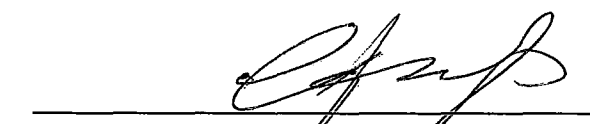
MEMÓRIA COOPERATIVA PARA DISTRIBUIÇÃO DE VÍDEO SOB DEMANDA

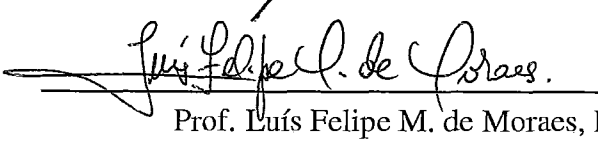
Edison Ishikawa

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

Aprovada por:

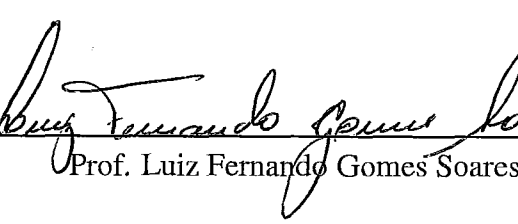


Prof. Claudio Luis de Amorim, Ph.D.

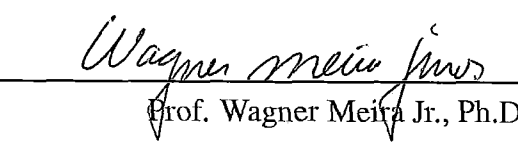


Prof. Luís Felipe M. de Moraes, Ph.D.

Prof. Jauvane Cavalcante de Oliveira, Ph.D.



Prof. Luiz Fernando Gomes Soares, Ph.D.



Prof. Wagner Meira Jr., Ph.D.

RIO DE JANEIRO, RJ-BRASIL

OUTUBRO DE 2003

ISHIKAWA, EDISON

Memória Cooperativa para Distribuição de Vídeo sob Demanda [Rio de Janeiro] 2003

XII, 103 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2003)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1 - Sistemas Distribuídos Multimídia

2 - Vídeo sob Demanda

3 - Gerência de Memória

I. COPPE/UFRJ II. Título (série)

A Neli.
A Helena, minha filha.
A memória de minha mãe
A meu pai
A Deus

Agradecimentos

A minha querida esposa Neli, que sempre me incentivou nesta longa jornada, compreendendo todas as dificuldades para a sua realização, principalmente pelo tempo que deixamos de estar juntos.

A minha filha Helena, que com seu sorriso e sua alegria de viver tornaram esta jornada ainda mais gratificante.

A minha mãe, que Deus a tenha, pelo exemplo de vida e dedicação.

A meu pai, pelo apoio que tive em todas as ocasiões de minha vida.

A toda a minha família e a todos os meus amigos pelo carinho e incentivo,

Ao meu orientador Claudio Amorim, que foi meu guia durante todo este período. Orientador sempre presente, nunca tive dificuldades em reunir-me com ele para tratar de qualquer assunto. Sempre paciente, não só leu e corrigiu meus artigos, como delineou o caminho da minha pesquisa. Sua dedicação ao LCP e a seus alunos é um exemplo a ser seguido.

A todos os colegas e funcionários do LCP e do PESC, em particular ao Leonardo e ao Lauro, companheiros nesta jornada, que sempre me ajudaram quando foi preciso.

A todos os amigos, colegas, alunos e funcionários do IME, em particular do DE/9, pela compreensão e pelo auxílio que me prestaram diante dos problemas de se terminar a Tese junto com as atividades docentes e administrativas.

Ao povo brasileiro, que através do Exército Brasileiro, permitiu que eu tivesse a oportunidade de fazer mais este curso durante minha carreira.

A Deus, a quem tudo devo.

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

MEMÓRIA COOPERATIVA PARA DISTRIBUIÇÃO DE VÍDEO SOB DEMANDA

Edison Ishikawa
Outubro/2003

Orientador: Claudio Luis de Amorim
Programa: Engenharia de Sistemas e Computação

A distribuição de vídeo digital é uma tecnologia recente se comparada com a tecnologia analógica, que já tem mais de um século de existência. Sua distribuição através de uma rede de comunicação de dados, possui um grande apelo, pois este pode ser feito de acordo com a demanda do usuário, sendo este serviço mais conhecido como Vídeo sob Demanda (*Video on Demand-VoD*). O grande obstáculo para a implantação de sistemas VoD é a falta de escalabilidade destes sistemas. Esta tese propõe uma nova forma de tornar escalável a distribuição de VoD. Mais especificamente, explorando a memória, ou dos clientes ou dos proxies, de forma cooperativa. Com este objetivo, projetamos e avaliamos duas novas técnicas para distribuição de vídeo sob demanda. A primeira técnica é baseada num modelo peer-to-peer, denominado Memória Cooperativa Distribuída (MCD), para a distribuição de VoD. A segunda técnica, baseada em uma CDN (Rede de Distribuição de Conteúdo), estende a primeira colapsando os *playout* buffers dos clientes VoD em proxies de vídeo. Devido a isto, o método de gerenciamento da cache do proxy foi denominado Memória Cooperativa Colapsada (MCC). A avaliação de desempenho experimental das duas técnicas mostraram que elas são mais escaláveis que as existentes. Através de detalhadas simulações, verificou-se que a curva de vazão total do sistema na MCD foi bem melhor que os resultados existentes na literatura para uma quantidade semelhante de clientes atendida. Para verificarmos que a MCC atinge melhores desempenhos do que proxies que não usam o conceito de memória cooperativa, usamos simulações detalhadas de execuções que estressam o sistema ao máximo de sua capacidade de trabalho. Nossa comparação mostra que MCC supera significativamente o desempenho de propostas anteriores. Além disso, modelamos o gerenciamento da MCC como um problema de decisão, demonstrando que o problema da gerência da MCC é NP-Completo. Com esta demonstração, também mostramos de onde a MCC extrai o seu desempenho para ultrapassar o desempenho obtido pelos esquemas anteriores.

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

COOPERATIVE MEMORY FOR ON DEMAND VIDEO DISTRIBUTION

Edison Ishikawa
October/2003

Advisor: Claudio Luis de Amorim
Department: Computing and Systems Engineering

Digital Video Distribution is a recent technology when compared with the analogic technology, which has more than a century of developments. Distributing video digitally through a data network has a great appeal. Users can do it on demand and this service is better known as Video-on-Demand (VoD). The challenge to these systems is the lack of scalability. This Thesis presents a new way to scale Video Distribution on demand. More specifically, we explore or the client memory or the proxy memory in a cooperative way. With this objective in mind, we designed and evaluate two new approaches for VoD Distribution. The first approach is based on a peer-to-peer model, called Distributed Cooperative Memory (MCD), for VoD distribution. The second approach, based on a Content Distribution Netowrk (CDN), extends the first approach by collapsing the client's playout buffers in video proxies. Due to this reason, the resulting system is called Collapsed Cooperative Memory (MCC). Experimental performance evaluation of each approach was quite satisfactory. Both approaches showed that they are more scalable than existing systems. Through detailed simulations of both approaches we showed that the system throughput has better performance, when compared with existing results in the literature, for the same number of clients. In order to verify that MCC perform better than proxies that do not use our concept of cooperative memory, we ran detailed simulations that stress the system (near) to the maximum workload. Our results show that MCC outperforms existing solutions. Besides, we modeled the MCC management as a decision problem, showing that this problem is NP-complete. With this proof, we also show from where MCC extract's its performance in order to surpass existing schemes.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A produção e distribuição de vídeo digital são tecnologias recentes se comparadas com a tecnologia analógica, que já tem mais de um século de existência. Atualmente, os sistemas de vídeo estão passando por uma transição para a tecnologia digital. A tecnologia digital é mais robusta, uma vez que a informação não se degrada com o tempo ou com a transmissão. Sua qualidade é maior a um custo menor, além de ser mais fácil de reproduzir e distribuir. Esta tendência à digitalização também é acelerada devido ao fato da tecnologia digital relacionar tecnologias que até recentemente eram estanques, como computadores, vídeo, comunicações, ensino, entretenimento, etc. É o que chamamos convergência. Podemos considerar que o vídeo digital é uma das tecnologias principais para esta convergência tecnológica, e que, inexoravelmente, entraremos na era da transmissão de vídeo digital. Embora nem todos os problemas técnicos para sua ampla utilização estejam resolvidos, o potencial desta tecnologia para a oferta de novos serviços e funcionalidades é quase que ilimitado.

A distribuição de vídeo digital através de uma rede de comunicação de dados, de acordo com a demanda do usuário, é um dos serviços que a convergência digital tornou possível. Este serviço, mais conhecido como Vídeo sob Demanda (*Video on Demand-VoD*), é uma solução eletrônica para combinar as facilidades de um serviço de videolocadora próximo ao usuário a um equipamento de videocassete (VCR), usando para isto uma rede de comunicações de dados de banda larga. Essencialmente, ele é um sistema de videolocadora eletrônico [4] com um enorme potencial comercial em áreas como serviços de entretenimento, ensino à distância, aprendizado interativo, catálogos de produtos, etc., tendo sido considerado uma das tecnologias com capacidade para alavancar o amplo uso das Redes Digitais de Serviços Integrados de Faixa Larga (B-ISDN) [79].

O grande obstáculo para a implantação de sistemas VoD é a falta de escalabilidade destes sistemas. Um vídeo MPEG-1 [33] requer em média uma largura de banda de 1,5

Mbps. Já um filme digitalizado em MPEG-2 [37] com uma qualidade próxima ao padrão NTSC utiliza uma taxa média de transmissão de 8 Mbps, isto sem contar com o advento do HDTV (High Definition TV) que irá requerer taxas de transmissão ainda mais altas. Construir servidores de VoD que suportem milhares de fluxos de vídeo com estas altas taxas de transmissão não é algo trivial e seu custo é muito alto. Além disso, redes de transmissão de dados capazes de transmitir simultaneamente os diversos fluxos de vídeo gerados por este mesmo servidor não estarão disponíveis e em operação a médio prazo.

Vários mecanismos para aumentar a escalabilidade do sistema foram propostos. Desde propostas que aumentam a escalabilidade do servidor através do uso de novos sistemas de arquivos, algoritmos de escalonamento e políticas de compartilhamento do *buffer* do servidor, passando por propostas que minimizam a largura de banda necessária ao serviço, maximizando o uso de protocolos multidestinatários (*multicast*) através de políticas de atendimento de pedidos em lote (*batching*) [1] e do uso de mecanismos como o *piggybacking* [59].

Mais recentemente, foram propostas soluções que exploram tanto a largura de banda do cliente como a sua capacidade de armazenamento. No primeiro caso, enquanto o cliente recebe por um canal o fluxo que pediu, em um outro canal ele recebe e armazena o restante do vídeo que estava sendo transmitido para outro cliente, podendo desta forma liberar o servidor mais rapidamente. Este esquema é conhecido por *patching* [53]. No segundo caso, o fluxo que passa por um cliente não precisa ser descartado, pelo contrário, ele pode ser reaproveitado retransmitindo-o para outro cliente. Este esquema forma longas cadeias de clientes ligados por um mesmo fluxo de vídeo donde o nome *chaining* [89] atribuído a este esquema.

Outra proposta na área é o uso de redes de distribuição de conteúdo (*Content Distribution Network-CDN*). O objetivo de uma CDN é evitar que os enlaces que compõem o *backbone*[1] da rede fiquem congestionados. Isto é possível devido à colocação de *proxies* na borda da CDN, mais especificamente entre a CDN e a rede de acesso. Os *proxies* podem ser considerados servidores secundários próximos aos clientes que procuram substituir o servidor primário armazenando as informações mais acessadas, de forma a fornecer esta informação no lugar do servidor primário. Isto evita o tráfego no *backbone* da rede. Além de evitar o congestionamento no *backbone* da rede, a capacidade de processamento e armazenamento dos *proxies* é somada à capacidade do servidor primário, o que resulta numa maior escalabilidade do sistema. A eficácia da CDN depende

[1] espinha dorsal

da probabilidade do *proxy* atender a requisição do cliente no lugar do servidor primário [11]. O uso de *proxies* para mídias contínuas (vídeo) com as políticas tradicionais de substituição de conteúdo provaram ser ineficientes. Isto se deve ao grande tamanho destas mídias em relação ao tamanho da *cache*. No entanto, técnicas novas foram desenvolvidas para gerenciar a *cache* de *proxies* de vídeo, mostrando a viabilidade de se utilizar uma CDN para oferecer serviços de VoD [24, 80, 88, 101, 10, 67, 60, 16].

Nesta tese, propomos uma nova forma de explorar a memória, ou dos clientes ou de *proxies* ou de ambos, de forma cooperativa. Este novo esquema permite que sistemas VoD atendam um grande número de usuários, mesmo em um contexto de escassez de recursos tanto no servidor como na largura de banda da rede de transmissão. A quantidade de memória que se consegue agregar somando-se às memórias dos *buffers* de todos os clientes do sistema é mais do que suficiente para armazenar o conteúdo dos vídeos mais populares do sistema, transformando-os em uma *cache* cooperativa. Por outro lado, o intenso fluxo de conteúdo entre os clientes pode congestionar a rede de interconexão dos clientes, donde estendemos este método de exploração da memória dos clientes para os *proxies* que ficam na borda da rede de distribuição de conteúdo, na fronteira entre a rede de distribuição e a rede de acesso.

Para quantificar o impacto dessa nova abordagem no desempenho de um sistema de VoD, projetou-se e avaliou-se o modelo *peer-to-peer* e o modelo com *proxies* da memória cooperativa. O primeiro modelo, *peer-to-peer*, denominado Memória Cooperativa Distribuída (MCD) [21, 22, 19, 20] explora os buffers dos clientes em um sistema de VoD cujos clientes estão interconectados à rede de distribuição por enlaces simétricos *full duplex*. A segunda arquitetura, baseada em uma CDN, estende a primeira em uma rede de acesso cujos enlaces com os clientes são assimétricos. Para isso, colapsa parte do *buffer* de cada cliente no ponto de acesso da rede de distribuição de conteúdo, mais especificamente em um *proxy* . Devido a isto, o método de gerenciamento da *cache* do *proxy* foi denominado Memória Cooperativa Colapsada (MCC) [25, 24, 23].

Para verificarmos que memórias cooperativas atingem melhores desempenhos do que os modelos atuais, usamos simulações detalhadas de execução que estressam o sistema VoD ao máximo. Com este objetivo, simulamos ambos os sistemas em seu horário de pico, o comumente conhecido horário nobre da televisão (*prime time*), com mais de 1000 clientes[2] por ponto de acesso, requisitando até 100 vídeos em um espaço de tempo

[2]A tecnologia atual já suporta DSLAM (Digital Subscriber Line Access Multiplexer), que seria o análogo a uma central telefônica para linhas digitais de acesso assimétrico, com mais de 1000 clientes. No contexto do horário nobre, supõe-se que todos os 1000 clientes da DSLAM irão acessar, no intervalo de

correspondente à duração do vídeo (1 hora).

Através das simulações, verificou-se que a MCD possui uma taxa de utilização do servidor VoD semelhante ao *chaining* e, portanto, muito melhor que um sistema *multicast*. A curva da vazão total no sistema também foi bem melhor do que a de *chaining*, que cresce linearmente com o aumento da taxa de chegada de clientes. A curva da MCD além de crescer numa taxa menor, mostrou uma tendência de se estabilizar e inclusive diminuir para uma taxa de chegadas mais alta de clientes. Além disto, um protótipo da MCD, o GloVE [55, 54, 56], foi implementado para funcionar em ambientes corporativos com uma rede local com *backbone* colapsado, em uma topologia conhecida como *fat tree* [9], cujo desempenho comprovou na prática a viabilidade da MCD.

Para compararmos MCC com os esquemas existentes, criamos a MCC com as políticas **LS-**, **LS** e **LS+** (as diferenças destas políticas são explicadas no capítulo 5). As políticas **LS-** e **LS** se aproximam bastante dos esquemas implementados em [60, 16, 67, 10], enquanto que **LS+** representa a nova política que implementamos. Nossa comparação mostra que MCC **LS+** supera o desempenho de **LS-** e de **LS**. Além disso, modelamos o gerenciamento da MCC como um problema de decisão, demonstrando que o problema da gerência da MCC é NP-Completo. Com esta demonstração, também mostramos de onde a MCC tira o seu desempenho para ultrapassar o desempenho obtido pelos esquemas citados anteriormente.

Todos estes resultados confirmam que a utilização da MCD e da MCC tem um grande potencial para melhorar a escalabilidade e o desempenho de sistemas para distribuição de vídeo sob demanda.

1.1 Contribuições da Pesquisa

O uso das memórias dos clientes VoD como uma área coletiva de memória a ser explorada em proveito de um Sistema de VoD é uma abordagem diferente do problema, e que de acordo com o levantamento bibliográfico realizado nunca foi estudado ou proposto anteriormente. Além disso, MCD é um esquema de gerenciamento *peer-to-peer* pioneiro em se tratando de distribuição de vídeo. A extensão da MCD para gerenciar *proxies*, a MCC, permitiu estabelecer um método de gerenciar a *cache* mais eficiente do que apenas virtualizar o buffer do cliente no *proxy* sem uma visão global do mesmo. Sendo assim, as contribuições mais importantes desta pesquisa podem ser resumidas nos seguintes pontos:

tempo de uma hora, um dos cem filmes disponíveis.

- Propor e avaliar um novo mecanismo para tornar escalável o serviço de VoD que explora o uso coletivo da memória dos clientes, em uma arquitetura *peer-to-peer*, aumentando a escalabilidade do sistema. Nossa contribuição especifica neste ponto é a MCD.

- Propor e avaliar um mecanismo de gerência da *cache* de *proxies* de vídeo, aumentando a escalabilidade e desempenho do sistema. Nossa contribuição específica neste ponto é a MCC.

1.2 Organização da Tese

O restante desta tese está organizado da seguinte forma. No capítulo 2 apresenta-se os conhecimentos básicos que dão suporte ao trabalho que se está propondo. São apresentados os fundamentos de um sistema de VoD, assim como as técnicas mais importantes relacionados com a exploração dos recursos do cliente em proveito do sistema e o uso de redes de distribuição de conteúdo.

Em seguida, no capítulo 3 propomos e avaliamos a Memória Cooperativa Distribuída (MCD), enquanto no capítulo 4 propomos a Memória Cooperativa Colapsada (MCC) e no capítulo 5 avaliamos o seu desempenho.

Os trabalhos relacionados são discutidos no capítulo 6

Por fim, no capítulo 7 apresentamos as principais conclusões da nossa pesquisa.

Capítulo 2

Sistemas de Distribuição de Vídeo sob Demanda

Neste capítulo descrevemos suscintamente os conhecimentos básicos necessários ao melhor entendimento da tese.

Uma maneira objetiva de se definir um serviço de Vídeo sob Demanda (Video on Demand - VoD) é compará-lo com uma locadora de vídeo eletrônica onde usuários escolhem os filmes que querem assistir através do portal da loja e o mesmo é enviado via Internet para exibição. Pode parecer simples, mas os desafios tecnológicos para se implementar um serviço deste tipo são bastante complexos, principalmente se forem levados em conta requisitos de Qualidade de Serviço (Quality of Service - QoS), como latência inicial entre o pedido do vídeo e sua exibição.

Em um serviço de VoD, a latência inicial deve ser pequena. Para manter esta latência pequena, a exibição do vídeo se inicia tão logo chegue uma quantidade de fluxo de vídeo suficiente para se decodificar e compensar eventuais atrasos na entrega dos trechos subsequentes. É o que se costuma chamar de envio de vídeo no modo *streaming*. Embora esta forma de entrega de vídeo diminua a latência de início de exibição, existe a necessidade de se manter um fluxo de vídeo com pouco *jitter*, i.e., baixa variação estatística do retardo na entrega dos trechos de vídeo subsequentes. Caso o *jitter* seja muito grande, a quantidade inicial de vídeo terá que ser maior para compensar os atrasos e, portanto, a latência inicial de exibição também será maior. A outra forma de entrega de vídeo é o modo *download*, que exige que todo o vídeo seja entregue antes de se iniciar sua exibição. Isto acarreta uma latência inicial alta, considerando-se que o tamanho deste tipo de mídia é grande e que o custo de um enlace com largura de banda suficiente para fazer um *download* em pouco tempo é muito alto [14].

Como um dos requisitos de um sistema de VoD é a baixa latência inicial, o modo

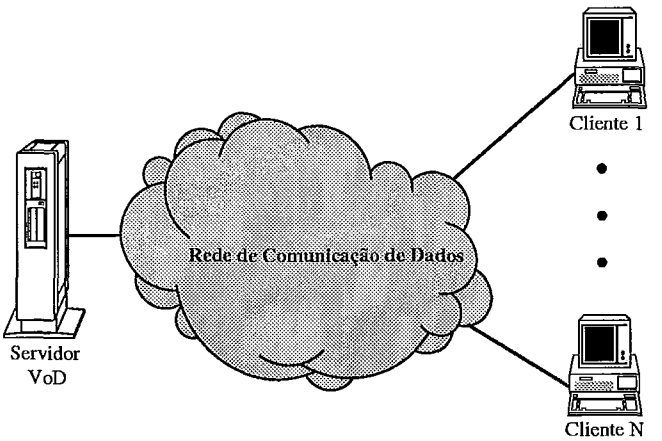


Figura 2.1: Arquitetura de um sistema de VoD

streaming é o mais indicado para entrega de vídeo para o cliente. Esta opção acarreta inúmeras questões a serem resolvidas no projeto de um sistema de VoD. Questões relacionadas com a alocação de recursos por um longo tempo e a garantia de entrega do fluxo de vídeo com baixo *jitter*, entre outras. Estas questões permeiam todo o sistema, desde o servidor até cada cliente. De uma maneira genérica e resumida, pode-se dizer que cada sessão de vídeo necessita uma determinada quantidade de recursos do servidor, que irá fornecer o fluxo de vídeo, e de recursos da infraestrutura de transmissão deste fluxo, basicamente, um canal que ocupe uma determinada largura de banda pelo tempo que durar o *streaming* de vídeo.

A próxima sessão é dedicada a detalhar cada componente de um Sistema de VoD convencional, bem como suas extensões para aumentar a escalabilidade, através do suporte a protocolos multidestinatários, doravante denominados apenas *multicast*. A sessão seguinte faz um levantamento das novas formas de distribuição de vídeo sob demanda que permitem escalar este serviço além do patamar atingido pelo modelo cliente/servidor.

2.1 Sistemas de Vídeo sob Demanda Convencional e com *Multicast*

Um sistema de VoD convencional segue o tradicional modelo cliente/servidor. Assim, um sistema de VoD é tipicamente constituído de um ou mais servidores VoD e de clientes VoD interconectados por uma rede de comunicação de dados (figura 2.1).

Um servidor VoD é constituído de um sub-sistema de armazenamento de mídia

contínua, *buffers* de memória, uma ou mais *CPU*s e pelo menos uma placa de rede para se conectar à rede de comunicação de dados. O funcionamento do sistema, que segue o modelo cliente/servidor, inicia-se com o cliente fazendo o pedido de um determinado vídeo e o servidor sendo encarregado de atender este pedido em tempo real. O equipamento do cliente possui um *playout buffer* para suportar uma certa variação nos atrasos da rede, bem como uma variação na taxa de consumo de quadros de vídeo. Além disso, o equipamento cliente também possui uma certa capacidade de processamento para interagir com o servidor e exibir o vídeo. Desta parte em diante, utilizaremos o termo "cliente"para nos referirmos ao "equipamento cliente".

2.1.1 Servidores VoD

O principal requisito de um servidor VoD é manter uma alta vazão contínua de fluxos de vídeo com um *jitter* mais baixo possível. Por isso, seu projeto é radicalmente diferente dos servidores convencionais.

Para exemplificar, quando um pedido chega ao servidor, ele tem de ser atendido em tempo real, ou seja, dentro de um espaço fixo de tempo após o qual degrada a qualidade do serviço percebida pelo usuário. Para atender este pedido, o servidor tem que assegurar que os outros pedidos em atendimentos também não sejam prejudicados. Para isso, o servidor faz um Controle de Admissão dos pedidos, para ter certeza que existem recursos suficientes para atender os pedidos em andamento e para este novo pedido [31].

Uma vez que o pedido foi admitido, ele precisa ser escalonado, ou seja, quando se iniciará o atendimento deste pedido de tal forma que os prazos de tempo real possam ser cumpridos e os prazos dos outros pedidos em atendimento também não sejam violados. Um grande problema para o servidor VoD é que normalmente os pedidos, uma vez iniciado seu atendimento, só serão finalizados após o envio do último quadro do filme, o que normalmente ocorre, em média, após noventa minutos. Ou seja, os recursos ocupados por um pedido demoram a ser liberados, o que dificulta escalar a taxa de atendimento de pedidos. Obviamente, para atender todos esses longos pedidos simultaneamente, o servidor multiplexa os recursos como CPU, largura de banda de entrada/saída, disco, etc. Para escalonar esses pedidos de forma a atender os diversos prazos (*deadlines*) e otimizar a ocupação dos recursos, algoritmos de escalonamento em tempo real como o EDF, *Earliest Deadline First*, [94] foram adaptados para uso num servidor VoD.

Os filmes, ou partes deles, podem estar em um mesmo disco. No entanto, para mandar um fluxo de vídeo no exato momento em que um pedido é escalonado, a cabeça de leitura

deve estar na posição correta, só que fazer este tipo de sincronização não é uma tarefa trivial. Por isso, o servidor tem um *buffer* que permite fazer o pré-carregamento (*prefetch*) do segmento de vídeo que será necessário mais à frente, escondendo a latência de acesso ao disco. Este mesmo segmento armazenado no *buffer* pode ser compartilhado por outros pedidos, maximizando a quantidade de pedidos atendidos por unidade de disco [73, 78, 49, 87].

O *buffer* do servidor VoD ajuda a esconder a latência do disco, mas para aumentar ainda mais a escalabilidade do servidor também é importante reduzir significativamente essa latência. Assim, o *layout* dos dados armazenados no disco, sua localização, distribuição e replicação em outros discos é de fundamental importância. Para isso, foram desenvolvidas várias técnicas como *striping* [81], *MET* [84], entre outras encontradas em [17, 46, 104], além de sistemas de arquivos especializados em armazenar mídias contínuas como o XFS da SGI [61], o *Tiger Video File Server* da Microsoft [77] e o *Tiger Shark* da IBM [99].

Como a demanda por recursos num servidor VoD é muito grande, iniciativas de servidores paralelos [2, 48] e *clusters* de computadores [74] também têm sido exploradas. Servidores de VoD comerciais de alto desempenho também já estão disponíveis. O SGI *Origin* 2000, por exemplo, é capaz de sustentar uma largura de banda contínua de 640 Gbps. Ou seja, a despeito do custo elevado de uma máquina como esta, o servidor deixa de ser o gargalo do sistema se custo não for problema. No entanto, como explorar esta largura de banda de saída, uma vez que as interfaces de rede não atingem esta taxa de transmissão, ainda é um problema. Para dar vazão a 640 Gbps são necessárias 533 placas de rede OC-24 (1.244,16 Mbps). Nem é preciso tanto para verificar que o gargalo está entre a rede e a saída do servidor. Realmente, um PC com barramento PCI de 64 bits a 100 MHz tem uma largura de banda de 6Gbps e uma placa de rede ATM (*Asynchronous Transfer Mode*) usual tem uma taxa de transmissão de 155 Mbps (OC-3) [53]. Além disso, ligar um servidor desses num só ponto da rede irá gerar congestionamentos, o mais plausível é usar uma quantidade maior de servidores, com menor largura de banda, mas distribuídos pela rede.

Na próxima sessão ver-se-á como explorar esta largura de banda limitada da placa de rede e da rede de transmissão e como diminuir a quantidade de dados a ser transportada.

2.1.2 Transmissão de VoD

A transmissão de mídia contínua no modo *streaming* implica em uma garantia da entrega deste fluxo com uma qualidade mínima. A especificação desta qualidade mínima inclui várias métricas, entre as quais pode-se destacar largura de banda média, baixa variação no tempo de entrega (*jitter*), taxa de pico de transmissão, taxa máxima de perdas na transmissão, ou através de prioridades relativas de acesso aos recursos da rede. Estas garantias são o que se convencionou chamar de Qualidade de Serviço (QoS). A aplicação, no caso o sistema de VoD, contrata o serviço de transmissão com as garantias mínimas de QoS para que o serviço atenda os requisitos de QoS especificados para que o usuário desfrute de uma exibição de vídeo com qualidade.

Atualmente, para se transmitir com garantia de QoS, duas tecnologias de transmissão podem ser utilizadas. O uso de Redes Digitais de Serviços Integrados de Banda Larga e os Serviços Diferenciados. A seguir, apresentamos um resumo dessas tecnologias para transmissão de vídeo.

Serviços Integrados

Esta abordagem pressupõe a **reserva de recursos** na rede ao longo do caminho de transmissão para a garantia de QoS. Para a reserva de recursos, a aplicação especifica uma QoS. Para a garantia desta QoS, a especificação é traduzida em termos de recursos de rede que devem ser reservados. Para que estes recursos sejam reservados, o sistema de transmissão usa um protocolo de sinalização para se comunicar com todos os elementos que farão parte deste novo fluxo , caso ele seja estabelecido. Caso os recursos necessários para esta nova conexão estejam disponíveis, eles serão reservados, garantindo a entrega com QoS, do contrário o serviço não será aceito. Embora esta técnica tenha um conceito simples para garantia de QoS, ela é de implementação complexa.

Serviços Diferenciados

Esta abordagem classifica os pacotes transmitidos pela rede em **classes de serviços**. Cada classe tem uma prioridade de transmissão. Classes com prioridades mais altas têm maior preferência no uso dos recursos da rede, em detrimento das classes de prioridades mais baixas. Em caso de escassez de recursos, como em um congestionamento na rede, os pacotes de maior prioridade serão privilegiados na transmissão, tendo uma probabilidade menor de degradação de QoS. Caso existam vários tráfegos de alta prioridade, certamente haverá degradação na QoS. Claramente, este é um serviço que não oferece garantias

severas de QoS, uma vez que não faz reserva explícita de recursos. Apesar disso, ele é uma opção aos Serviços Integrados, por não necessitar de um protocolo de sinalização e, portanto, de implementação menos complexa.

2.1.3 Multicast

Muitas aplicações transmitem os mesmos dados para vários destinatários, como uma carta circular enviada por e-mail ou quando vários usuários assistem à um mesmo segmento de vídeo. Transmitir um mesmo conteúdo várias vezes, é uma redundância desnecessária e um claro desperdício de recursos, no caso, largura de banda. Para evitar esta redundância, novos protocolos que permitem enviar simultaneamente o fluxo de vídeo de uma sessão para vários usuários foram propostos, como o SMART [6], o Mcast [76] e o IP multicast/MBone [91, 92].

Basicamente, os protocolos *multicast* procuram criar uma árvore *multicast* de menor custo ligando todos os participantes do grupo *multicast* e, ao mesmo tempo, respeitando o atraso máximo especificado. Como este problema é NP-completo, várias heurísticas foram propostas, baseadas em estratégias como a árvore de Steiner Restrita [97], e o CSPT (*Constrained Shortetst Path Tree*) [27], para citar algumas. Mais recentemente, pesquisas sobre multicast atacam o problema de diminuir a quantidade de estados gerados por uma sessão multicast no *backbone* da rede [43, 71].

No entanto, dois problemas permanecem, o primeiro é que o número de sessões *multicast* conseguidas por esses protocolos diminuem com o aumento do número de usuários. Isto porque, grupos maiores tendem a usar mais enlaces para formar a árvore *multicast*, esgotando com maior rapidez os recursos [26]. Por outro lado, sessões com muitos usuários tendem a atender mais usuários com menos sessões, o que economiza largura de banda. Ou seja, existe um compromisso entre muitas sessões *multicast* com poucos usuários, o que torna o serviço mais flexível, mas gera mais tráfego atendendo a um público menor, ou poucas sessões *multicast* com muitos usuários, o que permite atender a um público maior, mas com uma flexibilidade muito menor.

O segundo problema é que os usuários de um serviço VoD não se reunem em grupos para assistir a um vídeo em uma determinada hora, eles requisitam os vídeos de forma aleatória. Para resolver este problema foram criadas diferentes abordagens para atender os usuários, descritas a seguir:

- **Batching** - Esta técnica consiste em agrupar todos os pedidos para um mesmo vídeo num determinado espaço de tempo e atendê-los simultaneamente usando apenas

um fluxo de vídeo do servidor que será transmitido para vários usuários usando-se o *multicast* [1]. O problema desta abordagem é que o usuário que chegar primeiro terá que ficar esperando até o lote ser completado.

- **Bridging + Piggybacking** - *Bridging* permite o atendimento de vários pedidos separados por um pequeno intervalo de tempo (menor que o tempo total de vídeo que pode ser armazenado no buffer do servidor para um fluxo) usando o mesmo buffer do servidor [62]. Ele funciona da seguinte forma: se um pedido está sendo atendido por um fluxo de vídeo e o pedido seguinte é para o mesmo vídeo e o segmento pedido está no buffer mencionado, mas com uma pequena defasagem, ao invés de se criar um novo fluxo do disco, usa-se o conteúdo do buffer existente para transmitir ao outro usuário. Com o mecanismo de *bridging* consegue-se evitar a criação de mais um fluxo do disco do servidor, mas o fluxo que sai do servidor é distinto e defasado por um pequeno intervalo de tempo. A técnica de *piggybacking* foi criada para corrigir este problema. Ela simplesmente ajusta a taxa de transmissão dos dois fluxos, acelerando um e atrasando o outro, de tal forma que os dois fluxos possam ser substituídos por um único usando-se o multicast [59].

2.1.4 Compressão de Vídeo

Os mecanismos anteriores por si só não resolvem todos os problemas. A quantidade de dados a ser transportada ou armazenada, em se tratando de vídeo digital, é tão grande que torna impraticável o seu transporte ou armazenamento sem compressão.

Por isso, em 1988, o ISO/IEC criou um grupo para definir um padrão de compressão para áudio e vídeo, o MPEG (*Motion Picture Experts Group*) [12]. O primeiro padrão criado pelo grupo foi o MPEG-1[1] [33, 34, 35] cujo objetivo era produzir um vídeo com

1

A informação de vídeo no MPEG é estruturada como uma sequência contínua de quadros, como em um filme onde a película é formada por uma sequência de fotos. Para permitir o acesso aleatório ou funções de edição, estes quadros são agrupados em Grupo de Quadros (*group of frames* - GoF). Tipicamente um GoF é uma sequência de quadros IBBPBBPBBPBBP, os tipos de quadro são explicados mais adiante. Esta sequência é autocontida, ou seja, ela contém todas as informações necessárias para a visualização deste segmento de vídeo.

Tipos de Quadros no MPEG

I - *Intracoded*, é uma imagem completa comprimida comprimida com Transformada Discreta de Cossenos. É um quadro autocontido, no sentido de que não precisa de mais nenhum quadro para ser decodificado.

P - *Predictive*, é um quadro com predição de movimento, ou seja, para decodificá-lo precisa-se de informações de um quadro que foi exibido anteriormente, o quadro de referência, que pode ser um quadro I ou P. Um quadro P é de 30 a 50% menor que um Quadro I.

qualidade videocassete (352x240 para o padrão NTSC) com taxas de 1.2 Mbps. Levando-se em conta que o vídeo sem compressão, nesta baixa resolução, tem uma taxa de 60 Mbps, a tecnologia para se conseguir isto é bastante complexa. O padrão estabelecido a seguir foi o MPEG-2 [37, 38, 36], que pode ser considerado um superconjunto do MPEG-1. O principal objetivo do MPEG-2 é definir um formato padrão para descrever um fluxo de bits codificado. Ele não especifica o método de codificação ou a resolução do vídeo, por isso ele suporta desde vídeo com qualidade MPEG-1, passando por vídeo com qualidade *broadcast* a 6 Mbps, até vídeo com qualidade HDTV. Posteriormente, o padrão MPEG-4 [39, 40, 41] foi estabelecido. Este padrão permite distinguir objetos distintos dentro de um vídeo e também utiliza métodos de compressão mais eficientes, levando a um grau de compressão maior que o do MPEG-1 e do MPEG-2. Mais recentemente, um novo padrão de compressão foi criado, o H.264/MPEG-4 AVC [42]. Ele provê melhoras tanto na compressão (cerca de duas vezes mais eficiente do que o melhor padrão anterior) como na qualidade de percepção do vídeo (tanto em relação ao MPEG-2 como ao MPEG-4). Além disso, sua especificação prevê o seu uso em redes IP e redes sem fio para diversos tipos de aplicações, entre os quais VoD.

2.1.5 Clientes VoD

A maior parte da literatura sobre VoD se concentra no projeto de servidores e na transmissão de vídeo, sendo dada pouca atenção à interface com o usuário. O equipamento que faz a interface com o usuário é também chamado de *Set-Top-Box - STB*. Ele pode variar desde um equipamento que apenas exiba o vídeo assim que o receba a até um equipamento que rivalize com um computador pessoal ou mesmo ser um PC.

O cliente VoD basicamente possui duas funções:

1. enviar, receber e processar mensagens de controle;

2. receber, decodificar e exibir o vídeo.

Obviamente, um cliente tão simples exige que todo o resto do sistema seja muito mais complexo e dispendioso. Por exemplo, a inexistência de um *playout buffer*, ou apenas *buffer*, no cliente exige uma sincronização total no sistema, desde a geração do fluxo no servidor, passando pela transmissão, até chegar ao cliente, sem nenhuma variação no

B - *Bidirectional*, é um quadro com as diferenças entre um quadro anterior e outro posterior. Para decodificá-lo precisa-se ter o quadro anterior e já ter recebido o quadro seguinte. Um quadro B tem aproximadamente metade do tamanho de um quadro P.

atraso de entrega dos quadros de vídeo. Isto inviabiliza a utilização de equipamentos que fazem multiplexação estatística dos recursos, que são os mais viáveis economicamente.

Por outro lado, um sistema em que o servidor e o sistema de transmissão tenham uma baixa qualidade de serviço, obriga o cliente VoD a ser muito mais complexo para compensar as deficiências do resto do sistema. Normalmente, estas deficiências são corrigidas com um *buffer* maior.

Por exemplo, para contornar o problema do *jitter* da rede é preciso um *buffer* que possa acumular quadros a serem usados, caso o atraso na entrega aumente, ou que possa guardar os quadros, caso estes se adiantem, eliminando o risco de haver *overflow* e o conseqüente descarte de informação. Quanto maior o retardo máximo, maior o espaço necessário no *buffer*. Mas isto tem um custo, quanto maior o buffer, maior o tempo para enchê-lo até atingir o seu nível mínimo de operação e, portanto, maior a latência inicial de exibição do vídeo.

Resumindo, o *buffer* torna os requisitos tanto do servidor como da rede menos rígidos, ou seja, quanto maior o *buffer*, maior a flexibilidade dada ao servidor e à rede. Esta maior flexibilidade permite aumentar o número de clientes servidos pelo sistema. Por exemplo, se um servidor está com 50% de sua capacidade ociosa, e ainda faltam 60 minutos para os clientes terminarem a sessão, o servidor pode mandar o restante do vídeo da sessão em 30 minutos aproveitando sua capacidade ociosa[2] , ficando livre para atender os clientes que cheguem após estes 30 minutos, que pode ser o início do horário de pico [18]. Portanto, o *buffer* possibilita acumular quadros em períodos que eles são abundantes para gastá-los em períodos de escassez.

2.2 Novas Abordagens para Distribuição de Vídeo sob Demanda

As abordagens tradicionais de Sistemas VoD baseiam-se no modelo cliente/servidor. No modelo cliente/servidor puro há a necessidade de uma conexão dedicada para cada cliente, não havendo compartilhamento de recursos. Neste sistema, a necessidade de recursos cresce linearmente com o número de usuários. Como um fluxo de vídeo ocupa muitos recursos, basta um pequeno número de usuários para saturar o sistema de VoD. Portanto, o sistema cliente/servidor puro não é escalável. Por isso, o sistema de VoD convencional é estendido para suportar *multicast*. Com o *multicast*, um mesmo fluxo de

[2]Supondo-se que os clientes tenham largura de banda suficiente para receber este fluxo extra de vídeo

vídeo é compartilhado por vários clientes. No entanto, o *multicast* por si só não resolve o problema da escalabilidade. Um servidor de vídeo que forneça vários fluxos *multicast* também satura toda a largura da banda do seu enlace. Por isso, há a necessidade de se explorar novas abordagens com o objetivo de aumentar a escalabilidade de um sistema de VoD.

Recentes abordagens procuram explorar os recursos que estão na borda da rede, mais especificamente, explorar os recursos do cliente ou instalar recursos na fronteira, junto aos clientes, como os proxies, que são servidores de *cache*.

2.2.1 Explorando Recursos do Cliente

Basicamente, os recursos dos clientes que podem ser utilizados em proveito do sistema são sua largura de banda, tanto de descida (*download*) como de subida (*upload*), e a sua capacidade de armazenamento. Esta última capacidade varia, no pior caso, desde um cliente que possua somente a memória do *playout buffer*, até clientes com grande capacidade de armazenamento primário e secundário.

As técnicas que exploram os recursos dos clientes podem ser classificadas em duas grandes áreas:

- Técnicas que exploram a largura de banda de descida do cliente, e neste caso, precisam de uma capacidade extra de armazenamento no cliente para armazenar o conteúdo extra recebido do servidor.
- Técnicas que exploram a capacidade de armazenamento do cliente, e neste caso, precisam explorar também a largura de banda de subida do cliente para fazer uso do conteúdo armazenado.

Técnicas básicas que exploram a largura de banda de descida do cliente

Estas técnicas exploram o fato de que em alguns sistemas os clientes tem uma largura de banda de descida muito maior que a necessária para receber um vídeo. Assim, estes clientes podem receber dois ou mais fluxos de vídeo simultâneos.

A seguir são analisadas algumas dessas técnicas:

Patching [53, 100]. Esta abordagem explora a largura de banda do cliente. Neste esquema, o cliente ao solicitar um vídeo é informado que já existem outras sessões deste mesmo vídeo. Ao receber esta informação, o cliente se junta ao *multicast* da atual sessão e começa a guardar em disco o conteúdo que receber dessa sessão. O começo do filme

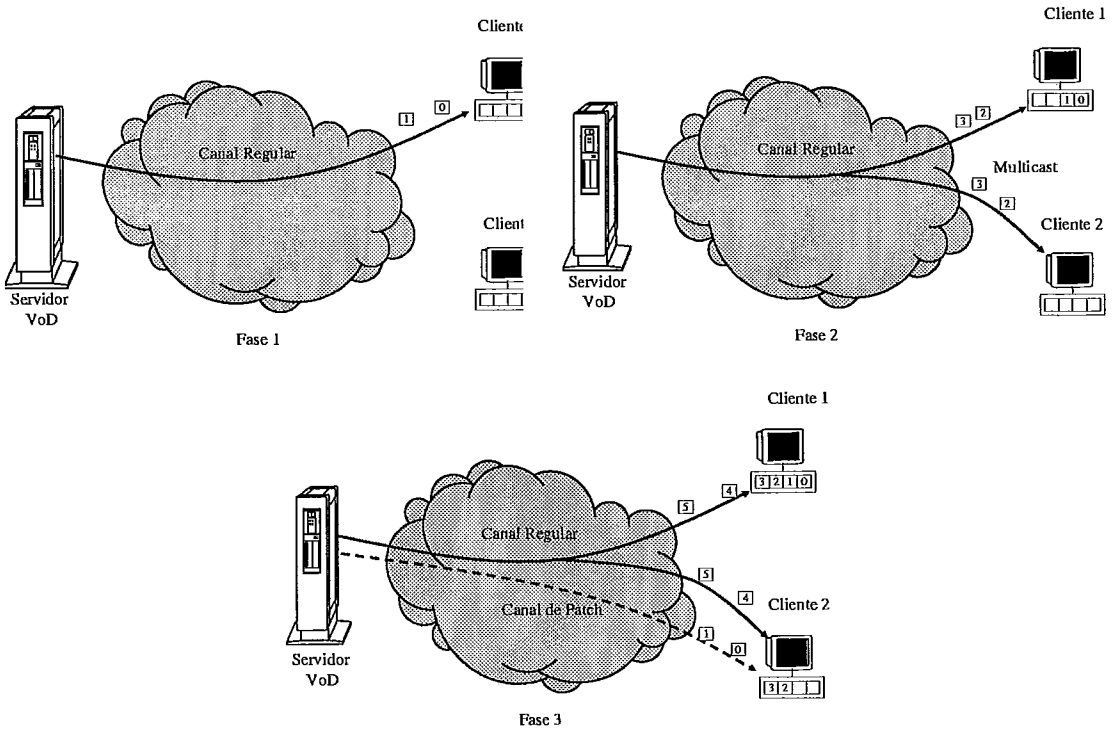


Figura 2.2: Patching

ou qualquer outro segmento que ele perdeu é solicitado ao servidor como um "remendo" em um outro canal, por isso a denominação *Patching* dada pelos seus autores (figura 2.2). Para isto, o cliente precisa ter banda suficiente para receber dados em dois ou mais canais simultaneamente. A vantagem deste esquema é que o remendo é atendido num período de tempo menor, liberando recursos do servidor mais rapidamente. *Skimming* [15] é outra técnica que se diferencia de *patching* por usar a largura de banda que sobra de um canal, quando da transmissão de um vídeo, para enviar um *patch*.

Técnicas básicas que exploram a capacidade de armazenamento do cliente

As técnicas que exploram a capacidade de armazenamento do cliente fazem, na verdade, o reuso do conteúdo que existe na memória de cada cliente. Para fazer este reuso, o cliente precisa retransmitir seu conteúdo para outro cliente. Desta forma, o cliente passa a exercer o papel também de servidor, permitindo o surgimento dos sistemas *peer-to-peer* para distribuição de VoD.

Chaining [89]. Uma abordagem diferente de utilização dos *playout buffers* é a sua utilização como linha de retardo. Neste caso, se um cliente chegar algum tempo depois de outro cliente que já foi atendido com o mesmo filme, o sistema não precisa criar uma

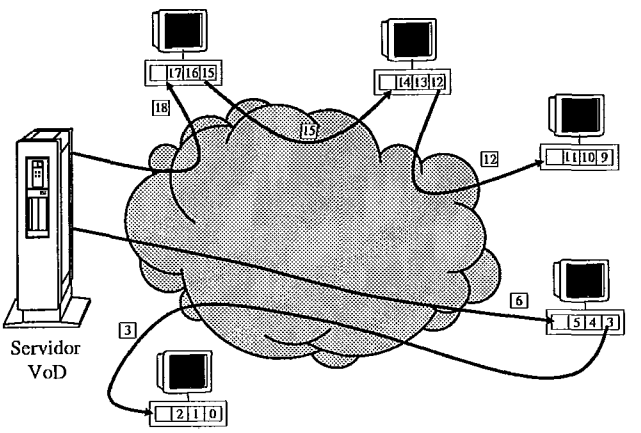


Figura 2.3: Chaining

nova sessão a partir do servidor para esse novo cliente, basta o sistema utilizar um fluxo que foi retardado pelo primeiro cliente e reenviá-lo ao novo cliente. Este procedimento pode ser estendido indefinidamente formando longas cadeias de *playout buffers*, sendo portanto denominado *Chaining* pelos seus autores (figura 2.3).

Embora *chaining* use apenas o buffer do cliente como uma linha de atraso, sem grandes preocupações com a gerência de memória dos buffers dos clientes, sua concepção permitiu o desenvolvimento das técnicas de distribuição de vídeo peer-to-peer.

Os autores de *chaining* estenderam ao extremo este conceito, propondo o *Earthworm* [52], onde disponibilizaram o *buffer* do servidor para evitar a descontinuidade de uma cadeia devido à falta de chegada de clientes que pudessem continuar fornecendo mais espaço em *buffer* para manter a cadeia (figura 2.4). No caso, evitar que o servidor recorra ao disco para criar mais um fluxo é a vantagem deste esquema. Desta forma, as cadeias seriam ainda mais longas, podendo dar a volta ao mundo, donde o nome do esquema se deriva.

Uma desvantagem de *Earthworm* e *Chaining* é que as cadeias podem se romper se um cliente, por qualquer motivo, não assistir ao filme até o seu final. Para eliminar esta possibilidade de falha, transferiram este *playout buffer* do cliente para o comutador da rede, usando o conceito de redes ativas [47, 13], sendo este esquema denominado Protocolo Multidestinatário com Cache [51] (figura 2.5).

2.2.2 *Proxies* de Vídeo em Redes de Distribuição de Conteúdo

As abordagens tradicionais de transmissão de vídeo fazem uso ou dos Serviços Integrados ou dos Serviços Diferenciados. No entanto, a oferta destes serviços é restrita,

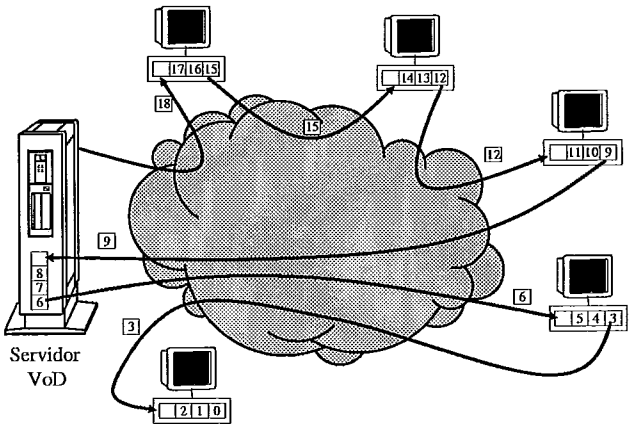


Figura 2.4: Earthworm

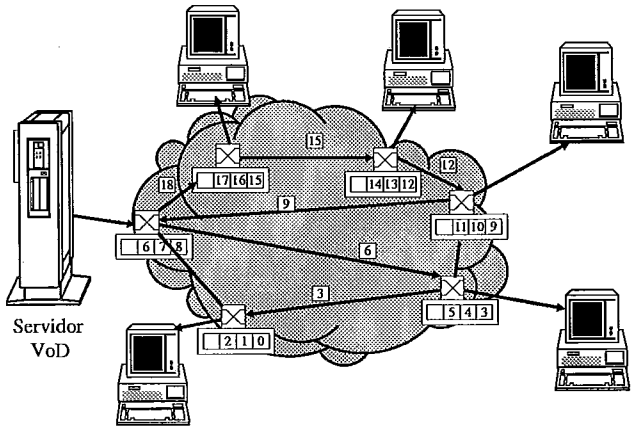


Figura 2.5: Protocolo Multidestinatário com Cache

pois exige um enorme investimento em toda a infra-estrutura de rede, que precisaria trocar todos os comutadores e roteadores para suportarem reservas ou diferenciação de serviço. Devido a este e outros problemas, o que atualmente se dispõe são algumas sub-redes que oferecem ou Serviços Integrados ou Serviços Diferenciados apenas na sua área de cobertura. Como no caminho entre um cliente e o servidor passa-se por várias sub-redes, gerenciadas por diferentes administradores, dificilmente se consegue um caminho que passe apenas por sub-redes que ofereçam um determinado tipo de serviço e cujos administradores concordem em prestar o serviço tal qual é solicitado, uma vez que isto implica em reservar recursos escassos para usuários externos à sub-rede [11]. Para prover um serviço de *streaming* de vídeo que independa deste ambiente, uma alternativa viável é o uso de Redes de Distribuição de Conteúdo (CDN). CDNs são usadas principalmente para distribuição de conteúdo estático, como arquivos HTML, imagens ou vídeos pela

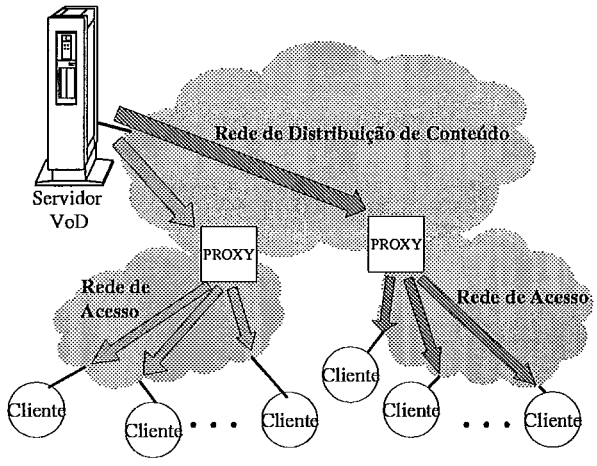


Figura 2.6: Arquitetura de uma Rede de Distribuição de Conteúdo - CDN

Internet. Uma vez que tal ambiente não oferece QoS, o uso de CDNs possibilita a entrega de conteúdo através de uma política de melhor esforço, com uma latência menor, um menor tráfego na rede e uma maior escalabilidade do sistema, e no caso do vídeo, com uma QoS melhor, mas sem nenhuma garantia da mesma. Alternativamente, em uma rede que ofereça QoS, uma CDN oferece as mesmas vantagens, mas desta vez para entrega de vídeo no modo *streaming* com QoS.

Redes de Distribuição de Conteúdo

Para oferecer um serviço com melhor QoS, uma Rede de Distribuição de Conteúdo (CDN) procura colocar o conteúdo o mais próximo do cliente, evitando caminhos longos até o servidor, que em algum ponto pode estar congestionado. Para isso, um servidor secundário é colocado na borda da rede, na fronteira entre a rede de distribuição e a rede de acesso (figura 2.6). Este servidor secundário, denominado *proxy*, replica o conteúdo do servidor primário. Isto traz diversas vantagens, entre elas o desempenho. Com um *proxy* próximo ao cliente a latência para a entrega do conteúdo é menor, a taxa de transmissão pode ser maior e, o tráfego no *backbone* da rede é menor. O resultado é que a escalabilidade do sistema também é maior, uma vez que à capacidade de processamento e de entrega de conteúdo do servidor primário são somadas a capacidade de processamento e de entrega dos *proxies*. A eficiência de um *proxy* é diretamente proporcional ao uso do conteúdo replicado que ele consegue armazenar. Por isso, um *proxy* procura armazenar apenas os conteúdos mais populares, de modo a manter sua taxa de acerto a mais alta possível. Aplicações que fazem entrega de conteúdo estático, como imagens, arquivos HTML e arquivos de áudio e vídeo, são as grande beneficiárias do uso de CDNs.

Proxies de Vídeo

Proxies de vídeo diferem dos *proxies* de conteúdo convencional devido às características da mídia vídeo que tornam o seu uso em *proxies* convencionais ineficiente. *Proxies* convencionais armazenam em sua *cache* arquivos HTML e imagens estáticas. Estes tipos de mídia caracterizam-se por ter, em média, arquivos de tamanho pequeno. Desta forma, o *proxy* convencional consegue gerenciar em sua *cache* diversos arquivos e imagens, visando maximizar a taxa de acertos com políticas que levam em conta a popularidade ou a frequência com que os objetos de mídia são acessados. Em particular, a mídia vídeo se caracteriza por ter um tamanho algumas ordens de grandeza maior, de tal forma que, com um *proxy* com o mesmo espaço de armazenamento, com poucas mídias de vídeo a *cache* já estará cheia. Com isso, o *proxy* armazena poucos vídeos e consequentemente tem uma baixa eficiência, ou seja, uma baixa taxa de acertos. Cabe ressaltar que para vídeos curtos (clips), devido ao seu tamanho ser da ordem de grandeza dos arquivos HTML ou de imagens estáticas, o *proxy* convencional costuma ser eficiente. Mas para vídeos longos, como filmes com uma hora ou mais de duração e com qualidade DVD, cujo tamanho chega a alguns giga bytes, há necessidade de *proxies* especializados.

Para contornar este e outros problemas, *proxies* que lidam com vídeos adotam mecanismos de gerência e políticas diferentes dos *proxies* convencionais. Ao invés de armazenar o objeto vídeo por inteiro, eles gerenciam e armazenam segmentos da mídia vídeo. Com isso, eles otimizam a utilização de espaço de armazenamento do *proxy* de vídeo, aumentando sua eficiência. A forma como estes segmentos são gerenciados levam a distinguir duas classes de *proxies* de vídeo. Os *proxies* de vídeo com gerenciamento de *cache* estática e os de gerenciamento de *cache* dinâmica.

Proxies de Vídeo com Cache Estática

Este tipo de *cache* não leva em conta a relação temporal que existe entre os diversos segmentos de um mesmo vídeo. O vídeo é dividido em vários segmentos e cada segmento é tratado como se fosse um objeto único distinto dos demais [80, 88, 101, 82].

Proxies de Vídeo com Cache Dinâmica

Este tipo de *cache* leva em conta a relação temporal que existe entre os diversos segmentos de um mesmo vídeo. O vídeo é dividido em vários segmentos e cada segmento se interrelaciona com outro do mesmo vídeo através de sua distância temporal [60, 16, 67, 10, 83, 85]. Desta forma, a distância temporal entre duas ou mais requisições pode ser

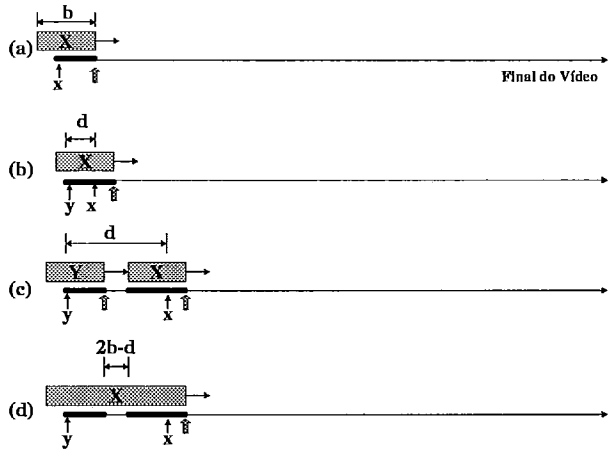


Figura 2.7: Ring buffers

escondida se o conteúdo entre estas requisições estiver na *cache* dinâmica. Com isso, basta apenas um único fluxo de vídeo do servidor para o *proxy*, que por sua vez enviará os diversos fluxos, atendendo às requisições dos clientes.

Caches de vídeo dinâmicas usam *ring buffers* para aproveitar a relação temporal entre os diversos segmentos de um mesmo vídeo. Neste ponto será apresentado suscintamente o funcionamento dos *ring buffers*, e no capítulo 4 ele será discutido em mais detalhes.

Ring buffers podem ser implementados como vetores circulares alocados na *cache* do *proxy*. A figura 2.7-a mostra o *ring buffer* X no início de um fluxo de vídeo. O parâmetro b é o tamanho do *ring buffer* dado pelo tempo médio de vídeo (supondo um fluxo VBR) que pode ser armazenado pelo ring buffer. À medida que o vídeo é enviado para o cliente o *ring buffer* desliza em direção ao final do vídeo. O fluxo de vídeo recebido do servidor, representado pela seta cheia da figura, alimenta o *ring buffer* da seguinte forma. À medida que chegam mais segmentos de vídeo do servidor (seta cheia da figura), estes trechos são postos no fim do *ring buffer*, enquanto que os segmentos no início do *ring buffer* já foram transmitidos ao cliente e, portanto, podem ser postos para fora do *ring buffer*. Desta forma, o *ring buffer* é gerenciado segundo uma política FIFO (primeiro a entrar, primeiro a sair). Este comportamento pode ser visto como uma janela deslizante sobre o fluxo de vídeo. *Ring buffers* preservam as relações temporais entre os segmentos de vídeo, uma vez que os segmentos de vídeo enviados ao cliente correspondem os segmentos seguintes, que não serão descartados, pois estão reservados (dentro do *ring buffer*).

Além deste relacionamento temporal dos segmentos internos ao *ring buffer*, *caches* dinâmicas também relacionam pedidos de clientes que chegam defasados por um intervalo de tempo. A figura 2.7-b mostra dois pedidos x e y que chegaram defasados de uma

distância d unidades de tempo. Como d é menor que b, o pedido y que chegou após o pedido x pode ser atendido pelo mesmo *ring buffer*.

A figura 2.7-c mostra dois pedidos que chegaram defasados por uma distância d maior que b. Neste caso, a *cache* dinâmica alocou outro *ring buffer* Y para atender o pedido y. Note que com esta decisão, a *cache* dinâmica teve que solicitar ao servidor de vídeo outro fluxo de vídeo para alimentar o *ring buffer* Y.

A figura 2.7-d mostra uma outra alternativa para atender o pedido y. Ao invés de solicitar outro fluxo de vídeo para alimentar o *ring buffer* Y, a *cache* dinâmica encadeou os dois *ring buffers* X e Y em um único *ring buffer*. Mas isto teve um custo, foi necessário utilizar um espaço de armazenamento extra da *cache* correspondente a $2b-d$. Além disto, para encadear o *buffer* é preciso que exista espaço de armazenamento em *cache* disponível para tal. Mais do que isso, para atender pedidos que chegam defasados por uma distância maior do que b, é necessário haver espaço em *cache* para alocar um *ring buffer*.

Capítulo 3

Memória Cooperativa Distribuída

Sistemas convencionais baseados no modelo cliente/servidor têm uma escalabilidade limitada. Aplicações, como portais de comércio eletrônico, quando precisam escalar para crescer e atender a um maior número de usuários, podem recorrer a Redes de Distribuição de Conteúdo (CDN). No entanto, nem todos dispõem de recursos para ter a infra-estrutura de uma CDN a sua disposição. Uma alternativa para distribuição de vídeo sob demanda mais econômica baseia-se no uso do modelo *peer-to-peer* para distribuição de conteúdo. Sistemas *peer-to-peer* formam redes de distribuição de conteúdo *ad hoc*, usando para isto os próprios clientes que fazem uso do serviço.

Apesar do modelo *peer-to-peer* não ser novo, por exemplo a Internet foi concebida originalmente como um sistema *peer-to-peer*, foi somente em meados do ano 2000 que a idéia de *peer-to-peer* foi apresentada como uma novidade [3]. Sistemas como Napster [64], Freenet [32], Gnutella [28] entre outros, se tornaram populares, ou seja, escaláveis, atendendo a um número imenso de usuários com pouca ou praticamente nenhuma infra-estrutura adicional para a distribuição de conteúdo. Neste contexto, surgiu a MCD (Memória Cooperativa Distribuída) para distribuição de vídeo, cuja concepção se iniciou em dezembro de 1999 e os primeiros resultados surgiram em meados do ano 2000 [21], assim como um pedido de patente foi depositado [20]. MCD surgiu como uma alternativa de VoD escalável aos sistemas de VoD convencionais

Este capítulo apresenta a MCD (Memória Cooperativa Distribuída) [21, 22, 19], uma das primeiras propostas na literatura de um sistema *peer-to-peer* para distribuição de vídeo sob demanda. A MCD inova no sentido de utilizar o espaço de armazenamento dos *buffers* não apenas como uma linha de retardo [52, 89], mas como uma área coletiva de memória e permitir que esta seja usada de forma cooperativa pelo sistema de VoD como um todo, visando a escalabilidade do sistema. A proposta inicial da MCD era um sistema *peer-to-peer* para ser utilizada na Internet. No entanto, clientes com acesso simétrico e com

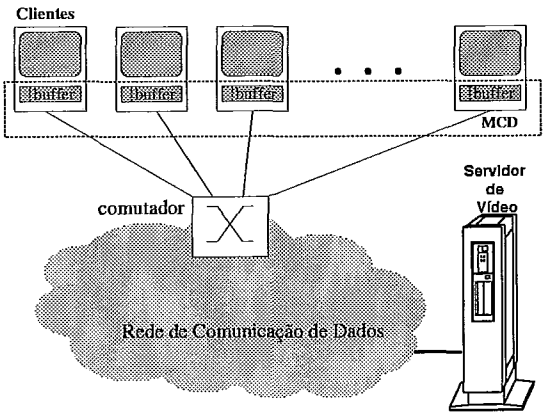


Figura 3.1: Arquitetura da MCD

grande largura de banda na Internet são exceções e não a regra. Como a MCD pressupõe um acesso simétrico nos clientes, uma versão da MCD para ambientes corporativos, onde clientes conectados a uma rede local com acesso simétrico são a regra, deu origem a uma Dissertação de Mestrado [54], cujo produto final foi um protótipo, o GloVE [55, 56], que confirmou os resultados obtidos por simulação da MCD.

3.1 Arquitetura do Sistema de VoD com a MCD

Esta seção inicia descrevendo a arquitetura e o protocolo de gerência e operação básica da MCD.

3.1.1 Arquitetura básica

A MCD supõe que cada cliente possui um conjunto de recursos mínimos para receber, armazenar um trecho do vídeo no *playout buffer*, decodificá-lo e exibí-lo, como em um *set-top-box* comum. A única e principal diferença é que este cliente também deve possuir uma largura de banda de subida no mínimo igual a de descida, uma vez que o cliente também deve agir como um servidor de vídeo.

O servidor de vídeo na MCD, por sua vez, também deve ser capaz de localizar o vídeo solicitado não só no seu sistema de armazenamento local, mas buscar inicialmente o conteúdo na memória cooperativa. Para isso, ele possui um módulo de gerência da MCD, que percorre a árvore de clientes para verificar se o conteúdo está ou não na memória cooperativa. A figura 3.1 mostra a arquitetura da MCD. Todos os elementos do sistema são interligados por uma rede de comunicação de dados.

3.1.2 Gerência e operação da MCD

A MCD utiliza basicamente a memória do *playout buffer* do cliente como parte de um espaço global de memória que pode ser utilizada em proveito de todo o sistema de VoD para aumentar sua escalabilidade. Portanto, é fundamental compreender como o *playout buffer* funciona e as estruturas de dados existentes nos clientes antes de estudar a operação da MCD.

Estrutura de dados

O *playout buffer* do cliente é dividido em partes com objetivos administrativos e sua divisão interna é indicada por alguns ponteiros e níveis de trabalho (figura 3.2). O *playout buffer* pode ser implementado de forma circular, de forma que não se precise fazer deslocamentos de avanços ou recúos de unidades de vídeo dentro do mesmo, apenas movimentos de ponteiros. Uma unidade de vídeo (**u.v.**) pode ser desde um quadro de vídeo, um GoF (grupo de quadros MPEG) ou qualquer outra unidade utilizada para indexar e gerenciar os trechos de um vídeo.

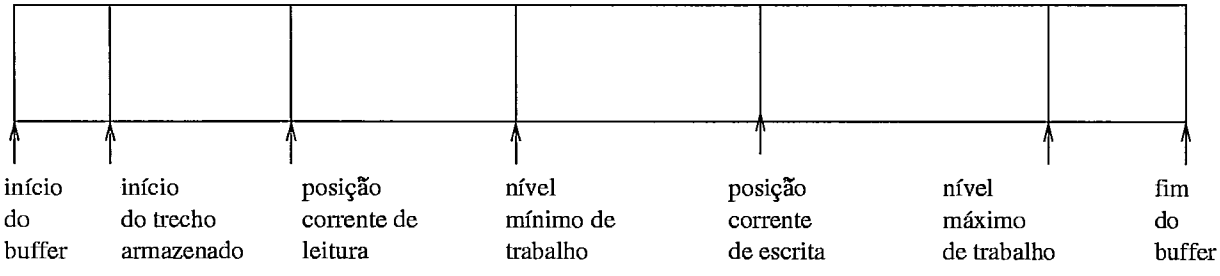


Figura 3.2: Divisão do *Playout Buffer*

Os ponteiros e níveis com suas funções são descritos a seguir:

1. **Início do trecho armazenado** - indica o início do trecho de vídeo armazenado no *playout buffer*, conforme as **u.v.** são descartadas o ponteiro é atualizado.

2. **Posição corrente de leitura** - indica a próxima **u.v.** a ser exibida. Este ponteiro pode coincidir com o ponteiro de início do trecho quando não houver quadros já lidos. O espaço dado pela diferença entre estes dois ponteiros define o espaço disponível para armazenar **u.v.** já lidas e o tamanho deste espaço pode ser pré-estabelecido ou configurado dinamicamente.

3. **Posição corrente de escrita** - indica a posição em que a próxima **u.v.** que chegar ao cliente deve ser armazenada.

4. **Nível mínimo de trabalho** - serve para garantir que não faltem **u.v.** para serem exibidos devido a atrasos na rede. Este nível tem como referência o ponteiro de posição corrente de leitura. Quando o ponteiro de posição corrente de escrita está abaixo do nível mínimo de trabalho, é gerado uma chamada de falha de vídeo futura que irá contatar o gerenciador da MCD que irá providenciar um novo fluxo para recompletar o nível mínimo.

5. **Nível máximo de trabalho** - indica possibilidade de *overflow*. O ponteiro de posição corrente de escrita deve se manter abaixo deste nível. Como o nível mínimo, também tem como base a posição corrente de leitura. Quando o ponteiro de posição corrente de escrita está acima do nível máximo de trabalho, é gerado uma chamada de pré-*overflow* que irá contatar o gerenciador da MCD que irá diminuir a taxa de transmissão para este cliente. Na impossibilidade disto, irá providenciar outro fluxo a uma taxa menor a partir da última **u.v.** recebida, ou um outro fluxo após um pequeno intervalo de tempo, o suficiente para esvaziar o buffer abaixo do nível de pré-*overflow* e iniciar a transmissão de outro fluxo de vídeo.

O cliente também tem um ponteiro que indica o objeto que está lhe mandando o fluxo, bem como os objetos para os quais manda o fluxo. Desta forma, os clientes se encadeiam segundo uma topologia em árvore.

Além disso, cada cliente mantém uma tabela, sempre atualizada, com as **u.v.** que possui no seu buffer e ponteiros para as suas respectivas posições no *playout buffer*.

A tabela de informações do cliente mantém o identificador do cliente, o tamanho do seu *playout buffer*, o nível mínimo e máximo de trabalho do *playout buffer*. Possui também um campo que é um ponteiro para o(s) próximo(s) cliente(s) e para o cliente anterior para controlar o encadeamento dos *playout buffers* e um campo com o quadro inicial e final que está no *playout buffer* usado quando o cliente está em pausa. Esta tabela mantém também o instante aproximado em que o cliente começou a receber o vídeo.

Caso o cliente interrompa a exibição do vídeo por qualquer motivo, ao recomeçar a exibição ele envia uma notificação ao servidor que registra o tempo da chegada desta notificação e subtrai dele o tempo de vídeo já transmitido para o cliente, atualizando o tempo de início de recebimento do vídeo. Este tempo agora é um tempo virtual, uma vez que é o tempo em que ele deveria ter começado a receber o vídeo para estar com o *playout buffer* no estado atual. Esta será a referência de tempo utilizado pelo servidor para saber quais **u.v.** estão no *playout buffer* do cliente.

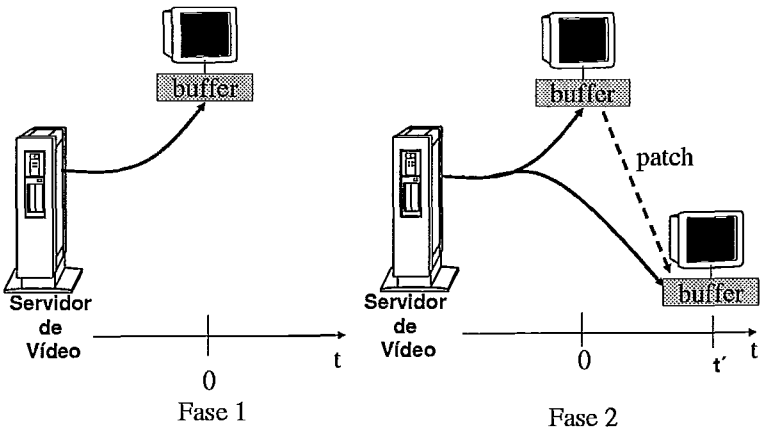


Figura 3.3: Derivação de um fluxo

Gerência e operação da MCD

O funcionamento da MCD será explicado usando-se um único vídeo em um único servidor, uma vez que isto simplifica o entendimento do mecanismo. No entanto, este mecanismo pode ser usado com diversos vídeos em um mesmo servidor, um vídeo em diversos servidores ou mesmo diversos vídeos em diversos servidores, sendo que estes servidores podem estar num *cluster* de forma centralizada ou distribuídos pela rede.

A gerência da MCD faz uso intensivo do *playout buffer* do cliente, por isso uma métrica a ser muito utilizada nesta proposta é o tempo do *playout buffer* do cliente em segundos, doravante referido apenas como TBC. Ele é calculado usando-se a taxa média de transmissão do fluxo, neste caso divide-se o tamanho do *playout buffer* pela taxa média de transmissão obtendo-se o tempo médio de vídeo armazenado no *playout buffer*. Como é uma média, por segurança, o tamanho do *playout buffer* é estimado para menos, não considerando o espaço de *overflow* do buffer. Desta forma procura-se garantir que nem um *byte* sequer da **u.v.** de interesse seja descartada antes que o fluxo reaproveitado seja mandado para o destino.

O primeiro cliente, ao solicitar um vídeo, envia um pedido ao gerente MCD que irá providenciar um fluxo do vídeo. Como não existe outro fluxo deste vídeo (é o primeiro pedido), quem vai fornecê-lo é o servidor. Quando o *playout buffer* atingir o seu nível mínimo de trabalho, o cliente inicia a exibição do vídeo. Até este ponto o funcionamento é semelhante aos demais sistemas VoD.

Ao chegar um novo pedido de vídeo, o gerenciador da MCD consulta a sua tabela com informações do cliente solicitante, doravante denominado apenas solicitante, a procura de um cliente provedor, chamado a partir deste ponto apenas provedor, que tenha começado a receber o vídeo até TBC segundos antes do tempo atual. Como podem existir vários clientes aptos a retransmitir o seu fluxo, a escolha do provedor pode ser feita da seguinte maneira:

- Se existir algum cliente cujo tempo de início relativo é menor que $TBC/2$ e ele já retransmite um fluxo, é este o provedor escolhido. Ou seja, o provedor coloca mais um cliente no seu *multicast*. Como o solicitante perdeu o início do vídeo, um outro cliente do *multicast*, o colaborador, manda um *patch* com o início do vídeo (Figura 3.3). Este tamanho, $TBC/2$, não é fixo, depende de um compromisso entre explorar mais o fluxo *multicast* retransmitido pelo provedor e a necessidade de se fazer muitos *patches* longos. Note que este esquema de *patch* difere do *patching* visto no Capítulo 2 [53], pois é mandado de cliente para cliente.

- Se não houver nenhum cliente na categoria anterior, o último cliente a ter recebido o início do fluxo e que tenha começado a receber a não mais tempo que TBC segundos é que irá retransmtir o vídeo, ou seja ele faz um encadeamento.

- O servidor fornece um novo fluxo, caso contrário.

A figura 3.4 exemplifica o funcionamento da MCD. Inicialmente, o primeiro cliente a chegar é o Cliente A. Portanto, ele recebe diretamente do servidor um fluxo regular (Figura 3.4-a). Em seguida, chega o cliente B. Como o provedor do cliente A é o próprio servidor, e este iniciou a transmissão do fluxo regular em um tempo menor que $TBC/2$, o cliente B é posto no *multicast*, já existente, do servidor e um *patch* é fornecido pelo cliente A (Figura 3.4-b). Note que, devido ao *patch* fornecido pelo cliente A, o cliente B fica com o mesmo conteúdo do cliente A, ou seja, é como se os dois clientes tivessem chegado ao mesmo tempo. Suponha que um novo cliente C chegue ao sistema após $TBC/2$ segundos do tempo em que o cliente A chegou. Neste caso, o cliente C é encadeado no cliente A (Figura 3.4-c). A figura 3.4-d mostra o cliente D chegando a não mais que $TBC/2$ segundos depois que o cliente C. Por isso, ele é posto no fluxo *multicast* do cliente A e recebe um *patch* do cliente C. Por fim, chega o cliente E muito tempo depois que o último cliente. Como ele não pode ser encadeado nem posto num *multicast* já existente, um novo fluxo regular do servidor é estabelecido (Figura 3.4-e).

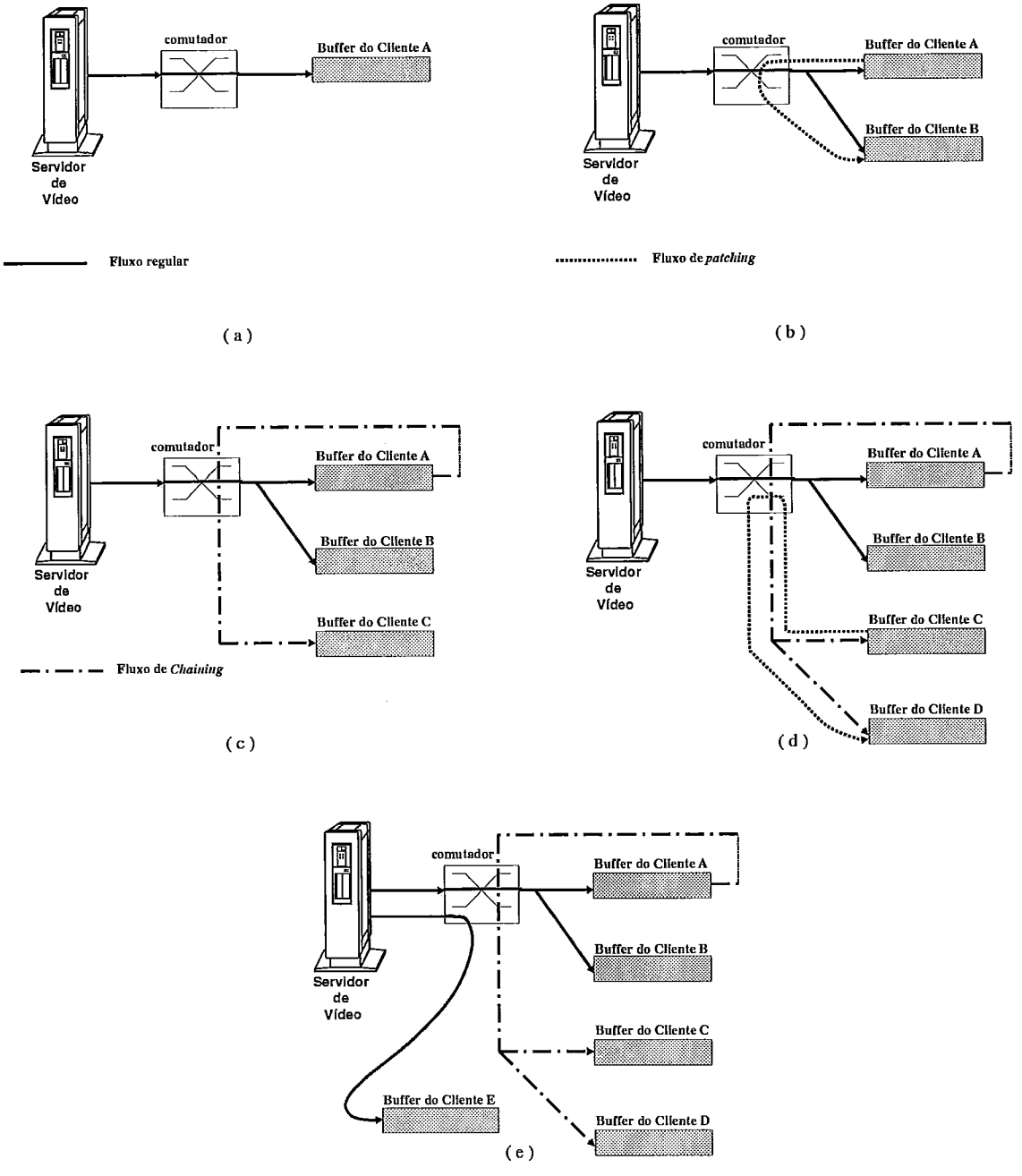


Figura 3.4: Exemplo de funcionamento da MCD

Como a largura de banda disponível no cliente pode ser menor que a do servidor e para diminuir o tráfego na rede, a MCD parte do princípio que o provedor só retransmite um fluxo *multicast*. Este detalhe também distingue a MCD de *chaining* [89, 52] na qual qualquer cliente pode retransmitir vários fluxos *unicast* com uma pequena defasagem entre si, congestionando a rede rapidamente. O restante da largura de banda de um cliente é usado para a gerência da MCD e envio de *patches*.

Caso o fluxo se rompa por algum motivo, o conteúdo do *playout buffer* será gradualmente consumido pela exibição do vídeo até atingir o seu nível mínimo. Neste momento, será gerada uma chamada ao gerenciador MCD que providenciará um fluxo para este *playout buffer*, vindo de outro provedor ou servidor. O cliente sabe qual é a última **u.v.** recebida, logo ele solicita à MCD o vídeo a partir da **u.v.** seguinte. Em qualquer caso, o nível mínimo de trabalho deve ser suficiente para armazenar um trecho de vídeo que dure o tempo para notificar o gerenciador MCD, tratar a notificação e enviar as ordens a todos os elementos envolvidos no estabelecimento ou derivação de um novo fluxo.

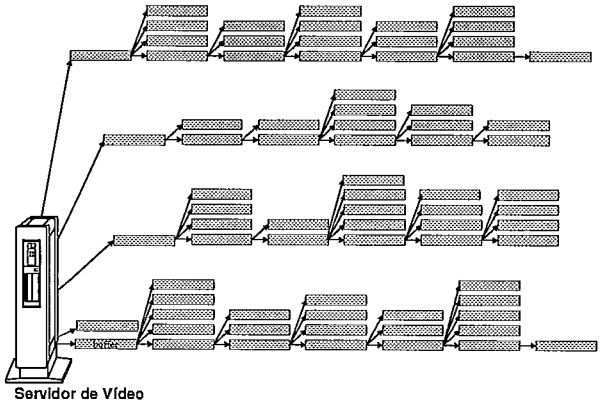


Figura 3.5: Conjunto de cadeias de buffers

É importante ressaltar que se espera da MCD um melhor desempenho quando existem várias cadeias de *buffers*, figura 3.5, ou seja, ele é mais apropriado para ser usado em vídeos muito populares e em horários de pico de utilização do serviço. Isto não quer dizer que ele não possa ser usado em situações com vídeos de pouca demanda, embora nestes casos a sua contribuição no desempenho e escalabilidade do sistema são desnecessárias.

3.2 Estudo do Desempenho da MCD

Nesta seção serão apresentados resultados obtidos de simulações que demonstram a viabilidade desta abordagem. Os resultados da simulação serão comparados com os resultados obtidos da implementação do GloVE [54], protótipo da MCD, resultado de uma Tese de Mestrado derivada deste trabalho. Inicialmente será descrito o ambiente de simulação.

3.2.1 Ambiente de Simulação

Supõe-se a existência de uma rede de comunicação de dados de banda larga (B-ISDN) que irá interligar os clientes ao servidor do sistema de VoD.

Nesta simulação não será levada em conta a topologia da rede, apenas que a capacidade da matriz de comutação interna a cada comutador ATM será muito maior que a de seus canais e portanto pode ser explorada em proveito do sistema. Os comutadores ATM estão totalmente interligados, ou seja, o *backbone* da rede não é o gargalo, apenas o enlace entre o servidor e a rede constitui o gargalo do sistema. Para simplificar a implementação do simulador, supõe-se que a rede presta apenas um serviço determinístico, onde todos os pacotes de uma conexão recebem a garantia de chegar ao destino com o limite de atraso respeitado e sem perda de pacotes devido a congestionamentos na rede.

Todos os clientes deste sistema possuem um *playout buffer* capaz de armazenar até 80 segundos de vídeo. O vídeo a ser transmitido segue o padrão MPEG-1 cuja vazão média de transmissão é de 1,5 Mbps. Desta forma, cada cliente possui um *buffer* de 16 MB e está conectado à rede através de um canal *full duplex* de 155 Mbps (Figura 3.6).

O servidor também está conectado à rede através de um canal de 155 Mbps. Com este canal é possível transmitir em média até 100 fluxos de vídeo MPEG-1. Por isso, supõe-se que o servidor é capaz de prover com folga até 100 canais lógicos. Um canal lógico é constituído de todos os recursos necessários para prover um fluxo de vídeo com uma qualidade pré-especificada, tais como largura de banda da unidade de disco, largura de banda do barramento interno do servidor, largura de banda da conexão, tempo de CPU, etc. Note que um computador *off-the-shelf* de baixo custo é capaz de atender com folga mais de 100 canais lógicos [57].

O objetivo de se usar esses parâmetros é demonstrar que a MCD pode escalar em redes ATM com velocidades relativamente baixas (155Mbps). De fato, o ambiente experimental

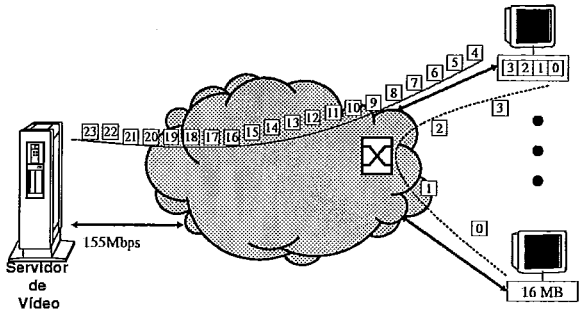


Figura 3.6: Ambiente da simulação

PARÂMETRO	NORMAL	VARIAÇÃO
Tamanho do vídeo (s)	3600	3600 e 7200
Tamanho da unidade de vídeo (2 GoFs)	1 s	-
Largura de banda do servidor (canais lógicos)	100	-
Buffer do cliente (s)	80	-
Taxa de requisição (requisições/min)	-	0.5-60
Número de clientes -	60 - 14400	

Tabela 3.1: Parâmetros usados na simulação da MCD

do protótipo GloVE utilizou um *switch Fast Ethernet* 3COM SuperStack II 3300, com suporte a *multicast* através do protocolo IGMP *Snooping*. Foi medido que o servidor de VoD com um enlace de 100 Mbps tem capacidade de suportar 56 fluxos simultâneos de vídeo MPEG-1 padrão NTSC-SIF (352x240, 29.97 quadros/s) [54]. Isto porque o servidor utilizado possuía um overhead de 0,2 Mbps, totalizando uma vazão de 1,7 Mbps. Para comparação, normalizou-se os resultados.

Cada simulação consistiu de requisições que chegam segundo uma distribuição de Poisson [45], com tempo entre chegadas exponencialmente distribuídos com média $1/\lambda$, onde λ é a taxa de chegadas. A cada chegada é criado um cliente com um identificador que irá consumir o fluxo de vídeo e possivelmente retransmití-lo. Como o objetivo deste estudo é verificar o comportamento da Memória Cooperativa Distribuída, assumir-se-á que o servidor contenha apenas um vídeo.

A carga de trabalho e os parâmetros do sistema estão resumidos na tabela 3.1. Os valores normais da simulação estão listados na coluna "Normal" e alguns parâmetros que são variados na simulação para análise de sensibilidade estão na coluna variação.

Para quantificar o desempenho do sistema foram escolhidas as seguintes variáveis a serem medidas:

- Taxa de bloqueio: é a percentagem de requisições não atendidas por falta de recursos do sistema. É dada pela razão entre o número de clientes que tiveram seus pedidos rejeitados e o total de clientes que requisitaram pedidos de vídeo (atendidos e rejeitados).
- Vazão: é dado pelo número médio de sessões *multicast* simultâneas.
- Taxa de utilização do servidor: é a taxa média de utilização de todos os canais lógicos do servidor. É dada pela razão do somatório do tempo de utilização de cada canal do servidor pelo somatório do tempo total de simulação de cada canal do servidor.

3.2.2 Ferramenta de Simulação

Foi implementado um simulador para avaliar o sistema de VoD que utiliza o conceito de Memória Cooperativa Distribuída. O simulador foi implementado em ANSI C++ e sua estrutura de objetos permite fácil expansão à medida que novas funcionalidades são agregadas ao sistema.

Para cada configuração foi realizada uma simulação com 7200 segundos de duração, o que corresponde ao horário nobre de 20:00 às 22:00. O intervalo de confiança foi conseguido pelo método das replicações independentes com nível de confiança de 90% para um intervalo de mais ou menos 10% em relação à média obtida. As curvas dos gráficos foram interpoladas com o método de Bezier.

3.2.3 Análise da Vazão do Sistema

A **u.v.** da simulação, por razões de simplificação, é composta de dois GoFs (Group of Frames), o que corresponde em média a 1 segundo de exibição de vídeo.

Analisando o gráfico da vazão (figura 3.7), é possível observar que o sistema convencional satura para uma taxa de chegadas superior a 2 chegadas/minuto, isto porque os canais lógicos disponíveis logo se esgotam.

O sistema que utiliza apenas encadeamento apresenta um crescimento linear da vazão em relação à taxa de chegada, ou seja, este sistema não é limitado pela quantidade de canais lógicos oferecidos pelo servidor nem pela largura de banda no enlace de acesso

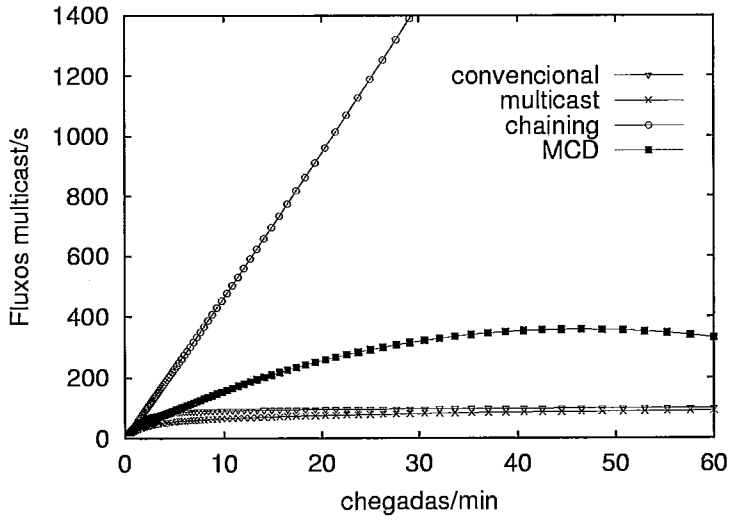


Figura 3.7: Vazão x Taxa de Chegada - vídeo de 60 minutos

do servidor, no entanto, ele exige uma largura de banda crescente da rede. Isto é uma limitação séria para a sua escalabilidade.

O sistema que usa o *multicast* usa menos largura de banda que o sistema convencional, pois consegue atender um ou mais usuários com um mesmo fluxo. Esta aparente vantagem desaparecerá quando analisarmos adiante a carga de trabalho do servidor de VoD.

Já o MCD, à medida que a taxa de chegadas aumenta, sua vazão aumenta até 45 chegadas/min, a partir deste ponto ele apresenta uma tendência de queda, pois passa a explorar a enorme quantidade de memória coletiva e os fluxos de *multicast* já existentes, o que confirma sua potencialidade em se conseguir um sistema de VoD escalável.

3.2.4 Análise da Taxa de Utilização do Servidor

Observando-se o gráfico da figura 3.8 nota-se que tanto o sistema convencional como o de *multicast* impõem uma carga de trabalho significativa no servidor, isto com apenas um vídeo. Este detalhe é importante porque o vídeo considerado tem 60 minutos de duração, ou seja, com 100 canais lógicos ele consegue atender todos os usuários, levando-se em conta um intervalo de 35 segundos. No entanto, na figura 3.9 é possível verificar que para um vídeo de 2 horas de duração, o *multicast* rapidamente consome todos os recursos do servidor com o aumento da taxa de chegada, acarretando uma alta taxa de bloqueio, enquanto a taxa de bloqueio do MCD permanece em zero.

Já o MCD diminui a taxa de utilização do servidor com o aumento da taxa de chegadas, sendo que seu desempenho tende a ser igual ao do *chaining*, com a vantagem de usar muito menos largura de banda. A taxa de utilização do servidor cai para menos de 1%

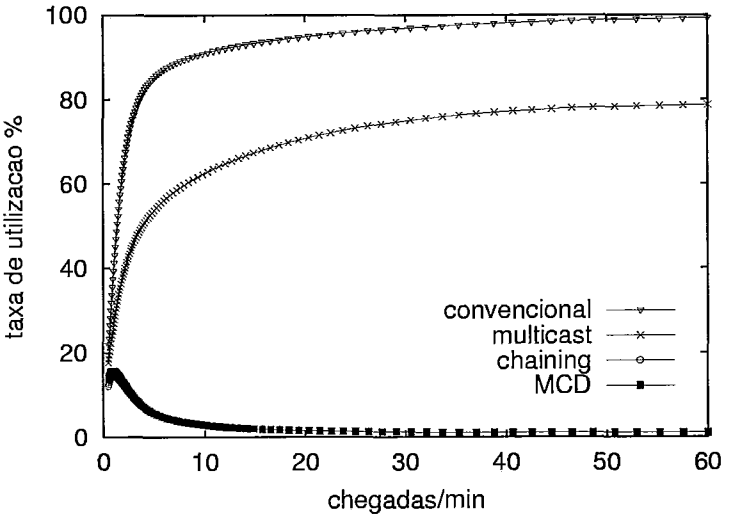


Figura 3.8: Taxa de Utilização x Taxa de Chegada - vídeo de 60 minutos

quando a taxa de chegadas é maior que 10 chegadas/minuto, o que implica em dizer que o sistema que utiliza o multicast consegue garantir a escalabilidade, em média, para apenas um vídeo e o MCD para vários vídeos, isso usando um servidor de baixo custo.

3.2.5 Comparação da largura de banda normalizada

Para se comparar as simulações obtidas com a MCD e com o protótipo GloVE [55, 56], normalizou-se os resultados obtidos em termos da largura de banda. Observando a figura 3.10 nota-se que o protótipo teve um desempenho melhor que a MCD. Isto se deve às otimizações realizadas no GloVE, que implementa a árvore de *playout buffers* de forma a maximizar o uso do *multicast*, que não estavam presentes na simulação da MCD original.

3.3 Discussão

Este capítulo apresentou a MCD, uma nova abordagem para se explorar o espaço de armazenamento existente nos clientes VoD em proveito do sistema de VoD. Os resultados obtidos através de detalhadas simulações, confirmados por experimentos reais com o GloVE, indicam que o uso da MCD tem um grande potencial para melhorar o desempenho e a escalabilidade de um sistema de VoD. A MCD é nitidamente melhor que os esquemas propostos anteriormente. As técnicas de gerência da MCD pesquisados até o momento são simples, apesar disto apresentam um desempenho melhor que um sistema de VoD convencional.

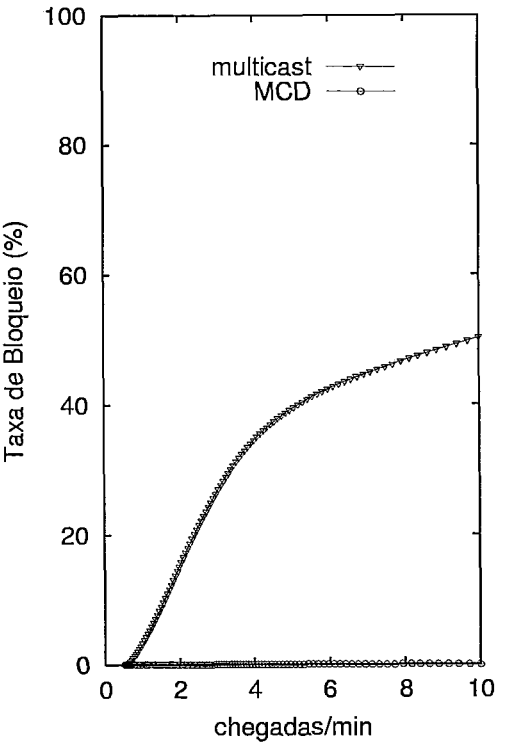


Figura 3.9: Taxa de Bloqueio x Taxa de Chegada - vídeo de 120 minutos

É importante ressaltar que a MCD permite a diminuição do custo do servidor de VoD e do enlace que o liga à rede, tornando viável a disponibilização de vídeo em grande escala com poucos recursos. Na verdade o que torna a MCD escalável é a quantidade de clientes que estão utilizando o sistema. Quanto mais clientes, mais memória disponível, melhor a qualidade do serviço e maior a eficácia do sistema.

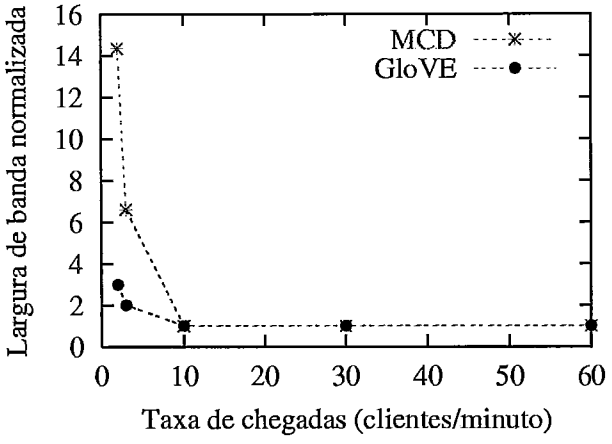


Figura 3.10: Largura de Banda Normalizada x Taxa de Chegada

Capítulo 4

Memória Cooperativa Colapsada

O estudo da Memória Cooperativa Distribuída, apresentada no capítulo anterior, revelou suas vantagens. Como vantagem principal, podemos notar a maior escalabilidade do sistema em relação aos sistemas anteriores. Como principal desvantagem, a quantidade de tráfego no *backbone* da rede de distribuição devido ao tráfego *ping-pong*[1] entre os clientes, o que impede a sua utilização em MANs e WANs. Com o objetivo de criar um sistema que agregasse as vantagens da MCD sem suas desvantagens, i.e., tornar possível seu uso em redes geograficamente maiores que uma LAN, pensou-se num esquema que agregasse a memória dos clientes em um ponto de acesso, o proxy, de tal forma que as vantagens da MCD fossem utilizadas também em uma WAN.

A primeira abordagem, a MCD, é baseada em um modelo peer-to-peer para distribuição de conteúdo. A vantagem desta abordagem é que ela não precisa de uma infra-estrutura pré-definida para transmissão e o armazenamento do conteúdo. Esta infra-estrutura é estabelecida de acordo com a demanda, com a chegada de novos clientes que vão agregando recursos a esta rede de distribuição de conteúdo (CDN) virtual. A segunda abordagem que propomos, a MCC, prevê a existência de uma rede de distribuição de conteúdo pré-estabelecida, conforme mostra a figura 4.1, com um *backbone* de rede que liga os servidores a diversos proxies na borda da rede, entre a rede de distribuição de conteúdo e a rede de acesso. Os proxies são encarregados de armazenar o conteúdo mais popular, evitando a sua retransmissão. A MCC, portanto, foi projetada para uma CDN real e funciona como um gerenciador da memória dos proxies.

O objetivo deste capítulo é explicar em detalhes o funcionamento da MCC, bem como o desempenho obtido de sua aplicação através de simulação.

[1] Tráfego entre clientes em pontos opostos da rede

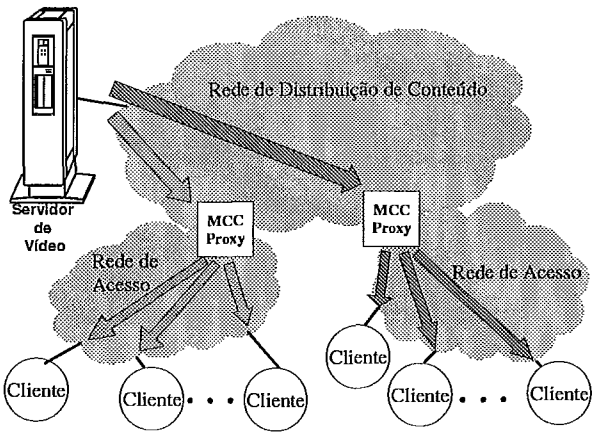


Figura 4.1: Arquitetura da MCC

4.1 Da MCD para MCC

A idéia da MCC é utilizar os princípios da MCD na sua implementação, preservando suas vantagens de escalabilidade, eliminando suas desvantagens, ou seja, reduzindo o tráfego no *backbone* da rede.

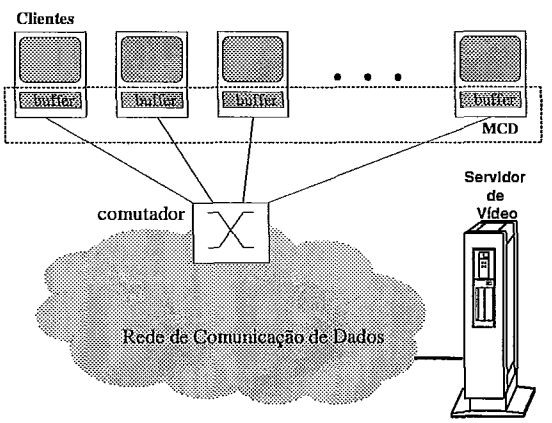


Figura 4.2: Arquitetura da MCD

A figura 4.2 mostra novamente a arquitetura da MCD, com seus *playout buffers* fazendo parte de uma única área de memória global, a ser utilizada pelo sistema de forma cooperativa, ou seja, sempre que possível, seu conteúdo poderia ser compartilhado com outros usuários. Esquemas de gerenciamento de memória distribuída costumam ser mais complexos do que um sistema centralizado. Além disso, os recursos dos clientes nem sempre estão a disposição da MCD. Já em um esquema centralizado, clientes solicitam recursos ao sistema, que ao serem liberados podem ser utilizados em proveito

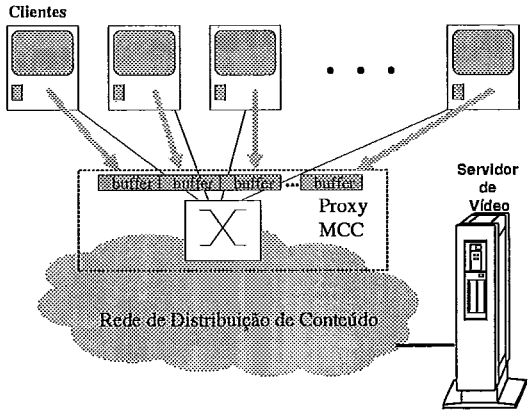


Figura 4.3: Colapsando a MCD

de outras atividades ou outros usuários. Devido a estas possibilidades de um sistema mais centralizado, visualizou-se a MCC como um simples deslocamento de parte do *play out buffer* dos clientes para um *proxy* (figura 4.3).

Existem basicamente duas maneiras de se implementar a gerência de memória de um *proxy* com *buffers* colapsados. A primeira, a mais fácil e direta, é através de *ring buffers*. *Ring buffers* nada mais são do que *buffers* implementados em uma estrutura de vetor circular. À medida que os clientes requisitem vídeos, *ring buffers* são alocados na memória do *proxy*. Para evitar a duplicação de conteúdo, pedidos próximos de um mesmo vídeo são servidos por um mesmo *ring buffer*. Além disso, *ring buffers* podem ter seu tamanho aumentado para atender mais pedidos próximos uns dos outros. Um óbice desta abordagem é que as implementações existentes não tratam os *ring buffers* de um *proxy* de forma global, procurando otimizar a utilização do espaço de memória e da largura de banda da rede, implementando operações, por exemplo de concatenação e de divisão de *ring buffers*. Talvez esta lacuna se deva ao fato de que operações de alocação, desalocação e expansão de *ring buffers* não necessitem de estruturas complexas que façam relacionamentos com o conteúdo dos diversos *ring buffers*. Por exemplo, para se alocar um *ring buffer*, expandí-lo ou desalocá-lo não há necessidade de verificar o estado dos outros *ring buffers*.

Já a segunda abordagem, a qual defendemos nesta tese, ao invés de lidar com diversos objetos *ring buffers*, utiliza um único objeto servidor de *buffers* que irá gerenciar todo o espaço de armazenamento do *proxy*. Para isto, este servidor de *buffers* possui uma estrutura de dados específica que permitirá não só a alocação do *buffer* colapsado, mas sua gerência de forma integrada com outros *buffers* colapsados, permitindo além das

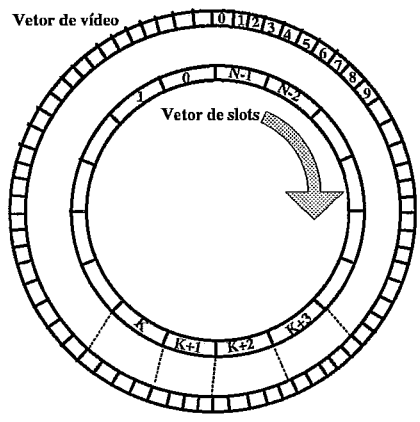


Figura 4.4: Vetor de *slots* e vetor de video

tradicionais operações com *ring buffers*, também operações de divisão de *buffers*, entre outras, para melhor administrar a memória do *proxy*.

4.2 Estrutura de Dados da MCC

A MCC mantém duas estruturas de dados que guardam meta-informações de cada vídeo armazenado no sistema: o vetor de vídeo e o vetor de *slots*. Estas duas estruturas são implementadas como vetores circulares que trabalham de forma coordenada, como mostra a figura 4.4. O vetor de vídeo mapeia as unidades de vídeo que compõem um fluxo de vídeo completo seqüencialmente. Ele contém somente os metadados do vídeo necessários para que a MCC acesse e gerencie a sua *cache*. O vetor de *slots* consiste de *slots* de tamanho fixo em unidades de tempo, que em conjunto compreendem toda a duração do vídeo.

Uma unidade de vídeo (**u.v.**) é a menor quantidade de vídeo que a MCC consegue manipular e gerenciar. Para mapear as **u.v.** na *cache*, cada unidade do vetor de vídeo contém um *timestamp* da **u.v**, seu tamanho e um ponteiro para a posição de memória em que está armazenado, caso esteja em *cache*. Toda vez que uma **u.v** tem seu status de prioridade alterado na *cache*, seu correspondente valor no vetor de vídeo é atualizado.

O vetor de *slots* serve para alocar *buffers* colapsados (figura 4.5) no *proxy*. Um *buffer* colapsado pode ser esquematicamente dividido em 4 partes. Cada parte é limitada por ponteiros que indicam o início (B), o fim (E), o limite entre a área de escrita e a de risco de overflow (O), o limite entre a área de escrita e a de risco de underflow (U) e o limite entre a área de risco de underflow e área de unidades de vídeos já lidas (R) do buffer. Cada

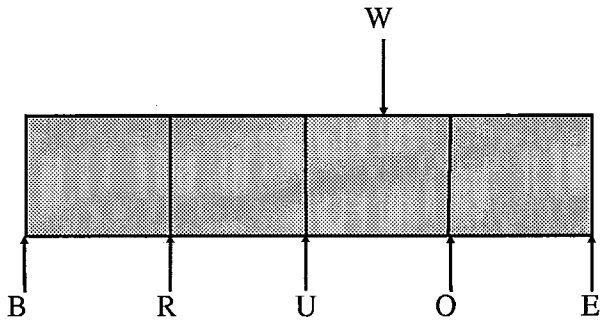


Figura 4.5: Esquema de um *buffer* colapsado

parte dessas, respectivamente nomeada como BS, RS, WS e ES é mapeada diretamente no vetor de *slots*, como mostra a figura 4.6.

Os *slots* do vetor de *slots* que pertencem a um vetor colapsado têm o seu conteúdo reservado na *cache*, ou seja, se estão na *cache* não podem ser descartados. Para este controle, o vetor de *slots* gira sincronamente em sentido horário em relação ao vetor de vídeo na mesma velocidade da exibição do vídeo. À medida que gira, algumas unidades de vídeo entram e outras saem do *buffer* colapsado. As que entram e têm seu conteúdo na *cache* são marcadas com prioridade 2, i.e, não podem ser descartadas. As que saem, voltam a ter prioridade 1, ou seja, presentes na *cache*, mas não reservadas. A prioridade 0, indica que a unidade de vídeo não está na *cache*. Obviamente, o fluxo de vídeo que chega do servidor e é colocado na posição indicada pelo ponteiro W, por estar dentro do *buffer* colapsado, tem sua prioridade atualizada para 2. Este controle é feito pelos ponteiros de controle do buffer colapsado, respectivamente B, R, U, O e E. Estes ponteiros coincidem com os limites dos *slots*. Por exemplo, toda vez que uma unidade de vídeo cruza o ponteiro de fim do *slot* BS, significa que ela está cruzando o ponteiro B do *buffer* colapsado e, portanto, está saindo do *buffer* e, logo, precisa ter sua prioridade atualizada para 1.

O ponteiro de *overflow* O, detecta se o ponteiro W cruzou seus limites, ou seja, se a taxa de escrita no *buffer* colapsado está mais rápida que a de consumo. Isto é feito testando-se as unidades de vídeo que cruzam o ponteiro O. Se a prioridade for 2, significa que W ultrapassou O e portanto existe um risco de *overflow*. Desta forma a MCC tem como gerar um sinal de alerta para o sistema, reduzindo ou interrompendo o fluxo que alimenta este *buffer* colapsado. De foma análoga trabalha o ponteiro de *underflow* U. O

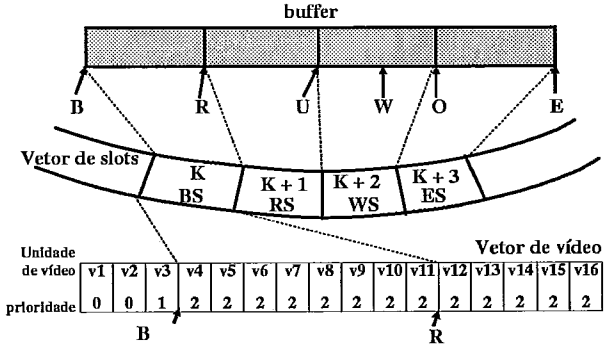


Figura 4.6: Projeção do Vetor de *Slot* sobre o Vetor de Vídeo

mais importante, é que este mecanismo de alarme permite que a MCC interaja com o servidor antecipadamente, permitindo o controle de fluxo do vídeo e conseqüentemente a QoS. Especificamente, os sinais de alarme permitem que o *proxy* antecipe reações de pedir mais unidades de vídeo ou de diminuir seu fluxo antes que sua falta ou excesso ocasionem perdas na exibição do mesmo.

4.3 Usando a estrutura de dados da MCC

Com a estrutura de dados apresentada anteriormente é possível gerenciar a memória do proxy. A questão é como implementar as operações de alocação, desalocação, monitoramento de ponteiros, etc na MCC. Mais precisamente, é preciso estabelecer um método para descobrir, em qualquer instante, qual *slot* mapeia determinado trecho de vídeo. Com esta informação pode-se alocar *buffers* colapsados. Outro método de suma importância é, dado um *slot*, descobrir onde estão seu limites iniciais e finais, ou seja, para quais unidades de vídeo os ponteiros dos *buffers* colapsados estão apontando.

A MCC usa os *timestamps* das unidades de vídeo para localizar precisamente a quais *slots* pertencem. A cada primeiro pedido para um determinado vídeo no proxy, MCC cria a dupla estrutura de vetores circulares e marca o tempo em que um vetor começou a se deslocar sincronamente em relação ao outro, sendo este tempo ST. Suponha um pedido para o mesmo vídeo que tenha chegado no tempo CT solicitando o vídeo a partir de uma unidade de vídeo com *timestamp* UTS. Com estes dados precisa-se determinar o slot RS do buffer colapsado que será alocado na MCC para fornecer o vídeo ao cliente. O slot RS correponde ao primeiro *slot* do *buffer* colapsado cujo conteúdo será transmitido

ao cliente. Ou seja, a unidade de vídeo com *timestamp* UTS deve estar neste *slot*. Para isto, calcula-se inicialmente a posição do vídeo subtraindo-se ST de CT mais a posição do trecho do vídeo requisitado dado pelo seu *timestamp* UTS. Em seguida, divide-se este valor pelo tamanho do *slot* SL, e como o vetor de *slots* pode ter dado várias voltas, divide-se (módulo - %) este valor pelo número de *slots* NS do vetor de *slots*. Portanto, temos a seguinte fórmula para calcular RS:

$$RS = [(CT - ST + UTS)/SL]\%NS \tag{4.1}$$

Uma vez determinado RS, achar os demais *slots* do *buffer* colapsado é simples. Por exemplo, para achar os limites do *slot* RS: Seja PT o período de uma rotação que coincide com a duração do vídeo. Dado ST, CT, PT e o número do *slot* RS calculado em 4.1, o número de voltas NC dada pelo vetor de *slots* pode ser calculado com a seguinte expressão:

$$NC = [CT - ST - (RS * SL)]/PT \tag{4.2}$$

O início do *slot*, IS, e término do *slot*, TS, são dados por:

$$IS = CT - ST - NC * PT - RS * SL \tag{4.3}$$

e

$$TS = IS - SL \tag{4.4}$$

Ambos, IS e TS retornam valores relativos ao início do vídeo. Como as unidades de vídeo podem ser acessadas tanto pelo *timestamp* como pelo seu número seqüêncial, tabelado no vetor de vídeo, a MCC tem como descobrir as unidades de vídeo apontadas pelos ponteiros de controle dos *slots* ES, WS e BS.

É importante ressaltar que as operações de atribuição de *slots* e monitoramento de ponteiros são freqüentes, e por isso, os algoritmos que a implementam devem ser muito eficientes. Com as fórmulas acima, essas operações são realizadas em $O(1)$, ou seja, não importa o tamanho do vídeo, a quantidade de slots ou qualquer outra variável, o tempo da operação sempre será constante.

4.4 Funcionamento da MCC

Nesta seção, descreve-se suscintamente como a MCC implementa operações básicas entre os *buffers* colapsados utilizando a estrutura de dados apresentada.

A figura 4.7 mostra como a MCC alocaria os *buffers* colapsados correspondentes a pedidos que estariam em *slots* vizinhos[2]. *Slots* que não pertencem a um *buffer* colapsado são denominados *slots* livres (FS).

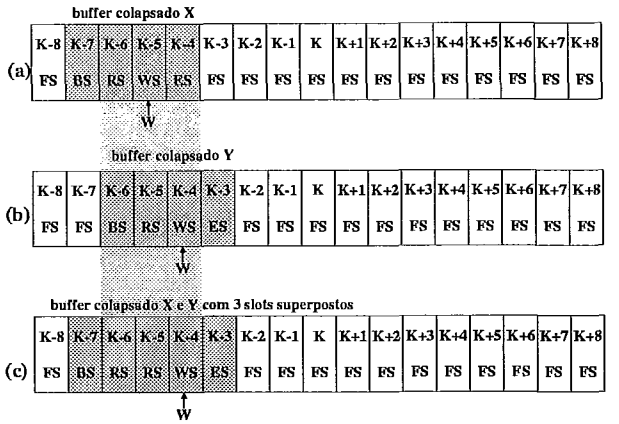


Figura 4.7: Superposição de *buffers* colapsados para pedidos em *slots* vizinhos

A figura 4.8 mostra como a MCC alocaria os *buffers* colapsados correspondentes a pedidos que estariam em *slots* distintos, separados por um *slot*. Note que o *slot* que os separa, apesar de fazer parte do *buffer* colapsado é denominado *slot* de ligação (LS). O *slot* LS se distingue dos outros tipos de *slot*, por manter o seu conteúdo reservado, e seus limites não correponderem a nenhum ponteiro de controle, ou seja, do jeito que a unidade de vídeo entra em LS ela sai, sem alterar suas prioridades.

A figura 4.9 mostra como a MCC alocaria os *buffers* colapsados correspondentes a pedidos que estariam em *slots* distintos, separados por dois *slots*.

A figura 4.10 mostra como a MCC alocaria os *buffers* colapsados correspondentes a pedidos que estariam em *slots* distintos, separados por três *slots*. Ou seja, na verdade os *buffers* colapsados não se superpõem, mas eles se justapõem.

A figura 4.11 mostra como a MCC alocaria os *buffers* colapsados correspondentes a pedidos que estariam em *slots* distintos, separados por quatro *slots*. Neste caso, os *buffers* colapsados estão separados por um *slot* e um dos *slots* de ligação, LS, não pertencerá

[2]O ponteiro R do *slot* RS do *buffer* colapsado atende os pedidos dos clientes com um fluxo de vídeo. Portanto a distância entre pedidos pode ser dada pela distância entre slots RS dos buffers colapsados.

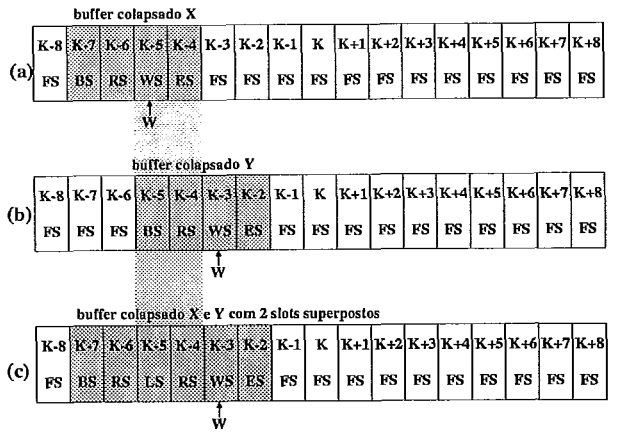


Figura 4.8: Superposição de *buffer* colapsados para pedidos em *slots* distintos separados por um *slot*

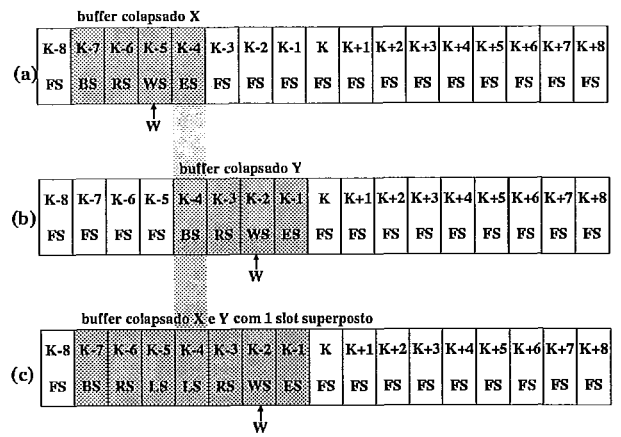


Figura 4.9: Superposição de *buffer* colapsados para pedidos em *slots* distintos separados por dois *slots*

a um *buffer* colapsado, mas mesmo assim seu conteúdo será reservado. Este *slot* está hachurado para diferenciá-lo dos *slots* LS que pertencem a *buffers* colapsados.

Por fim, a figura 4.12 mostra como a MCC alocaria os *buffers* colapsados correspondentes a pedidos que estariam separados por sete *slots*. Este exemplo mostra que o encadeamento de *buffers* colapsados pode-se estender enquanto exista memória disponível no *proxy*. A figura 4.12-c também mostra que, se for necessário, os slots LS alocados podem ser removidos para liberar espaço de memória em cache.

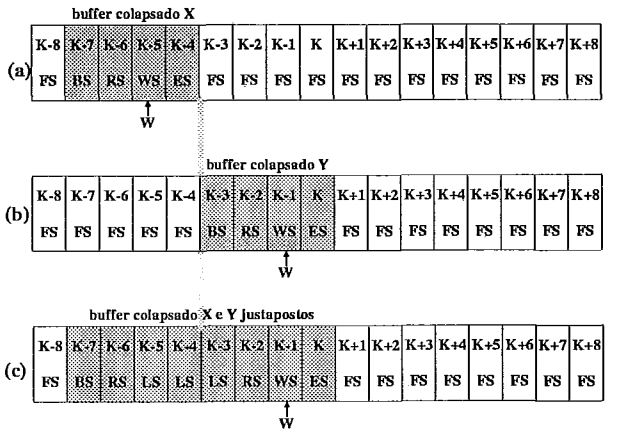


Figura 4.10: Superposição de *buffers* colapsados para pedidos em *slots* distintos separados por três *slots*

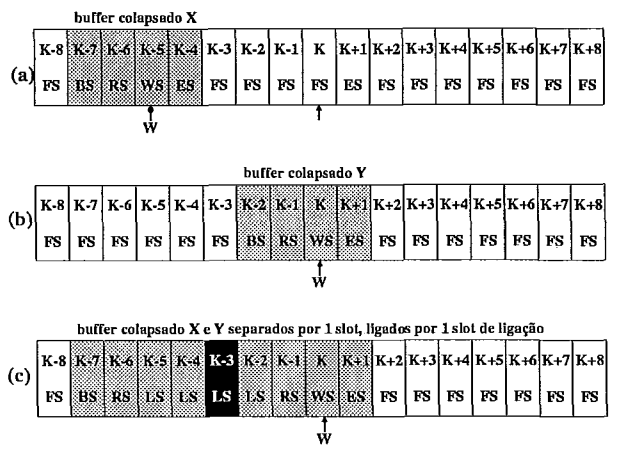


Figura 4.11: Encadeamento de *buffers* colapsados para pedidos em *slots* distintos separados por quatro *slots*

4.5 Política de substituição na *cache* do *proxy*

O conteúdo sob controle dos *buffers* colapsados e dos *slots* *LS* são reservados e não podem ser descartados. No entanto, quando não existir mais espaço de memória na *cache* para continuar armazenando as unidades de vídeo que chegam dos diversos fluxos vindos do servidor, unidades de vídeo na *cache* precisam ser descartadas para dar lugar as novas unidades que chegam. As unidades de vídeo que não estão reservadas, i.e., com prioridade 1, são descartadas a começar pelos vídeos menos populares e entre unidades de vídeo pertencentes a um mesmo vídeo, as que estão no final do vídeo têm maior prioridade de descarte. Desta forma, se houver espaço em *cache*, a tendência é preservar os prefixos dos vídeos mais populares.

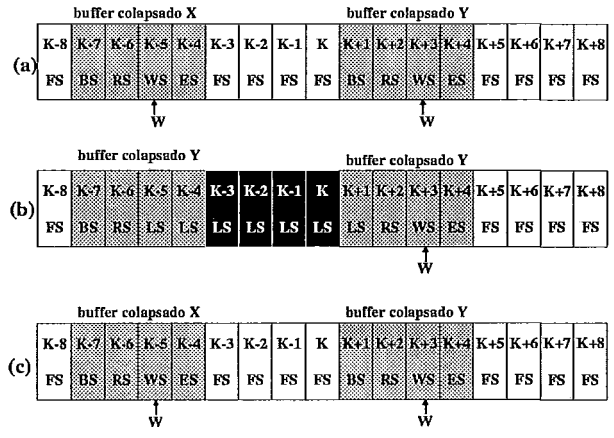


Figura 4.12: Encadeamento de *buffers* colapsados para pedidos em *slots* distintos separados por sete *slots*

Por outro lado, o conteúdo do *proxy* pode estar todo reservado por *buffers* colapsados e *slots* de ligação. Isto ocorre em horários de pico, em que vários usuários estão acessando o sistema, ocasião em que o *proxy* é exigido ao máximo. Nesta situação, resta à MCC liberar *slots de ligação*, desfazendo o encadeamento dos buffers colapsados, liberando *slots LS* e em troca estabelecendo um novo fluxo para substituir o conteúdo reaproveitado dos *slots LS*.

A figura 4.13 mostra com um exemplo o funcionamento da MCC, desde a chegada dos primeiros pedidos até o momento em que a *cache* fica cheia e o algoritmo de substituição da MCC tem que entrar em ação. A figura 4.13-a mostra o estado inicial de um vetor de *slot*, nos quais todos os *slots* estão livres. Em seguida, a primeira requisição chega ao *proxy*, e um instantâneo do vetor de *slot* mostra o primeiro *buffer* colapsado alocado (figura 4.13-b). A seta mais grossa indica o fluxo de vídeo regular vindo do servidor de VoD para alimentar o *buffer* colapsado e a seta mais fina indica o deslocamento do vetor de *slot* para a direita, sobre um vetor de vídeo fixo, não mostrado na figura, cujo começo se encontra no lado esquerdo da figura e cujo fim do lado direito da mesma.

A figura 4.13-c mostra o vetor de *slots* três *slots* de tempo depois. Nesta figura, observa-se que o prefixo do vídeo que saiu do *buffer* colapsado, e está sombreado com um tom diferente de cinza, não foi descartado. Desta forma, se um pedido chegar logo em seguida, não é necessário requisitar um fluxo de *patching* do servidor para recompletar o conteúdo que poderia ter sido descartado e faria falta num possível encadeamento.

A figura 4.13-d mostra a chegada de mais um pedido, com a conseqüente alocação de mais um *buffer* colapsado que é encadeado com o primeiro, não necessitando de um novo

fluxo regular do servidor, pois o prefixo foi mantido em *cache*. O tamanho desta parte inicial do prefixo será avaliada no capítulo seguinte.

A figura 4.13-i mostra o estado do vetor de *slots* pouco antes de chegar um novo pedido e, neste exemplo, considera-se que a *cache* está cheia e com todo o seu conteúdo reservado, i.e., *slots* LS terão que ser liberados para alocar um novo *buffer* colapsado e, se houver mais *slots* de ligação a serem descartados, este novo *buffer* colapsado deverá ser encadeado com a cadeia já existente, evitando-se um novo fluxo regular do servidor. Ou seja, há a necessidade de quatro *slots* para o *buffer* colapsado mais um para o *slot* LS, totalizando a necessidade de se liberar cinco *slots*. Existem três alternativas. A mais simples é recusar o pedido deste novo cliente, o que causaria um bloqueio deste pedido. Como a MCC tem como atender estes pedidos, liberando *slots* LS, isto a torna mais flexível dos que os esquemas de *ring buffers* existentes, em que não é possível gerenciar o espaço de memória dos *ring buffers* já alocados.

O exemplo mostra as duas outras alternativas de liberação de *slots* LS. A alternativa da figura 4.13-j1, libera uma seqüência de seis *slots* LS, dos quais cinco são usados. Além disso, um novo fluxo regular do servidor é estabelecido para alimentar a cadeia à esquerda da figura. Este fluxo terá o custo de sua largura de banda vezes a sua duração, ou seja, vinte *slots*, que é o quanto falta para chegar ao final do vídeo. A alternativa da figura 4.13-j2 libera exatamente cinco *slots*, no entanto usa duas seqüências de *slots* LS, uma de dois *slots* e outra de três *slots*. Neste caso, o custo da substituição foi de dois fluxos regulares, um que durará vinte e cinco *slots* e outro que durará nove *slots*. Supondo uma largura de banda unitária, o custo total seria de trinta e quatro *slots*, muito superior aos vinte *slots* da primeira opção. Logo a primeira opção é a melhor. Este exemplo deixa bem claro, que sempre que possível, é melhor liberar a menor quantidade de sequências de *slots* LS. Também fica claro no exemplo, que o custo de uma seqüência de *slots* LS é dada pelo seu tempo de uso até o final do vídeo, ou seja, é preferível descartar seqüências grandes, no entanto, isto se contrapõe ao fato de que pode haver seqüências menores cujo custo de substituição por um fluxo regular é menor. Como este exemplo simples demonstra, a alocação e liberação de *slots* LS é um problema de otimização.

4.6 Fomulação do problema de liberação de *slots* de ligação (LS)

O problema de substituição de *slots* de ligação pode ser formulado da seguinte forma:

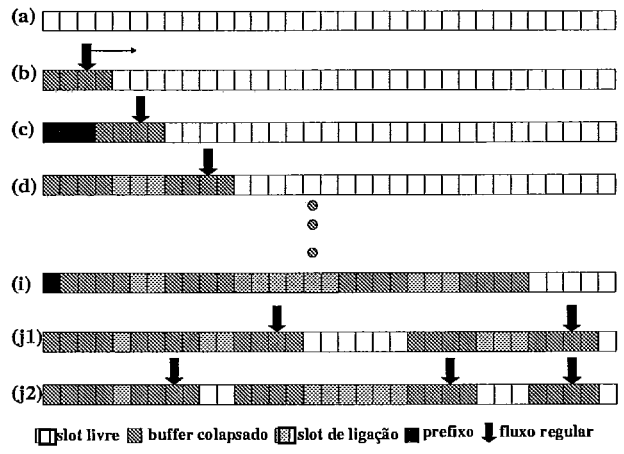


Figura 4.13: Problema de substituição de *slots* de ligação

Instância do problema: Uma coleção de seqüências de *slots* LS, o número de *slots* LS que precisa ser liberado e o custo de ser liberar cada seqüência de *slots* LS.

Questão: Existe uma combinação de sequências de *slots* LS que satisfaça simultaneamente a três condições:

1. a soma do tamanho das seqüências de *slots* LS a serem liberados é maior ou igual ao número de *slots* que tem que ser liberados?

2. o custo total das seqüências liberadas é menor do que um valor dado d?

3. o número de sequências de *slots* LS é o menor possível?

Para provar que este problema é NP-completo, reformular-se-á o problema original na forma de seu dual, como se segue. Suponha que uma seqüência de *slots* LS corresponda à um buraco de tamanho em número inteiro de *slots* que deve ser preenchido. Cada buraco tem um custo igual à sua distância d até o fim do vídeo. A distância d é dada em número de *slots*. A quantidade de *slots* a ser liberada corresponde à mesma quantidade de fichas e cada ficha tem o tamanho de um buraco, ou seja, um *slot*.

A figura 4.14 exemplifica a transformação do problema de substituição de *slots* de ligação no seu problema dual. A figura 4.14-i mostra o mesmo estado do vetor de slots da figura 4.13-i, no momento em que chega mais um cliente. A figura 4.14-k mostra a necessidade de se reservar mais 5 *slots*. Quatro *slots* para alocar um *buffer* colapsado e um *slot* de ligação. O vetor de *slots* possui três seqüências de *slots* a, b e c alocadas. Como a *cache* está cheia, uma ou mais das três seqüências deverão ser liberadas. O dual deste problema é mostrado na figura 4.14-k. A quantidade de *slots* a ser liberada é representada

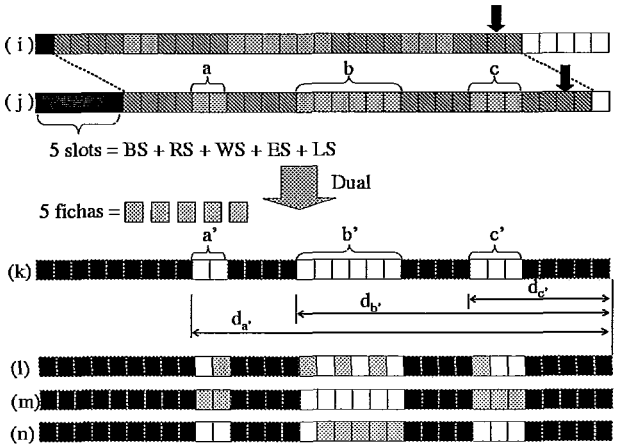


Figura 4.14: Dual do problema de substituição de *slots* de ligação

por fichas (5 fichas pertas na figura) e os slots que podem ser liberados são representados por buracos (quadrados em branco na figura). Em cada buraco só cabe um número inteiro de fichas. No exemplo existem três buracos, a', b' e c'. Cada buraco dista do fim de uma distância de $d_{a'}$, $d_{b'}$ e $d_{c'}$ O problema agora é arranjar as 5 fichas nos três buracos de forma que a soma das distâncias dos buracos com pelo menos uma ficha seja mínima e o número de buracos com uma ou mais fichas também seja mínimo. As figuras 4.14-l, 4.14-m e 4.14-n mostram três arranjos possíveis para o problema dual. A solução que minimiza tanto a distância dos buracos como o número de buracos com fichas é a figura 4.14-n. Desta forma o problema dual pode ser formulado da seguinte forma:

Instância do problema dual: Um conjunto S de buracos com tamanho inteiro, f fichas com tamanho unitário, b o número de buracos e d o custo total dos buracos com pelo menos uma ficha. Cada buraco tem um valor dado por sua distância à um ponto definido.

Problema dual: Colocar as f fichas no menor número de buracos possível, cuja soma das distâncias tenha um valor mínimo?

Solução: S', subconjunto de S que contenha pelo menos uma das f fichas. Minimizar a cardinalidade de S'. Minimizar a distância total dos buracos ao fim.

Problema de decisão: Existe um subconjunto S' de S tal que sua cardinalidade seja menor que um valor b' e a soma das distância dos buracos à um ponto seja menor que um valor d'?

O problema assim formulado corresponde a um problema NP-completo bem conhecido, o SR7 - Alocação de Capacidades (Capacity Assignment) [63]. Portanto, obter sempre a solução ótima é muito custosa, principalmente quando se trabalha com

restrições de tempo real, no caso um sistema de VoD, no qual se deseja uma política de substituição dinâmica de *slots LS* implementada por um algoritmo rápido e eficiente.

4.7 O algoritmo de substituição de *slots* de ligação (LS)

Seja f o número de *slots* que precisam ser liberados para alocar um *buffer* colapsado incluindo os *slots LS* para encadeá-lo, se for necessário. Seja t o tamanho da seqüência de *slots LS* e d distância desta seqüência. Define-se como custo de uma seqüência LS uma função $F(d, f, t)$ igual a razão $d/minínimo\{f, t\}$. O objetivo é minimizar a soma de F para as diversas seqüências escolhidas. Intuitivamente, um valor mínimo de F corresponde à uma seqüência de *slots LS* perto do final do vídeo e que será substituído por um fluxo regular do servidor por um pequeno período de tempo. Além disso, procura-se não valorizar em excesso seqüências muito longas, que se escolhidas podem levar a distorções como a escolha de uma seqüência de *slots LS* distante do final do vídeo, mas que teve o valor F reduzido pelo seu tamanho t. Estabelecendo como divisor o mínimo entre f e t, valoriza-se seqüências que estejam próximas do fim e ao mesmo tempo possuam um tamanho maior ou igual a t.

Desta maneira, procura-se evitar selecionar uma seqüência de *slots LS* cuja substituição seja muito cara (fluxo regular por um longo período), mesmo que ela libere muitos slots, no lugar de uma seqüência mais barata (substituída por um fluxo regular de curta duração).

```
getFreeSlot( f ) {
        freeSlots=0;

        while( freeSlots < t or queue.empty()==TRUE) {
                linkSlotSequence=queue.getFirst();
                freeSlots += linkSlotSequence.size;
                freeSlot(linkSlotSequence);
        }
        return freeSlots;
}
```

Figura 4.15: Algoritmo de substituição de *slots LS*

A implementação do algoritmo de substituição usa um algoritmo guloso que ordena as seqüências de *slots LS* por seu valor $F(d, f, t)$. Este algoritmo, mostrado na figura 4.15, é sempre executado toda vez que a *cache* está cheia e há a necessidade de se remover

slots LS para abrir espaço na *cache*. A complexidade do algoritmo está relacionada à complexidade de se inserir e remover seqüências de *slots* LS de uma fila de prioridades, que tem complexidade de $O(logN)$ [93].

4.8 Discussão

Este capítulo apresentou a MCC, uma nova abordagem para gerenciar a *cache* de *proxies* de vídeo dinâmicas. Esta abordagem colapsa parte do *playout buffer* do cliente de VoD no *proxy* ao qual está ligado. Com isto, forma a *cache* a ser gerenciada pelo *proxy*. Isto é, equivale a dizer que a *cache* do *proxy* complementa o *playout buffer* do cliente trabalhando de forma cooperativa com este. Abordagens tradicionais de gerenciamento de *cache* que simulam *playout buffers* dos clientes para gerenciar a *cache*, denominados *ring buffers*, instanciam estes *buffers* na *cache* do *proxy* independentemente uns dos outros. MCC criou uma estrutura de dados original para instanciar *buffers* colapsados que permite o controle do relacionamento temporal entre os mesmos. Além disso, a modelagem do problema utilizando esta estrutura de dados permitiu mostrar que o problema de substituição de slots de ligação é NP-completo. A partir desta constatação, projetou-se um algoritmo guloso de substituição de *slots* de ligação. O capítulo seguinte apresenta os resultados obtidos na simulação da MCC, que mostram a melhora no desempenho da MCC utilizando-se o algoritmo proposto.

Capítulo 5

Avaliação Experimental da MCC

5.1 Metodologia

Para avaliar o desempenho da MCC e poder ter uma base de comparação com propostas anteriores, implementou-se três versões da MCC, a **LS-**, a **LS** e a **LS+**. A versão **LS-** que não utiliza *slots LS*; a versão **LS** que utiliza *slots LS*, mas não possui política de substituição para esses *slots*; e a versão mais completa **LS+** que possui política de substituição para *slots LS*. Embora a comparação direta com os resultados de trabalhos anteriores não ser possível, devido à falta de dados para reproduzir os experimentos, depreende-se da literatura que as versões **LS-** e **LS** aproxima-se bastante dos esquemas ditos ring buffers [60, 16, 67, 10, 83, 85].

5.1.1 Simulação

Para avaliar o desempenho da MCC na entrega de *streaming* de vídeo sob demanda utilizou-se o simulador NS-2 [58]. As simulações usaram um *trace* da primeira hora do filme Platoon de Oliver Stone, cuja definição corresponde ao formato *wide-screen* qualidade DVD com compressão MPEG-2 e uma taxa média de transmissão de 8 Mbps. Supôs-se que o servidor de vídeo está conectado ao *proxy* MCC através de uma CDN, cujos enlaces até o *proxy* conseguem sustentar até 1000 fluxos simultâneos de vídeo. Cada cliente é conectado ao proxy através de um enlace ADSL (*Assimetrical Digital Subscriber Line*), cuja largura de banda de descida é o suficiente para um fluxo de vídeo com esta qualidade.

Para quantificar o desempenho do sistema foram escolhidas as seguintes variáveis a serem medidas:

- Taxa de bloqueio: é a percentagem de requisições não atendidas por falta de recursos do sistema. É dada pela razão entre o número de clientes que tiveram seus

pedidos rejeitados e o total de clientes que requisitaram pedidos de vídeo (atendidos e rejeitados).

- Taxa de utilização: é a taxa média de utilização do servidor. É dada pela razão do somatório do tempo de utilização de cada canal do servidor pelo somatório do tempo total de simulação de cada canal do servidor. Como só existe um servidor e um *proxy*, esta medida é igual a taxa de ocupação da largura de banda no *backbone* da CDN.

- Taxa de pico de utilização: é a maior taxa de utilização atingida no servidor em qualquer tempo da simulação. É contabilizado contando-se o número de canais ocupados do servidor ao fornecer um novo fluxo de vídeo. Se este número for maior que um registrado anteriormente, o valor do número máximo de canais ocupados é atualizado. Ao final da simulação o número máximo de canais ocupados é dividido pelo número total de canais do servidor para se obter a taxa de pico.

- Latência: é o tempo médio de espera que o cliente experimenta entre o pedido do vídeo e o início de sua exibição.

As requisições do usuário são modeladas como um processo de Poisson. Cada cliente escolhe um vídeo de acordo com a distribuição Zipf [1]. Inicialmente, simulou-se a MCD com vários tamanhos de *slots* e vários tamanhos da parte inicial dos prefixos para se determinar os tamanhos a serem utilizados nas demais simulações. Em seguida, foram feitas três séries de simulações. A primeira com apenas um vídeo para estudar o comportamento da MCC diante de restrições de tamanho da *cache* e diferentes taxas de chegadas com as políticas de gerenciamento da *cache* **LS-**, **LS** e **LS+**. Na seqüência simulou-se a MCC com 100 vídeos com a distribuição Zipf assumindo o parâmetro 0,271 [1] e intervalos entre chegadas de 3,1 e 2,5 segundos. Finalmente, simulou-se a MCC com 100 vídeos escolhidos de acordo com os parâmetros Zipf igual a 0, 0,271 e 1 para diversos tamanhos de *cache*. Todas as simulações foram executadas 30 vezes obtendo-se uma confiança de no mínimo 95% para um intervalo de confiança menor que 1% do valor da medida.

Além disso, foram testados vários tamanhos de *slots*, para se determinar um tamanho compatível com o desempenho esperado e também o tamanho da parte do prefixo que fica antes do primeiro *buffer* colapsado, o pré-prefixo. Estes experimentos, bem como seus resultados são descritos a seguir.

5.2 Determinação de parâmetros de simulação da MCC

Nesta seção é mostrado como se determinou o tamanho de *slot* a ser utilizado nas simulações bem como o tamanho do pré-prefixo.

5.2.1 Considerações sobre o tamanho do slot

Inicialmente simulou-se a MCC com *slots* de 4, 8, 16, 32 e 64 segundos com um vídeo. O sistema foi estressado para simular condições de horários de pico, considerando-se uma *cache* com tamanho de 10% do tamanho total do vídeo e chegada de 1000 clientes em intervalo de uma hora. Foi assumido que o *jitter* era zero, como muitos trabalhos nesta área [10]. É claro que quanto maior for o *jitter*, maior terá que ser o buffer colapsado. No entanto, o estabelecimento do tamanho do *slot* pode ser independente do tamanho do *buffer* colapsado, desde que seu tamanho seja determinado em termos de quantidade de *slots*. Por exemplo, para absorver *jitters* maiores o *slot buffer* pode ter a sua área WS constituída por dois *slots*, mas isto está fora do escopo desta tese.

A figura 5.1-a mostra que a taxa de bloqueio dos clientes no sistema para *slots* de 4 e 8 segundos se mantém quase que constante e, após 16 segundos, a taxa de bloqueio começa a crescer rapidamente. Este fato pode ser melhor compreendido se compararmos o esquema de gerenciamento de memória da MCC com os sistemas tradicionais de paginação de memória virtual [72]. Para esta comparação supõe-se que requisições de clientes são palavras em uma página de memória virtual e que o intervalo entre requisições é o *offset*[1] entre uma palavra e outra. Neste caso, claramente a maior parte do conteúdo da página não é usado. Com o crescimento da página, a probabilidade de que uma página tenha pelo menos uma requisição aumenta e conseqüentemente teremos que carregar várias páginas grandes na memória, onde na verdade apenas algumas palavras são efetivamente utilizadas, aumentando o desperdício de memória. Por outro lado, diminuindo-se o tamanho das páginas, a probabilidade de uma página ter pelo menos uma requisição, em uma aplicação com este comportamento, diminui. Portanto serão necessárias mais páginas, considerando-se a mesma quantidade de requisições, porém com menor tamanho, diminuindo-se o desperdício de memória.

Voltando a analisar a MCC. *Slots* grandes implicam em *buffers* colapsados grandes, que usam muita memória, isto é, armazenam muito conteúdo que logo enchem a *cache*. *Slots* pequenos implicam em *buffers* colapsados pequenos, que armazenam

[1] deslocamento

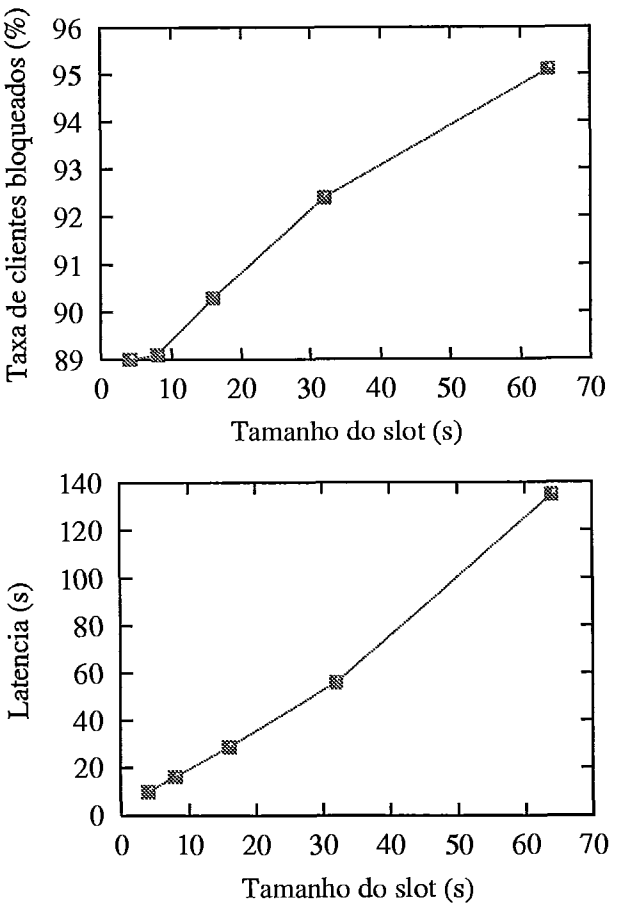


Figura 5.1: (a)Relação entre tamanho de slot e taxa de bloqueio, e (b) relação entre tamanho de slot e a latência de exibição

menos conteúdo e fazem o mesmo papel de um *buffer* com *slots* grandes. Embora com *buffers* colapsados pequenos muitas partes do vídeo não fiquem em *cache*, e consequentemente haja necessidade de mais fluxos regulares do servidor, este espaço extra que foi economizado na *cache* dá à MCC uma maior flexibilidade para administrar a memória, alocando este espaço economizado para encadear *buffers* colapsados onde o custo seja mais vantajoso.

Na figura 5.1-(b) também se nota que com o crescimento do *slot*, a latência também cresce linearmente. Isto ocorre porque os pedidos que chegam em um mesmo *slot* são tratados como se estivessem em esquema de lotes (*batching*), onde os clientes são agrupados para serem servidos e, se o conteúdo não estiver na *cache*, o *buffer* colapsado tem que ser preenchido até atingir seu nível normal de trabalho, o que também leva tempo.

Escolheu-se *slots* de 8 segundos porque a taxa de bloqueio foi quase a mesma da obtida com *slots* de 4 segundos e a latência média entre a requisição do vídeo e a exibição

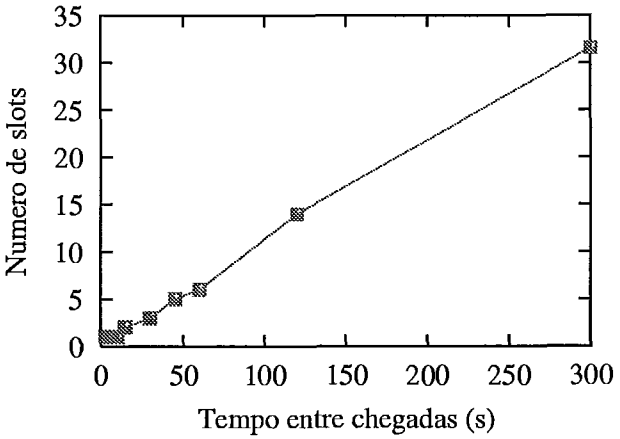


Figura 5.2: Relação entre as distâncias em números de slot entre *buffers* colapsados e tempo médio entre chegadas

para o cliente foi em média de apenas 16 segundos, o que não representa um grande perda em termos de QoS para o cliente. Além disso, levou-se em conta que *slots* pequenos demais armazenam poucas unidades de vídeo, tornando o gerenciamento de *slots* muito custoso em termos de processamento e espaço em memória para a estrutura de controle dos *slots*. Por exemplo, um *slot* de apenas dois segundos, com vídeos MPEG-2 tem-se em média de 3 a 4 unidades de vídeo (GoF) por *slot*.

5.2.2 Considerações sobre a política de alocação de pré-prefixos

A figura 5.2 mostra que se um vídeo é muito popular, ou seja, chegam muitas requisições separadas por um pequeno intervalo de tempo, os *buffers* colapsados estão muito próximos e, ou não precisam de *slots* LS para serem concatenados, ou precisam de poucos. Deste modo quase todo o vídeo estará em *cache* e seu prefixo também, exceto o início do prefixo que está saindo do último *buffer* colapsado, o pré-prefixo. Como este início está fora do *buffer* colapsado e portanto não está reservado, a MCC tem que adotar uma política que preserve este trecho inicial do prefixo, enquanto uma nova requisição não chega.

Uma política simples de ser implementada é manter a prioridade de descarte deste início de prefixo a mais baixa possível, só descartando ele em caso de última necessidade. A questão é até quando esta parte inicial do prefixo deveria ser mantida com prioridade tão baixa de descarte. Note que à medida que o tempo passa, esta parte inicial do prefixo cresce, ocupando espaço em *cache*. Outra preocupação é o fato de que se esta parte do prefixo for descartada, na realidade a *cache* não estará mais armazenando o

PARÂMETRO	NORMAL	VARIAÇÃO
Duração do vídeo (h)	1	-
Quantidade de vídeos	1 ou 100	-
Largura de banda do servidor (canais lógicos)	1000	-
Buffer colapsado (slots)	4	-
Tamanho do slot (s)	8	-
Tempo entre chegadas (s)	-	3,1 - 150
Tamanho da *cache* (% do conteúdo no servidor)	-	10 - 100
Parâmetro da distribuição Zipf	0,271	0 - 1

Tabela 5.1: Parâmetros usados na simulação da MCC

prefixo, aumentando a latência do sistema e gerando a necessidade de fluxos de *patch* para encadear com o buffer colapsado anterior. Por isto, da figura se tirou o valor de 32 slots para esta parte inicial do prefixo. Com isto, vídeos muito populares manterão sempre o prefixo em *cache*, e vídeos não tão populares também manterão esta parte inicial do prefixo em *cache* até 32 slots, se nenhuma requisição chegar neste tempo, esta parte do vídeo terá a sua prioridade de descarte aumentada. No entanto, caso a *cache* necessite descartar também esta parte inicial do prefixo, ela iniciará descartando os que têm maior tamanho, i.e., os menos populares.

5.3 Resultados de desempenho da MCC

Nesta seção são mostrados os desempenhos da MCC para um vídeo e para 100 vídeos. Em seguida são analisados detalhadamente as diferenças de desempenho entre as políticas implementadas (**LS-**, **LS** e **LS+**).

A carga de trabalho e os parâmetros do sistema estão resumidos na tabela 5.1. Os valores normais de simulação estão listados na coluna "Normal"e alguns parâmetros que são variados na simulação para análise da sensibilidade estão na coluna "Variação".

5.3.1 Simulações com um vídeo

Para se estudar o desempenho da MCC, simulou-se inicialmente a MCC com 1 servidor oferecendo apenas 1 vídeo. Os tempos entre chegadas variaram de 3.2 segundos até 150 segundos. O tamanho da *cache* do proxy foi variado de 10 a 100% do tamanho total do único vídeo armazenado no servidor.

MCC com política LS-

O problema em encadear *ring buffers* é que uma vez que se aloca *slots* de ligação para encadear dois *buffers*, este espaço permanece alocado até o final da exibição do vídeo. Isto acontece mesmo quando a *cache* do *proxy* está cheia. Para evitar *slots* de ligação ocupando espaço em cache, espaço este que poderia ser utilizado para alocar mais *ring buffers* e atender mais clientes, **LS-** opta por simplesmente não utilizar *slots* de ligação. Desta forma, **LS-** só encadeia *ring buffers* que estão ou superpostos ou justapostos, ou seja, não gasta nenhum espaço em cache além do mínimo necessário. Resumindo, em se tratando de *ring buffers*, a política **LS-** é a mais conservadora de todas.

Analisando-se a taxa de bloqueio para **LS-** com apenas um vídeo (figura 5.3) observa-se que a taxa de bloqueio é alta para intervalos entre chegadas pequenos. Isto porque o espaço de memória da cache é rapidamente consumida alocando-se *ring buffers* para os pedidos iniciais. Em conseqüência, os pedidos que chegam a posteriori encontram a cache cheia e, portanto, não havendo como se alocar mais *ring buffers*, os pedidos são bloqueados. Observa-se também que a taxa de bloqueio é alta para caches pequenas. Caches pequenas permitem alocar poucos *ring buffers* e em conseqüência a cache fica rapidamente cheia. Com isso muitos pedidos não podem ser atendidos. No entanto, a taxa de bloqueio para **LS-** é baixa nos demais casos.

Analisando-se a taxa de utilização de **LS-** para apenas um vídeo (figura 5.4) observa-se que ela é baixa para tempo entre chegadas pequenos, uma vez que neste caso a maior parte dos *ring buffers* fica ou justaposta ou superposta. Com isto, basta apenas um fluxo regular do servidor para alimentar todos os *ring buffers*. O mesmo acontece para caches grandes, uma vez que neste caso, quase todo ou todo o vídeo fica armazenado em cache. Nos demais casos a taxa de utilização é maior, sendo superior a 0.25% em grande parte da superfície do gráfico, e em alguns pontos superior a um por cento. É importante observar que esta taxa de um por cento se refere a taxa de utilização de um único vídeo em um servidor com mil canais lógicos. Ou seja, uma taxa de 1% equivale a dez canais lógicos ficarem ocupados durante 100% do tempo, durante a simulação de uma hora.

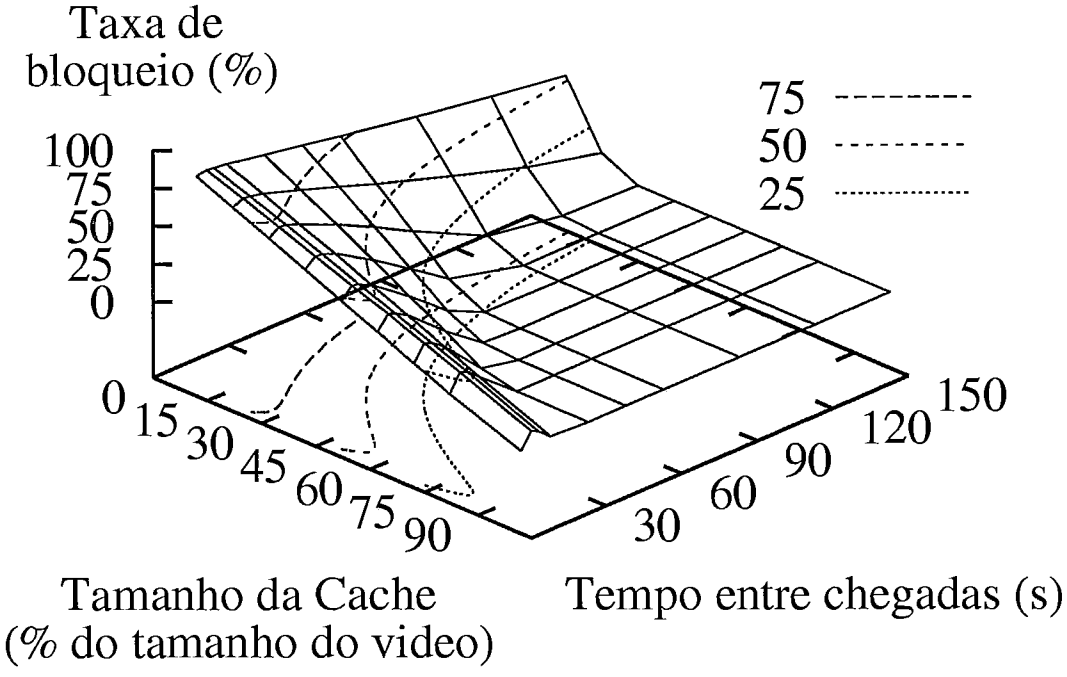


Figura 5.3: Taxa de Bloqueio da MCC política **LS-**

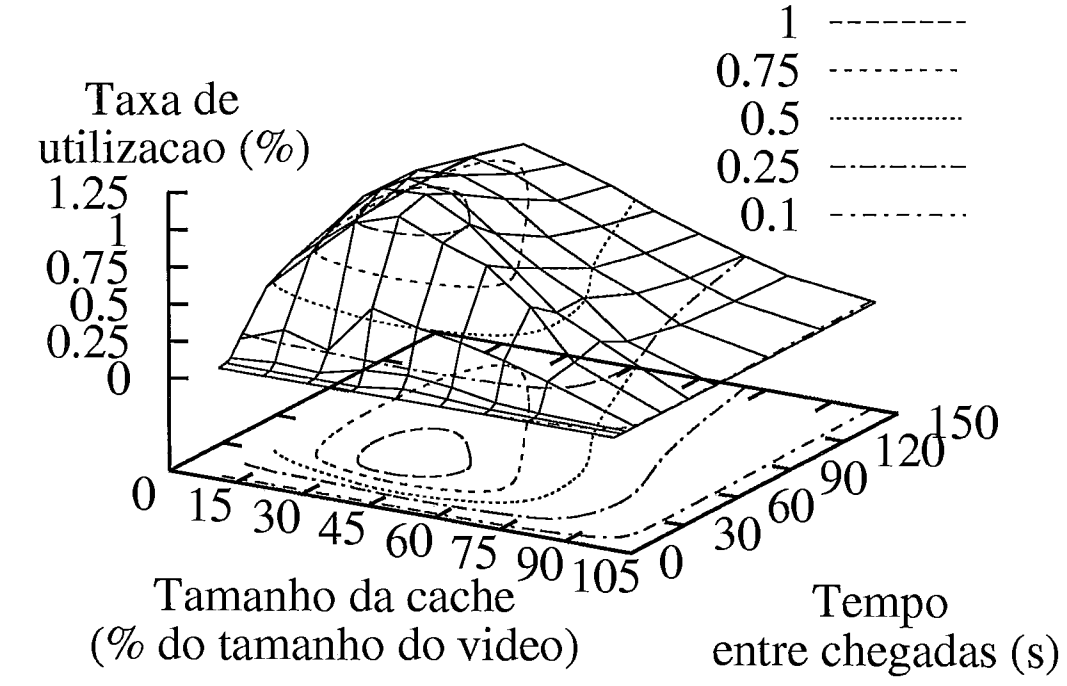


Figura 5.4: Taxa de Utilização da MCC política **LS-**

A figura 5.5 é o gráfico da taxa de pico de utilização. Este gráfico mostra que esta taxa é superior a 2.5 % para caches entre 30 e 90% do tamanho do vídeo e tempo entre chegadas superior a 30 segundos. Esta taxa é superior à taxa de utilização por que em algum instante houve uma combinação de fluxos regulares e fluxos de *patches* que atingiu um pico registrado no gráfico.

O gráfico da figura 5.6 mostra que quanto menor for a cache e maior o intervalo entre chegadas, maior a latência. Para caches pequenas, não há muito espaço para armazenar o prefixo do vídeo, logo ele é descartado. Se o prefixo e o último *ring buffer* estão contíguos no momento em que chega um novo pedido, o novo *ring buffer* é alocado sobre o prefixo, eliminando a necessidade de um novo fluxo de vídeo. Neste caso, o prefixo só é descartado se não houver espaço em cache, uma vez que manter o prefixo significa economizar um novo fluxo regular. Além disso, ter o prefixo em cache significa mandar imediatamente o fluxo de vídeo desde o início para o cliente, o que diminui a latência. Para intervalos entre chegadas grandes, o prefixo e o último *ring buffer* alocado deixam de ser contíguos após algum tempo. O algoritmo da gerência da cache, neste caso, só deixa o prefixo em cache se houver espaço na mesma. Se faltar espaço em cache, ele descarta o prefixo com a mesma prioridade das outras unidades de vídeo não reservadas. Este procedimento não faz muito sentido se analisado isoladamente, mas se considerarmos vários vídeos na cache, os vídeos mais populares, por terem tempo de intervalos entre chegadas menores, não terão o seu prefixo descolados do último *ring buffer* e, portanto, terão o seu prefixo com uma prioridade maior de permanecer em cache. Caso contrário, terão seu prefixo com uma prioridade menor de permanecer em cache. Isto explica a maior latência para tempo entre chegadas maiores,i.e., para tempo entre chegadas grandes maior a probabilidade do prefixo ser descartado. Uma vez que o prefixo foi descartado, ele primeiro terá que ser enviado para o proxy que em seguida enviará ao cliente, aumentando a latência inicial de exibição. Apesar da latência ser maior, ela fica limitada a menos de meio minuto.

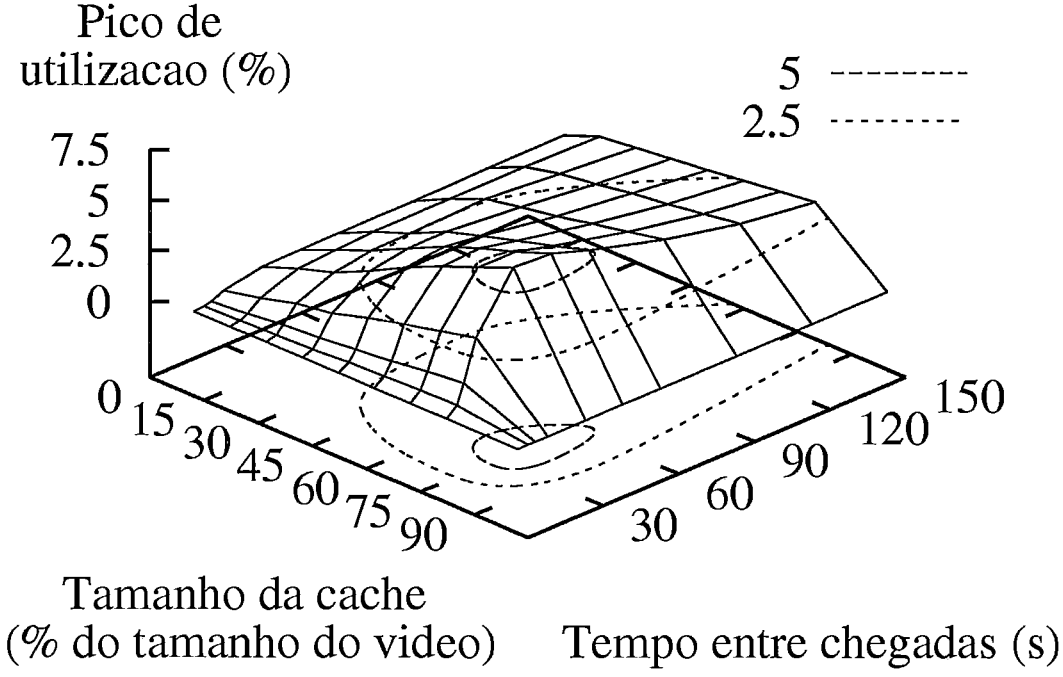


Figura 5.5: Taxa de Pico de Utilização da MCC política **LS-**

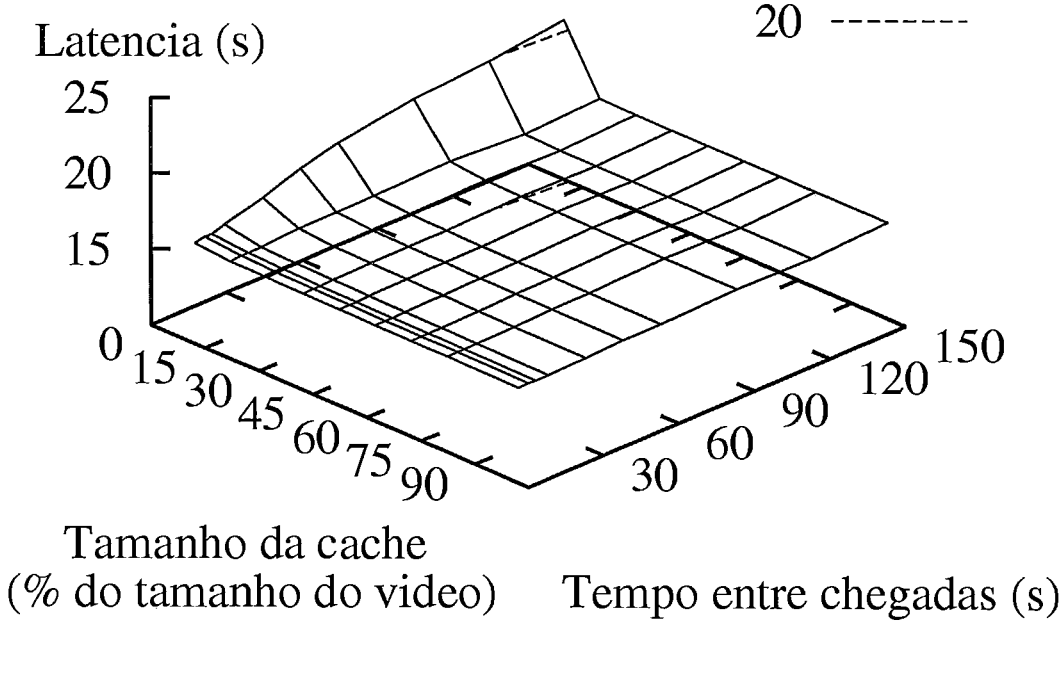


Figura 5.6: Latência Média para as políticas da MCC política **LS-**

MCC com política LS

A grande vantagem em se encadear *ring buffers* está no fato de se trocar largura de banda do servidor por memória na *cache* do *proxy*. Como esta memória fica alocada até o final da exibição do vídeo, um limite máximo na quantidade de memória para encadear *ring buffers* deve ser estabelecido. Este limite evita que vídeos pouco populares sejam encadeados. Como vídeos poucos populares tem um grande intervalo entre pedidos, evitando-se o encadeamento entre *ring buffers* alocados para estes pedidos, também evita-se gastar memória demais para economizar um fluxo regular do servidor. Como o prefixo foi estabelecido em 32 *slots*, o número máximo de *slots* de ligação para encadeamento também foi estabelecido em 32 *slots*. Isto garante que ao se encadear *ring buffers* com *slots* de ligação, estes *slots* estarão com seu conteúdo em *cache* (no prefixo). Com isto não há a necessidade de se recompletar o conteúdo com um fluxo de *patch*.

A figura 5.7 mostra que a taxa de bloqueio da política **LS** é alta, mesmo para caches grandes e intervalos grandes de tempo entre chegadas. Isto se deve ao fato da política LS gastar memória do proxy para encadear *ring buffers*. Com isto, após atender alguns pedidos, logo a cache fica cheia. Com a cache cheia, não há espaço para alocar novos *ring buffers* e, em conseqüência, a taxa de bloqueio é alta. Em compensação, a taxa de utilização é baixíssima (figura 5.8). Isto acontece por que fluxos regulares do servidor foram substituídos por espaço de memória em cache (foram alocados *slots* de ligação para encadear *ring buffers*). Em conseqüência da taxa de utilização baixa, a taxa pico de utilização também é muito baixa (figura 5.9). A figura 5.10 mostra que a latência para esta política também se mantém baixa, isto por que o prefixo do vídeo é mantido em cache enquanto há espaço em cache para isto.

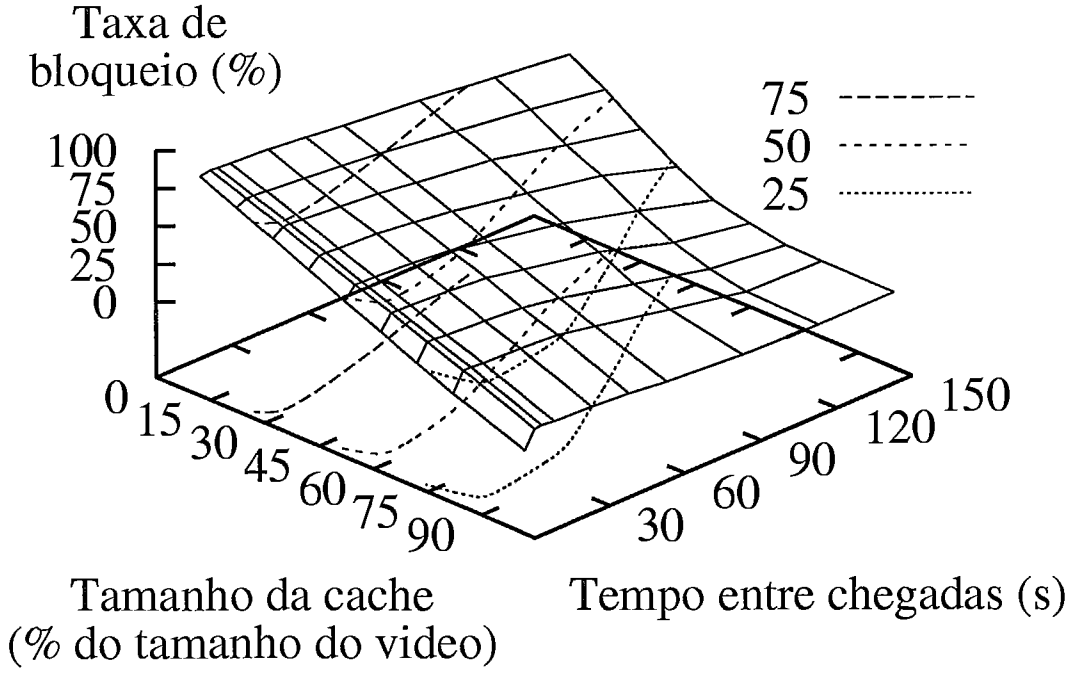


Figura 5.7: Taxa de Bloqueio da MCC política **LS**

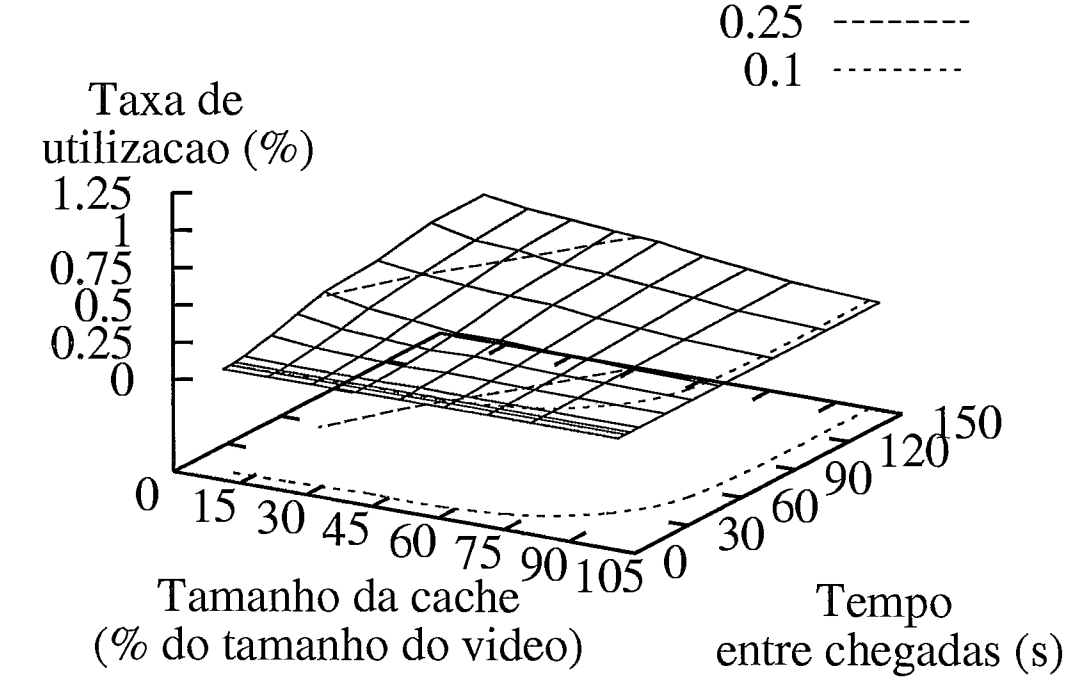


Figura 5.8: Taxa de Utilização da MCC política **LS**

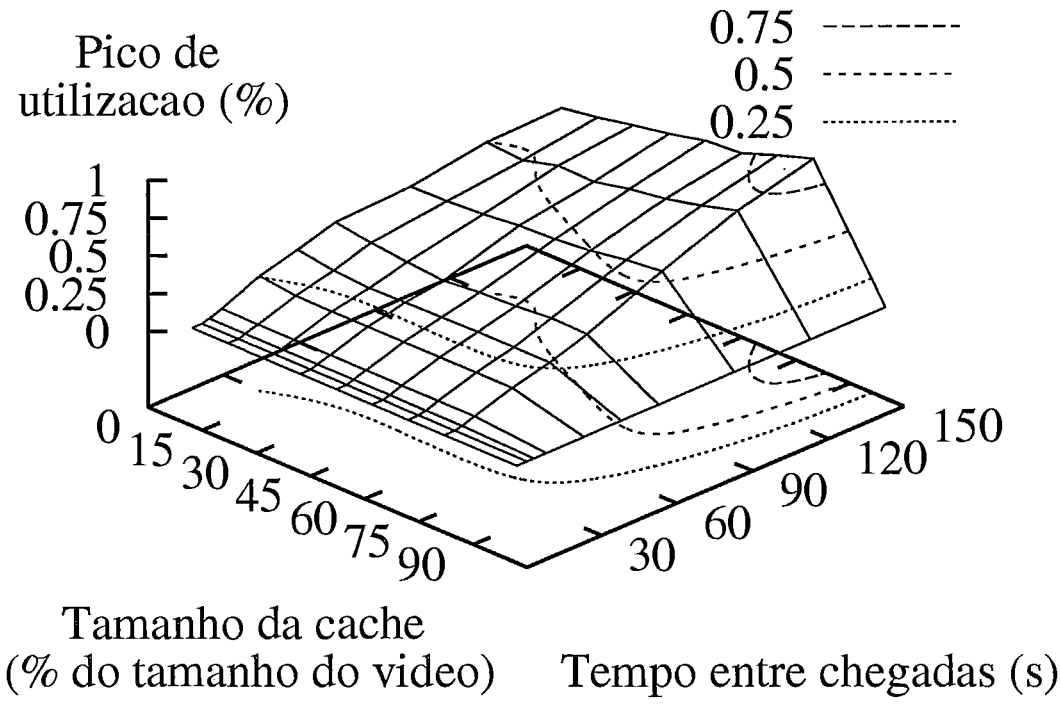


Figura 5.9: Taxa de Pico de Utilização da MCC política **LS**

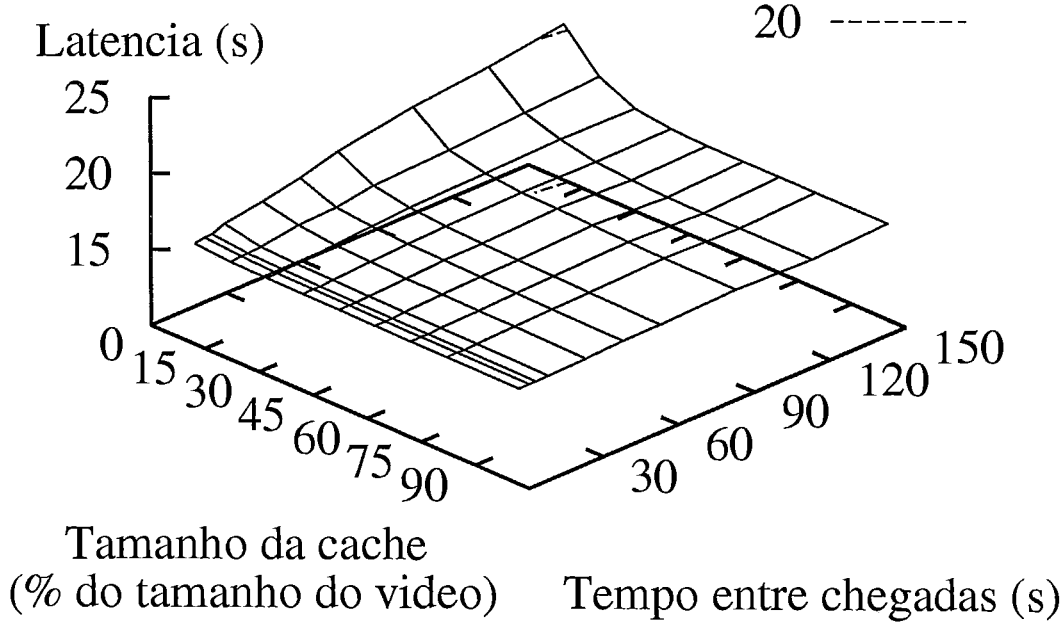


Figura 5.10: Latência Média para as políticas da MCC política **LS**

MCC com política LS+

O problema do esquema de *ring buffers* é que ele aloca espaço em memória da cache para encadear *ring buffers*, e este espaço permanece alocado até o final da exibição do vídeo. Para contornar este problema, *ring buffers* limitam o tamanho máximo de encadeamento (para fins de comparação, nesta Tese, **LS-** limita o encadeamento em zero *slots* de ligação e **LS** em 32 *slots* de ligação). A política **LS+**, proposta nesta Tese, procura gerenciar a área de memória usada para o encadeamento de *ring buffers*. Com isto, caso a cache fique cheia, *slots* de ligação podem ser liberados antes do final do vídeo para atender mais pedidos.

Analisando a taxa de bloqueio da política **LS+** (figura 5.11) verificamos que sua taxa de bloqueio é bastante semelhante a de **LS-** (uma observação mais precisa mostra que **LS+** tem uma taxa de bloqueio ligeiramente inferior a **LS-**). Em relação a **LS**, **LS+** possui uma taxa de bloqueio bem menor.

Em relação à taxa de utilização (figura 5.12) observa-se que **LS+** e **LS-** tem uma superfície bastante semelhante, por isso, as duas superfícies são sobrepostas na figura 5.15 que será analisada mais adiante. Além disso, **LS** tem uma baixa taxa de utilização tanto em relação a **LS+** como **LS-** devido a sua alta taxa de bloqueio.

A figura 5.13 mostra que a taxa de pico de utilização de **LS+** tem um pico menor que **LS-**, mas superior a **LS**. Essa baixa taxa de pico de utilização de **LS** é devido à sua alta taxa de bloqueio. A figura 5.14 mostra que a latência média de **LS+** permanece restrita a menos de 25 segundos, como nas outras políticas.

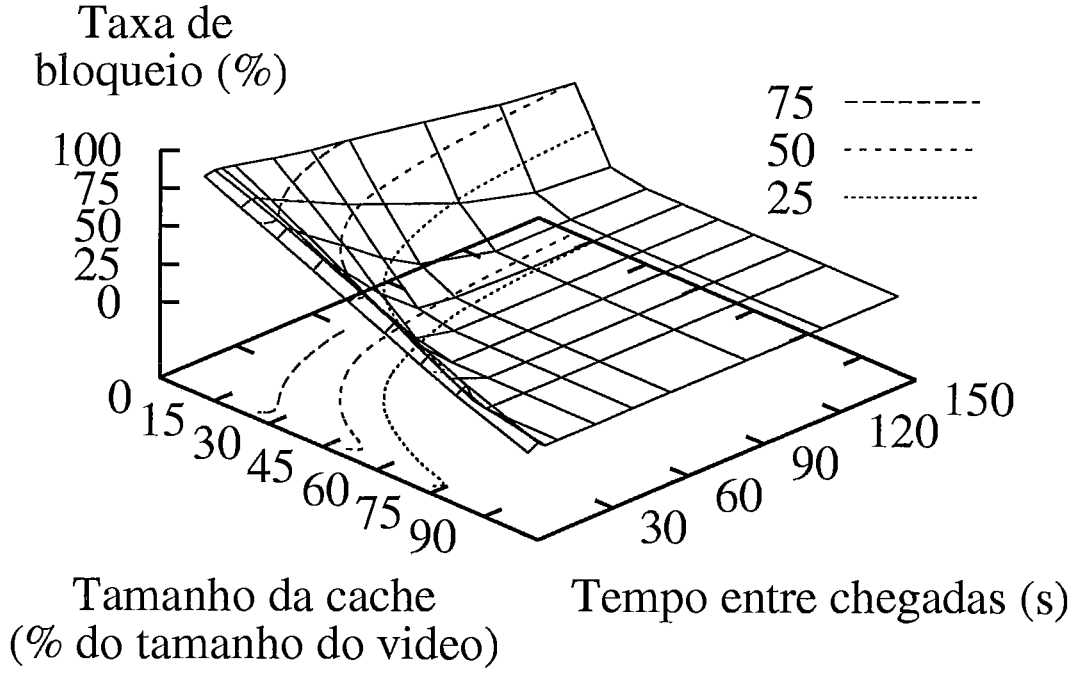


Figura 5.11: Taxa de Bloqueio da MCC política **LS+**

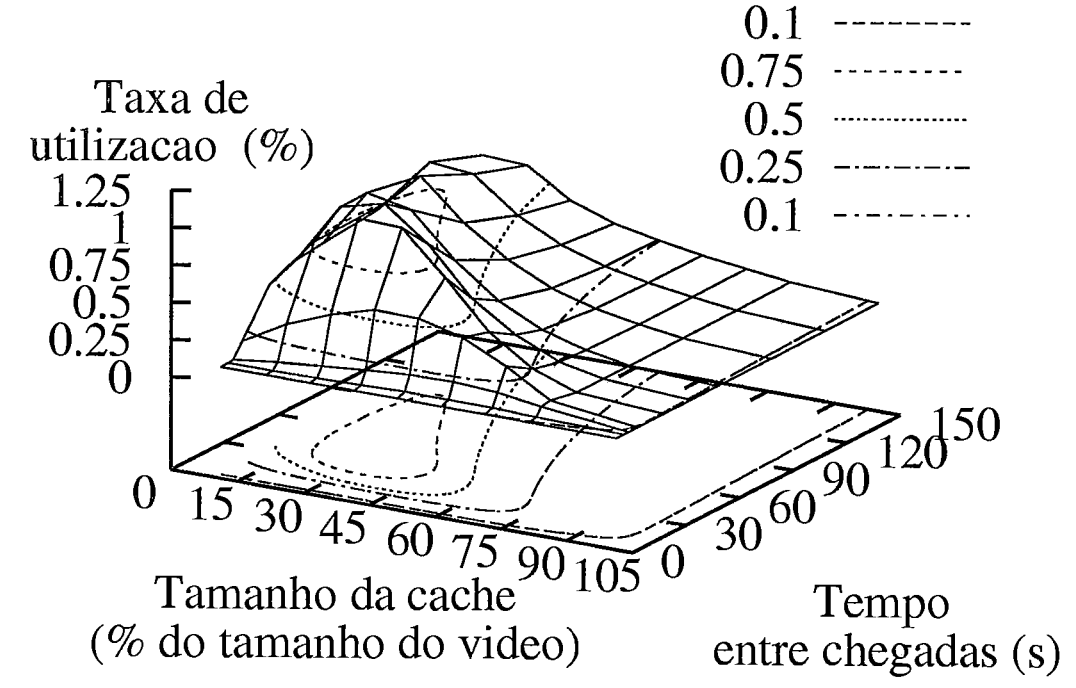


Figura 5.12: Taxa de Utilização da MCC política **LS+**

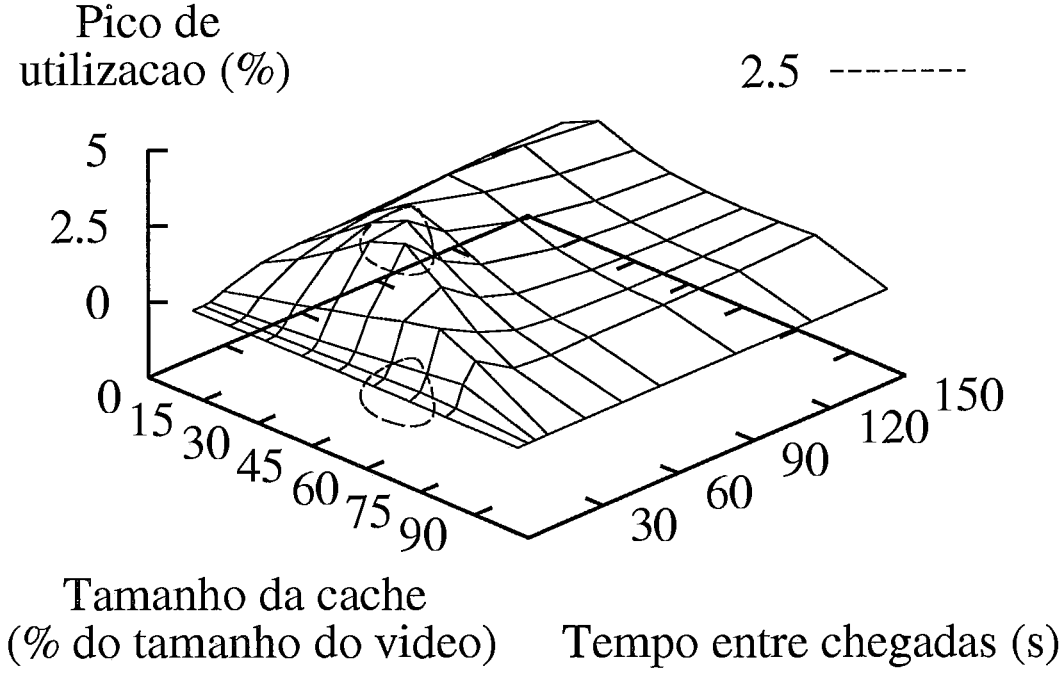


Figura 5.13: Taxa de Pico de Utilização da MCC política **LS+**

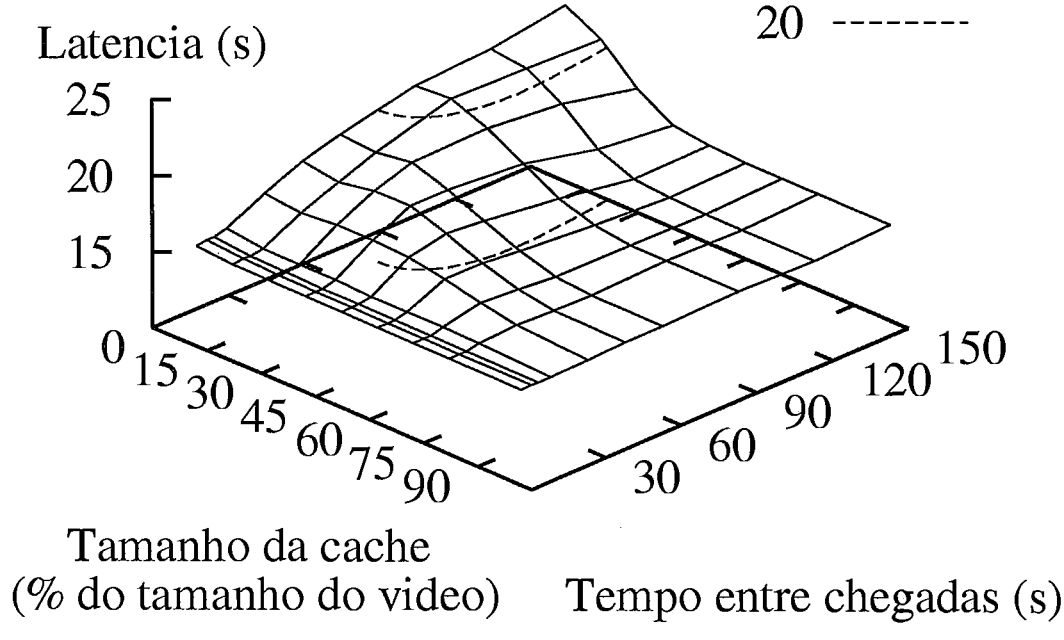


Figura 5.14: Latência Média para as políticas da MCC política **LS+**

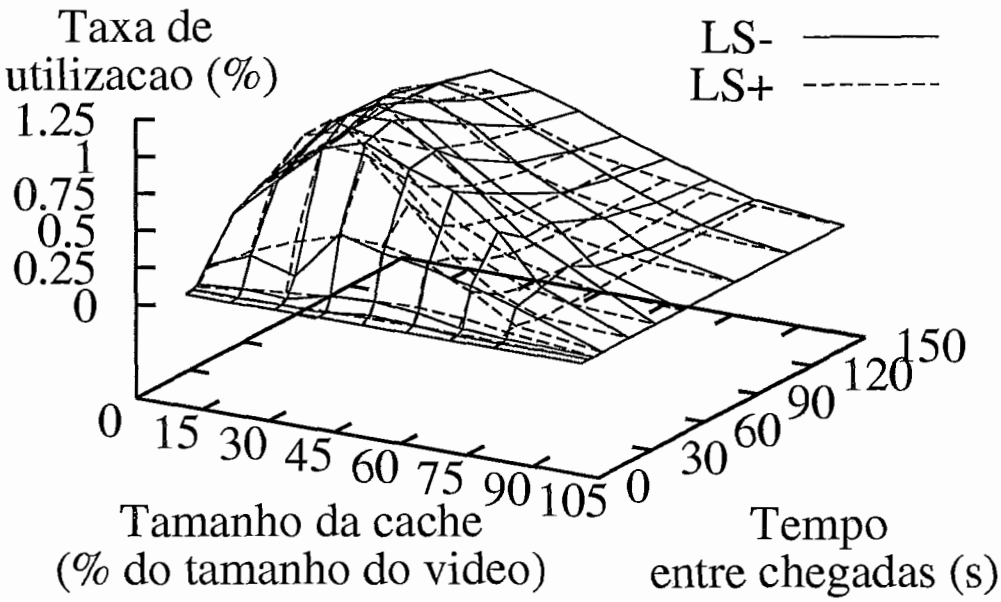


Figura 5.15: Comparando a taxa de utilização de **LS-** e **LS+**

Resumindo as observações acima, nota-se que a grande vantagem de **LS+** sobre **LS-** se encontra numa região com tamanho de *cache* nem grande nem pequeno e tempo entre chegadas maiores que o tamanho de um *buffer* colapsado. Em relação ao tempo entre chegadas, **LS+** só começa a mostrar desempenho quando este tempo entre chegadas é maior do que 4 *slots* (no caso desta simulação, 32 segundos), que é quando os *slots* de ligação começam a ser utilizados pela MCC **LS+**. Por isso, para se observar melhor a diferença de desempenho entre **LS+** e **LS-**, realizou-se um corte no gráfico da figura 5.15 no tempo entre chegadas de 150 segundos e outro com tamanho da *cache* em 20%.

5.3.2 Análise da MCC com 1 vídeo na faixa de operação de melhor desempenho

Taxa de bloqueio dos clientes

A figura 5.16-a mostra a taxa de bloqueio para um tempo médio entre chegadas constante, variando-se apenas o tamanho da *cache* do *proxy*. O gráfico mostra que a taxa de bloqueio da MCC com política **LS+** e **LS-** estão muito próximas. Para *caches* menores que 30% do conteúdo total do servidor, **LS+** é pouco melhor que **LS-**. Esta diferença é explicada pelos efeitos que as duas políticas causam no prefixo. Devido ao fato de **LS-** não alocar slots de ligação para encadear buffers colapsados, estes não podem ser desalocados e, portanto, o tamanho do prefixo se mantém estático. Por outro lado, a política **LS+** aloca *slots* de ligação para encadear *buffers* colapsados. Ao desalocar *slots* de ligação, existe uma probabilidade de que a cadeia esteja sobre o prefixo do vídeo. Precisamente, este trecho do vídeo compreendido pelos *slots* de ligação desalocados pode ser descartado. Este fato ocorre com mais freqüência para *caches* pequenas, onde unidades de vídeo reservadas enchem a *cache* mais rapidamente. Desta forma, descartando parte do prefixo, *LS+* consegue atender mais pedidos que *LS-*, tendo portanto uma taxa de bloqueio menor para *caches* pequenas. Por outro lado, a política *LS* apresenta maior taxa de bloqueio, porque consome o espaço de armazenamento da *cache* alocando *slots* de ligação. Como estes *slots* de ligação não são descartados em caso de falta de espaço em memória, logo a *cache* fica cheia e grande parte dos pedidos não são atendidos.

A figura 5.17-a mostra a taxa de bloqueio para um tamanho fixo de *cache*, no caso 20%, variando-se o tempo médio entre chegadas dos clientes de 3,1 segundos a 150 segundos. Nesta figura, observa-se que para pequenos intervalos de chegada a taxa de bloqueio é alta e praticamente igual para as três políticas. Como nesta faixa de intervalo entre chegadas, nem **LS** nem **LS+** usam *slots* de ligação, o gerenciamento da *cache* é igual para as as três políticas. Além disso, com este intervalo, muitos *buffers* colapsados são alocados em um curto espaço de tempo e, a *cache* de 20% logo fica cheia, o que leva as três políticas a terem uma mesma taxa de bloqueio alta.

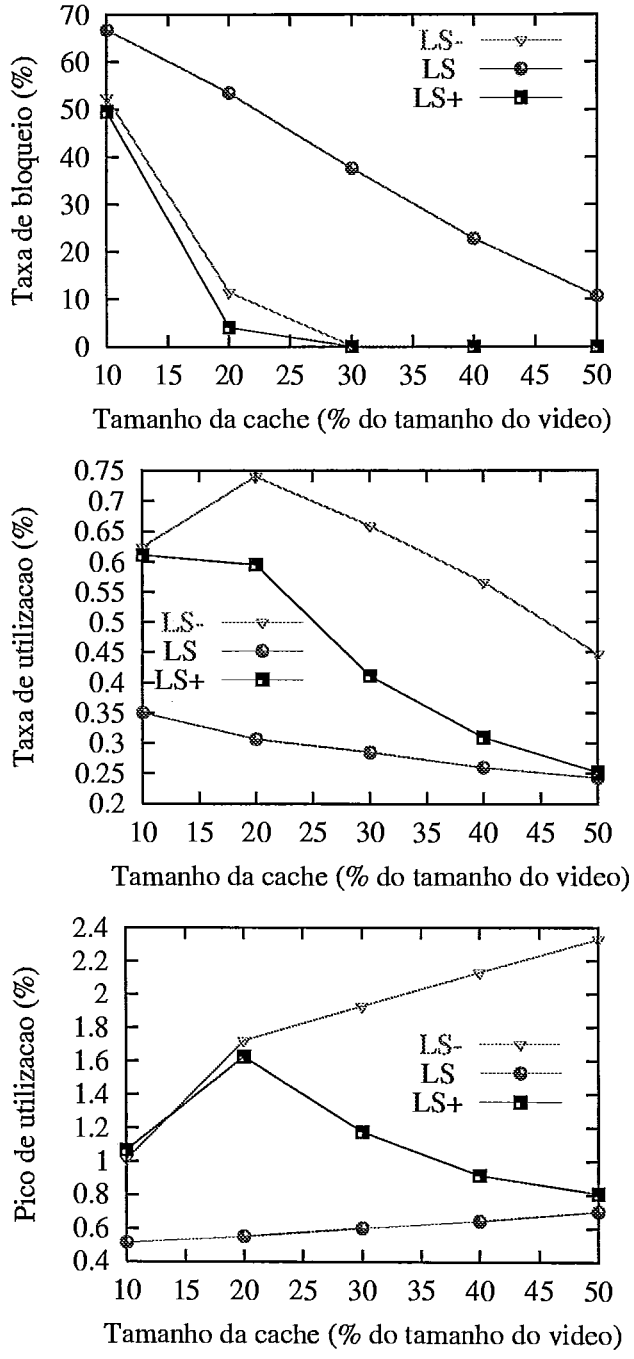


Figura 5.16: Um video com tempo entre chegadas igual a 150 segundos. (a) taxa de bloqueio, (b) taxa de utilização e (c) taxa pico de utilização, respectivamente de cima para baixo

Com intervalos de tempo maiores, **LS** e **LS+** passam a fazer uso dos slots de ligação. Com isso a taxa de bloqueio começa a se diferenciar. Além disso, com intervalos entre chegadas maiores, menos pedidos chegam em um mesmo intervalo de tempo e, obviamente, menos pedidos precisam ser atendidos e menos memória da *cache* é utilizada. Com isso, para as três políticas a curva apresenta uma derivada negativa. **LS** apresenta uma pequena taxa de queda, uma vez que a *cache* fica rapidamente cheia com os *slots* de ligação, e como a quantidade de pedidos diminui, o número de clientes rejeitados é menor. **LS-** apresenta uma taxa de queda maior, uma vez que chegando menos pedidos, menos espaço em *cache* é reservado para os *buffers* colapsados, e portanto, proporcionalmente mais pedidos são atendidos. **LS+** consegue uma taxa de bloqueio menor que **LS-** porque consegue diminuir o tamanho do prefixo do vídeo dinamicamente. À medida que a *cache* vai ficando cheia e falta espaço para alocar mais *buffers* colapsados, parte do prefixo é descartado.

Taxa de utilização do servidor

A figura 5.16-b mostra a taxa de utilização para uma *cache* de 20%. Para tempos pequenos entre chegadas de clientes (abaixo de 10 s) o desempenho das três políticas é igual, pois, como explicado anteriormente, **LS** e **LS+** não chegam a fazer uso dos *slots* de ligação. Para tempos maiores entre chegadas, **LS** apresenta uma baixa taxa de utilização, fruto do pequeno número de clientes atendidos (taxa de bloqueio alta) e os *buffers* colapsados que atendem estes clientes estão, em grande quantidade, encadeados por *slots* de ligação. Como *slots* de ligação substituem canais regulares do servidor, a política **LS** conseqüentemente usa menos canais regulares.

Entre 10 s e 60 s, **LS+** tem uma taxa de utilização ligeiramente maior que **LS-**. Como neste intervalo quase todos os buffers colapsados estão encadeados, a diferença de desempenho vem do fato que **LS+** diminui o prefixo em *cache* quando há falta de espaço em *cache*, e conseqüentemente atende mais clientes. Isto gera a necessidade de novos fluxos regulares de vídeo do servidor o que leva a uma maior taxa de utilização do mesmo.

Para intervalos acima de 60 s, **LS+** tem uma taxa de utilização menor que **LS-**. Nesta faixa de operação, **LS+** pode utilizar os *slots* de ligação economizando canais regulares do servidor, o que leva **LS+** a ter um desempenho melhor que **LS-**.

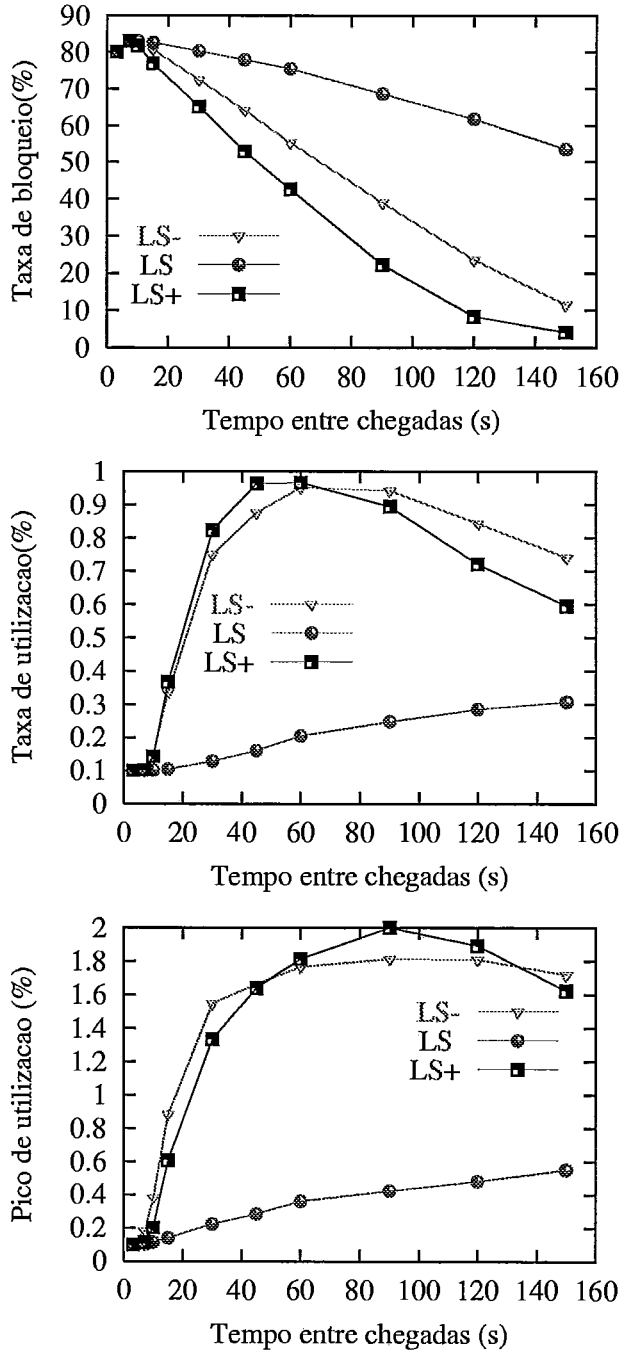


Figura 5.17: Um video com tamanho de *cache* igual a 20%, (a) taxa de bloqueio, (b) taxa de utilização e (c) taxa pico de utilização, respectivamente de cima para baixo

A figura 5.17-b mostra que **LS+** tem um melhor desempenho que **LS-** para um intervalo médio entre chegadas de 150 s e para quase todos os tamanhos de *cache*. Novamente, com este intervalo de tempo entre chegadas **LS+** faz uso intensivo dos *slots* de ligação, o que explica seu desempenho. O desempenho aparentemente melhor de **LS** com respeito à taxa de utilização, na verdade é reflexo do seu péssimo desempenho no que diz respeito a sua alta taxa de bloqueio.

Taxa de pico de utilização do servidor

A taxa média de utilização do servidor reflete diretamente no tráfego médio no *backbone* da CDN. No entanto, só a média é uma medida muito vaga para se avaliar o desempenho total do sistema. Um sistema pode ter uma taxa de utilização média baixa e ter uma péssima qualidade de serviço. Basta que em pequenos intervalos de tempo sua taxa de utilização seja tão alta que provoque congestionamentos na rede, perdas de pacotes, etc. degradando a qualidade do serviço. Normalmente isto ocorre quando muitos clientes se conectam ao sistema e todos experimentam e reclamam da baixa qualidade do serviço. Devido a isto, mediu-se nas simulações a taxa de pico gerada por cada um dos três esquemas.

Novamente, para tempos entre chegadas menores que 10 s, a figura 5.16-c mostra que a taxa pico para os três esquemas é igual. A explicação é a mesma, para este intervalo entre chegadas nem **LS** nem **LS+** usam *slots* de ligação, tendo o mesmo comportamento que **LS-**. Acima deste intervalo, **LS** apresenta uma menor taxa de pico porque atende muito menos clientes dos que os outros dois esquemas. Ainda, as taxas de pico para **LS-** e **LS+** com uma *cache* de 20% para intervalos maiores que 10 s estão bastante próximas. No entanto, quando se observa o desempenho de **LS+** e **LS-** para um intervalo de 150 s (figura 5.17-c), variando-se o tamanho da *cache*, **LS+** tem um desempenho melhor que **LS-** para *caches* maiores que 20%. Isto se deve ao fato de em havendo espaço em *cache*, **LS+** usa este espaço para alocar *slots* de ligação, economizando fluxos regulares do servidor.

5.3.3 Simulações com 100 vídeos

Zipf igual a 0,271, em condições de carga máxima, LS-, LS+ e LS

Para avaliar o desempenho dos três esquemas da MCC com 100 vídeos, simulou-se o comportamento do sistema em um horário de pico, com intervalos entre chegadas de 3,1 e 2,5 segundos. Os clientes chegam de acordo com o processo de Poison e escolhem os vídeos de acordo com a distribuição Zipf de parâmetro 0,271 [1].

É importante ressaltar que as simulações efetuados com um vídeo foram realizadas para se estudar o comportamento da MCC com os diferentes esquemas de gerenciamento. No entanto, em um sistema real, vários vídeos são oferecidos simultaneamente pelo servidor. Da mesma forma, o *proxy* deve armazenar trechos distintos de vários vídeos ao mesmo tempo. Desta forma, cada vídeo terá uma taxa diferente de pedidos dependendo de sua posição na escala de popularidade, dada pela distribuição Zipf. Além disso, o espaço em *cache* para cada vídeo não é definido previamente. Existe o espaço de memória da *cache* do proxy de vídeo que será compartilhado por todos os vídeos. Ou seja, o tamanho de *cache* utilizado para as simulações com um vídeo, que era a porcentagem do tamanho do vídeo, tem pouca ou nenhuma relação com o tamanho da *cache* que é dada pela porcentagem da quantidade total de vídeos no servidor (100 vídeos).

A figura 5.18 mostra o desempenho da MCC com um intervalo entre chegadas de 3,1 segundos. Ou seja, em média, em uma hora o proxy recebe aproximadamente 1000 clientes. Com 100 vídeos o desempenho de **LS** e **LS+** é melhor do que **LS-** para *caches* entre 20% e 60 % do conteúdo total armazenado no servidor. Para *caches* muito grandes, o esquema utilizado pela MCC não faz diferença. Para *caches* muito pequenas (10 %) **LS** apresenta uma alta taxa de bloqueio, ou seja, 40% dos pedidos são bloqueados (aproximadamente 400 clientes). Esta é uma medida relevante, uma vez que *caches* reais tem um espaço de armazenamento muito menor que o conteúdo total existente no servidor.

Aumentando-se a taxa de chegadas (figura 5.19) observa-se que **LS+** permanece com taxa de bloqueio zero para qualquer tamanho de *cache*. O que não ocorre com **LS-** e **LS**. O que mostra que a MCC com o esquema **LS+** apresenta uma solução de compromisso para *caches* pequenas que é melhor que **LS-** e **LS**: Taxa de bloqueio zero, menor taxa de utilização do servidor e menor taxa de pico do que **LS-**.

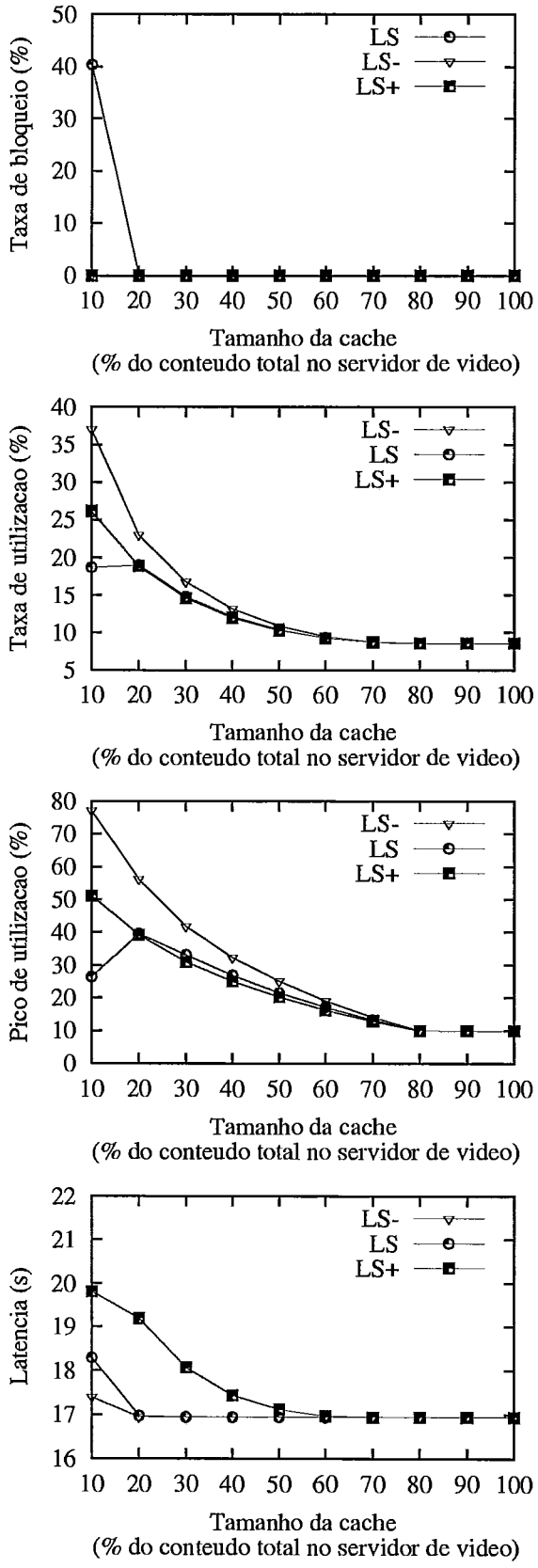


Figura 5.18: **LS-**, **LS** e **LS+** com intervalo entre chegadas de 3,1 segundos

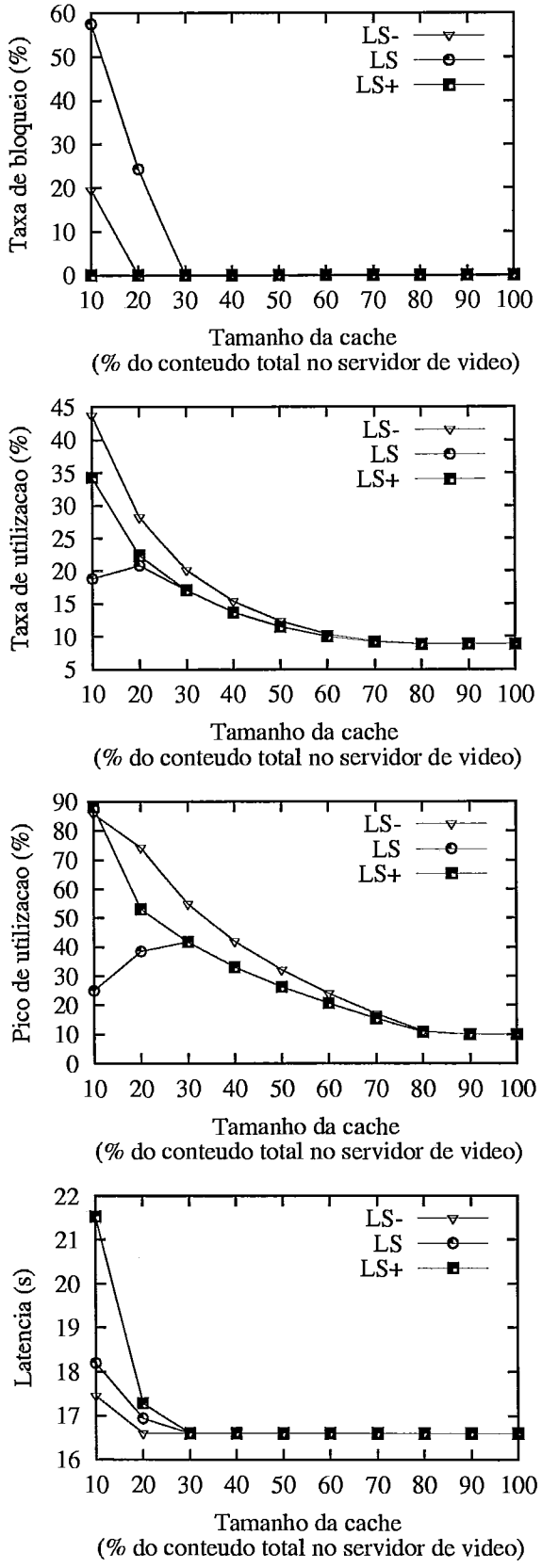


Figura 5.19: **LS-**, **LS** e **LS+** com intervalo entre chegadas de 2,5 segundos

Em relação latência, para caches com tamanho igual a 10 %, **LS+** teve uma latência em média 1,5 segundos maior diminuindo-se o intervalo entre chegadas de 3,1 para 2,5 s. Isto acontece por que com um intervalo menor entre chegadas mais clientes chegam em um menor tempo, com isto a cache fica cheia mais rapidamente levando o prefixo a ser descartado mais cedo. Sem o prefixo na cache, a latência se torna maior. Além disso, deve-se levar em conta que **LS+** teve uma taxa de bloqueio muito menor que as demais políticas, ou seja, atendeu a mais clientes, que por falta de espaço em cache não puderam ser atendidos com o prefixo, o que acarretou em uma maior latência. No entanto, para caches iguais ou maiores do que 20 %, existe um espaço maior na cache para se armazenar prefixos. Além disso, com um intervalo entre chegadas menor, mais pedidos chegam em menos tempo, aumentando a probabilidade de se reutilizar o conteúdo da cache. Neste caso em particular mais pedidos podem ser atendidos pelo mesmo prefixo, donde a menor latência para intervalos entre chegadas de 2.5 s.

Variando Zipf, condições de carga máxima e LS+

Para se avaliar a MCC com diferentes parâmetros da distribuição Zipf, simulou-se a MCC **LS+** com Zipf igual a 0, 0,271 e 1. A figura 5.20 mostra que a MCC com parâmetro Zipf igual a zero tem o melhor desempenho. Isto não é nenhuma surpresa, uma vez que com este parâmetro a distribuição Zipf concentra os pedidos de vídeos com uma maior intensidade nos vídeos mais populares. O contrário ocorre com o parâmetro 1, onde a distribuição Zipf degrada para uma distribuição onde os pedidos são uniformemente distribuídos entre os 100 vídeos oferecidos.

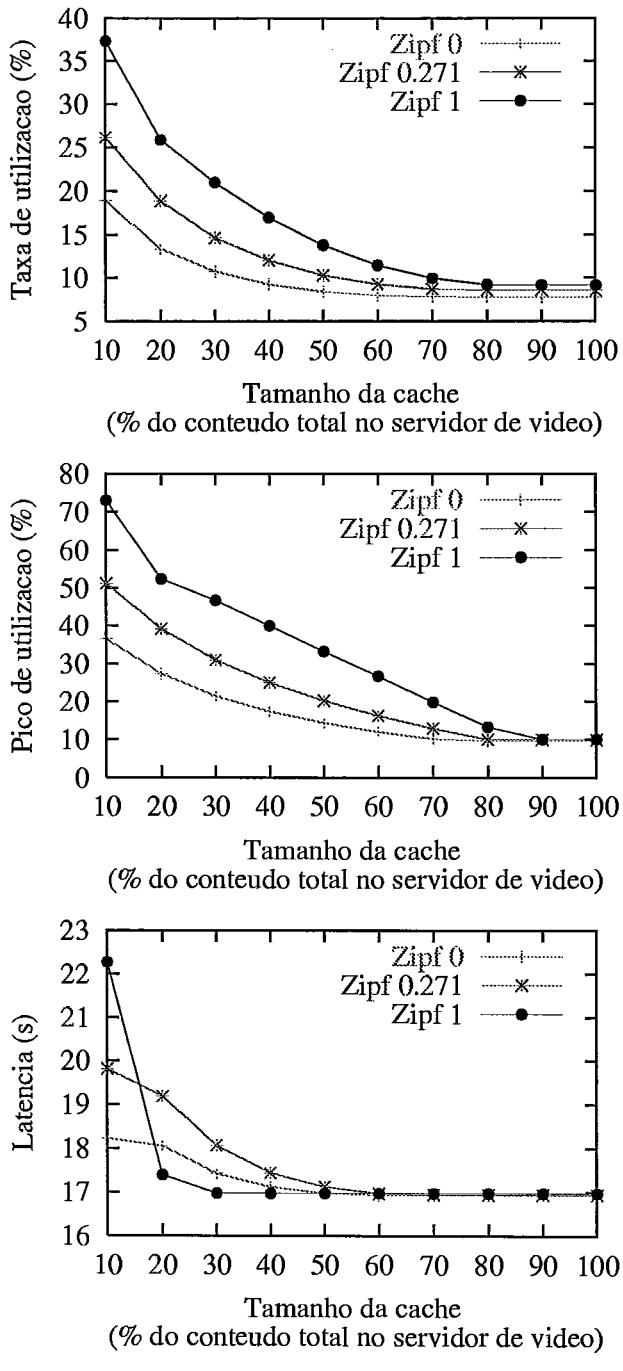


Figura 5.20: **LS+** variando-se o parâmetro da distribuição Zipf

5.4 Discussão dos resultados

Este capítulo apresentou os resultados experimentais obtidos com a MCC. Os resultados das simulações mostraram que a MCC **LS+** consegue um desempenho melhor que os esquemas **LS-** e **LS**, que simulam os esquemas de *proxies* dinâmicos com *ring buffers*. Em particular, MCC **LS+** se mostrou muito mais escalável do que MCC **LS** e **LS-**. Os resultados para um *proxy* com uma *cache* com o tamanho de 10% do conteúdo total do servidor (100 vídeos) e tempo entre chegadas de 2.5 segundos mostraram que MCC **LS+** teve uma taxa de bloqueio igual a zero, enquanto os outros esquemas, **LS-** e **LS**, tiveram uma taxa de bloqueio de aproximadamente 20% e 60% respectivamente. Mesmo atendendo mais clientes, a taxa de utilização e a taxa pico de utilização de **LS+** se mantiveram praticamente iguais ou menores do que as obtidas com **LS-**. Além disso, a latência aumentou em média apenas 4,5 segundos a mais que **LS-** e 3,5 segundos a mais que **LS**. Estes resultados corroboram a modelagem de MCC **LS+** como um problema NP-completo. Os esquemas atuais de alocação de espaço em cache para alocar (*ring buffers*) não levam em conta a possibilidade de se gerenciar o espaço extra para encadear dois ou mais *ring buffers*. Com isto, os esquemas de *ring buffers* tradicionais precisam ser muito conservadores no momento de alocar espaço em cache para encadear *ring buffers*. Isto se deve ao fato de que o erro de se alocar muito espaço pode se estender durante toda a exibição do vídeo, que pode durar algumas horas. Com isso, só *ring buffers* próximos são encadeados (o que normalmente ocorre apenas para vídeos muito populares), não aproveitando todo o espaço de armazenamento disponível no *proxy*. MCC **LS+** não só mostra que este espaço na cache alocado para encadeamento pode ser gerenciado, como ele aumenta a escalabilidade do sistema de VoD, permitindo também o encadeamento de vídeo menos populares enquanto houver espaço em *cache*.

Capítulo 6

Trabalhos relacionados

Este capítulo discute os trabalhos relacionados à MCD e à MCC. Em relação à MCD, são discutidas as propostas encontradas na literautura de distribuição de vídeo segundo o modelo *peer-to-peer*. No que diz respeito à MCC, são comparadas as propostas de gerenciamento de *proxies* de vídeo em Rede de Distribuição de Conteúdo (CDN).

6.1 Trabalhos relacionados a MCD

A **MCD** [21, 20, 22, 19], uma das propostas desta tese, é uma técnica que gerencia os buffers dos clientes como uma área de Gerenciamento Global de Memória para sistemas de VoD. Para otimizar o reuso do conteúdo dos buffers dos clientes, a MCD faz uso extensivo de *chaining* e uma versão *peer-to-peer* de *patching*, o *patching* fornecido por um cliente e não pelo servidor. Desta forma, MCD combina chaining com batching, possibilitando o aproveitamento de técnicas de *multicast*, conseguindo a mesma escalabilidade de *chaining* com um tráfego muito menor. Um protótipo da MCD para ambientes de redes locais com *backbone* colapsada foi implementada, o GloVE [55, 56], comprovando a viabilidade prática desta técnica.

Como em sistemas VoD o tradicional modelo cliente/servidor não se mostrava escalável, a alternativa peer-to-peer começava a se mostrar viável no final do século passado. Inicialmente, surgiram sistemas que começaram a explorar a capacidade do cliente em proveito do sistema. Primeiro surgiu *chaining* [89], uma técnica destinada a encadear os clientes de forma que o conteúdo do *playout buffer* do cliente não fosse descartado, mas fosse reaproveitado sendo transmitido a outro cliente. Com isto os *playout buffers* dos clientes foram transformados em linhas de retardo que poderiam ser encadeados formando longas cadeias de clientes. Em seguida surgiu *patching* [53], que explora a largura de banda da interface de rede do cliente, permitindo que um cliente

receba ao mesmo tempo mais de um fluxo de vídeo. Desta forma, um cliente que entre em um *multicast* já estabelecido, consegue recuperar o trecho de vídeo perdido através de um remendo, o *patch* fornecido pelo servidor de VoD.

Quando a idéia da MCD surgiu, ela estava dissociada das técnicas já citadas, embora as utilizasse. MCD surgiu mais como uma nova aplicação da Memória Compartilhada Distribuída em Software (SW-DSM) [50, 70, 75] para distribuição de vídeo. Como a aplicação de VoD só realiza leituras e o acesso é muito regular, os protocolos tradicionais de SW-DSM poderiam ser bastantes simplificados, e o foram, dando origem a MCD. No mesmo período, após a publicação dos primeiros trabalhos da MCD, surgiram vários artigos e iniciativas da indústria para a distribuição de vídeo utilizando o modelo *peer-to-peer*. Abaixo citamos os trabalhos encontrados, após pesquisa bibliográfica, e suas principais diferenças em relação à MCD.

SpreadIt [29, 30]. Esta técnica foi proposta para distribuição de vídeo ao vivo. Basicamente ele emprega *chaining* para a construção de uma árvore de *multicast* ao nível da aplicação. Como na transmissão de vídeo ao vivo a parte perdida da transmissão não interessa ao usuário, basta colocar o cliente na árvore gerada por *chaining* para este receber o vídeo.

Coopernet [96, 95]. Esta técnica também usa *chaining* para distribuição de vídeo, mas para ser utilizado em eventos que causam um aumento rápido e dramático na taxa de requisições que chegam ao servidor, denominado *flash crowds*. Na ocorrência de *flash crowds*, o serviço de entrega de vídeo, que normalmente segue o modelo cliente/servidor, passa a utilizar o modelo peer-to-peer para aumentar a escalabilidade no atendimento dos pedidos que chegam.

P2Cast [103, 102]. O esquema *peer-to-peer* de P2Cast é muito semelhante ao da MCD, apesar de atualmente só funcionar com vídeos CBR (Continuous Bit Rate). A principal diferença entre P2Cast e a MCD é a política para construir a árvore de *chaining*. Enquanto a MCD procura construir uma árvore com políticas que visem ou a maximizar o multicast ou a maximizar o *chaining* ou a ter um meio termo entre as duas políticas, P2Cast constrói a árvore de *chaining* procurando o caminho mais rápido e menos congestionado.

Allcast, vTrails, BlueFalcon [5, 98, 8]. Existem diversas iniciativas da indústria em tornar viável a distribuição de vídeo em um modelo *peer-to-peer* pela Internet que dizem prover vídeo ao vivo ou sob demanda. Apesar de não ser possível fazer comparações específicas devido à falta de informações publicadas dos respectivos serviços, da pesquisa

do material de divulgação se concluiu que os esquemas também se baseiam em *chaining*.

Redes de *Overlay* [69]. O modelo peer-to-peer é frequentemente usado para oferecer serviços que poderiam ser oferecidos em nível de protocolo IP. É uma estratégia muito útil e economicamente viável para contornar a dificuldade de implementar novos serviços. Por exemplo, para se diponibilizar o *multicast* sobre uma rede IP [90] sem a necessidade de comprar novos roteadores, protocolos em nível de aplicação poderiam fornecer este serviço. De fato, a MCD e seus similares *peer-to-peer* podem ser considerados redes de overlay específicas para um determinado tipo de aplicação, uma vez que estabelecem uma nova rede sobre a existente para a distribuição de vídeo.

6.2 Trabalhos relacionados à MCC

A Memória Cooperativa Colapsada [24] teve origem na observação de que o esquema *peer-to-peer* da Memória Cooperativa Distribuída poderia ser usada para gerenciar a *cache* de um *proxy* de vídeo. Desta forma se conseguiria agregar as vantagens da escalabilidade de um sistema *peer-to-peer* com a economia de tráfego no *backbone* da rede proporcionada por um *proxy*. MCC implementa a gerência da *cache* como se esta fosse um prolongamento do *playout buffer* do cliente. Desta forma, o *playout buffer* do cliente é colapsado no *proxy* na forma de um *ring buffer*, doravante denominado apenas *buffer* colapsado, e, no caso da MCC, este pode cooperar com outros *buffers* colapsados. A novidade de MCC está no fato dela administrar todos os *buffers* colapsados de um vídeo com uma estrutura de dados que permite além de uma visão global, operações de relacionamento entre *buffers* colapsados e otimizações não conseguidas por esquemas anteriores. De fato, mostramos através de simulações que MCC consegue um desempenho superior aos esquemas anteriores que usam *ring buffers*.

A literatura existente a respeito de *proxies* é muita vasta. Podemos dividí-la em duas. Proxies convencionais, que tratam de mídias texto e imagens estáticas e *proxies* de mídias contínuas. MCC se enquadra na segunda categoria. Por sua vez, esta categoria pode ser subdividida em *proxies* de mídia contínua com *cache* estática e dinâmica. *Proxies* de mídia continua com *cache* estática não levam em conta as relações temporais entre os diversos segmentos em que um vídeo pode ser subdividido. Proxies de vídeo com *cache* dinâmica levam isto em conta. Portanto, MCC se enquadra na categoria de *proxies* de mídias contínuas dinâmicas.

6.2.1 *Proxies* convencionais

O gerenciamento de *caches* de *proxies* convencionais é bastante diferente da de *proxies* de vídeo. A principal diferença é que *proxies* convencionais manipulam objetos e *proxies* de vídeo manipulam segmentos de objetos. Como um único objeto de vídeo é capaz de ocupar toda a *cache*, uma granularidade mais fina é necessária, por isso o uso de segmentos de objetos de vídeo. Uma das principais métricas de desempenho de todo algoritmo de gerenciamento de *cache* é a sua taxa de acerto (*hit*). Para que esta taxa seja alta, uma das estratégias é manter em *cache* o que é mais acessado. No caso de *proxies* convencionais, mantém-se em *cache* as páginas HTML estáticas e imagens mais populares. No caso do vídeo, a questão não se resume a manter em *cache* os vídeos mais populares, mas quais segmentos deste vídeo devem permanecer em *cache*. Por exemplo, vale mais a pena manter os segmentos iniciais (prefixo) do vídeo que o seu sufixo [88] Quando a *cache* está cheia, em um *proxy* convencional, a preocupação é com a escolha do objeto vítima do descarte. Em um *proxy* de vídeo, é com o segmento do objeto que será descartado. Em um *proxy* convencional, algoritmos de substituição simples procuram escolher um objeto que tenha a menor probabilidade de ser requisitado novamente. Como estes algoritmos não preveêm o futuro, suas escolhas se baseiam em políticas que levam em conta a freqüência de acesso ao objeto ou no tempo que este objeto está em *cache*. Exemplos desta políticas são os algoritmos LFU (Least Frequently Used) e LRU (Least Recently Used) respectivamente.

Algoritmos mais complexos para gerenciar *caches* de *proxies* convencionais levam em conta não só a popularidade, mas o custo do objeto. Este custo está relacionado com o tamanho do objeto e o custo de trazê-lo até o *proxy*. Manter um objeto grande na *cache* é caro. É mais barato manter um objeto pequeno em *cache*. Por outro lado, manter um objeto grande em *cache* é mais econômico do que se for preciso trazê-lo de novo. Conseqüentemente, é mais barato e mais fácil trazer um objeto pequeno de volta. Neste caso, o algoritmo de substituição procura maximizar a economia em manter um objeto em *cache*. Logo, o objeto que se traduz por uma menor economia de custos é descartado. A família dos algoritmos denominados GreedyDual usa esta política de substituição em *proxies* convencionais [68]. Young [65, 66] prova que esta família de algoritmos *on-line* (algoritmos cuja resposta não depende das requisições futuras) é k-competitiva para objetos de tamanho fixo. Ou seja, esta garantia de performance é a melhor possível para qualquer algoritmo *on-line* determinístico. Cao [68] prova que GreedyDual continua sendo otimamente competitivo para objetos de diferentes tamanhos. GreedyDual* [86]

melhora GreedyDual acrecentando à função de custo a popularidade do objeto. Ou seja, se o objeto grande for muito popular a economia será grande. Para um objeto pequeno produzir a mesma economia, sua popularidade deve ser muitas vezes maior que a de um objeto grande. MCC LS+ é parecido com GreedyDual se levarmos em conta que sua função custo para escolher a seqüência de *slots* de ligação a ser descartada é proporcional à sua distância até o final do vídeo e inversamente proporcional ao seu tamanho. A distância até o final do vídeo é o custo de se estabelecer um fluxo regular do servidor durante o tempo dado pela distância da seqüência de *slots* de ligação até o fim do vídeo. O tamanho da seqüência de *slots* de ligação é a quantidade de memória da cache a ser liberada caso a seqüência seja a escolhida.

6.2.2 *Proxies* de Vídeo com *Cache* Estática

MiddleMan [80]. Esta é uma das primeiras abordagens que tratam de *proxies* especializados exclusivamente em vídeo. MiddleMan fragmenta um objeto de vídeo em diversos blocos de mesmo tamanho. Estes blocos podem ser espalhados por diversos *proxies*, aumentando-se o espaço de armazenamento da *cache*. Estes blocos são a unidade de substituição na *cache*. MiddleMan testa diversas políticas de substituição na *cache*, sendo que o melhor resultado foi obtido com uma política LRU modificada. Esta modificação leva em conta o histórico dos últimos acesso a *cache*. Como o acerto na *cache* pode-se dar em qualquer *proxy*, a latência para se obter o segmento de vídeo varia com a distância até o *proxy*. Ou seja, não existe QoS, sendo assim um serviço de melhor esforço.

Prefix Caching [88]. Esta abordagem leva em conta que os usuários na Internet costuman assistir apenas o começo do vídeo e, devido à má qualidade na exibição ou ao desinteresse pelo conteúdo, não vêem o vídeo até o final. Desta forma, armazenar apenas o prefixo no *proxy*, atende a três métricas importantes. Primeiro, diminui a latência para se buscar a parte inicial do vídeo. Segundo, poupa largura de banda no servidor e na rede, uma vez que os prefixos constituem a maior parte das requisições. Terceiro, consegue uma boa taxa de acertos na *cache*.

Silo, Rainbow and Caching Token [101]. Este trabalho é um misto entre as abordagens anteriores. Usa segmentos para gerenciar objetos de vídeo, mas estes segmentos têm tamanho variável. O tamanho do segmento é definido previamente, de acordo com sua posição no vídeo. Estes segmentos podem estar armazenado em qualquer *proxy* da rede e não estão replicados pelos *proxies*. O descarte dos segmentos leva em

conta, além da popularidade do vídeo, se eles fazem parte do prefixo ou não. Este esquema não garante uma latência baixa para o prefixo, uma vez que este pode estar em um *proxy* distante. No entanto, poupa largura de banda no servidor, apesar de não diminuir muito o tráfego no *backbone* da rede, uma vez que o *proxy* que irá fornecer o conteúdo não é necessariamente o mais próximo do cliente. Este esquema também consegue uma boa taxa de acertos, mas é um serviço de melhor esforço.

Lazy segmentation [82]. Este trabalho difere de Silo por atrasar a definição do tamanho do segmento o máximo possível, levando em conta para isso o comportamento dos clientes em tempo real e não a posição do segmento no vídeo.

6.2.3 *Proxies* de Vídeo com *Cache* Dinâmica

Soccer [60, 16]. Soccer usa duas políticas distintas para gerenciar a *cache*; uma é estática e a outra dinâmica. Na *cache* estática Soccer implementa a segmentação de objetos de vídeo, com segmentos de mesmo tamanho. Para aumentar a taxa de acertos na *cache* e levando em conta o fato que o acerto de um segmento implica que o próximo segmento também será requisitado, Soccer agrupa segmentos em *chunks*. Os *chunks* têm o tamanho de k segmentos. O gerenciamento da *cache* é feita da seguinte maneira: Cada *chunk* é armazenado na *cache* independentemente dos outros chunks. A unidade de substituição da *cache* é o segmento. Soccer usa prefixos de chunks na sua *cache*, i.e., se segmentos de um chunk devem ficar na *cache*, a prioridade é para os seus segmentos iniciais, ou seja, o prefixo. Quando um segmento de um *chunk* é acessado, nenhum outro segmento deste *chunk* pode ser descartado, isto garante os próximos acertos (*hits*) se os segmentos estiverem em *cache*. Se algum segmento de um *chunk* for escolhido para ser descartado pelo algoritmo de substituição, o último segmento do prefixo do *chunk* é que deve ser descartado.

O gerenciamento dinâmico de soccer é feito através de *ring buffers*. *Ring buffers* são usados para esconder a distância temporal entre duas requisições. Soccer usa *ring buffer* em memória para armazenamento de curto prazo e *cache* estática para armazenamento de longo prazo em disco. Soccer também limita o tamanho do *ring buffer*. A distância máxima temporal entre duas requisições que serão servidas por um mesmo *ring buffer* é chamada de distância temporal entre *ring buffers*. O *ring buffer* pode ser alimentado tanto pelo servidor como pela *cache* estática em disco. O fluxo que chega do servidor é usado tanto para alimentar o *ring buffer* como a *cache* estática em disco.

O *proxy* pode servir o cliente usando tanto a *cache* estática como a dinâmica,

dependendo dos recursos disponíveis no sistema.

Soccer funciona segundo o esquema **LS**, mas com um serviço do tipo melhor esforço. Se houver espaço em *cache*, o *ring buffer* é alocado. O *ring buffer* pode ser alimentado tanto por um fluxo vindo do servidor com por uma *cache* estática na memória secundária do próprio *proxy*. Se não houver espaço em *cache*, o pedido do cliente é atendido pela *cache* estática. Por usar o esquema **LS**, Soccer não é tão eficiente no uso da memória primária como a MCC **LS+**. Além disso, o gerenciamento da memória primária é independente do gerenciamento da memória secundária. Tanto que a primeira usa *cache* dinâmica e a segunda estática. Embora a simulação da MCC feita para esta tese não use explicitamente a memória secundária, nada impede que a MCC também a use. Esta integração entre os doís níveis de memória poderia ser feita de duas formas. A primeira, como em Soccer, complementando a *cache* dinâmica em memória primária com a *cache* estática na memória secundária. A outra, seria integrando as duas memórias, de forma que ambas trabalhassem de forma síncrona e dinâmica. Neste caso, a estrutura vetor de slots e vetor de vídeo permite que todo armazenamento do *proxy* seja gerenciado segundo um paradigma análogo ao do gerenciamento de memória virtual. O que não pode estar na memória primária é posto na memória secundária. O critério para decidir quando uma unidade de vídeo deve estar em qual memória seria dado pelos *slots* a que estas unidades de vídeo pertencem. Por exemplo, se as unidades de vídeo pertencem a *slots* do tipo RS, significa que elas serão enviadas em seguida para o cliente, logo devem permanecer na memória primária. Assim que a unidade de vídeo sai do slot RS e entra no *slot* BS, significa que esta unidade de vídeo já foi enviada ao cliente e portanto pode ser paginada para a memória secundária. Da mesma forma, unidades de vídeo pertecentes a *slots* LS, não necessitam estar na memória primária, pois não serão usadas em seguida. Logo, se faltar espaço na memória primária, estas unidade de vídeo são candidatas a serem paginadas para a memória secundária.

Coordenação entre servidor e *proxy* para entrega de vídeo [67, 10]. Este trabalho é semelhante ao Soccer e enfatiza a coordenação entre o servidor e o *proxy* para a entrega de vídeo. Esta coordenação é obtida através de um *ring buffer*. Desta forma, o *proxy* só serve o cliente se tiver como alocar um *ring buffer* e este *ring buffer* puder ser alimentado pelo servidor, garantindo desta forma uma QoS na entrega do vídeo.

Em relação ao tratamento dado aos prefixos dos vídeos [88, 7] Venkatramani [10] estende o trabalho de [67] ao complementar o esquema de *ring buffers* com um esquema que estabelece o conceito de *working set* [72] ideal de prefixos, com prefixos de tamanho

variável. Tanto o working set como o tamanho do prefixo são estabelecidos de acordo com a popularidade do vídeo. O esquema é implementado tomando por base um valor c que é a menor unidade de alocação na *cache*. Ou seja, todas as alocações na *cache* são em múltiplos desta unidade. O algoritmo consiste em calcular a taxa agregada de transmissão no *backbone* da rede (Rw) para cada incremento no tamanho do prefixo em unidades de c. Rw é basicamento o custo para transmitir o sufixo do vídeo (enviado pelo servidor) dividido pelo distância temporal entre dois pedidos. Ou seja, mantendo-se a distância temporal entre dois pedidos e aumentando-se o tamanho do prefixo, Rw decresce monotonicamente. O algoritmo gulosamente enche a *cache* com prefixos cada vez maiores de forma que o custo Rw seja minimizado. Este algoritmo foi comparado com a solução ótima dada pela modelagem deste esquema como um problema de programação dinâmica que é NP-completo. A comparação entre as duas soluções mostrou que o algoritmo guloso também conseguia a solução ótima. O problema deste algoritmo, é que ele precisa ser executado de tempos em tempos para recalcular quais prefixos ficarão na *cache* e seus respectivos tamanhos. Diferentemente, na MCC **LS+** a definição do tamanho do prefixo e de quais ficarão em *cache* é feita sem a necessidade de um algoritmo específico. A própria gerência dos *slots* de ligação e alocação dos *buffers* colapsados dinamicamente define o tamanho do prefixo que ficará em *cache*.

Por exemplo, se um vídeo for muito popular, a distância entre dois pedidos é pequena e portanto os buffers colapsados estarão todos encadeados sem a necessidade do uso de slots de ligacão. Neste caso todo o vídeo estará em *cache*, ou seja, o tamanho do prefixo é igual ao tamanho do vídeo. Para vídeos um pouco menos populares, os *buffers* colapsados estarão encadeados por seqüências pequenas de *slots* de ligação. Para vídeos menos populares ainda, a sequência de *slots* de ligação é maior, uma vez que a distância entre pedidos é maior. Caso falte espaço na *cache*, a MCC **LS+** procura desalocar seqüências de *slots* de ligação com o melhor custo benefício para o sistema, ou seja, seqüências que estão próximas do fim do vídeo e grandes o suficiente para liberar o espaço requisitado. Desta forma, após encher a *cache*, o tamanho dos prefixos é diminuido à medida que novos *buffers* colapsados tenham que ser alocados e encadeados. Desta forma, a tendência é termos prefixos de tamanho variável, de acordo com a sua popularidade.

Verscheure [67] propõe um esquema semelhante a Soccer, mas com um serviço com QoS. Ou seja, ele não prevê uma *cache* secundária estática com o objetivo de atender a mais clientes. Todo pedido que chega é atendido por um *ring buffer* ou o pedido é bloqueado. Além disso Verscheure é explícito em afirmar que o *ring buffer* trabalha de

forma coordenada com o servidor de vídeo. Ou seja, se o fluxo de vídeo é entregue com determinada QoS para o *ring buffer*, ele também poderá ser entregue ao cliente com a mesma QoS. No entanto, como Soccer, o esquema de Verscheure funciona segundo a política **LS**.

Chen [83] funciona de maneira semelhante a soccer. A diferença é que caso chegue um pedido por um vídeo mais popular que o de menor popularidade em *cache* e não haja espaço em *cache*, o *ring buffer* do vídeo de menor popularidade e que esteja mais distante do final do vídeo é desalocado. O pedido que perdeu o *ring buffer* passa a ser servido pela *cache* estática em disco com um serviço do tipo melhor esforço ou diretamente pelo servidor. MCC **LS+** é mais eficaz, pois apenas descarta os slots de ligação e continua mantendo o serviço contratado pelo cliente.

Chan [85] mostra analiticamente, através de uma função de custo, que esquemas que usam *ring buffers* em *proxies* são capazes de fornecer serviços de VoD a um custo mais baixo.

6.2.4 Outros trabalhos

Como visto anteriormente, existem diversos trabalhos na área de distribuição de vídeo utilizando CDN. CDN implica em utilizar-se *proxies* na fronteira da rede de distribuição. No entanto, alguns trabalhos se preocupam não com a gerência da cache do proxy, mas com a forma com que o vídeo é entregue pelo proxy ao cliente. Almeida [44] desenvolve um modelo de custos para armazenar vídeos em *proxies* que utilizam o protocolo *bandwidth skimming* [15] para entrega de vídeo ao cliente. *Proxies* MCC não utilizam nenhum protocolo especial para entregar o fluxo de vídeo aos clientes.

Capítulo 7

Conclusões

Sistemas de distribuição de Vídeo sob Demanda eram tidos como uma das aplicacões que iriam revolucionar o modo de vida a que estamos acostumados. A decisão do que assistir, do quando assistir e do onde assistir não estaria mais limitada à programação das emissoras de televisão ou de salas de exibição. Além disso, poderia ser uma das aplicações mais rentáveis para as concessionárias de telecomunições que poderiam vender serviços de entrega de vídeo sob demanda. O serviço de VoD possui um alto valor agregado e um grande potencial de demanda, levando-se em conta que a sociedade cada vez mais valoriza o setor de serviços, incluso aqui o setor de entretenimento e o de educação à distância. No entanto, este tipo de aplicação consome tantos recursos da infra-estrutura de comunicações de dados e dos servidores de vídeo que sistemas escaláveis e economicamente viáveis se tornaram um desafio.

Esta tese teve por objetivo propor uma solução que aumentasse a escalabilidade dos sistemas de VoD. Para isto, este trabalho explora a capacidade de esquemas cooperativos de memória para distribuição de vídeo sob demanda. Com esse enfoque, projetamos e avaliamos duas formas de distribuição de vídeo sob demanda, uma usando o modelo *peer-to-peer* de distribuição de vídeo, a MCD, e outra inserida no contexto de uma rede de distribuição de conteúdo (CDN), a MCC.

MCD foi uma das primeiras propostas de distribuição de VoD *peer-to-peer*. Em particular, MCD revelou-se ideal para ambientes corporativos, onde pode explorar a capacidade de redes locais com grande largura de banda. Com a MCD, é possível implantar um sistema de VoD escalável economicamente viável. Uma rede local com comutadores Ethernet ou Fast Ethernet, requisito da MCD, já é uma realidade em muitas empresas. O servidor de VoD, antes restrito a máquinas de alto desempenho, com a MCD, tem a possibilidade de viabilizar um sistema de VoD escalável, utilizando para tanto computadores pessoais de baixo custo. De fato, um protótipo da MCD, o GloVE, foi

implementado e testado no Laboratório de Computação Paralela (LCP) da COPPE/UFRJ, com a configuração descrita, como parte de uma Dissertação de Mestrado [54] fruto deste trabalho. GloVE demonstrou na prática a viabilidade de sistemas de VoD *peer-to-peer*, onde a sinergia dada pela cooperação dos diversos *playout buffers* dos clientes possibilitou sistemas de VoD escaláveis e de baixo custo.

Para a distribuição de Vídeo sob Demanda em redes geograficamente mais amplas do que uma rede local, projetou-se a MCC. A idéia da MCC é colapsar parte da memória do *playout buffer* dos clientes em um proxy. Desta forma, a *cache* do proxy complementaria o *playout buffer* do cliente. Além desta cooperação entre o *playout buffer* do cliente e a *cache* do proxy, MCC otimiza o gerenciamento da cooperação entre os *buffers* colapsados na *cache* do *proxy*. Para esta otimização, provamos que o gerenciamento da MCC é um problema NP-completo. Com base nisto, elaborou-se a política **LS+** para a substituição de *slots* na *cache* do *proxy*. Através de simulações detalhadas, demonstrou-se que os ganhos de desempenho obtidos por **LS+** são plenamente satisfatórios. Em particular MCC **LS+**, mostrou-se mais escalável que **LS-** e **LS**, utilizados pelos esquemas existentes.

Desta forma, o estudo realizado nesta tese demonstra que a idéia de se utilizar a memória de forma cooperativa para distribuição de vídeo, em contexto bem definidos, como em uma rede local com *backbone* colapsada para a MCD e em uma CDN para a MCC, pode melhorar o desempenho dos sistemas de distribuição de vídeo sob demanda.

Diversos trabalhos e pesquisas já foram feitos na área. Os avanços conseguidos são significativos, mas ainda há muito o que ser feito para se viabilizar sistemas de VoD que sejam escaláveis. Da mesma forma que trabalhos anteriores não esgotaram este assunto, este trabalho também não tem esta pretensão. Muito trabalho há que ser feito na área para que aplicações do tipo VoD se tornem viáveis economicamnete. Além disso, nossos resultados se limitam a sistemas de VoD que ou possuam grande largura de banda no *backbone* da rede e no acesso dos clientes ou em Redes de Distribuição de Conteúdo cujos clientes possuem no mínimo um acesso ADSL que permita receber o vídeo em tempo real.

7.1 Trabalhos Futuros

Futuramente, pretende-se investigar a implementação de operações de videocassete tanto para a MCD como na MCC. Também pretende-se estudar o comportamento da MCC face a cooperação entre proxies, bem como formas de otimizar esta interação.

Outro trabalho bastante interessante, em andamento, é a implementação do protótipo

da MCC **LS+**. A implementação deste protótipo apresenta desafios interessantes, como a integração do gerenciamento da memória primária e secundária sob um único esquema, análogo ao gerenciamento de memória virtual. Este trabalho vem sendo desenvolvido no LCP como parte de uma Dissertação de Mestrado.

Uma das dificuldades de se implantar sistemas de VoD é a falta de redes de acesso com largura de banda suficiente para a entrega de vídeos com uma qualidade aceitável. A tecnologia de redes sem fio é uma alternativa economicamente promissora para a instalação de novas redes de acesso. Desta forma, outra proposta interessante é pesquisar a viabilidade da MCD e da MCC face à limitação da largura de banda e compartilhamento do meio de transmissão em redes sem fio. Ainda em redes sem fio, o uso da MCD e da MCC em redes móveis é uma outra alternativa interessante.

Por último, uma proposta mais ousada seria adaptar tanto a MCD como a MCC para redes sem fio em malha (*mesh networks*). Neste tipo de rede, os equipamentos dos clientes possuiriam interfaces de rede sem fio de pequeno alcance, mas grande largura de banda. Desta forma, vários canais de comunicação poderiam ser estabelecidos com os vizinhos e uma quantidade enorme de informação poderia ser transmitida em redes deste tipo. Tanto o uso da MCD como da MCC em redes de malhas sem fio parecem ser bastante promissoras.

Referências Bibliográficas

[1] A. Dan, D. Sitaram, P. Shahabuddin. "Dynamic batching policies for an on-demand video server". *Multimidia Systems*, v. 3, n. 4, pp. 112–121, 1996.

[2] A. Krikelis. "Parallel Multimedia Servers". *IEEE Concurrency*, v. 6, n. 1, pp. 16–19, 1998.

[3] A. Oram. *Peer-to-peer: harnessing the power of disruptive technologies*, chapter 1, pp. 3. O'Reily & Associates Inc, 2000.

[4] A. Srivastava, A. Kumar, A. Singru. "Design and analysis of a video-on-demand server". *Multimedia Systems*, v. 5, n. 4, pp. 238–254, 1997.

[5] Allcast. "http://www.allcast.com".

[6] B. Gauthier, Le Boudec, P. Oechslin. "SMART: a Many-to-Many MUlticast Protocol for ATM". *IEEE Journal on Selected Areas in Communications*, v. 15, pp. 458–472, março 1997.

[7] B. Wang, S. Sen, M. Adler, D. Towsley. "Optimal Proxy Cache Allocation for Efficient Streaming Media Distribution". In: *Proceedings of IEEE INFOCOM 2002*, New York, NY, USA, junho 2002.

[8] BlueFalcon. "http://www.bluefalcon.com".

[9] C. E. Leiserson. "Fat Tree: Universal Networks for Hardware-Efficient Supercomputing". *IEEE Transactions on Computers*, pp. 272–285, outubro 1985.

[10] C. Venkatramani, O. Verscheure, P. Frossard, K.-W. Lee. "Optimal Proxy Management for Multimedia Streaming in Content Distribution Networks". In: *Proceedings of International Workshop on Networking and Operating System Support for Digital Audio and Video*, Miami, Florida, maio 2002.

[11] D. C. Verma. *Content Distribution Networks an Engineering Approach.* John Wiley & Sons, 2002.

[12] D. Le Gall. "MPEG: A video compression standard for multimedia applications". *Communications of the ACM*, v. 34, n. 4, pp. 46–58, abril 1991.

[13] D. Tennehouse et all. "A survey of active network research". *IEEE Communications Magazine*, pp. 80–86, janeiro 1997.

[14] D. Wu, Y.T. Hou, W. Zhu, Y. Zhang, J.M. Peha. "Streaming Video over the Internet: Approach and Directions". *IEEE Transactions on Circuits and Systems for Video Technology*, v. 11, n. 3, pp. 282–300, março 2001.

[15] Derek Eager, Mary Vernon, John Zahorjan. "Bandwidth Skimming: A Technique for Cost-Effective Video-on-Demand". In: *Proceedings of Multimedia Computing and Networking 2000*, San Jose, CA, janeiro 2000.

[16] E. Bommaiah, K. Guo, M. Hofmann, S. Paul. "Design and Implementation of a Caching System for Streaming Media over the Internet". In: *Proceedings of IEEE Real-Time Technology and Applications Symposium - RTAS*, Washingto, DC, maio 2000.

[17] E. Chang, H.G. Molina. "Reducing Initial Latency in Media Servers". *IEEE Multimedia*, v. 4, n. 3, pp. 50–61, 1997.

[18] E. Chang, H.G. Molina. "MEDIC: A Memory & Disk Cache for Multimedia Clients". In: *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Florence, Italy, junho 1999.

[19] E. Ishikawa, C. Amorim. "Cooperative Video Cache for Interactive and Scalable VoD Systems". In: *Proceedings of International Conference on Networking, ICN'01*, Colmar, France, julho 2001.

[20] E. Ishikawa, C. L. Amorim. "Memória Cooperativa Distribuída para Sistemas de Video sob Demanda Interativa e Escalável". *Pedido de patente depositado no INPI - PI0001810-4*, Março 2000.

[21] E. Ishikawa, C. L. Amorim. *Memória Cooperativa Distribuída: Uma solução escalável para Sistemas de Vídeo sob Demanda interativos.* Technical report, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, outubro 2000.

[22] E. Ishikawa, C. L. Amorim. "Memória de Vídeo Cooperativa para Sistemas de VoD peer-to-peer ". In: *Anais do 19 Simpósio Brasileiro de Redes de Computadores, SBRC 2001*, Florianópolis-SC, Brasil, maio 2001.

[23] E. Ishikawa, C. L. Amorim. "Memória Cooperativa Distribuída Colapsada para Sistemas de Mídia-sob-Demanda Interativa e Escalável". *Pedido de patente depositado no INPI - PI0201115-8*, abril 2002.

[24] E. Ishikawa, C. L. Amorim. "Collapsed Cooperative Video Cache for Content Distribution Networks". In: *Anais do 21 Simpósio Brasileiro de Redes de Computadores, SBRC'03*, pp. 249–264, Natal-RN, Brasil, maio 2003.

[25] E. Ishikawa, C. L. Amorim. "Near Optimal Video Cache Management in Streaming Proxies". *COPPE/UFRJ, Relatório Técnico ES-610-03*, Agosto 2003.

[26] F. Meylan et all. "Análise Comparativa de Algoritmos de Roteamento Multicast para Comunicacao Multimidia em Redes ATM". In: *Anais do 17 Simposio Brasileiro de Redes de Computadores -0 SBRC'99*, pp. 175–190, 1999.

[27] G. Waters. "A New Heuristic for ATM Multicast Routing". In: *Proceedings of 2nd IFIP on Performance Modelling and Evaluation of ATM Networks*, pp. 8/1–8/9, julho 1994.

[28] GNUTELLA. "Gnutella". *http://www.oreillynet.com/topics/p2p/gnutella/*.

[29] H. Deshpande, M. Bawa, H. Garcia-Molina. *Streaming Live Media over a Peer-to-Peer Network.* Technical report, Stanford Universisty, Department of Computer Science, abril 2001.

[30] H. Deshpande, M. Bawa, H. Garcia-Molina. *Streaming Live Media over Peers.* Technical report, Stanford Universisty, Department of Computer Science, março 2002.

[31] H.M. Vin, A. Goyal, A. Goyal, P. Goyal. "An Observation-based Admission Control Algorithm for Multimedia Servers". In: *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pp. 234–243. Springer-Verlag, 1994.

[32] I. Clark, O. Sandberg, B. Wiley, T. Hong . "Freenet: A distributed anonymous information storage and retrieval system". In: *Proceedings of the ICSI Workshop on Design Issues in Anonimity and Unobservability*, 2000.

[33] ISO/IEC. "ISO/IEC 11172-1:1993 Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 1: Systems". 1993.

[34] ISO/IEC. "ISO/IEC 11172-1:1993 Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 2: Video". 1993.

[35] ISO/IEC. "ISO/IEC 11172-1:1993 Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio". 1993.

[36] ISO/IEC. "ISO/IEC 13818-1:1998 Information technology – Generic coding of moving pictures and associated audio information – Part 3: Audio". 1998.

[37] ISO/IEC. "ISO/IEC 13818-1:2000 Information technology – Generic coding of moving pictures and associated audio information: Systems". 2000.

[38] ISO/IEC. "ISO/IEC 13818-1:2000 Information technology – Generic coding of moving pictures and associated audio information: Video". 2000.

[39] ISO/IEC. "ISO/IEC 14496-1:2001 Information technology – Coding of audio-visual objects – Part 1: Systems". 2001.

[40] ISO/IEC. "ISO/IEC 14496-1:2001 Information technology – Coding of audio-visual objects – Part 2: Video". 2001.

[41] ISO/IEC. "ISO/IEC 14496-1:2001 Information technology – Coding of audio-visual objects – Part 3: Audio". 2001.

[42] ITU-T. "ITU-T Rec H.264/ISO/IEC 11496-10, Advanced Video Coding, Final Committee Draft, Document JVT-E022". setembro 2002.

[43] J.-H. Cui, L. Lao, D. Maggiorini, M. Gerla. "BEAM: A Distributed Aggregated Multicast Protocol Using Bi-directional Trees". In: *Proceedings of IEEE ICC2003*, Anchorage, Alaska, USA, maio 2003.

[44] J. M. Almeida, D.L. Eager, M. Ferris, M. K. Vernon. "Provisioning Content Distribution Networks for Streaming Media". In: *Proceedings of 21st Annual Joint Conference of IEEE Computer and Communication Societies (INFOCOM 2002)*, New York, USA, junho 2002.

[45] J. M. Almeida, J. Krueger, D.L. Eager, M. K. Vernon. "Analysis of Educational Media Server Workloads". In: *Proceedings of 11^{th} International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001)*, Port Jefferson, NY, USA, junho 2001.

[46] J. R. Santos, R. Muntz. "Performance Analysis of the RIO (Randomized I/O) Storage Server". In: *Proceedings of the ACM Multimedia*, pp. 303–309, 1998.

[47] J.M. Smith et al. "Activating Networks: A Progress Report". *IEEE Computer*, v. 32, n. 4, pp. 32–41, abril 1999.

[48] J.Y.B. Lee. "Parallel Video Servers: A Tutorial". *IEEE Multimedia*, v. 5, n. 2, pp. 20–28, 1998.

[49] K. Lee, J.B. Kwon, H.Y. Yeom. "Exploiting Caching for Realtime Multimedia Systems". In: *Proceedings of International Conference on Multimedia Computing Systems, ICMCS'99*, Florence, Italy, junho 1999.

[50] K. Li, P. Hudak. "Memory coherence in shared virtual memory systems". In: *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*, pp. 229–239, agosto 1986.

[51] K.A. Hua, D. A. Tran, R. Villafane. "Caching multicast protocol for on-demand video delivery". In: *Proceedings of IS&T/SPIE conference on Mutimedia Computing and Networking*, San Jose, CA, USA, janeiro 2000.

[52] K.A. Hua, S. Sheu, J. Z. Wang. "Earthworm: A Network Memory Management Technique for Large-Scale Distributed Multimedia Applications". *IEEE INFOCOM'97*, 1997.

[53] K.A. Hua, Y. Cai, S. Sheu. "Patching: A Multicast Technique for True Video-on-Demand Services". *ACM Multimedia'98*, 1998.

[54] L. B. Pinho. *Implementação e Avaliação de um Sistema de Vídeo sob Demanda baseado em Cache de Vídeo Cooperativa.* Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2002.

[55] L. B. Pinho, E. Ishikawa, C. L. Amorim. "GloVE: A Distributed Environment for Low Cost Scalable VoD Systems". In: *Proceedings of 14th Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2002*, Vitória-ES, Brasil, outubro 2002.

[56] L. B. Pinho, E. Ishikawa, C. L. Amorim. "GloVE: A Distributed Environment for Scalable Video-on-Demand Systems". *International Journal of High Performance Computing Applications [IJHPCA]*, v. 17, n. 2, maio 2003.

[57] L. Bertini et all. "Análise de Desempenho do Servidor de Vídeo ALMADEN-VOD". In: *Anais do XXVI Seminário Integrado de Software e Hardware, SEMISH'99*, pp. 259–275, 1999.

[58] L. Breslau et al. "Advances in Network Simulation". *IEEE Computer*, pp. 59–67, maio 2000.

[59] L. Golubchik, J.C.S. Lui, R.R. Muntz. "Adaptive piggybacking: a novel technique for data sharing in video-on-demand storage servers". *Multimidia Systems*, v. 4, n. 3, pp. 140–155, 1996.

[60] M. Hofmann, T.S. Eugene Ng, K. Guo, S. Paul, H. Zhang. *Caching Techniques for Streaming Media over the Internet.* Technical report, Bell Laboratories, abril 1999.

[61] M. Holton, R. Das. "XFS: A Next Generation Journalled 64-bit Filesystem With Guaranteed Rate I/O". *SGI - Silicon Graphics Inc*, 1996.

[62] M. Kamath, K. Ramamritham, D. Towsley. "Continuous media sharing in multimedia database systems". In: *Proceedings of the 4th International Conference on Database Systems for Advanced Applications (DAS-FAA'95)*, Singapore, 1995.

[63] M. R. Garey, D. S. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman Company, USA, 1979.

[64] NAPSTER. "Napster protocol specification". *http://opennap.souceforge.net/napster.txt*, março 2001.

[65] N.E. Young. "On-Line File caching". In: *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, California, janeiro 1998.

[66] N.E. Young. "On-Line File caching". *Algorithmica*, v. 33, n. 3, pp. 371–383, 2002.

[67] O. Verscheure, C. Venkatramani, P. Frossard, L. Amini. "Joint server scheduling and proxy caching for video delivery". *Computer Communications*, v. 25, n. 4, pp. 413–423, abril 2002.

[68] P. Cao, S. Irani. "Cost-Aware WWW Proxy Caching Algorithms". In: *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, Monterey, California, USA, dezembro 1997.

[69] P. Keleher, B. Bhattacharjee, B. Silaghi. "Are virtualized Overlay Networks TooMuch of a Good Thing?". In: *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, Cambridge, MA, USA, março 2002.

[70] P. Keleher et all. "TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems". In: *Proceedings of the 1994 Usenix Conference*, pp. 115–131, janeiro 1994.

[71] P.C.S. Vidal, O.C.M.B. Duarte. "Reducing the Reservation Establishment Time in IP Tunnels by using Staged Refresh Timers". In: *Proceedings of 2nd IEEE International Symposium on Network Computing and Applications*, Cambridge, Massachusetts, USA, abril 2003.

[72] P.J. Denning, K. C. Kahn. "A study of program locality and lifetime functions". In: *Proceedings of the Fifth ACM Symposium on Operating System Principles*, novembro 1975.

[73] P.S. Yu, J.L. Wolf, H. Shachnai. "Design and Analisys of a look-ahead scheduling scheme to support pause-resume for VoD applications". *Multimedia Systems*, n. 3, pp. 137–149, 1995.

[74] R. Rooholamini, V. Cherkassky. "ATM-Based Multimedia Servers". *IEEE Multimedia*, v. 2, n. 1, pp. 39–52, 1995.

[75] R. Samanta et all. "Home-based SVM protocols for SMP clusters: design and performance". In: *Proceedings of the fourth International Symposium on High-Performance Computer Architecture*, fevereiro 1998.

[76] R. Stadler, M. Borla, C. Aurrecoechea. *Mcast: Design of the service and its management*. Technical report, Columbia University, dezembro 1997.

[77] R.L. Haskin, F.B. Schmuck. "The Tiger Shark File System". *IBM Almaden Research Center*, 1996.

[78] R.T. Ng, J. Yang. "An analisys of a buffer sharing and prefetching techniques for multimedia systems". *Multimedia Systems*, n. 4, pp. 55–69, 1996.

[79] S. A. Barnnet, G. J. Anido. "A Cost Comparison of Distributed and Centralized Approaches to Video-on-demand". *IEEE Journal on Selected Areas in Communication*, v. 14, n. 6, 1996.

[80] S. Acharya, B. Smith. "MidleMan: A Video Caching Proxy Server". In: *Proceedings of International Workshop on Networking and Operating System Support for Digital Audio and Video*, Chapel Hill, NC, junho 2000.

[81] S. Berson, R. Muntz, S. Ghandeharizadeh, X. Ju. "Staggered Stripping: A Flexible Technique to Display Continuous Media". *Multimedia Tools and Applications*, n. 1, pp. 127–148, 1995.

[82] S. Chen, B. Shen, S. Wee, X Zhang. "Adaptive and Lazy Segmentation Based Proxy for Streaming Media Delivery". In: *Proceedings of 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV'03*, Monterey, CA, USA, junho 2003.

[83] S. Chen, B. Shen, Y. Yan, S. Basu. *Shared Running Buffer based Proxy Caching of streaming Sessions*. Technical report, HP Laboratories, Palo Alto, março 2003.

[84] S. Ghandeharizadeh, L. Ramos. "Continuous Retrieval of Multimedia Data Using Parallelism". *IEEE Transactions on Knowledge and Data Engineering*, v. 5, n. 4, pp. 658–669, 1993.

[85] S.-H. G. Chan, F. Tobagi. "Distributed Servers Architecture for Networked Video Services". *IEEE/ACM Transactions on Networking*, v. 9, n. 2, abril 2001.

[86] S. Jin, A. Bestavros. "GreedyDual* Web Caching Algorithm - Exploiting the Two Sources of Temporal Locality in Web Request Streams". *International Journal on Computer Communications*, v. 24, n. 2, pp. 174–183, fevereiro 2001.

[87] S. Kang, H. Y. Yeom. "Modeling the Caching Effect in Continuous Media Servers". *Performance Evaluation*, agosto 1999.

[88] S. Sen, J. Rexford, D. Towsley. "Proxy Prefix Caching for Multimedia Systems". In: *Proceedings of IEEE Infocom 99*, abril 1999.

[89] S. Sheu, K.A. Hua, W. Tavanapong. "Chaining: A Generalized Batching Technique for Video-On-Demand Systems". In: *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, Ottawa, Canada, junho 1997.

[90] S. Shi, J. Turner. "Routing in Overlay Multicast Networks". In: *IEEE INFOCOM 2002*, junho 2002.

[91] S.E. Deering, D.R. Cheriton. "Host Groups: A Multicast Extension to Internet Protocol". *Internet RFC 966*, dezembro 1985.

[92] S.E. Deering, D.R. Cheriton. "Multicast Routing in Datagram Internetworks and Extended LANs". *ACM Transactions on Computer Systems*, v. 8, n. 2, 1990.

[93] T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms*. McGrawHill, 1990.

[94] T. P. Jimmy, B. Hamidzadeh. *Interactive Video-on-Demand Systems - Resource Management and Scheduling Strategies*. Kluwer Academic Publishers, Norwell, MA, 1998.

[95] V. N. Padmanabhan, K.Sripanidkulchai. "Distributed Streaming Media Content Using Cooperative Network". In: *Proceedings of 12^{th} International Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV'02*, Miami, FL, USA, maio 2002.

[96] V. N. Padmanabhan, K.Sripanidkulchai. "The Case for Cooperative Networking". In: *Proceedings of First International Workshop on Peer-to-Peer Systems*, Cambridg, MA, USA, março 2002.

[97] V.P. Kompella, J.C. Pascale, G.C Polyzos. "Multicast Routing for Multimedia Communication". *IEEE/ACM Transactions on Networking*, v. 1, n. 3, pp. 286–292, junho 1993.

[98] vTrails. "http://www.vtrails.com".

[99] W.J. Bolosky et all. *The Tiger Video Fileserver.* Technical report, Microsoft, Redmond, VA, 1996.

[100] Y. Cai, K.A. Hua. "An Efficient Bandwidth-Sharing Technique for True Video on Demand Systems". In: *Proceedings of ACM Multimedia'99*, outubro/novembro 1999.

[101] Y. Chae, K. Guo, M. M. Buddhikot, S. Suri, E. Zegura. "Silo, Rainbow, and Caching Token: Schemes for Scalable, Fault Tolerant Stream Caching". *IEEE Journal on Selected Areas in Communications, Special Issue on Internet Proxy Services*, 2002.

[102] Y. Guo, K. Suh, J. kurose, D. Towsley. "A Peer-to-Peer On-Demand Streaming Service and Its Performance Evaluation". In: *Proceedings of 2003 IEEE International Conference on MM & Expo, ICME-2003*, Baltimore, MD, julho 2003.

[103] Y. Guo, K. Suh, J. kurose, D. Towsley. "P2Cast: Peer-to-Peer Patching Scheme for VoD Service". In: *Proceedings of 12th World Wide Web Conference (WWW-03)*, Budapest, Hungary, maio 2003.

[104] Y. Rompogiannakis et all. "Disk Scheduling for Mixed-Media Workloads in a Multimedia Servers". *ACM Multimedia'98*, pp. 297–302, 1998.