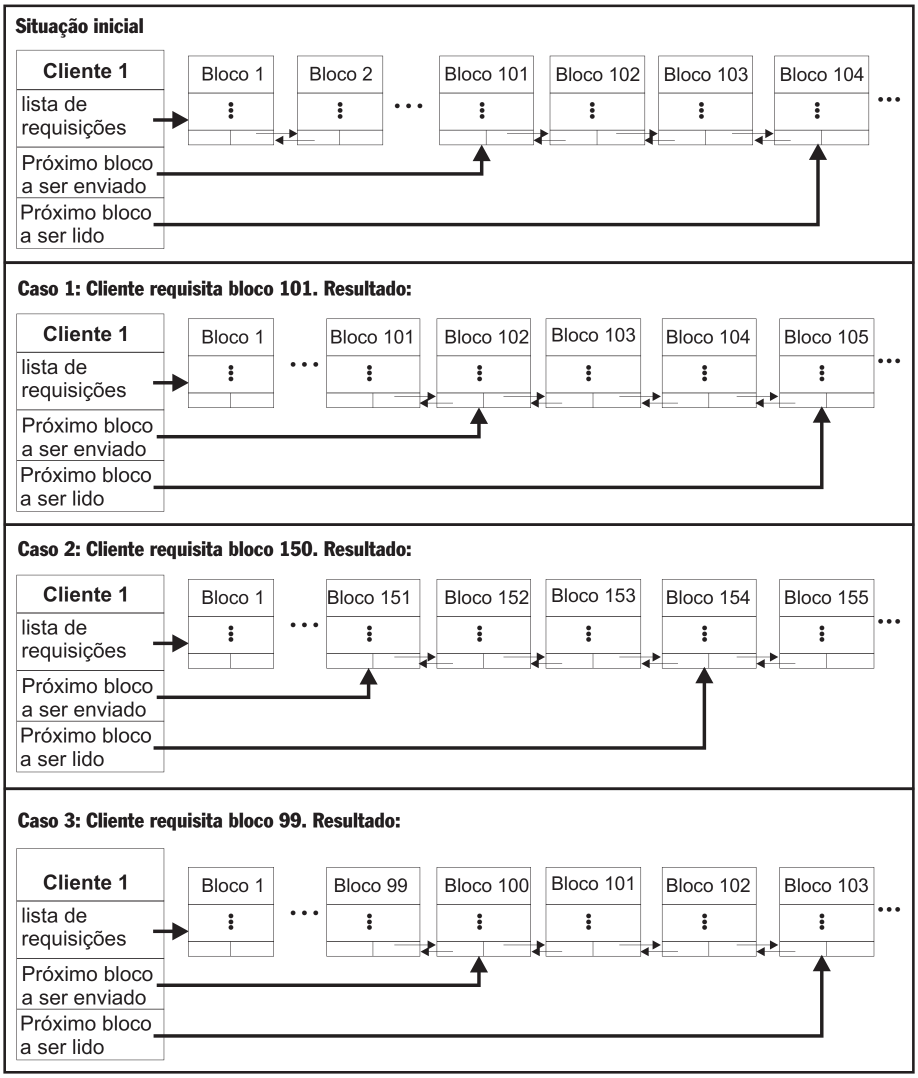

Figura 5.13: Exemplo do gerenciamento de *buffers*

aleatória, a mudança no padrão de acesso do cliente não altera o padrão de acesso no nível físico (discos). Entretanto, a partir deste momento, a QoS para qualquer um dos clientes admitidos pode ser violada, pois durante a admissão (processo descrito na seção 5.1.5) é considerado o acesso seqüencial do conteúdo para todos os clientes. Mas como a probabilidade de vários clientes alterarem o padrão de acesso ao mesmo tempo é pequena, pode ser que a carga gerada para o preenchimento dos *buffers* não cause problemas no atendimento dos demais clientes. Uma solução para suportar estas variações no tempo de requisição dos blocos poderia ser, por exemplo, limitar o número de usuários abaixo da capacidade máxima do servidor de modo a garantir a QoS.

5.1.5 Controle de Admissão

Conforme descrito anteriormente, a idéia do controle de admissão proposto é utilizar todos os tempos envolvidos no atendimento de um pedido, juntamente com as informações *a priori* dos vídeos armazenados e as informações de cada cliente, para prever se um novo cliente pode ser admitido. Para isto, o servidor, no momento de cada admissão, faz uma simulação de forma a verificar tal condição.

Através da união de todas as listas (uma para cada cliente), obtém-se a lista de eventos (pedidos) de todos os clientes. Para os clientes já admitidos só são considerados os pedidos a partir do tempo de chegada do novo cliente (pedidos ainda não processados), conforme ilustra a Figura 5.14.

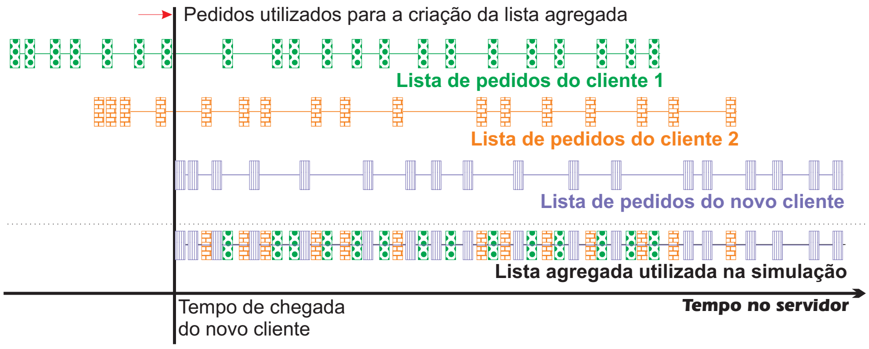

Figura 5.14: Pedidos utilizados na criação da lista agregada

A Figura 5.15 ilustra um exemplo da criação da lista de eventos para um cliente já admitido (cliente 1) no momento de admissão do novo cliente. Suponha que o *buffer* no servidor seja de três posições e que o intervalo entre o tempo corrente (chegada do novo cliente) e o último pedido efetuado pelo cliente 1 seja de $215ms$. Como pode ser observado, o próximo bloco (já armazenado no *buffer* do servidor) a ser enviado para o cliente 1 é o bloco 100 e o próximo pedido de leitura deste cliente é pelo bloco 103 no tempo t_{103}, sendo assim o tempo atual do cliente (localização dentro do seu fluxo de dados), conforme descrito na linha 5 da Figura 5.16, é $t_{102} + 215ms = 72720ms$. Desta forma, o evento de requisição do bloco 103 será daqui a $t_{evento_{103}} = t_{103} - 72720ms = 435ms$, o pedido para o disco do bloco 104 será daqui a $t_{evento_{104}} = t_{104} - 72720ms = 1080ms$ e assim por diante.

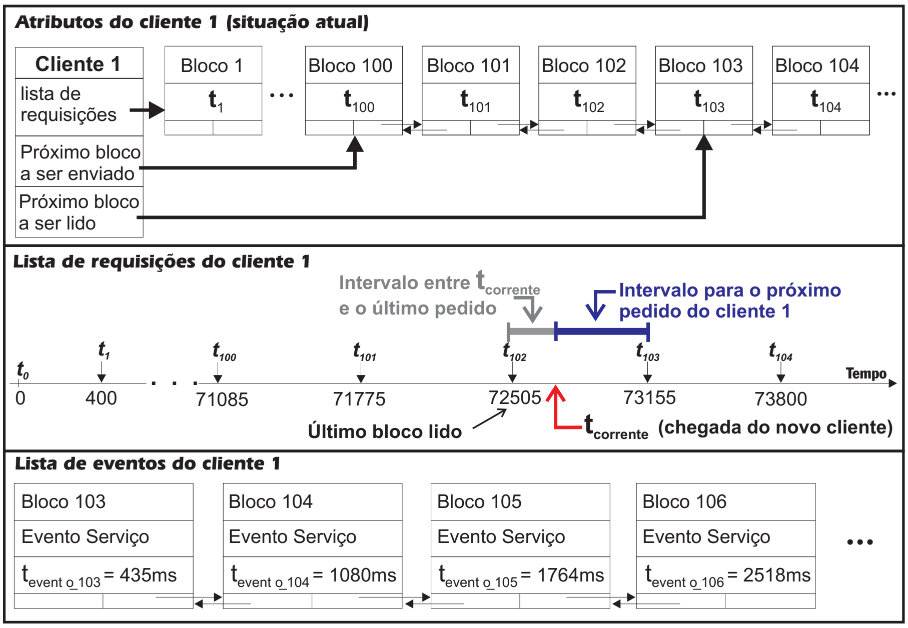

Figura 5.15: Exemplo da lista de eventos para um cliente

O algoritmo utilizado para a simulação do sistema pode ser visualizado na Figura 5.17. Este algoritmo pode ser dividido em duas fases. Na primeira, é criada a lista de eventos, de acordo com o algoritmo descrito na Figura 5.16, e são obtidas as informações necessárias para a simulação, entre elas, as informações atuais dos *buffers* e tempos de envio (RTT) de cada cliente. Na segunda fase, é feita a simulação do sistema, utilizando as informações obtidas, incluindo os tempos de serviço de cada um dos discos utilizados pelo servidor, e a lista de eventos criada.

Inicialmente só existem eventos de pedidos a serem atendidos pelos discos (tipo 1). À medida que a simulação avança, novos eventos são inseridos nesta lista: eventos de serviço efetuados pelos discos (tipo 2) e eventos de chegada dos blocos em cada um dos clientes (tipo 3), como pode ser visualizado na Figura 5.4.

Para cada evento existe um conjunto de ações a serem executadas. Considere $buffer\_servidor_i$ o contador que indica a quantidade de blocos no *buffer* do cliente i localizado no servidor; $buffer\_cliente_i$ o contador que indica a quantidade de blocos no *playout buffer* do cliente i e $N\_hiccups_i$ o contador que indica o número de *hiccups* ocorrido para o cliente i.

```
Função gera_lista_de_eventos( )
tcorrente = tempo atual no servidor;
percorre lista_de_clientes_admitidos e para cada cliente
  pega ponteiro para próximo bloco a ser lido do disco;
  se (cliente->id não for igual a id do novo cliente)
    Tempo_atual[cliente->id] = cliente->tempo_do_bloco_anterior
                             + (tcorrente - tempo_do_último_pedido);
    buffer_servidor[cliente->id]= pega_qtde_blocos_no_buffer_do_servidor();
  fim-se
  senão
    // cliente está começando agora
    Tempo_atual[ cliente->id ] = 0;
    buffer_servidor[ cliente->id ] = 0;
  fim-senão
  buffer_cliente[cliente->id] = pega_qtde_blocos_no_buffer_do_cliente(cliente->id);
  RTT[cliente->id] = pega_RTT_do_cliente(cliente->id);
  QtdeBlocos[cliente->id] = pega_qtde_de_blocos_do_video(cliente->id);
  avança ponteiro para o próximo cliente;
fim-percorre

enquanto tiver algum pedido ainda não processado
dos clientes da lista_de_clientes_admitidos
  pega_o_próximo_pedido_entre_todos_os_clientes();
  aloca estrutura para evento;
  evento->tipo  = chegada de um pedido de um cliente;
  atribui valores para demais variáveis (id do cliente, disco )
  evento->tempo = tempo_do_próximo_pedido - Tempo_atual[cliente->id];
  arruma ponteiros para o evento anterior e próximo evento;
fim-enquanto
retorna lista_de_eventos;
fim-função
```

Figura 5.16: Algoritmo (em português estruturado) para criação da lista de eventos

Caso o evento seja do tipo:

1. chegada de um pedido: se $buffer\_servidor_i > 0$, é inserido um evento do tipo 3 para o cliente i. Logo após, o valor de $buffer\_servidor_i$ é decrementado e é inserido um evento do tipo 2, solicitando a leitura do próximo bloco a ser armazenado no *buffer* do servidor. No processamento do evento do tipo 1, o valor de $buffer\_cliente_i$ é decrementado. Se este valor ficar negativo, o valor de $N\_hiccups_i$ é atualizado.

2. serviço de um bloco: se $buffer\_servidor_i < 0$, indica que o bloco já foi requisitado pelo cliente, então é inserido um evento do tipo 3 para o cliente i. Logo após, o valor de $buffer\_servidor_i$ é incrementado e o tamanho da fila do disco responsável pelo atendimento do bloco corrente é decrementado.

3. chegada de um bloco: o contador $buffer\_cliente_i$ é incrementado. Se o valor de $buffer\_cliente_i = 0$, é calculado o intervalo que o *playout buffer* ficou vazio.

Como no protótipo do servidor são disparadas várias *threads* em paralelo para a leitura dos blocos de dados, cada *thread* atualiza o tempo de serviço de acordo com o número de *threads* ativas no começo do seu atendimento. Para a inserção de um evento do tipo 2, o tempo de serviço de um pedido simulado pelo controle de admissão é obtido de acordo com o tamanho da fila do disco (igual ao número de *threads* ativas), utilizando as informações dos discos coletadas em tempo real. Para a inserção de um evento do tipo 3, o tempo de chegada de um bloco de dados é deslocado utilizando o RTT coletado por cada cliente durante os passos iniciais do protocolo de comunicação.

Com o conhecimento do tempo de requisição de cada bloco (para todos os clientes) e dos tempos de atendimento para cada requisição, é possível verificar durante a simulação a situação dos *buffers* de cada cliente, tanto no servidor, quanto no cliente, e se algum dos requisitos de QoS (número de *hiccups* e intervalo máximo vazio do *playout buffer*) é violado caso o novo cliente seja admitido. Comentários sobre o custo computacional são apresentados na seção 6.2.

```
Função controle_de_admissão( )
lista_de_eventos = gera_lista_de_eventos( );
pode_admitir = verdadeiro;
enquanto (não for final da lista_de_eventos) e (pode_admitir == verdadeiro)
  se (evento->tipo == chegada de um pedido de um cliente)
    se (buffer_servidor[ evento->cliente ] > 0)
      // insere evento de envio do bloco para cliente
      tipo    = chegada de um bloco em um cliente;
      cliente = evento->cliente;
      tempo   = tempo_da_transmissão + RTT[evento->cliente];
      insere_evento( tipo, cliente, tempo );
    fim-se
    decrementa buffer_servidor[evento->cliente];
    decrementa buffer_cliente[evento->cliente];
    // verifica se o buffer do cliente esvaziou
    se (buffer_cliente[evento->cliente] == -1)
      incrementa N_hiccups[evento->cliente];
      proporcao_de_falhas = N_hiccups[evento->cliente]x100/QtdeBlocos[evento->cliente];
      se ( proporcao_de_falhas > NumberOfRequestsWhenEmptyBuffer)
          pode_admitir = falso;
      tempo_inicial[evento->cliente] = evento->tempo;
    fim-se
    se (buffer_cliente[evento->cliente] < 0)
      incrementa N_Total_de_Falhas[evento->cliente];
    // insere evento de leitura para próximo bloco
    incrementa tam_fila_disco[evento->disco];
    tipo    = serviço de um bloco;
    cliente = evento->cliente;
    tempo   = tempo_do_disco(evento->disco,evento->tempo) +
              tserv_disco( evento->disco,tam_fila_do_disco[evento->disco] );
    insere_evento( tipo, cliente, tempo );
  fim-se
  senão se (evento->tipo == serviço de um bloco)
    se (buffer_servidor[ evento->cliente ] < 0)
      // insere evento de envio do bloco para cliente (está esperando)
      tipo    = chegada de um bloco em um cliente;
      cliente = evento->cliente;
      tempo   = tempo_da_transmissão + RTT[evento->cliente];
      insere_evento( tipo, cliente, tempo );
    fim-se
    incrementa buffer_servidor[evento->cliente];
    decrementa tam_fila_do_disco[evento->disco];
  fim-se
  senão se (evento->tipo == chegada de um bloco em um cliente)
    incrementa buffer_cliente[evento->cliente];
    // verifica se o buffer do cliente estava vazio
    se (buffer_cliente[evento->cliente] == 0)
      intervalo_vazio = evento->tempo - tempo_inicial[evento->cliente];
      Tempo_buffer_vazio[ evento->cliente ] += intervalo_vazio;
      se (Maior_Intervalo_buffer_vazio[evento->cliente] < intervalo_vazio)
          Maior_Intervalo_buffer_vazio[evento->cliente] = interval_vazio;
      se (Maior_Intervalo_buffer_vazio[evento->cliente] > MaxIntervalEmpty)
          pode_admitir = falso;
    fim-se
  fim-se
fim-enquanto
retorna pode_admitir;
fim-função
```

Figura 5.17: Algoritmo (em português estruturado) para controle de admissão

5.2 Outras modificações efetuadas no Servidor RIO

Várias modificações foram efetuadas no servidor RIO de modo a incorporar novas características e melhorar o seu desempenho, como por exemplo, a alteração na política de escalonamento de pedidos descrita na seção 5.1.3. Nesta seção são apresentadas as alterações realizadas no protocolo de comunicação e no policiamento de tráfego.

O protocolo de comunicação da versão original é descrito na Figura 5.18. Como pode ser observado, vários passos precisam ser executados até que o cliente possa começar a exibição do conteúdo. De modo a diminuir esta latência inicial durante o acesso ao servidor RIO, o protocolo de comunicação foi modificado para a versão apresentada na Figura 5.8. Na versão atual, o pedido de abertura de uma sessão já retorna o tamanho do bloco de dados utilizado pelo servidor, o pedido de abertura de um fluxo já retorna a quantidade máxima de pedidos ativos para este fluxo aberto e a abertura de um objeto já retorna o tamanho do objeto solicitado. Com isto, obteve-se uma diminuição de 3 RTTs durante a comunicação do cliente com o servidor.

Com relação ao policiamento do tráfego, na versão original, os pedidos de um cliente eram policiados de forma que apenas um pedido era repassado ao componente *Router* dentro de um intervalo de tempo, calculado no momento da sua admissão, conforme descrito na seção 4.4. Na versão atual, foi implementado um mecanismo chamado de *Leaky Bucket*, descrito na Figura 5.19.

O mecanismo implementado possui dois parâmetros: a capacidade de fichas que cada cliente pode armazenar (representada por F) e a taxa de geração destas fichas (representada por r). Para cada pedido a ser enviado para o *Router*, deve ser consumida uma ficha armazenada. Desta forma, a fila de pedidos armazena os pedidos caso a taxa de chegada de pedidos seja maior que a taxa de geração de fichas e a quantidade de fichas armazenadas seja 0. Caso a taxa de geração de fichas seja maior que a taxa de chegada de pedidos, a quantidade de fichas armazenadas vai sendo incrementada até o seu limite máximo F. Com este mecanismo, todo o

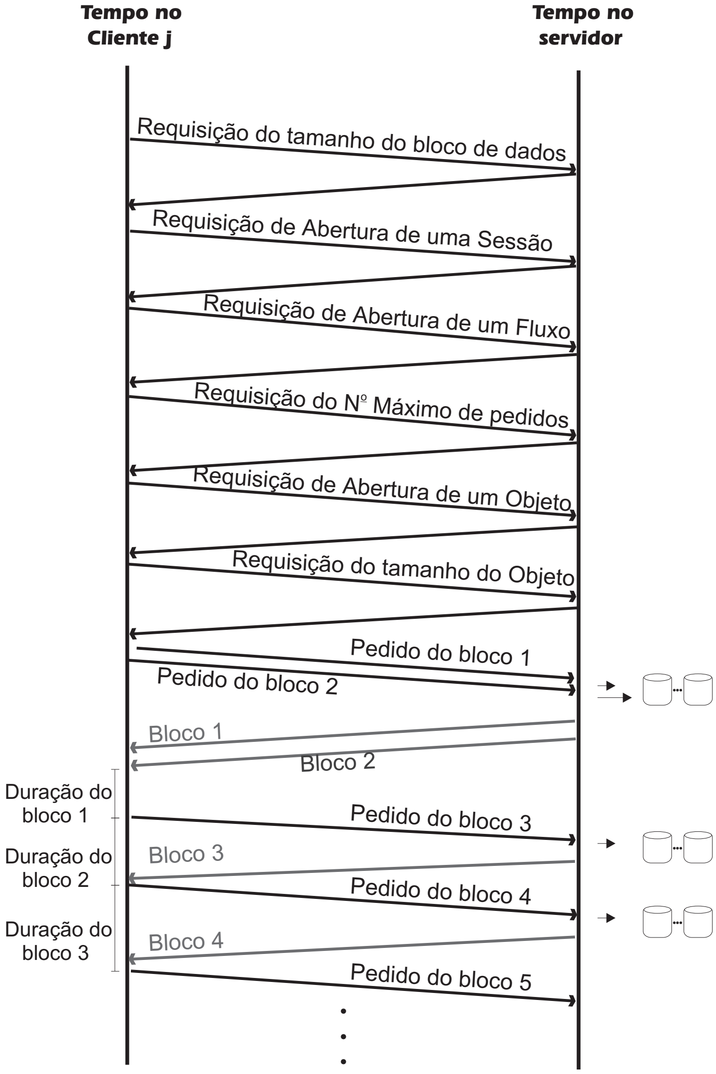

Figura 5.18: Protocolo de comunição da versão original do servidor RIO

fluxo que atravessa o regulador, é policiado, de forma que as rajadas (*bursts*) passam de forma controlada. A quantidade máxima de pedidos que podem ser repassados em qualquer intervalo de tempo t é $rt + F$. O valor de F é configurado durante a inicialização do sistema e através da taxa solicitada pelo cliente no momento de admissão é configurado o valor de r. Este mecanismo está implementado dentro do componente fluxo, ilustrado na Figura 5.12.

Este mecanismo oferece flexibilidade para fazer os pedidos para os discos, possibilitando a leitura de um maior número de blocos em um curto intervalo de tempo. A vantagem deste mecanismo em relação ao da versão original é que cada cliente pode acumular fichas durante a sua execução, como por exemplo, quando a exibição

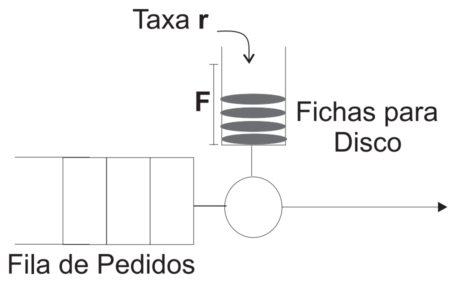

Figura 5.19: Mecanismo *Leaky Bucket*

do vídeo é pausada. No caso de operações que alteram o padrão de acesso do cliente, tais como operações de videocassete, onde os *buffers*, tanto do servidor, quanto do cliente são esvaziados, através deste mecanismo é possível o preenchimento mais rápido destes *buffers*.

5.3 Resumo

Conforme visto no capítulo 4, na versão original do servidor RIO, não era utilizado o mecanismo de *prefetching*. Desta forma, a cada bloco requisitado era feita a sua leitura e o envio imediato para o cliente. A cada intervalo de tempo, calculado na admissão do cliente, era repassado apenas um pedido de leitura de bloco para os discos. Além disso, não existia a infra-estrutura de medição e o controle de admissão utilizado era um controle simples, onde era indicado, na configuração do sistema, a capacidade do servidor (bits/segundo) e a banda reservada para os fluxos sem restrições de tempo. Na abertura de um fluxo de tempo real, através da largura de banda requisitada pelo novo candidato, era verificado se a banda já alocada, para os demais clientes, juntamente com a largura de banda requisitada não ultrapassava a capacidade do servidor disponível para os fluxos com restrições de tempo.

A Figura 5.20 ilustra a arquitetura atual do servidor RIO, onde pode ser visualizado um *buffer* no servidor para cada um dos clientes, utilizado pelo gerenciamento de *buffers*, implementado para o *prefetching* dos blocos de dados. Para cada cliente, existe um *Leaky Bucket* responsável pelo controle do envio dos pedidos de leitura para cada um dos discos utilizados.

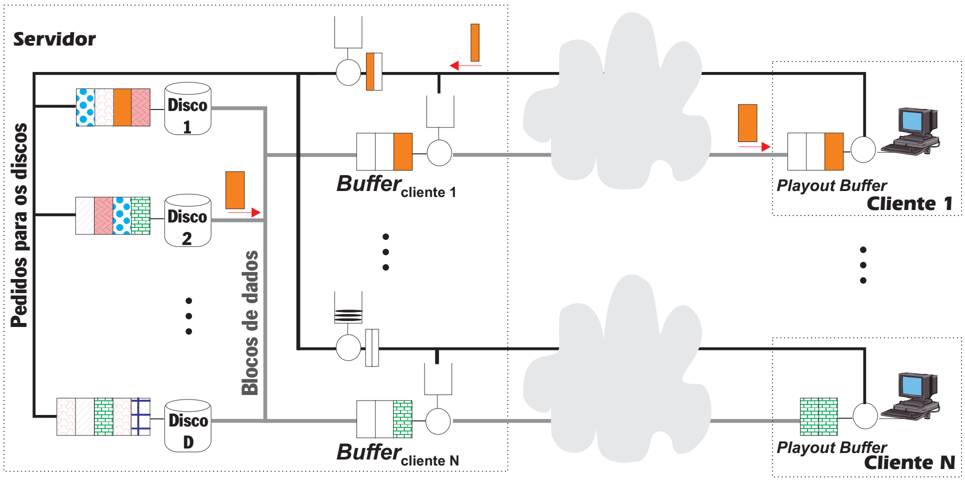

Figura 5.20: Arquitetura atual do servidor RIO

Através do ambiente criado é possível o estudo e a coleta de diversas medidas, como por exemplo, tempo de serviço e de espera na fila dos discos, tempo para geração da lista de requisições e tempo do processo de admissão de um cliente, entre outras.

Para cada cliente, a partir das informações *a priori* do vídeo selecionado, dos passos executados pelo protocolo de comunicação e dos tempos envolvidos neste processo, é criada a lista de requisições dos blocos do objeto. Utilizando as medidas obtidas em tempo real do sistema, juntamente com a lista de eventos criada a partir das listas de requisições de todos os clientes, o controle de admissão proposto executa uma simulação para verificar se o novo candidato pode ser admitido sem violar os requisitos de QoS, número máximo de *hiccups* e intervalo máximo que o *playout buffer* do cliente pode ficar vazio, para nenhum cliente.

Na versão atual foram implementadas novas funcionalidades e também alterações de modo a melhorar o protótipo do servidor RIO, como por exemplo, a redução de 3 RTTs no protocolo de comunição original e a mudança na disciplina de escalonamento dos pedidos para os discos de FIFO para a política utilizada no sistema operacional Linux (*elevator*) que faz o ordenamento dos pedidos para reduzir o tempo de *seek*.

Capítulo 6

Medidas e Resultados

Neste capítulo são apresentados os resultados obtidos nas simulações realizadas com o modelo criado através da ferramenta Tangram-II [28], os testes e as medidas obtidas durante a avaliação dos mecanismos implementados no servidor multimídia.

6.1 Simulação

Para o processo de simulação foi criado um modelo, ilustrado na Figura 6.1, que descreve o funcionamento do servidor e do gerenciamento de *buffers* implementados neste trabalho. Tais mecanismos foram descritos no capítulo 5.

Através deste modelo, é possível observar o comportamento do cliente a ser admitido e do servidor, como por exemplo, o comportamento das filas dos discos do sistema e do *playout buffer* do novo cliente juntamente com a carga gerada pelos clientes já admitidos. Além disto, é possível observar o impacto do retardo da rede entre o servidor e o novo cliente e também o impacto da mudança no padrão de acesso do cliente.

6.1.1 Modelo

Os componentes do modelo criado são: o objeto Clientes Admitidos, o objeto Novo Cliente, os objetos Discos, o objeto Fluxo, o objeto Rede e o objeto Roteador. Cada um desses objetos é detalhado a seguir. De uma forma geral, este modelo descreve o sistema real utilizado no servidor desenvolvido com algumas simplificações.

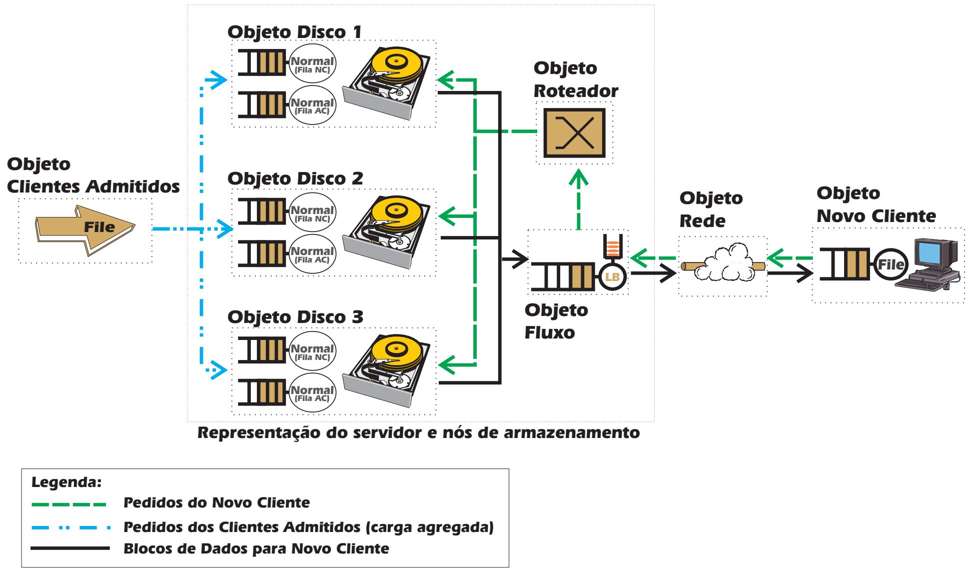

Figura 6.1: Modelo do sistema

Neste modelo, apenas o comportamento do novo cliente (isto é, de um cliente típico em um sistema com n clientes) é analisado para verificar se existe a possibilidade de ocorrerem falhas (bloco a ser visualizado não estar disponível no cliente quando solicitado) durante a sua execução, caso ele seja admitido. Além disto, é modelada a rede entre este cliente e o servidor, por onde os pedidos e os blocos de dados são transmitidos. Como o objetivo é analisar as falhas devido aos retardos aleatórios do servidor ou da rede, não é feita a simulação de perdas e nem de inversão da ordem de chegada dos pacotes transmitidos pela rede.

Os demais clientes são modelados em conjunto, a carga gerada por eles é representada pelo objeto Clientes Admitidos. Este objeto gera um fluxo agregado para os discos de forma a analisar o desempenho do sistema.

Do lado do servidor, além dos discos, são modelados o componente Fluxo do Novo Cliente e o Roteador para simular o atendimento dos pedidos do novo cliente.

Objeto Clientes Admitidos

O objeto Clientes Admitidos simula a carga agregada de todos os clientes já admitidos pelo sistema. Este objeto apenas gera os pedidos para cada um dos discos, sendo que a escolha de cada disco é feita aleatoriamente (como no servidor RIO). A distribuição utilizada para modelar o comportamento deste objeto é a distribuição FILE, cujo parâmetro indica o arquivo de *trace* que contém os intervalos entre os eventos deste objeto a serem disparados durante toda a simulação. Cada evento indica a requisição de um bloco por um dos clientes admitidos. O método utilizado para a criação dos *traces* e os vídeos utilizados são apresentados na seção 6.1.2.

Objeto Novo Cliente

O objeto Novo Cliente, como o próprio nome já indica, simula a carga gerada pelo novo cliente que está solicitando a abertura de um vídeo para o servidor. Assim como no objeto Clientes Admitidos também é utilizado um *trace*, contendo os intervalos entre os pedidos, para simular o seu comportamento (visualização dos blocos de dados). Este objeto possui três variáveis de estado:

- um vetor que representa o *playout buffer* no cliente, onde são armazenados os blocos de dados recebidos do servidor;
- um contador que indica quantas posições do *playout buffer* estão preenchidas com os blocos de dados e
- uma variável que indica qual bloco está sendo visualizado no momento corrente.

De acordo com os intervalos indicados no *trace*, são disparados os eventos de visualização dos blocos de dados do vídeo selecionado, onde cada evento representa a requisição do próximo bloco pelo aplicativo de exibição do conteúdo. Neste momento, as seguintes ações são executadas:

1. se o bloco de dados repassado ao aplicativo no evento anterior está armazenado no *buffer*, o contador que indica a quantidade de blocos no *buffer* é decrementado;

2. a variável que indica o próximo bloco a ser visualizado é incrementada, e então é verificado se este bloco já está armazenado no *buffer*, em caso negativo é atualizado o contador de falhas, que indica o número de vezes que o próximo bloco a ser visualizado não se encontra no cliente;

3. uma mensagem é enviada para o servidor (através do objeto Rede), simulando a requisição do próximo bloco que irá preencher a posição do *buffer* do cliente, agora vazia.

Para simular o envio dos blocos de dados, o servidor, através do objeto Rede, envia mensagens para o novo cliente, contendo o número do bloco transmitido. A cada mensagem recebida pelo Novo Cliente, é verificado se o número do bloco enviado está dentro do intervalo dos blocos que devem ser armazenados no *buffer* do cliente. Caso o bloco tenha chegado atrasado (depois do momento de sua exibição), nenhuma ação é executada. Caso contrário, o bloco de dados é armazenado no *buffer* e o contador que indica a quantidade de blocos no *buffer* é incrementado.

Objeto Rede

O objeto Rede é o responsável pelo envio das requisições do cliente para o servidor e pelo envio dos blocos de dados do servidor para o cliente. Para isto, este objeto possui duas filas, uma para as requisições e outra para os blocos. O atendimento destas filas é realizado por um servidor infinito, onde para cada mensagem recebida (tanto do Novo Cliente, quanto do servidor), o evento de serviço é "clonado"(um novo evento é escalonado pelo simulador, não sendo necessária a espera do disparo do evento escalonado anteriormente).

A distribuição utilizada para modelar o retardo da rede no modelo é a distribuição Normal (*Gaussian*). Esta distribuição foi escolhida com base em [31], onde foram realizadas diversas medições para calcular o RTT e através da análise dos dados coletados foi observado que a distribuição Normal é uma boa aproximação para a modelagem do retardo da rede.

Os parâmetros média e variância utilizados nas simulações realizadas foram calculados de forma que o coeficiente de variação fosse o mesmo coeficiente obtido em

[31]. Para cada simulação, de acordo com o coeficiente de variação de [31] e a média do retardo a ser simulado, é obtida a variância a ser utilizada.

Objeto Fluxo

O objeto Fluxo simula o componente Fluxo responsável pelo atendimento dos pedidos do Novo Cliente. Este objeto possui quatro variáveis de estado:

- um vetor que representa o *lookahead buffer* do cliente no servidor, onde são armazenados os blocos de dados recebidos dos discos;
- um contador que indica quantas posições deste *buffer* estão preenchidas com os blocos de dados;
- um contador que indica a quantidade de blocos que o Novo Cliente está esperando, ou seja, a quantidade de blocos requisitados pelo Novo Cliente que ainda não foram enviados;
- uma variável utilizada para indicar qual o próximo bloco a ser requisitado para os discos.

A cada mensagem recebida do Novo Cliente (através do objeto Rede), se o bloco de dados requisitado estiver no *buffer*, uma mensagem é enviada para o cliente, simulando o envio do bloco de dados, e o contador que indica o número de blocos armazenados no *buffer* é decrementado. Caso contrário, o contador que indica a quantidade de blocos a ser enviada para o Novo Cliente é incrementado. Logo após, em ambos os casos, é feita a requisição do próximo bloco para os discos através do objeto Roteador.

A cada mensagem recebida por um dos discos, é verificado se o bloco lido já foi requisitado pelo cliente ou deve ser armazenado no *buffer* devido ao mecanismo de *prefetching*. Caso o bloco já tenha sido solicitado é enviada uma mensagem simulando o envio do bloco de dados e o contador que indica a quantidade de blocos a ser enviada para o Novo Cliente é decrementado. Caso contrário, o bloco é armazenado no *buffer* e o contador que representa a quantidade de blocos armazenados é incrementado.

Objeto Roteador

O objeto Roteador faz o redirecionamento dos pedidos efetuados pelo componente Fluxo para cada um dos discos do sistema. A cada mensagem recebida do objeto Fluxo, é escolhido aleatoriamente um dos discos para o atendimento desta requisição.

Objetos Disco 1, Disco 2 e Disco 3

Cada um dos discos do sistema é modelado por um objeto Disco. A distribuição utilizada para modelar o tempo de serviço é a distribuição Normal, com os parâmetros média e variância do tempo de serviço para a leitura de um bloco de dados. Em algumas simulações também foi utilizada a distribuição Lognormal para verificar a sensibilidade do modelo.

Diversas amostras foram coletadas de cada um dos discos utilizados pelo servidor, através da ferramenta *hdparm* disponível no Linux, que extrai a taxa de leitura em MB/seg. A partir dos valores extraídos foram computados a variância e o tempo médio de serviço de um bloco de dados (usando o mesmo tamanho utilizado pelo servidor - 128KB). Os valores obtidos estão descritos na Tabela 6.1.

	Taxa Média (MB/seg)	**Tempo médio de serviço (ms)** t_{medio_i}	**Variância** σ_i^2
Disco 1 - IDE	9,39	14,131	9,189
Disco 2 - SCSI	13,25	9,432	0,0071
Disco 3 - IDE	15,79	8,054	4,405

Tabela 6.1: Medidas dos discos coletadas com o *hdparm*

Devido à necessidade do armazenamento do número do bloco de dados solicitado pelo Novo Cliente para a leitura, que identifica o bloco a ser armazenado no *buffer*, foi necessária a modelagem de duas filas de atendimento (filas NC e AC) para cada disco.

A fila de atendimento dos pedidos do Novo Cliente (fila NC) é representada por um vetor, onde cada posição armazena o número do bloco a ser lido, e por um contador que indica o tamanho atual da fila ($cont\_NC$). Já a fila de atendimento dos pedidos dos Clientes Admitidos (fila AC) é representada apenas por um contador indicando o tamanho atual desta fila ($cont\_AC$).

Para cada fila existe um evento de serviço associado, cuja distribuição é Normal e os parâmetros média e variância são proporcionais ao tamanho da fila, pois as duas filas são servidas por um único disco. A política de escalonamento dos pedidos de cada fila é FIFO.

Para a fila dos pedidos do Novo Cliente, os parâmetros da distribuição Normal, média e variância do tempo de serviço do disco i, $\mathcal{N}(t_{medio\_NC_i}, \sigma^2_{NC_i})$, são

$$t_{medio\_NC_i} = t_{medio_i} \times \frac{cont\_NC + cont\_AC}{cont\_NC} \tag{6.1}$$

$$\sigma^2_{NC_i} = \sigma^2_i \times \left(\frac{cont\_NC + cont\_AC}{cont\_NC}\right)^2 \tag{6.2}$$

e para a fila dos pedidos dos Clientes Admitidos, os parâmetros da distribuição Normal, $\mathcal{N}(t_{medio\_AC_i}, \sigma^2_{AC_i})$, são

$$t_{medio\_AC_i} = t_{medio_i} \times \frac{cont\_NC + cont\_AC}{cont\_AC} \tag{6.3}$$

$$\sigma^2_{AC_i} = \sigma^2_i \times \left(\frac{cont\_NC + cont\_AC}{cont\_AC}\right)^2 \tag{6.4}$$

Para cada disco, a cada mensagem recebida do objeto Roteador, um pedido é armazenado na fila NC e o contador $cont\_NC$ é incrementado, e a cada evento de serviço da fila NC, uma mensagem é enviada para o objeto Fluxo, simulando a conclusão do processo de leitura de um bloco de dados, e o contador $cont\_NC$ é decrementado. Caso a mensagem recebida seja do objeto Clientes Admitidos, o contador $cont\_AC$ é incrementado e a cada evento de serviço da fila AC o contador $cont\_AC$ é decrementado.

6.1.2 Geração dos *Traces*

Para a geração dos *traces* utilizados na simulação foi implementado um programa em linguagem C que, através dos arquivos contendo a duração de cada bloco de um filme (processo descrito no capítulo 5), gera um arquivo contendo o intervalo entre os pedidos de todos os clientes simulados no *trace* agregado. A Tabela 6.2 lista todos os vídeos (formato MPEG 1) utilizados para simular os clientes, tanto na geração dos *traces*, quanto nos experimentos, descritos na seção 6.2.

Vídeo	**Duração (minutos)**	**Número de blocos (128KB)**	**Tempo médio de duração de um bloco (ms)**
13th Floor	100,37	8021	750,82
Final Fantasy	104,97	8481	742,62
Matrix	136,25	11016	742,10
Starwars I	125,88	10175	742,29
Stigmata	95,83	7654	751,22
What Dreams	111,99	8916	753,64
Palestra 1	105,52	7326	864,15
Palestra 2	89,89	7182	750,97

Tabela 6.2: Filmes utilizados nos *traces*

O *trace* gerado varia de acordo com o número de clientes desejados para a simulação. Para cada cliente é escolhido aleatoriamente: (a) um vídeo; (b) o bloco corrente (que simula o bloco que o cliente está visualizando no momento de admissão de um novo cliente); (c) o tempo até o próximo pedido de leitura (tempo entre 0 e a duração do bloco corrente) e (d) o tamanho do buffer do cliente entre 1 e 7 posições (para determinar os próximos pedidos para os discos). A partir destes parâmetros é criada uma lista de pedidos para cada cliente e, em seguida, é criada uma única lista ordenada com todos os pedidos. Através desta lista ordenada, é criado um arquivo (*trace*) contendo os intervalos entre os pedidos. Como descrito anteriormente, este *trace* é utilizado na distribuição FILE da ferramenta Tangram-II para simular a carga dos clientes no sistema.

6.1.3 Resultados

Várias medidas podem ser obtidas através da simulação do modelo descrito na seção anterior, tais como o tamanho médio das filas dos discos de acordo com a carga gerada pelos clientes, a quantidade de clientes que podem ser admitidos considerando o comportamento do objeto Novo Cliente e a quantidade de falhas no novo cliente para um dado tamanho de *playout buffer*. Estas medidas auxiliam, por exemplo, na configuração do tamanho do *playout buffer* no cliente e na configuração do tamanho do *buffer* para cada cliente no servidor, conforme descrito a seguir.

Configuração do tamanho do *playout buffer* no cliente

De acordo com o vídeo selecionado e o retardo da rede entre o cliente e o servidor é possível estimar o tamanho mínimo do *playout buffer* no cliente de modo a compensar as flutuações da rede. Para estimar este tamanho mínimo, foram executadas diversas simulações, desconsiderando os retardos aleatórios no atendimento dos pedidos no servidor (simulação sem a carga gerada pelo objeto Clientes Admitidos). Sendo assim, o atendimento dos pedidos do novo cliente é feito sem atrasos pelo servidor, na chegada da requisição o bloco solicitado é enviado imediatamente para o cliente.

Os dois exemplos desta seção apresentam a escolha do *buffer* do cliente para os vídeos Matrix e Stigmata (através dos arquivos de *traces* que contêm os tempos de consumo de cada bloco). Para cada valor de retardo da rede é calculado o tamanho do *buffer* para que não ocorram falhas durante a exibição de cada filme. Os valores de retardo utilizados foram 150ms, 200ms e 500ms.

A Figura 6.2 mostra os resultados obtidos para o filme Matrix, enquanto a Figura 6.3 mostra os resultados obtidos para o filme Stigmata. Para cada simulação foram executadas 10 rodadas com o intervalo de confiança igual a 95%.

Como pode ser observado, quanto maior o retardo da rede, maior deverá ser o tamanho mínimo do *playout buffer* no cliente. Para que não ocorram falhas durante a exibição do vídeo Matrix, é necessário um *buffer* para 9, 10 e 13 blocos, considerando os retardos de 150ms, 200ms e 500ms. Enquanto que para o filme Stigmata é

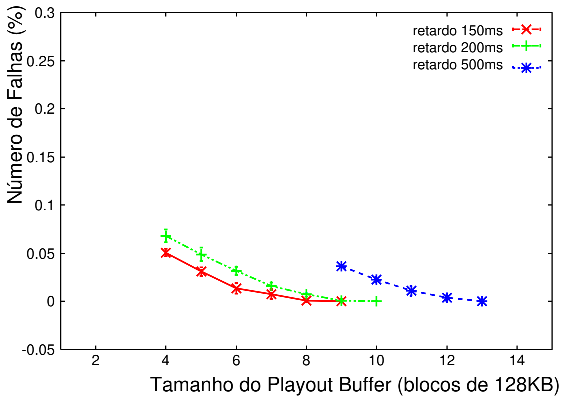

Figura 6.2: Tamanho mínimo do *playout buffer* para o filme Matrix (simulação)

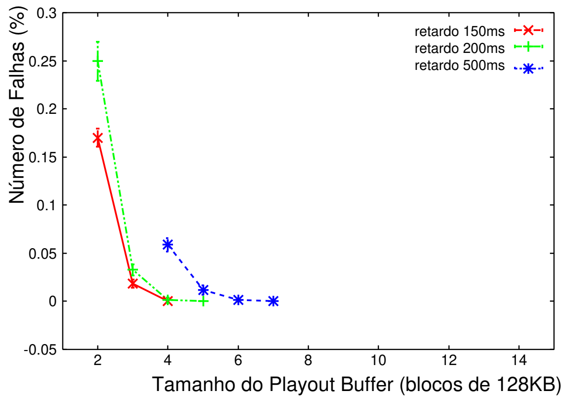

Figura 6.3: Tamanho mínimo do *playout buffer* para o filme Stigmata (simulação)

necessário um *buffer* para 4, 5 e 7 blocos considerando os mesmos retardos.

A diferença dos tamanhos calculados para cada filme depende da seqüência de duração dos blocos. Como visto anteriormente, o tempo de resposta entre o pedido por um bloco de dados e o recebimento deste deve ser menor ou igual ao tempo de consumo de todos os blocos armazenados no *buffer* do cliente. Desta forma, o tamanho mínimo do *playout buffer* depende da menor seqüência de blocos cuja soma dos tempos de consumo é igual ou superior ao tempo de resposta.

Para comparar os resultados apresentados acima, foi criado um modelo analítico simples, que é uma aproximação do modelo descrito anteriormente. Este novo modelo possui dois objetos: uma fila simulando o *playout buffer* do cliente e um servidor infinito para simular o retardo da rede, conforme ilustra a Figura 6.4.

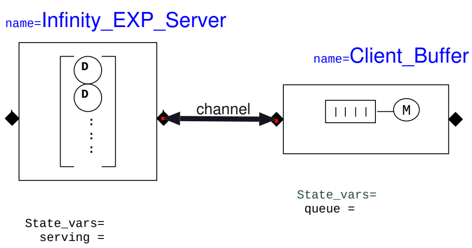

Figura 6.4: Modelo aproximado para resolução analítica

O objeto fila ($Client\_Buffer$) possui dois eventos de serviço, ambos com distribuição Exponencial de taxa λ, onde $\frac{1}{\lambda}$ é o tempo médio de consumo de um bloco.

Cada disparo destes eventos simula a visualização de um bloco de dados. O tempo médio de consumo depende do filme selecionado, sendo 751,22 ms e 742,1 ms para os vídeos Stigmata e Matrix, respectivamente. Um dos eventos está habilitado quando o tamanho da fila é maior que zero, neste caso, no disparo deste evento a variável de estado que indica o tamanho da fila ($fila$) é decrementada e uma mensagem simulando o próximo pedido de bloco é enviada para o servidor. O outro evento de serviço é habilitado quando a fila está vazia, caso onde é feita a atribuição do valor 1 à variável de estado $Falha$, indicando que uma falha ocorreu pois o bloco de dados não estava armazenado no *buffer* no momento de sua exibição. A cada mensagem recebida do objeto servidor, caso o bloco recebido não esteja atrasado (recebido após a sua exibição), a variável de estado $fila$ é incrementada e o valor da variável de estado $Falha$ é alterado para 0.

O objeto servidor infinito ($Infinity\_EXP\_Server$) possui uma variável de estado que indica a quantidade de pedidos que deve ser enviada para o cliente ($pedidos$). A cada mensagem recebida do objeto cliente, o valor desta variável é incrementado e a cada mensagem enviada, o seu valor é decrementado. O evento de serviço do servidor é modelado por uma distribuição Exponencial com taxa $\mu * pedidos$, onde $\mu = \frac{1}{RTT}$ (sendo RTT o *round trip time* entre o servidor e o cliente). Os valores de RTT utilizados foram 300ms, 400ms e 1000ms.

Após a resolução analítica deste modelo obtém-se a probabilidade da variável de estado $Falha$ ser igual a 1, isto é, a proporção de falhas em estado estacionário. A partir disto é possível calcular o número de falhas durante a exibição do vídeo multiplicando o valor desta probabilidade pelo tempo de duração do filme e pela taxa de consumo.

A Figura 6.5 mostra os resultados obtidos pela resolução analítica para o filme Matrix, enquanto a Figura 6.6 mostra os resultados obtidos para o filme Stigmata.

É importante observar que no primeiro modelo, utilizando simulação, com um intervalo de confiança de 95% a resposta indica se ocorrerão ou não falhas durante a exibição. No segundo modelo, utilizando resolução analítica, a resposta indica a probabilidade de falhas para um dado tamanho de *buffer* e retardo de rede.

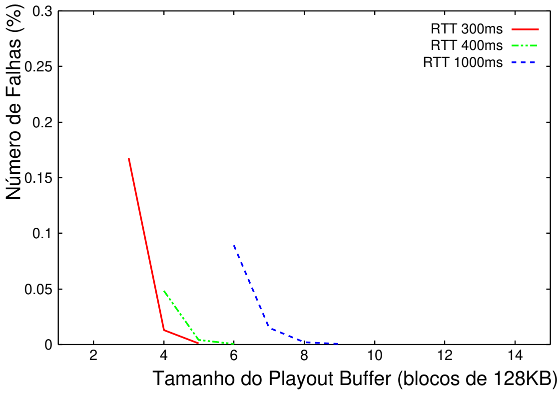

Figura 6.5: Tamanho mínimo do *playout buffer* para o filme Matrix (resolução analítica)

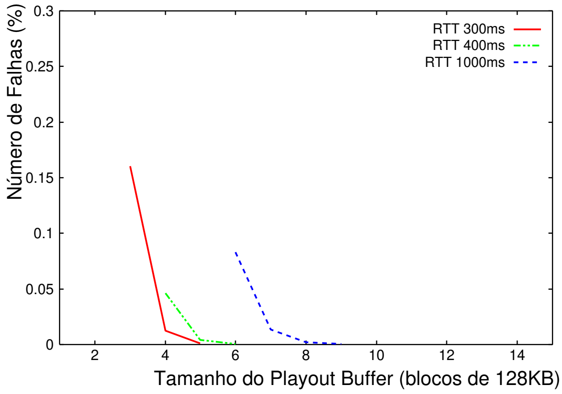

Figura 6.6: Tamanho mínimo do *playout buffer* para o filme Stigmata (resolução analítica)

Por exemplo, considerando $P_{falha} <= 10^{-06}$, através da solução analítica, os tamanhos dos *buffers*, para o filme Matrix, seriam de 5, 6, e 9 posições, para os retardos de 150ms, 200ms e 500ms. Para os *buffers* de tamanho 9, 10 e 13, calculados através da simulação para este mesmo filme, as probabilidades de falha seriam de 10^{-11}, 10^{-11} e 10^{-10}, respectivamente.

Assim como é possível configurar o tamanho do *playout buffer* no cliente para compensar as flutuações na rede, é possível configurar o tamanho deste *buffer*, de acordo com a carga atual do servidor (gerada pelos demais clientes), para também compensar as flutuações nos tempos de atendimento dos discos do sistema. Para isto, é necessária a simulação do modelo com a carga agregada do objeto Clientes Admitidos.

Tamanho do *buffer* no servidor × número de clientes admitidos

Várias medidas foram obtidas para um dado número de clientes admitidos e tamanho de *buffer* no servidor para cada cliente. O vídeo utilizado pelo Novo Cliente nas simulações foi o Stigmata. Para cada retardo da rede, o tamanho do *playout buffer* no cliente foi configurado utilizando o tamanho mínimo extraído através dos resultados da seção anterior.

Utilizando discos com taxas de transferência diferentes (como no servidor usado neste trabalho), o número máximo de usuários admitidos pelo sistema é limitado pelo disco com menor taxa. Sendo assim, para calcular o número de clientes admitidos pelo sistema foi considerado que o servidor possui três discos iguais, com tempo médio de serviço igual a 8,054 ms e variância 4,405 (todos iguais ao disco 3 - IDE).

As figuras 6.7, 6.8, 6.9, 6.10 e 6.11 mostram a proporção de falhas para o Novo Cliente de acordo com o número de clientes atendidos pelo sistema, considerando o tamanho do *buffer* no servidor de 5, 10, 20, 255 e 1020 posições, respectivamente. Para cada simulação foram executadas 5 rodadas com o intervalo de confiança igual a 95%.

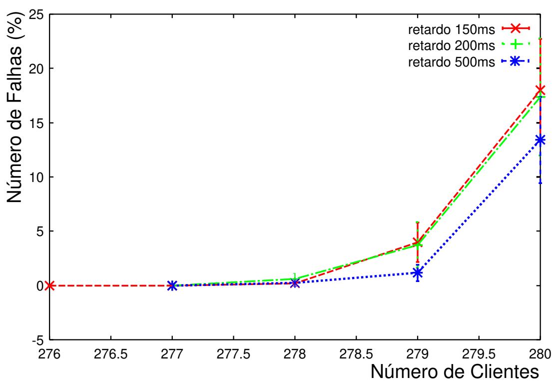

Figura 6.7: Número de Falhas *versus* Número de Clientes para tamanho do *buffer* no servidor = 5

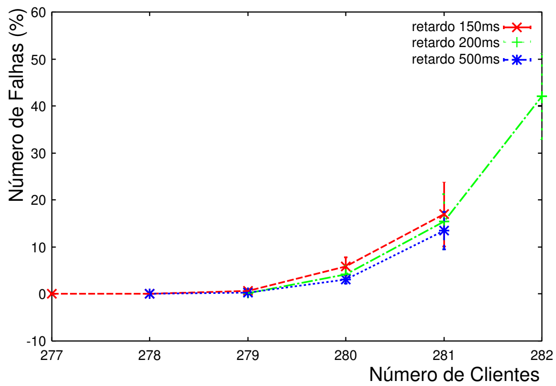

Figura 6.8: Número de Falhas *versus* Número de Clientes para tamanho do *buffer* no servidor = 10

Como pode ser observado, o fato de aumentar o tamanho do *buffer* no servidor, permite que os pedidos de cada cliente sejam servidos antecipadamente, e com isto, o número de clientes admitidos pelo sistema é maior. Solicitando os pedidos mais cedo não aumenta os requisitos de banda, pois a quantidade de tráfego gerada pelos vídeos é independente do tempo de requisição dos blocos (os pedidos estão apenas deslocados no tempo). O custo para aumentar o limite de tempo de resposta é o tamanho adicional do *buffer* necessário no cliente e no servidor e o aumento na latência inicial do cliente.

Aumentando o tamanho do *buffer* no servidor de 5 para 10, 20, 255 e 1020 posições, considerando o número de falhas igual a 0%, obteve-se um ganho de apro-

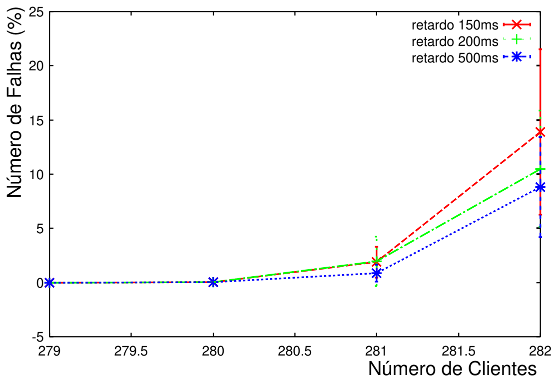

Figura 6.9: Número de Falhas *versus* Número de Clientes para tamanho do *buffer* no servidor = 20

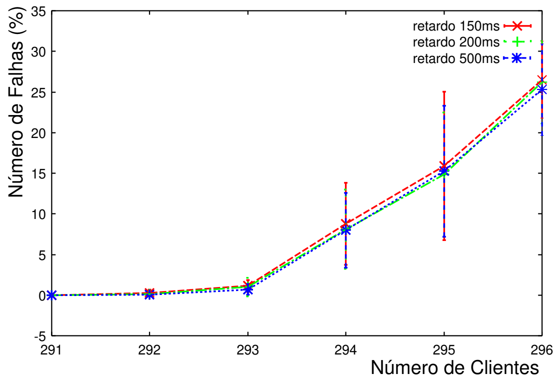

Figura 6.10: Número de Falhas *versus* Número de Clientes para tamanho do *buffer* no servidor = 255

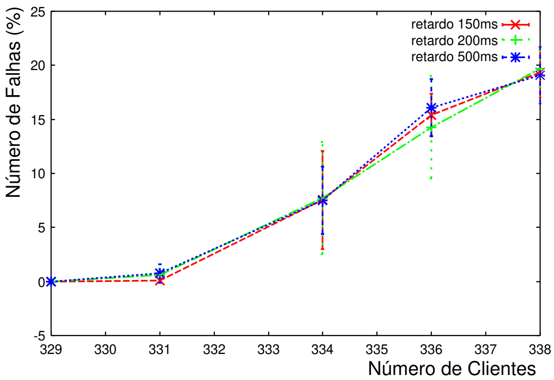

Figura 6.11: Número de Falhas *versus* Número de Clientes para tamanho do *buffer* no servidor = 1020

ximadamente 0,36%, 1,09%, 5,43% e 19,20% no número de clientes admitidos, respectivamente.

Outro aspecto observado é a diminuição da proporção de falhas com o aumento do retardo da rede simulado. Isto ocorre pois o *playout buffer* no cliente é configurado de acordo com o retardo. Neste caso quanto maior o retardo, maior é o tamanho do *buffer*, o que também ajuda a compensar as flutuações dos tempos de atendimento dos discos.

Comparando com um controle de admissão simples

Caso fosse feito um controle de admissão simples utilizando a taxa de leitura dos discos para verificar o número máximo de clientes que poderia ser admitido, o limite seria de 248 clientes, considerando a taxa de 1,5 Mbits/seg para cada cliente. Como o tempo médio de serviço de um bloco de 128KB (ou 1Mbit) é de 8,054ms, em um segundo cada disco poderia ler 124,1619 blocos, então a taxa dos três discos seria de 372,4857 blocos/segundo. Divindo a taxa total pela taxa de cada cliente, obtém-se o número de clientes suportados pelo sistema.

Considerando o limite de 248 clientes obtido acima, através da simulação do modelo com o controle de admissão mais complexo e utilizando o gerenciamento de *buffers*, para os tamanhos de *buffer* no servidor de 5, 10, 20, 255 e 1020 posições (no caso onde o número de falhas é igual a 0%), obteve-se um ganho de aproximadamente 11,29%, 11,69%, 12,50%, 17,34% e 32,66% no número de clientes admitidos, respectivamente.

6.2 Experimentos realizados com o protótipo do servidor RIO

Nesta seção é detalhada a plataforma utilizada para os testes no protótipo, os experimentos realizados para analisar a implementação no servidor e os resultados obtidos.

6.2.1 Plataforma de testes

A plataforma usada está ilustrada na Figura 6.12. Como pode ser observado são utilizadas duas máquinas servidoras de discos (máquinas 1 e 3), onde um processo *Storage Server* é executado por cada uma delas, e uma máquina onde é executado o processo *Server* (máquina 2). Para a execução dos clientes foram utilizadas as outras quatro máquinas. Todas as máquinas utilizam o sistema operacional Linux *Red Hat*

7.2 (kernel versão 2.4.7) e estão conectadas a uma rede de 100 Mbps, através de um *hub* de 100 Mbps.

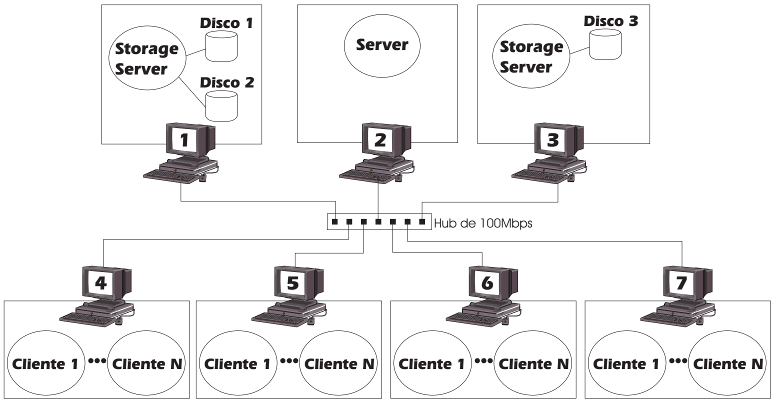

Figura 6.12: Plataforma de testes

A Tabela 6.3 descreve as características de cada uma das máquinas.

Máquina	**Processador**	**RAM**
1	AMD K6 350MHz	256M
2	Pentium IV 1.6GHz	1G
3	Pentium III 600MHz	384M
4	Pentium III 600MHz	256M
5	Pentium III 1GHz	512M
6	Pentium IV 1.6GHz	256M
7	Pentium IV 1.8GHz	1G

Tabela 6.3: Descrição das máquinas utilizadas nos experimentos

6.2.2 Clientes

Para a geração da carga de trabalho foi implementado um programa em linguagem C que dispara vários clientes. A chegada entre os clientes é baseada em um processo de *Poisson*, com taxa de 11,11 clientes/minuto. Esta taxa foi obtida de [48], onde é considerada a chegada de 1000 clientes a cada T unidades de tempo, sendo T igual a duração total de um vídeo. Para os testes realizados, o valor de T é igual a 90 minutos.

Durante a execução deste programa, são gerados intervalos de acordo com o processo de *Poisson* para a inicialização do próximo cliente. Para cada cliente é escolhida aleatoriamente a máquina onde este deve ser disparado e, após a espera do intervalo gerado (através da chamada de sistema *usleep*), o cliente é inicializado na máquina escolhida (através da chamada de sistema *rsh*).

Devido à decodificação do vídeo exigir um consumo considerável de processamento e memória, o cliente *riomtv* foi alterado para somente simular a exibição do vídeo e não exibí-lo propriamente. Cada cliente inicializado escolhe aleatoriamente o vídeo a ser solicitado ao servidor, dentre os descritos na Tabela 6.2, e o tamanho do seu *playout buffer*. Este processo da escolha do *playout buffer* foi realizado separadamente através da simulação do modelo, descrito na seção 6.1.3, de acordo com o vídeo e o retardo da rede. Neste caso, como a rede é local, o RTT utilizado para o cálculo do tamanho do *playout buffer* foi de 1ms.

Logo após, são executados os passos definidos pelo protocolo de comunicação descrito na seção 5.1.3. Caso o cliente seja admitido, são enviados os pedidos dos blocos para preencher o seu *buffer*. Após a sinalização enviada pelo servidor, indicando que a exibição do conteúdo pode iniciar, o cliente simula a visualização de cada bloco de acordo com o tempo de consumo coletado anteriormente para o vídeo selecionado (a partir de um arquivo contendo tais tempos). Desta forma, após a simulação da exibição do bloco corrente, é enviado o pedido do próximo bloco a ser armazenado no seu *buffer*.

Com a finalidade de verificar o comportamento durante a sua execução, cada cliente gera um arquivo de transações (log), com informações sobre o recebimento dos blocos e falhas ocorridas. Este arquivo contém, por exemplo, o número de vezes que o *buffer* do cliente ficou vazio (número de *hiccups*), o tempo total e o maior intervalo vazio.

Para a realização dos experimentos, o servidor RIO foi modificado para enviar apenas um pacote de 1500 bytes aos clientes, simulando o envio do bloco de dados de 128KB, de modo que o gargalo não fosse a rede, mas os discos utilizados pelo servidor. Isto foi feito com o objetivo de estressar o servidor e não a rede.

6.2.3 Resultados dos testes realizados no protótipo

Vários experimentos foram realizados de forma a verificar o número de clientes admitidos pelo sistema e a QoS oferecida para cada cliente.

Em todos os experimentos o tamanho do *buffer* no servidor para cada cliente é de cinco posições, o parâmetro de QoS que indica o intervalo máximo que o *playout buffer* do cliente pode ficar vazio é de 2000 ms e o parâmetro que indica número máximo permitido de *hiccups* é de 10% dos blocos do vídeo.

O controle de admissão só começa a atuar a partir do 19º cliente. Este valor foi baseado no maior tempo médio de serviço obtido, entre os três discos utilizados, durante alguns testes realizados no servidor RIO. Dado que $t_{serv\_disco_1}$ foi de 27,85ms, a taxa de leitura deste disco é de 35,91 blocos/segundo. Considerando que a taxa total de leitura é limitada pela menor taxa de todos os discos, a taxa total de leitura do servidor é de 107,735 blocos/segundo. Para uma taxa de consumo de 1,5 Mbits/segundo por cliente, o número máximo de clientes admitidos seria 71. Utilizou-se $\frac{1}{4}$ deste valor para a extração do número mínimo de clientes admitidos sem a execução do processo de simulação.

O resultado dos experimentos 1 ao 6 pode ser visualizado na Tabela 6.4. Nestes experimentos o *playout buffer* de cada cliente é o tamanho mínimo obtido através da simulação do modelo utilizando a ferramenta Tangram-II. Os clientes que simulam a visualização do vídeo Matrix possuem um *playout buffer* de três posições, enquanto os demais clientes possuem um *buffer* de duas posições.

Para alguns clientes, o parâmetro de QoS violado foi o número permitido de *hiccups*. No entanto, como pode ser observado pela Tabela 6.4, em todos estes experimentos o maior intervalo que o *playout buffer* de um cliente ficou vazio não ultrapassou 473,57ms. Pelos arquivos de log, observou-se que todos os blocos que

Experimento	Número de Clientes			Intervalo Máximo Vazio
	Disparados	Admitidos	QoS não garantida	
1	85	72	3	296,17ms
2	85	68	2	152,37ms
3	85	73	0	-
4	85	68	1	473,57ms
5	85	70	0	-
6	85	71	3	270,83ms

Tabela 6.4: Resultado dos experimentos 1 ao 6

ocasionaram *hiccups* chegaram durante o intervalo de sua exibição.

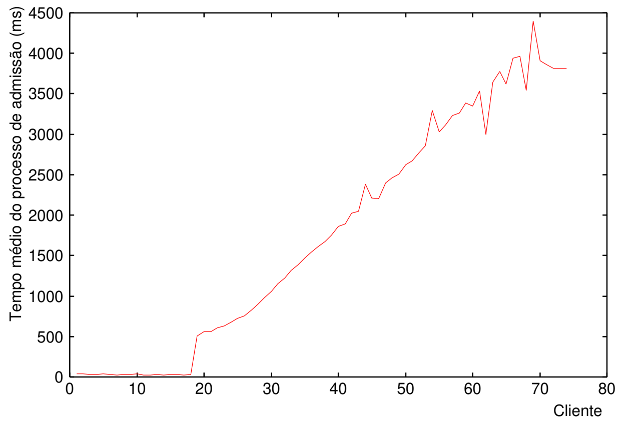

Figura 6.13: Tempo médio de admissão - Experimentos 1 ao 6

A Figura 6.13 apresenta o tempo médio gasto no processo de admissão ($\overline{t_{adm}}$) para o i-ésimo cliente a ser admitido durante os experimentos 1 ao 6. Este tempo inclui o tempo de geração da lista de requisições do i-ésimo cliente, o tempo de geração da lista de eventos com i clientes ($i - 1$ clientes já admitidos pelo servidor mais o novo candidato) e o tempo gasto na simulação desta lista.

O valor de $\overline{t_{adm}}$ é influenciado pela quantidade de clientes já admitidos. Quanto maior o número de clientes, maior é a quantidade de eventos a serem simulados. Cada evento corresponde a um bloco de dados de um objeto. A quantidade de blocos de dados de cada filme pode ser visualizada na Tabela 6.2. Através da Figura

6.13 é possível observar que o tempo gasto no processo de admissão aumenta a uma taxa aproximadamente constante de 80ms/cliente.

Outro aspecto que influencia no valor de $\overline{t_{adm}}$ é a taxa de chegada entre os clientes. Nos experimentos realizados a taxa de chegada dos clientes foi de 11,11 clientes/minuto (1 cliente a cada 5,4 segundos), logo todos os clientes admitidos ainda estavam no início da exibição do vídeo durante o processo de admissão de todos os clientes disparados (os clientes chegam em média durante os 9 minutos desde a admissão do primeiro cliente). Caso o intervalo entre as chegadas dos clientes fosse maior, poder-se-ia observar uma redução no número de eventos simulados, pois os clientes admitidos já teriam consumido uma maior quantidade de blocos de dados e, com isto, o número de eventos originados por estes clientes seria menor (apenas a quantidade de blocos restante do ponto atual até o final do vídeo).

Como um dos objetivos deste trabalho foi a implementação de um controle detalhado do sistema para permitir o estudo e a coleta de diversas medidas, não houve a preocupação com o custo computacional do algoritmo de controle de admissão proposto. No entanto, como o tempo de execução do algoritmo é proporcional ao número de eventos a serem simulados, pode-se utilizar técnicas para a redução de eventos, como por exemplo, a simulação de blocos de dados maiores.

A Figura 6.14 apresenta o tempo médio de espera para o processo de admissão de acordo com o número de clientes na fila de espera durante os experimentos 1 ao 6. Este valor também é influenciado pelo intervalo entre as chegadas dos clientes e pela quantidade de eventos simulados. Cada cliente na fila espera o tempo do processo de admissão de todos os clientes na sua frente. Quanto maior o número de clientes, maior é o número de eventos simulados e, conseqüentemente, maior é o tempo de espera.

Uma alternativa que poderia minimizar o tempo de espera seria a simulação simultânea de um grupo de novos candidatos (em *batch*). Alguns problemas seriam: (1) o tamanho ideal do grupo de novos clientes (evitando grupos muito grandes) e (2) o critério de admissão (caso algum parâmetro de QoS seja violado, quais clientes devem ser rejeitados e quais clientes podem ser admitidos).

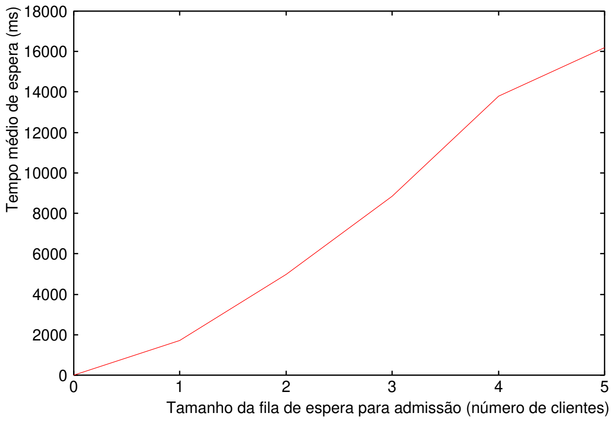

Figura 6.14: Tempo médio de espera - Experimentos 1 ao 6

A Tabela 6.5 apresenta o resultado dos experimentos 7 ao 12. Nestes experimentos o *playout buffer* de cada cliente foi alterado para 20 posições. O tamanho do *buffer* no servidor foi mantido em 5 posições para cada cliente devido à quantidade mínima de memória necessária a ser alocada para um dado número de clientes em cada nó de armazenamento do servidor. Para um bloco de dados de 128KB e o tamanho do *buffer* do servidor de 20 posições para cada cliente, se fossem admitidos 85 clientes, a quantidade de memória (RAM) mínima, apenas para os *lookahead buffers*, deveria ser de 212.5MB. No entanto, um dos nós de armazenamento usado possui 256MB.

Como pode ser observado na Tabela 6.5, o número de clientes admitidos aumentou e a QoS foi garantida para todos os clientes. Através dos logs, observou-se que alguns clientes tiveram falhas, mas dentro dos níveis estabelecidos de QoS (6 clientes no Experimento 7 e 3 clientes no Experimento 8 tiveram 1 *hiccup* durante a sua execução).

A Figura 6.15 apresenta o tempo médio gasto no processo de admissão ($\overline{t_{adm}}$) para o i-ésimo cliente a ser admitido e a Figura 6.16 apresenta o tempo médio de espera para o processo de admissão durante os experimentos 7 ao 12.

De modo a verificar o número de clientes suportados pelo servidor, foi disparado mais um experimento (Experimento 13), desta vez sem controle de admissão, com 90 clientes, cada um com um *playout buffer* de 20 posições. Após a finalização dos

Experimento	**Número de Clientes**			**Intervalo Máximo Vazio**
	Disparados	**Admitidos**	**QoS não garantida**	
7	110	75	0	-
8	100	85	0	-
9	100	78	0	-
10	100	82	0	-
11	100	80	0	-
12	100	83	0	-

Tabela 6.5: Resultado dos experimentos 7 ao 12

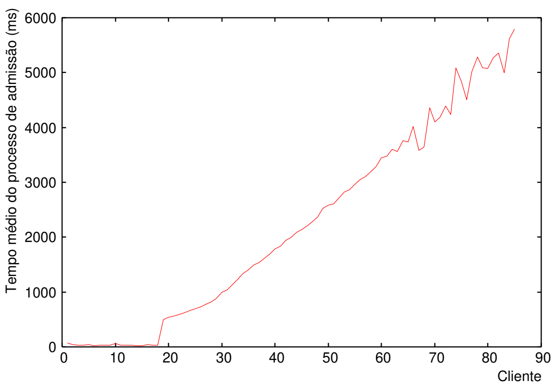

Figura 6.15: Tempo médio de admissão - Experimentos 7 ao 12

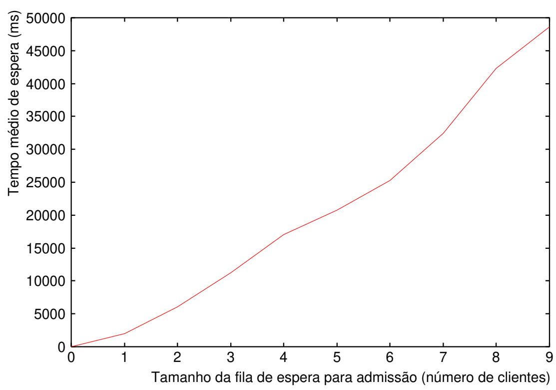

Figura 6.16: Tempo médio de espera - Experimentos 7 ao 12

clientes, verificou-se que ambos os requisitos de QoS, número de *hiccups* e intervalo máximo vazio do *playout buffer*, foram violados nos 90 clientes, indicando um bom funcionamento do controle de admissão nos testes anteriores.

Durante a realização dos experimentos, verificou-se que a carga nas máquinas onde o processo *Storage Server* era executado atingia valores altos e prejudicava o desempenho do servidor, sendo que o responsável pela maior parte desta carga era o sistema operacional. Com isto, observou-se uma grande variabilidade nos tempos de serviço dos discos, conforme ilustrado nas Figuras 6.17, 6.18 e 6.19. Uma das soluções encontradas para tentar minimizar a carga foi desabilitar o *swap* e limitar o número de *threads* em paralelo para no máximo 10 *threads* por disco.

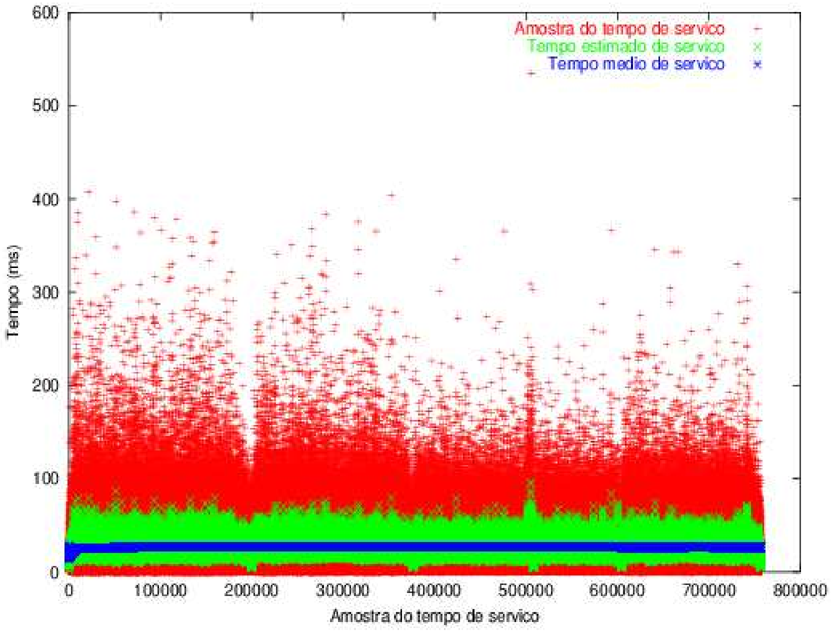

Figura 6.17: Amostras e médias do tempo de serviço do Disco 1

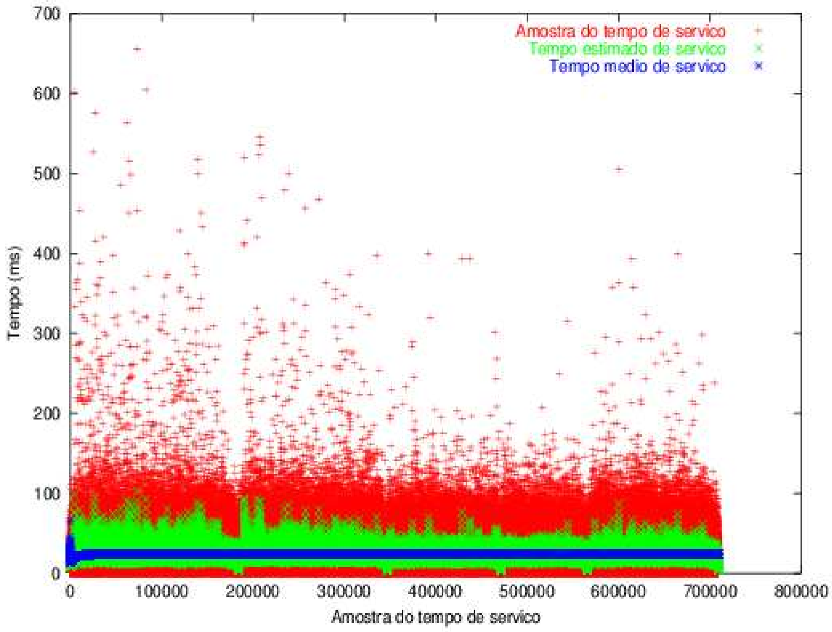

Figura 6.18: Amostras e médias do tempo de serviço do Disco 2

Conforme descrito no capítulo 5, de modo a obter a média do tempo de serviço dos discos ao longo do tempo para, por exemplo, utilizar no controle de admissão, durante a execução do servidor é estimado dinamicamente o tempo de serviço do

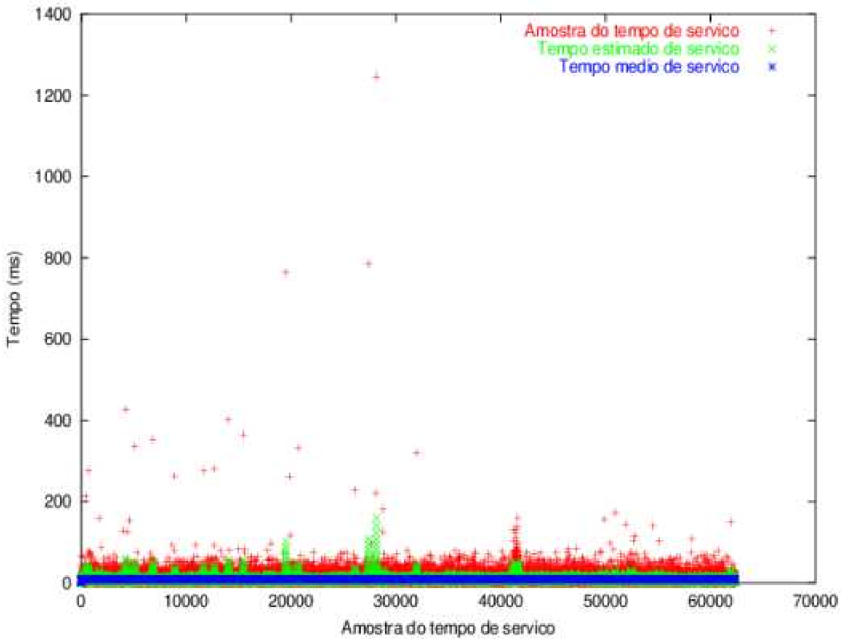

Figura 6.19: Amostras e médias do tempo de serviço do Disco 3

disco i ($\overline{t_{serv\_disco_i}}$) através da equação 5.1:

$$tempo\_medio = (1 - \tau) \times tempo\_medio + \tau \times amostra$$

A seguir são apresentadas as distribuições obtidas tanto das amostras de $\overline{t_{serv\_disco_i}}$, quanto das amostras do tempo de serviço.

As Figuras 6.20, 6.21 e 6.22 apresentam as distribuições do tempo de serviço de cada disco obtidas a partir de algumas amostras de $\overline{t_{serv\_disco_i}}$, independente do número de *threads* ativas. Tais amostras foram coletadas ao longo da execução dos experimentos 1 ao 13. O valor esperado dos tempos estimados de serviço foram 20,19ms, 19,47ms e 8,25 ms, dos discos 1, 2 e 3, respectivamente. Como pode ser observado, a carga do sistema é limitada pelos disco 1 e 2. O ideal em servidores multimídia é o uso de discos com mesma capacidade (ou bem próxima) para melhor utilização de todos os recursos do sistema.

As Figuras 6.23, 6.24 e 6.25 apresentam as distribuições do tempo de serviço de cada disco obtidas a partir das amostras dos tempos de serviço, independente do número de *threads* ativas. Para o Disco 1 a média foi de 26,86ms, para o Disco 2 foi de 24,27ms e para o Disco 3 foi de 9,09ms.

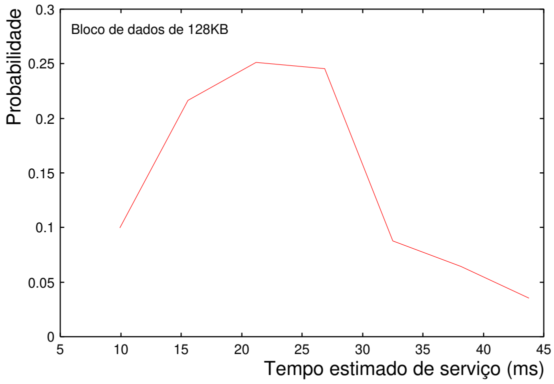

Figura 6.20: Distribuição do tempo estimado de serviço do Disco 1

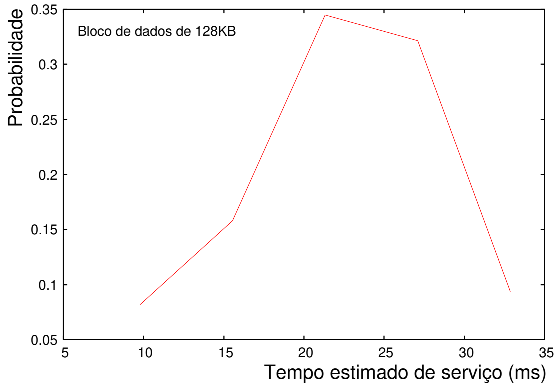

Figura 6.21: Distribuição do tempo estimado de serviço do Disco 2

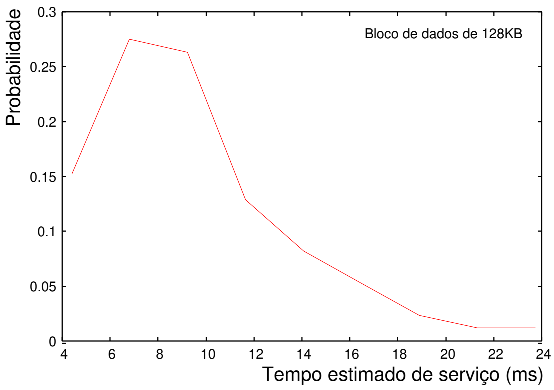

Figura 6.22: Distribuição do tempo estimado de serviço do Disco 3

Em alguns experimentos, observou-se que o controle de admissão admitia mais clientes do que o número de clientes que poderiam ser atendidos sem problemas de QoS. Isto deve-se ao fato da variabilidade dos tempos de serviço, mas também do intervalo entre chegada de clientes. Como todos os clientes chegavam em um curto intervalo de tempo, aproximadamente 9 minutos, as medidas dos discos ainda não refletiam a carga total já admitida durante as simulações nos processos de admissão dos novos candidatos, fazendo com que o controle fosse mais otimista e atendesse um maior número de usuários. Entretanto, o inverso também ocorreu devido a alguns valores maiores do tempo de serviço, fazendo com que o controle fosse mais pessimista e admitisse um menor número de clientes.

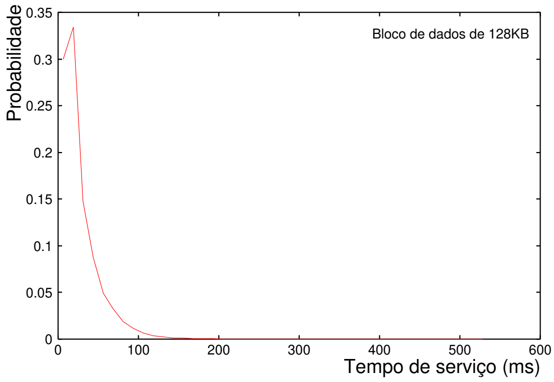

Figura 6.23: Distribuição do tempo de serviço do Disco 1

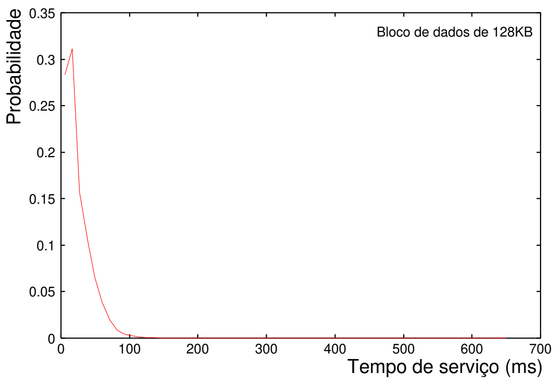

Figura 6.24: Distribuição do tempo de serviço do Disco 2

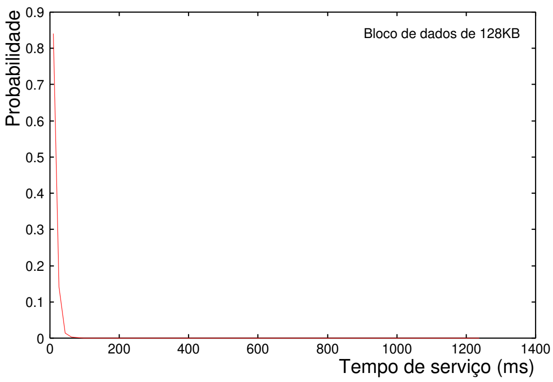

Figura 6.25: Distribuição do tempo de serviço do Disco 3

Além das medidas obtidas dos discos, outro fator que influencia na admissão ou não dos candidatos é o padrão de acesso nos discos dos clientes ativos durante a admissão do novo usuário. Durante alguns experimentos, observou-se que mesmo quando um cliente i não fosse admitido, o seu subseqüente (cliente $i+1$) poderia ser aceito.

Este padrão de acesso aos discos também pode influenciar de outra maneira. Caso um cliente acesse o mesmo vídeo requisitado por outro cliente e o intervalo de chegada entre eles seja pequeno, é possível que os pedidos do segundo cliente não façam acesso aos discos, já que o sistema operacional possui uma *cache* dos dados lidos recentemente, e com isto, seja reduzido o tempo de serviço. No entanto esta característica não é levada em consideração no processo de admissão.

6.2.4 Comparando com o controle de admissão da versão original

Se fosse executado o controle de admissão simples e sem o gerenciamento de *buffers*, para uma taxa de consumo de 1,5Mbits/segundo por cliente e considerando a capacidade total do servidor disponível para os fluxos de tempo real, três casos poderiam ser considerados:

1. controle através dos tempos estimados, como a menor taxa de leitura (Disco 1) foi de 49,53 blocos/segundo obtida durante os experimentos 1 ao 13, a taxa total de leitura seria de 148,59 blocos/segundo e o número máximo de clientes admitidos seria 99;

2. controle através dos valores obtidos pela ferramenta *hdparm*, onde o pior tempo de serviço foi de 15,79ms, ou seja, uma taxa de leitura de 63,33 blocos/segundo, sendo a taxa total de leitura de 189,99 blocos/segundo e 126 o número máximo de clientes que poderia ser admitido pelo servidor;

3. controle através dos tempos médios obtidos pelas amostras do tempo de serviço, onde o número de clientes admitidos seria 74.

Como pode ser observado, considerando o controle da versão original, o número de clientes admitidos poderia ser superior ao número de clientes obtido com o novo controle de admissão. No entanto, através dos experimentos, pode-se observar que não haveria garantia de QoS caso fosse utilizado este controle simples para a admissão de novos usuários, mesmo com o uso dos *buffers* no servidor. A reserva de recursos na versão original é feita utilizando a taxa de consumo fornecida por cada cliente. Caso a reserva fosse feita considerando a taxa de consumo igual a taxa de pico de cada cliente, a QoS poderia ser garantida, mas os recursos do sistema seriam subutilizados.

6.2.5 Comparando com o modelo de simulação

Com a finalidade de comparar os resultados obtidos nos testes com os resultados obtidos do modelo, foram realizadas simulações do modelo usando os mesmos parâmetros coletados durante os experimentos. Para isto foram considerados os tempos estimados de serviço de cada disco ($t_{serv\_disco_1} = 20,19$ ms, $t_{serv\_disco_2} = 19,47$ ms, $t_{serv\_disco_3} = 8,25$ ms) e o tamanho do *buffer* no servidor de cinco posições. Simulando o modelo na ferramenta Tangram-II, com o tamanho do *playout buffer* no objeto Novo Cliente de duas posições (equivalente aos experimentos 1 ao 6), o número de clientes admitidos poderia ser maior ou igual a 111 clientes (considerando os mesmos parâmetros de QoS utilizados nos testes), enquanto que nos experimentos, o número máximo de usuários admitidos foi 73. Com 111 clientes utilizados na simulação, o objeto Novo Cliente teve 3,14% de *hiccups* e o intervalo máximo que o *playout buffer* ficou vazio foi de 677,11ms.

Para verificar a sensibilidade do modelo, a variância do tempo de serviço de cada disco foi alterada de modo que o coeficiente de variação fosse igual a 3 (sem alterar as médias utilizadas). Além disto, a distribuição usada para modelar cada disco também foi alterada para Lognormal. Após as simulações verificou-se que com estes novos parâmetros, o número de clientes admitidos reduziu para 101 clientes. Com 101 clientes utilizados na simulação, o objeto Novo Cliente teve 7,08% de *hiccups* e o intervalo máximo que o *playout buffer* ficou vazio foi de 724,91ms.

A diferença entre os resultados obtidos na simulação e nos experimentos realizados deve-se ao fato da simplificação feita para a modelagem do sistema. Enquanto que na emulação realizada pelo servidor são analisados todos os clientes para verificar se a QoS é garantida, no modelo esta verificação é feita analisando apenas um cliente típico (objeto Novo Cliente). Ou seja, na emulação efetuada pelo controle de admissão a condição para que o novo candidato não seja admitido é de que pelo menos um cliente tenha a sua QoS violada.

Seja f a probabilidade de pelo menos 1 cliente em N clientes ter a QoS violada e p a probabilidade de um cliente escolhido aleatoriamente entre os N clientes ter a sua QoS violada. Então, $f = 1 - (1 - p)^N$.

Para exemplificar, suponha $p = 0.07$. A Figura 6.26 mostra o gráfico fXN para este valor de p. É possível verificar que para uma larga faixa de valores de N, a probabilidade de pelo menos 1 dos N clientes apresentar falha é alta, mesmo sendo baixa a probabilidade de um cliente aleatório ter a sua QoS violada.

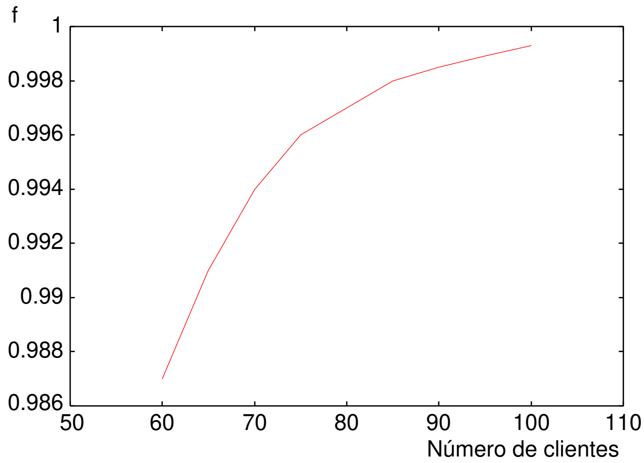

Figura 6.26: fXN

6.2.6 Resumo dos resultados

A Tabela 6.6 apresenta o resumo dos resultados dos experimentos realizados no protótipo, do controle de admissão simples da versão original e das simulações do modelo utilizando os tempos de serviço obtidos durante os testes no servidor.

	Número de Clientes	
	Admitidos	**QoS não garantida**
Controle de admissão implementado (controle fino)		
Experimento 1 ao 6	68 - 73	0 - 3
Experimento 7 ao 12	75 - 85	0
Experimento 13	90	90
Controle de admissão simples (utilizando tempos coletados)		
tempos estimados	99	sem garantia
tempos médios	74	sem garantia
tempos coletados com *hdparm*	126	sem garantia
Simulações do modelo (utilizando tempos estimados)		
Dist. Normal	111	0
Dist. Lognormal	101	0

Tabela 6.6: Resumo dos resultados

Capítulo 7

Aplicações

O servidor desenvolvido neste trabalho será utilizado pelo curso de Tecnólogo em Informática do consórcio Centro de Educação a Distância do Estado do Rio de Janeiro (CEDERJ). O CEDERJ [8] é uma parceria da Secretaria do Estado de Ciência e Tecnologia, da Universidade Estadual do Rio de Janeiro (UERJ), da Universidade Estadual do Norte Fluminense (UENF), da Universidade Federal do Rio de Janeiro (UFRJ), da Universidade Federal Fluminense (UFF), da Universidade Federal Rural do Rio de Janeiro (UFRRJ), da Universidade do Rio de Janeiro (UNIRIO) e das prefeituras municipais com o objetivo de proporcionar ao aluno do ensino médio, condições de realizar um curso de graduação em uma universidade pública e de qualidade, sem que necessite para isso se deslocar de sua cidade.

Todos os cursos do CEDERJ oferecem aos alunos que não dispõem de recursos computacionais em casa as condições para sucesso do ensino/aprendizagem, com material didático (impresso e na *web*), recursos de informática, laboratórios e biblioteca nos pólos regionais (ilustrados na Figura 7.1).

Os cursos abrangem diversas áreas de conhecimento. A área de informática estará utilizando o servidor para armazenamento e distribuição dos vídeos gravados dos seus cursos. Para isto foi desenvolvido pelo grupo de pesquisa LAND [28] um novo cliente de vídeo, chamado *riommclient*, com suporte para sincronização de *slides* com vídeo.

Figura 7.1: Pólos regionais [8]

Cada aula é filmada e digitalizada. Logo após, é criado um arquivo com informações sobre a sincronização com os *slides* disponibilizados pelo professor. Este arquivo é armazenado no servidor juntamente com o vídeo.

O cliente *riommclient*, durante a requisição do vídeo para o servidor, identifica se existem *slides* associados ao vídeo selecionado e o formato destes *slides*, que pode ser *html* ou *obj*. Para a visualização do conteúdo no formato *html*, é utilizado o navegador Netscape, e para o formato *obj*, é utilizada a ferramenta Tgif [9]. No *riommclient*, o aplicativo usado para exibição do vídeo é o *mplayer* [18].

Com a utilização do arquivo contendo as informações sobre a sincronização, o cliente *riommclient* envia comandos para o visualizador dos *slides* durante toda a exibição do vídeo de modo a atualizar os *slides* de acordo com a seqüência de vídeo sendo visualizada. Além disto, o cliente *riommclient* fornece uma interface gráfica, desenvolvida na linguagem JAVA, para interação com o aluno. Através desta interface, o aluno pode avançar, pausar e retroceder a exibição do vídeo, entre outros comandos. Sendo que qualquer um destes comandos também altera a exibição dos *slides*, fazendo com que os dois conteúdos (vídeo e *slides*) estejam sempre sincronizados.

A Figura 7.2 mostra a interface do cliente *riommclient*, a ferramenta Tgif e o aplicativo *mplayer* durante uma demonstração disponibilizada através do servidor. E a Figura 7.3 mostra o mesmo ambiente em uma aula do curso de Tecnólogo em

Informática do CEDERJ.

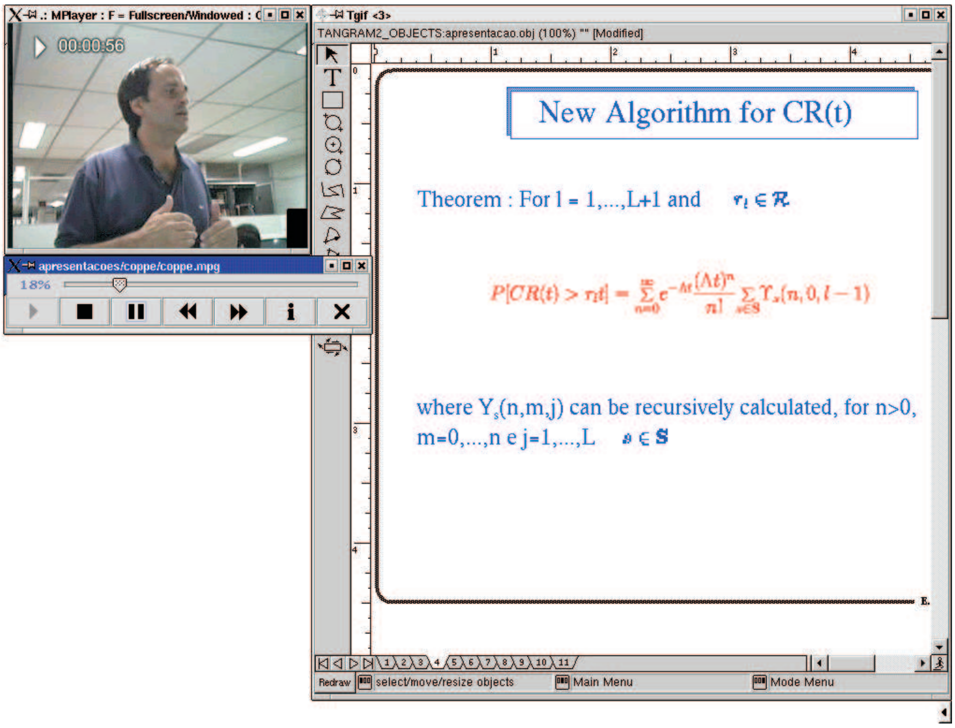

Figura 7.2: Ambiente disponível

Figura 7.3: Aula do curso de Tecnólogo em Informática do CEDERJ

Capítulo 8

Conclusões

Devido às características das aplicações de mídia contínua e às limitações dos recursos utilizados por um servidor multimídia, é necessário que se faça um controle rigoroso dos recursos de forma que os clientes possam ter a sua QoS satisfeita. A utilização dos recursos do servidor deve ser feita de forma eficiente, para atender ao maior número possível de usuários, e obedecer os requisitos estreitos de tempo das aplicações de mídia contínua, garantindo a QoS para todos os usuários. Esta é a principal diferença entre o uso de servidores destinados às aplicações de mídia contínua e o uso de servidores convencionais, tais como servidores de banco de dados, já que estes não levam em consideração as restrições temporais deste tipo de aplicação.

Neste trabalho foi proposto e implementado um conjunto de mecanismos que visam à melhoria do desempenho do servidor multimídia e à garantia de QoS para os usuários do sistema. Tais mecanismos foram incorporados no servidor RIO [46], que foi desenvolvido inicialmente na UCLA.

Um dos recursos criados foi o gerenciamento de *buffers* no servidor e no cliente, cujo principal objetivo foi compensar as flutuações da carga dos discos e da rede e aumentar o número de clientes atendidos. Para a garantia de QoS, foi desenvolvido um controle de admissão, cuja idéia principal é fazer uma previsão em forma de simulação para verificar a possibilidade de admissão do novo cliente, considerando a carga dos clientes já admitidos, juntamente com a carga deste novo candidato. Esta simulação, que prevê o funcionamento do servidor, utiliza: (a) as informações *a priori* dos vídeos armazenados para obter a lista de requisições de cada cliente de acordo com o vídeo selecionado; (b) os estados dos *buffers* (no servidor e no cliente) de cada cliente do sistema; (c) informações da rede e (d) as medidas em tempo real extraídas do servidor. Através da simulação, é possível verificar se o novo cliente pode ser admitido no sistema sem violar a QoS oferecida para nenhum cliente. Os parâmetros de QoS analisados foram o número máximo de congelamentos (*hiccups*) permitidos e o intervalo máximo vazio do *playout buffer* de cada cliente durante a sua execução. Tais parâmetros de QoS são configurados na inicialização do servidor.

Vários experimentos foram realizados de modo a verificar o desempenho do servidor com os novos mecanismos implementados. Os resultados obtidos mostraram que o aumento tanto no tamanho do *buffer* do servidor, quanto no *buffer* do cliente, ocasiona o aumento do número de clientes admitidos pelo servidor. E que o controle de admissão garantiu, na maioria dos experimentos, os requisitos de QoS para os clientes. Observou-se, no entanto, uma grande variabilidade nas medidas extraídas do servidor que podem influenciar no controle de admissão.

Em paralelo, foi desenvolvido um modelo simplificado, na ferramenta Tangram-II, que representa o funcionamento do servidor e o gerenciamento de *buffers*. Este modelo serviu de guia para a implementação e ajustes dos mecanismos supracitados. Através de simulações, observou-se o impacto do retardo da rede, entre o servidor e o novo candidato à admissão, o ganho obtido com a utilização de *buffers* no servidor para cada cliente e o impacto da carga gerada por todos os usuários, de acordo com a limitação das taxas dos discos.

Embora seja óbvio que se deve utilizar discos com mesma capacidade para que o sistema obtenha uma melhor utilização, nos experimentos os discos possuiam capacidades distintas por serem estes os recursos disponíveis no laboratório. Mesmo assim pode ser verificado que a técnica do RIO é robusta, apresentando resultados satisfatórios quanto ao número de clientes admitidos mesmo com a diferença no tempo de resposta dos discos utilizados.

Como resultado final, foram feitas diversas melhorias no servidor RIO, implementando novas funcionalidades, e um ambiente para estudo e coleta de dados. Diversas medidas podem ser extraídas auxiliando na configuração do sistema.

É importante enfatizar que este trabalho não foi apenas experimental, mas também serviu de base para um grande empreendimento no Estado do Rio de Janeiro. O servidor desenvolvido neste trabalho será utilizado para ensino à distância no curso de Tecnólogo em Informática do CEDERJ.

8.1 Trabalhos Futuros

Algumas linhas de atuação podem estender o trabalho aqui apresentado, como por exemplo:

- o compartilhamento dos *buffers* no servidor para que seja possível a redução de pedidos aos discos e o atendimento de um maior número de usuários;
- estudar o parâmetro de estimativa do tempo de serviço dos discos (parâmetro τ da equação 5.1) e o seu efeito no cálculo das medidas obtidas em tempo real do servidor;
- o uso de *proxy* e de técnicas de compartilhamento de recursos de comunicação para redução do tráfego na rede;
- o uso de mecanismos para alterar a qualidade do vídeo de acordo com os recursos disponíveis;
- estudar a distribuição que melhor se adeqüe ao tempo de serviço dos discos;
- explorar outros aspectos no modelo criado, como por exemplo, alternativas para compensar as diferenças nas taxas de transferências dos discos (uso de replicação, alocação baseada com pesos de acordo com a taxa de transferência disponível, etc).

Referências Bibliográficas

[1] Elevator I/O scheduler (Linux). URL http://tux.u-strasbg.fr/j13/features-2.3.html.

[2] Berson, S., Muntz, R. R., e Wong, W. R. Randomized Data Allocation for Real-time Disk I/O. In *First IEEE Computer Society International Conference (COMPCON'96)* (1996), pp. 286–290.

[3] Bianchi, G., Capone, A., e Petrioli, C. Throughput Analysis of End-to-End Measurement-Based Admission Control in IP. In *Proceedings of IEEE INFOCOM* (2000), pp. 1461–1470.

[4] Bovet, D. P., e Cesati, M. *Understanding the Linux Kernel.* O'Reilly & Associates, Inc., 2001.

[5] Breslau, L., Jamin, S., e Shenker, S. Comments on the Performance of Measurement-Based Admission Control Algorithms. In *Proceedings of IEEE INFOCOM* (2000), pp. 1233–1242.

[6] Breslau, L., Knightly, E. W., Shenker, S., Stoica, I., e Zhang, H. Endpoint Admission Control: Architectural Issues and Performance. In *Proceedings of SIGCOMM* (2000), pp. 57–69.

[7] Casetti, C., Kurose, J., e Towsley, D. An adaptative algorithm for measurement-based admission control in integrated services packet networks. In *Computer Communications 23* (2000), pp. 1363–1376.

[8] CEDERJ. Centro de Educação de Ensino Superior a Distância do Estado do Rio de Janeiro. URL http://www.cederj.rj.gov.br.

[9] Cheng, W. C. Tangram graphic interface facility. URL http://bourbon.cs.umd.edu:8001/tgif/.

[10] chi Feng, W., Krishnaswami, B., e Prabhudev, A. Proactive Buffer Management for the Streamed Delivery of Stored Video. In *ACM Multimedia'98* (1998), pp. 285–290.

[11] cker Chiueh, T., e Vernick, M. An Empirical Study of Admission Control Strategies in Video Servers. In *Proceedings of International Conference on Parallel Processing* (1998), pp. 313–320.

[12] Côté, G., Erol, B., Gallant, M., e Kossentini, F. H.263+: Video Coding at Low Bit Rates. In *IEEE Transactions on Circuits and Systems for Video Technology* (1998), vol. 8, pp. 849–866.

[13] de Souza e Silva, E., Gail, H. R., Golubchik, L., e Lui, J. C. S. Analytical models for mixed workload multimedia storage servers. In *Performance Evaluation 36-37* (1999), pp. 185–211.

[14] de Souza e Siva, E., Leão, R. M. M., Ribeiro-Neto, B., e Campos, S. Performance Issues of Multimedia Applications. In *Performance Evaluation of Complex Systems: Techniques and Tools* (September 2002). Editores: Maria Carla Calzarossa e Salvatore Tucci, LNCS Volume 2459.

[15] Fabbrocino, F., Santos, J., e Muntz, R. An Implicitly Scalable, Fully Interactive Multimedia Storage Server. In *Second International Workshop on Distributed Interactive Simulation and Real Time Applications* (1998), pp. 92–101.

[16] Ferrari, D., e Verma, D. C. A Scheme for Real-Time Channel Establishment in Wide-Area Networks, April 1990.

[17] Gemmell, D. J., Vin, H., Kandlur, D. D., e Rangan, P. V. Multimedia Storage Servers: A Tutorial. In *IEEE Computer* (1995), pp. 40–49.

[18] Gereoffy, A. mplayer - Movie Player for Linux. URL http://www.mplayerhq.hu/homepage/.

[19] Ghandeharizadeh, S., e Muntz, R. Design and Implementation of Scalable Continuous Media Servers. In *Parallel Computing, Special Issues on Applicatons, Parallel Data Servers and Applications* (1998), pp. 91–122.

[20] Jepson, W., Liggett, R., e Friedman, S. An Environment for Real-time Urban Visualization. In *Proceedings of the Symposium on Interactive 3D Graphics* (1995).

[21] Jian, X., e Mohapatra, P. Efficient Admission Control Algorithms for Multimedia Servers. In *Multimedia Systems* (1999), vol. 7, pp. 294–304.

[22] Kim, I.-H., Kim, J.-W., seung Won Lee, e Chung, K.-D. Measurement-Based Adaptative Statistical Admission Control Scheme for Video-on-Demand Servers. In *Proceedings of 15th IEEE International Conference on Information Networking* (2001), pp. 471–478.

[23] Knightly, E. W. H-BIND: A New Approach to Providing Statistical Performance Guarantees to VBR Traffic. In *Proceeding of IEEE INFOCOM'96* (1996), pp. 1091–1099.

[24] Knightly, E. W. On the Accuracy of Admission Control Tests. In *Proceedings of IEEE ICNP'97* (1997), pp. 125–133.

[25] Knightly, E. W., Wrege, D. E., Liebeherr, J., e Zhang, H. Fundamental Limits and Tradeoffs of Providing Deterministic Guarantees to VBR Video Traffic. In *Proceedings of ACM SIGMETRICS* (1995), pp. 98–107.

[26] Kurose, J. F., e Ross, K. W. *Computer Networking A Top-Down Approach Featuring the Internet.* Addison Wesley, 2001.

[27] Kwon, J. B., e Yeom, H. Y. An Admission Control Scheme for Continuous Media Servers Using Caching. In *Proceedings of the IEEE International Performance, Computing and Communications Conference (IPCCC'00)* (2000), pp. 456–462.

[28] LAND. *Laboratory for modeling, analysis and development of networks and computing systems.* URL http://www.land.ufrj.br, April 2001.

[29] LEE, J. Y. B. Buffer Management and Dimensioning for a Pull-Based Parallel Video Server. In *IEEE Transactions on Circuits and Systems for Video Technology'01* (2001), vol. 11, pp. 485–496.

[30] LEE, S.-H., WHANG, K.-Y., MOON, Y.-S., E SONG, I.-Y. Dynamic Buffer Allocation in Video-on-Demand Systems. In *Proceedings of ACM SIGMOD'01* (2001), pp. 343–354.

[31] ÉLTETÖ, T., E MOLNÁR, S. On the Distribution of Round-trip Delays in TCP/IP Networks. In *Proceedings of the 24th Conference on Local Computer Networks LCN'99* (1999), pp. 172–181.

[32] LUI, J. C. S., E WANG, X. Q. An Admission Control Algorithm for Providing Quality-of-Service Guarantee for Individual Connection in video-on-Demand System. In *Proceedings of Fifth IEEE Symposium on Computers and Communications (ISCC)* (2000), pp. 456 –461.

[33] MAKAROFF, D., NEUFELD, G., E HUTCHINSON, N. An Evaluation of VBR Disk Admission Algorithms for Continuous Media File Servers. In *Proceedings of the 5th ACM Multimedia Conference* (1997), pp. 143–154.

[34] MAPPED. Mecanismos de Adaptação e Controle para Recuperação e Transmissão Multimídia com Aplicação ao Centro de Educação Superior a Distância do Estado do Rio de Janeiro. Projeto aprovado no Edital 11/01 do CNPq.

[35] MPEGTV. mtvp - MPEG Movie Player and VCP Player for Linux. URL http://www.mpegtv.com.

[36] MUNTZ, R., SANTOS, J., E BERSON, S. A Parallel Disk Storage System for Realtime Multimedia Applicatons. In *International Journal of Intelligent Systems, Special Issue on Multimedia Computing System* (1998), vol. 13, pp. 1137–1174.

[37] MUNTZ, R., SANTOS, J., E FABBROCINO, F. Design of a Fault Tolerant Real-time Storage System for Multimedia Applications. In *3rd IEEE International Computer Performance and Dependability Symposium (IPDS'98)* (1998), pp. 174–183.

[38] Pancha, P., e Zarki, M. E. MPEG Coding for Variable Bit Rate Video Transmission. In *IEEE Communications Magazine* (1994), pp. 54–65.

[39] Reddy, A. L. N., e Wijayaratne, R. Techniques for improving the throughput of VBR streams. In *Proceedings of ACM/SPIE Conference on Multimedia Computing and Networking* (1999), vol. 3654.

[40] Reisslein, M., e Ross, K. W. Call Admission for Prerecorded Sources with Packet Loss. In *Proceedings of IEEE Journal on Selected Areas in Communications'97* (1997), pp. 1167–1180.

[41] Sahu, S., Zhang, Z.-L., Kurose, J., e Towsley, D. On the Efficient Retrieval of VBR Video in a Multimedia Server. In *Proceedings of IEEE Conference on Multimedia Computing and Systems (ICMCS)* (1997), pp. 46–53.

[42] Salehi, J. D., Zhang, Z.-L., Kurose, J., e Towsley, D. Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing. In *IEEE/ACM Transactions on Networking* (1998), vol. 6, pp. 397–410.

[43] Santos, J., e Muntz, R. Performance Analysis of the RIO Multimedia Storage System with Heterogeneous Disk Configurations. In *6th ACM International Multimedia Conference (ACM Multimedia 98)* (1998), pp. 303–308.

[44] Santos, J., Muntz, R., e Ribeiro-Neto, B. Comparing Random Data Allocation and Data Striping in Multimedia Storage Servers. In *Proceedings of ACM SIGMETRICS* (2000), pp. 44–55.

[45] Santos, J. R., e Muntz, R. Comparing Random Data Allocation and Data Striping in Multimedia Storage Servers. Relatório Técnico 980038, UCLA Computer Science Departament, 1998.

[46] Santos, J. R. G. *RIO: A Universal Multimedia Storage System Based on Random Data Allocation and Block Replication.* PhD thesis, UCLA, 1998.

[47] Siler, M., e Walrand, J. Measurement-Based Admission Control and Monitoring for Statistical Rate-Latency Guarantees. In *Proceedings of the 38th Conference on Decision & Control* (1999), pp. 4426–4431.

[48] Tan, H., Eager, D., Vernon, M., e Guo, H. Quality of Service Evaluations of Multicast Streaming Protocols. In *Proc. of ACM Sigmetrics* (2002).

[49] Vin, H. M., Goyal, A., Goyal, A., e Goyal, P. An Observation-Based Admission Control Algorithm for Multimedia Servers. In *Proceedings of the First IEEE International Conference on Multimedia Computing and Systems (ICMCS)* (1994), pp. 234–243.

[50] Vin, H. M., Goyal, P., Goyal, A., e Goyal, A. A Statistical Admission Control Algorithm for Multimedia Servers. In *Proceedings of ACM Multimedia* (1994), pp. 33–40.

[51] yu Qiu, J., Cetinkaya, C., Li, C., e Knightly, E. W. Inter-Class Resource Sharing using Statistical Services Envelopes. In *Proceedings of IEEE INFOCOM'99* (1999).

[52] Zhanga, Z.-L., Kurose, J., Salehi, J., e Towsley, D. Smoothing, Statistical Multiplexing and Call Admission Control for Stored Video. In *Proceedings of IEEE Journal on Selected Areas in Communication'97* (1997), vol. 15, pp. 1148–1166.