SISTEMA DE MEMÓRIA CACHE DE VÍDEO SEGMENTADO PARA PROTOCOLO HTTP DINÂMICO

Nelson Rodrigo Pérez Benítez

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Claudio Luis de Amorim

Rio de Janeiro
Setembro de 2014

SISTEMA DE MEMÓRIA CACHE DE VÍDEO SEGMENTADO PARA PROTOCOLO HTTP DINÂMICO

Nelson Rodrigo Pérez Benítez

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Claudio Luis de Amorim, Ph.D.

Prof. Luis Felipe Magalhães de Moraes, Ph.D.

Prof. Alexandre Sztajnberg, D.Sc.

Prof. Edison Ishikawa, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2014

Benítez, Nelson Rodrigo Pérez

Sistema de Memória Cache de vídeo segmentado para protocolo HTTP Dinâmico/Nelson Rodrigo Pérez Benítez. – Rio de Janeiro: UFRJ/COPPE, 2014.

XIII, 47 p.: il.; 29, 7cm.

Orientador: Claudio Luis de Amorim

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2014.

Referências Bibliográficas: p. 44 – 47.

1. HTTP. 2. cache. 3. streaming. I. Amorim, Claudio Luis de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Aos meus pais, Nelson e Lidia

Agradecimentos

Gostaria de agradecer em primeiro lugar aos meus pais, Nelson Pérez e Lidia Benítez, por terem me dado todas as oportunidades e o incentivo constante ao estudo.

Ao meu orientador, o Professor Claudio Luis de Amorim, pelo suporte durante todo este tempo; as suas análises e contribuições foram essenciais para este trabalho.

Da mesma forma, não posso deixar de agradecer ao Lauro Whately, pelos conselhos e constantes observações e debates que com certeza contribuíram bastante para o formato final deste trabalho. Obrigado também ao Diego Dutra, que me ajudou em vários momentos.

Finalmente, gostaria de extender os agradecimentos a todos os meus professores do Programa de Engenharia de Sistemas e Computação.

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

SISTEMA DE MEMÓRIA CACHE DE VÍDEO SEGMENTADO PARA PROTOCOLO HTTP DINÂMICO

Nelson Rodrigo Pérez Benítez

Setembro/2014

Orientador: Claudio Luis de Amorim

Programa: Engenharia de Sistemas e Computação

A transmissão de conteúdo multimídia tem se tornado bastante popular na última década. Devido ao o grande consumo de recursos de rede que a distribuição deste tipo de dados requer, é de grande interesse o desenvolvimento de técnicas que resultem em uma economia destes recursos. Neste trabalho apresentamos a adaptação e testes de um algoritmo originalmente desenvolvido para a cache de *streams* contínuos de vídeo, agora para a transmissão via HTTP dinâmico.

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

CACHE MEMORY SYSTEM OPTIMIZED FOR ADAPTIVE STREAMING TECHNIQUES OVER HTTP PROTOCOL

Nelson Rodrigo Pérez Benítez

September/2014

Advisor: Claudio Luis de Amorim

Department: Systems Engineering and Computer Science

Multimidia content transmission has flourished during the last decade. But due to the large consumption of network resources that the distribution of such data requires, is of great interest to develop techniques that result in an economy of these resources. In this work, we present the adaptation and tests of an algorithm originally developed with the purpose of caching continuous streaming media. This algorithm has been translated from its original form to better work with modern industry adopted dinamic HTTP transmission models.

Sumário

Lista de Figuras

Acrônimos

CDN Content Delivery Network

CPU Central Processing Unit

GB Gigabyte

GOP Group of Pictures

HD High Definition

HLS HTTP Live Streaming

HTTP Hypertext Transfer Protocol

IP Internet Protocol

LRU Least Recently Used

MB Megabyte

MCC Memória Cooperativa Colapsada

MCD Memória Cooperativa Distribuída

MPEG Moving Picture Experts Group

P2P Peer-to-Peer

RAM Random Access Memory

RTCP Real-time Transoport Control Protocol

RTMP Real Time Messaging Protocol

RTP Real-time Transport Protocol

RTSP Real-time Streaming Protocol

UDP User Datagram Protocol

VoD Video on Demand

WAN Wide Area Network

Capítulo 1

Introdução

A rede global de computadores conhecida hoje em dia como a Internet sofreu grandes mudanças e experimentou bastante crescimento desde as suas simples origens como a interconexão de várias redes acadêmicas e militares[1] na década dos 70. O crescimento desde então se deu de forma acelerada e vários protocolos que operam nas diferentes camadas foram propostos, utilizados e abandonados pelo caminho.

Durante a década dos 90 ocorreu a introdução da Internet ao público em geral. Com a legalização do uso comercial do que antes tinha sido uma rede exclusivamente dedicada à pesquisa, surgiu prontamente a necessidade de oferecer um conteúdo mais sofisticado aos potenciais clientes. Existe finalmente um incentivo econômico para disponibilizar informação em formato multimídia: imagens, som e vídeo para um número cada vez maior de pessoas. Também na década dos 90, foram desenvolvidos os primeiros padrões de compressão de vídeo, mas o alto custo da infraestrutura (em especial na última milha) disponível restringe a distribuição massiva deste tipo de mídia pela internet.

Já no início dos anos 2000, com a popularização da banda larga, uma maior penetração dos computadores pessoais e o grande crescimento da rede como um todo possibilitaram a popularização da distribuição de vídeo.

Rapidamente foram desenvolvidas tecnologias[2] que permitiram melhorar aspectos da codificação, transmissão e exibição do conteúdo multimídia. Com mais largura de banda disponível a um preço acessível e um maior poder de processamento disponível, o vídeo que antes devia ser completamente descarregado da rede, em uma operação que podia levar horas, para logo assim ser exibido; agora podia ser decodificado e exibido a medida que a transferência ocorre. Surgiram assim os primeiros exemplos viáveis de *streaming* na rede.

Mas a popularização do conteúdo multimídia somente aumentou a demanda global por este tipo de conteúdo em uma espécie de ciclo de realimentação positiva, incentivado também pela disseminação do acesso à banda larga, a ubiquidade das câmeras e filmadoras digitais e o surgimento de mais possibilidades e modelos co-

merciais. Esse processo continua hoje em dia, a tal ponto que a maior parte do tráfego IP (Internet Protocol) na rede já é de vídeo[3], e essa tendência continua a crescer.

Estima-se que para 2018 até 79% de todo o tráfego de consumo na internet seja de vídeo sobre IP. E esse número não inclui vídeo enviado na forma de compartilhamento de arquivos via peer-to-peer (P2P). Se consideradas todas as formas de vídeo (TV, vídeo sob demanda e P2P) elas contabilizariam entre 80% e um 90% do tráfego global de pacotes para 2018 [3]. E com a multiplicação dos dispositivos móveis e de vestir, cada vez com mais funções multimídia, essa tendência só tende a crescer.

Todo este tráfego, no entanto, possui um alto grau de redundância. Muitos programas são assistidos em horário de pico por um grande número de pessoas. Alguns vídeos tornam-se virais em questão de horas[4]. E mesmo assim, na maioria dos casos as conexões entre cliente e servidor são feitas no modo *unicast*, ou seja um envio ponto a ponto, duplicando grande parte do conteúdo sendo enviado pelas conexões principais da rede, também denominadas *backbone*. Isso é um problema pois todo tráfego na rede tem um custo associado. Grandes provedores de acesso e sistemas autônomos, como são denominadas as redes privadas de cuja interconexão está formada a internet, cobram pela quantidade de bytes trafegados. Reduzir o volume de tráfego na rede, não somente trás uma melhor qualidade de serviço ao cliente, como também oferece economia para o gerador de conteúdo.

Grandes companhias geradoras de conteúdo portanto, precisam estar preparadas para atender uma demanda extremadamente flexível. A solução para atender este problema normalmente é o uso de algo conhecido como rede de distribuição de conteúdo. Uma rede de distribuição de conteúdo, ou CDN pela sua sigla em inglês (Content Delivery Network) pode ser definida como um sistema interconectado de servidores de cache que usa a proximidade geográfica como critério para a entrega do conteúdo web.

Aproveitando o baixo custo do armazenamento em relação à largura de banda, estas redes fazem uso de servidores específicos distribuídos ao redor do mundo, com a finalidade de manter cópias de rápido acesso para os clientes.

Este tipo de solução provou ser altamente eficiente no caso de conteúdo web estático. Mas o conteúdo multimídia possui características únicas que apresentam um desafio maior. Isso porque o tamanho de um arquivo de mídia audiovisual típico previne que ele seja tratado como um documento qualquer por um servidor de cache, já que um número relativamente pequeno deste tipo de arquivos seria capaz de esgotar rapidamente com os recursos de armazenamento de um servidor de cache web normal.

Sendo pouco prático o armazenamento do documento completo no servidor ca-

che, a opção mais prática então consiste em armazenar partes deste documento, e descartar outras.

Com a intenção de usar técnicas de cache de vídeo, e com isso evitar a criação de novos fluxos de vídeo do servidor de origem, foi desenvolvido no Laboratório de Computação Paralela um sistema com um algoritmo próprio para o envio de conteúdo multimídia. Pelo fato de usar a memória dos clientes como recurso do sistema, a versão inicial do sistema foi denominada Memória Cooperativa Distribuída. Uma versão posterior foi adaptada as redes WANs (Wide Area Network) e denominada Memória Cooperativa Colapsada.

Todas as implementações da Memória Cooperativa sempre trabalharam com *streaming* real de vídeo. No *streaming* como já foi explicado, o servidor oferece um fluxo contínuo de dados que é decodificado e exibido pouco depois de ser recebido pelo cliente. Nos últimos anos, no entanto, as técnicas de *streaming* foram combinadas com o protocolo HTTP, para gerar uma série de métodos híbridos de envio denominados genericamente "HTTP adaptativo".

A proposta deste trabalho então é de estudar a adaptação do algoritmo da Memória Cooperativa a este novo paradigma, assim como a implementação de um protótipo funcional e a avaliação do seu desempenho, em comparação a alguma solução convencional de cache na web.

1.1 Organização do texto

O restante deste trabalho está organizado da seguinte forma. No capítulo 2 apresentamos os conhecimentos básicos de web cache e armazenamento e transmissão de conteúdo multimídia. No capítulo 3 apresentamos com mais detalhes o funcionamento do sistema de cache original no qual este trabalho de adaptação foi baseado. Os detalhes do trabalho de adaptação são apresentados no capítulo 4. O capítulo 5 apresenta o ambiente experimental e descreve os resultados obtidos. Finalmente as conclusões são apresentadas no capítulo 6

Capítulo 2

Conhecimentos Básicos

Neste capítulo serão contextualizadas as diferentes técnicas de transmissão desenvolvidas e utilizadas para a transferência de conteúdo multimídia. Uma classificação geral dessas técnicas será apresentada e finalmente será explicado como a técnica descrita nesse documento difere das demais.

2.1 Memória Cache

O termo cache no contexto da computação refere-se à um repositório destinado à informação de uso frequente. Este repositório geralmente possui uma capacidade reduzida, mas em compensação um tempo de acesso menor se comparado à fonte original dos dados.

Em geral, sempre que existir uma diferença muito grande do tempo de operação entre as partes de um sistema de processamento de informação, a introdução de uma cache em um lugar estratégico consegue uma melhoria (as vezes dramática) da eficiência geral do sistema.

No caso da hierarquia de memória de uma CPU por exemplo, o processador geralmente opera em escalas de tempo da ordem de nanosegundos. O tempo de acesso aos registradores geralmente oscila entre 0.25 e 0.5 ns. Enquanto que um acesso à memória principal demora no mínimo 100 vezes mais[5] com tempos que oscilam entre 50 e 250 ns. A introdução de uma memória com menor capacidade (de uns poucos Megabytes) mas operando dentro do circuito integrado da CPU e portanto com um tempo de acesso reduzido, tem como resultado uma melhoria significativa em termos de desempenho.

Quando o processador encontra a informação que precisa na cache, economizando tempo, dizemos que houve um "acerto"ou *cache hit*. Se pelo contrário, a informação não estiver na cache, esse evento denomina-se *cache miss*. Quando isso acontece, um bloco de tamanho fixo em bytes deve ser trazido da memória principal e armazenado na cache [5].

Devido a estruturação dos programas e a maneira sequencial na qual as instruções são processadas pela CPU, sabemos que os blocos possuem uma alta *localidade temporal*, ou seja um bloco recentemente solicitado possui maiores chances de ser requerido novamente no futuro próximo. Devido à *localidade espacial* sabemos também que os endereços de memória próximos possuem uma grande probabilidade de ser solicitados em seguida.

É possível fazer um paralelo entre o sistema CPU-memória e o cliente-servidor. Em ambos os casos temos uma unidade de processamento operando em uma frequência relativamente alta, quando comparada ao tempo de acesso à unidade de armazenamento correspondente.

Assim, da mesma forma que a introdução de uma memória de menor capacidade, mas alta velocidade de acesso perto da CPU resulta na economia de tempo para a mesma, a introdução de um ou mais níveis de memória cache entre o *browser* e o servidor podem resultar em tempos de carregamento muito menores e consequentemente melhorar a experiência do usuário.

No escopo deste trabalho sempre que usarmos o termo "cache"estaremos nos referindo à memória cache na web.

Ainda dentro da web, é importante diferenciar dois tipos de memória cache. Temos a memória cache local, ou do *browser*, que é um espaço geralmente no disco onde os *browsers* guardam elementos não dinâmicos como imagens e páginas estáticas. Por outro lado existem servidores cache que funcionam como intermediários na rede, repassando o tráfego HTTP para os servidores de origem, mas guardando cópias locais dos documentos, e servindo diretamente aos clientes caso o documento já se encontre armazenado ali. Este servidor intermediário é conhecido vulgarmente como *proxy*. É importante ressaltar que o termo *proxy* refere-se a qualquer intermediário, não necessariamente implicando a presença de cache. Mas para o escopo deste trabalho no entanto, podemos assumir que sempre que nos referirmos a um *proxy*, ele estará fazendo cache de conteúdo.

2.2 Web cache

Servidores cache reduzem o consumo de largura de banda, diminuem a latência percebida pelo usuário e aliviam a carga de processamento nos servidores de origem de forma transparente para os usuários. Hoje em dia, a presença de servidores cache em vários pontos da rede entre o cliente e o servidor de origem é muito comum. Dependendo da sua localização ao longo do trajeto entre o cliente e o servidor no entanto, os servidores cache geralmente recebem nomes diferentes. Um servidor cache colocado mais próximo do cliente, na saída de uma rede institucional por exemplo é conhecido como um *forward proxy*. Da maneira contrária, se a cache

estiver colocada mais perto do servidor, ele é comumente conhecido como *reverse proxy* ou proxy reverso. A figura 2.1 ilustra 3 exemplos de localização de servidores cache, mostrando um proxy reverso utilizado para aliviar a carga no servidor de origem. Um proxy no provedor de acesso funciona de maneira transparente ao usuário. Finalmente a rede institucional dentro da qual o cliente encontra-se pode optar por colocar um servidor cache em modo de *forward proxy*.

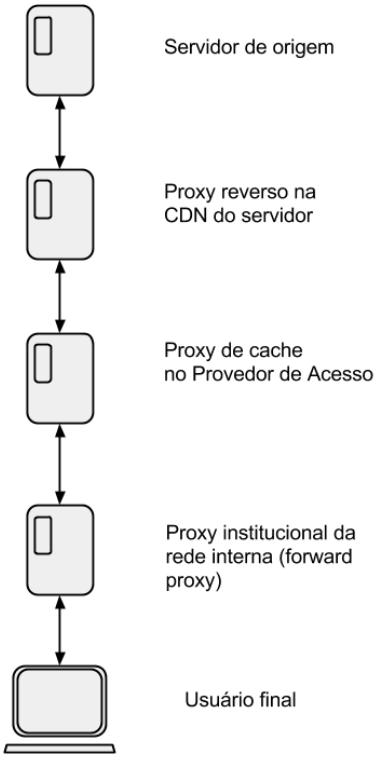

Figura 2.1: Possíveis localizações de um servidor cache no caminho entre o cliente e o servidor.

O cache de conteúdo web no entanto tem várias diferenças importantes com respeito ao cache de memória principal no nível do sistema. Os objetos não possuem um tamanho uniforme, os custos de obtenção da informação são maiores e nem todo conteúdo web é passível de ser guardado em cache, já que pode ter sido dinamicamente gerado. Além disso, ao cachear a informação corremos sempre o risco de estar exibindo conteúdo antigo, afinal mesmo um documento estático pode ser atualizado eventualmente pelo seu autor.

Além desses problemas, é também importante lembrar que a latência somente é diminuida para documentos que são encontrados na cache. Para os casos de *cache miss* o custo de obtenção do documento em termos de latência é ainda maior que se o servidor cache não estivesse sendo usado, devido ao tempo de processamento adicional que implica ter um servidor intermediário.

Por isso é de suma importância uma correta otimização no algoritmo de remoção

de cache. Idealmente é desejável guardar em cache os documentos mais populares e de maior tamanho.

2.3 Conteúdo multimídia

A distribuição de conteúdo multimídia em uma rede de comutação de pacotes apresenta vários desafios. Em primeiro lugar pelo tamanho que este tipo de dados requer. Para um simples arquivo de vídeo com uma duração de 10 minutos e uma resolução de 1280x720 e uma taxa de 23.98 quadros por segundo, teríamos um arquivo final de pouco mais de 18 GB caso ele não fosse comprimido. Isso corresponde à aproximadamente o espaço equivalente a 4 discos DVD-R.

Felizmente o sistema visual humano pode tolerar certo grau de perda de informação, e por isso é possível fazer uso de técnicas de compressão com perda. Coisa que não sería possível com outro tipo de dados, como informação textual por exemplo. Com isso é possível reduzir o tamanho do arquivo para algo mais fácilmente manipulável. A importância da compressão fica mais do que evidente depois de analizarmos estes números. O mesmo vídeo mencionado acima, se comprimido com o padrão H.264 pode chegar a ter um tamanho de somente 250 MB sem uma degradação aparente na qualidade.

Além disso, como a Internet não oferece garantia de entrega de dados, vários protocolos precisam ser combinados e/ou desenvolvidos para garantir um mínimo de qualidade de serviço.

No que diz respeito à integridade dos dados na recepção, em geral o conteúdo multimídia pode ser preparado para ser tolerante a perdas. Ou seja, o sistema pode ser projetado de tal forma que o decodificador seja capaz de continuar extraindo e exibindo as imagens mesmo com a perda de alguns pacotes[6].

Aplicações de áudio e/ou vídeo em tempo real (como a transmissão de um programa de radio ou televisão) e conteúdo sob-demanda podem tolerar um certo nível de atraso. Por outro lado, aplicações interativas como chamadas telefônicas, videoconferências e jogos exigem que o atraso seja sempre mínimo para não degradar a experiência do usuário. Um tempo de entrega superior aos 250 milissegundos é geralmente considerado fora do limite aceitável [7].

Um fator problemático no entanto é a variação do atraso, conhecido como *jitter*. Para compensar o efeito do jitter o cliente cria um buffer no qual armazena o video antes de decodificá-lo. A idéia então é que as variações de atraso possam ser escondidas graças ao buffer. Se o *jitter* for muito grande, ele pode ser atenuado até certo ponto pela ampliação dos buffers, a custo de um atraso adicional no início da reprodução.

2.4 Compressão de vídeo

Como ja foi visto anteriormente, a compressão do vídeo é essencial para que ele possa ser transmitido de forma eficiente pela rede. Vários algoritmos de compressão tem sido desenvolvidos, principalmente pelo MPEG (Moving Picture Experts Group) um grupo de trabalho formado por líderes da indústria com o objetivo de desenvolver técnicas de compressão multimidia.

Um vídeo é composto por uma sequência de imagens, que quando exibidas em rápida sucessão criam a impressão de movimento. Geralmente existe uma grande quantidade de informação redundante tanto dentro de uma imagem individual, como entre imagens sucessivas. Ao explorar a redundância de informação existente dentro de uma imagem individual, dizemos que estamos fazendo compressão "espacial"ou "*intra-frame*". Se a redundância de informação entre quadros sucessivos é utilizada para economizar bits enviados, dizemos que é feita uma compressão "temporal"ou "*inter-frame*".

Desta forma podem ser distinguidos geralmente 3 tipos de quadros. Os quadros com codificação intra, ou quadros I são aqueles que não dependem de outros quadros para poder ser corretamente decodificados. Os quadros com codificação preditiva no entanto, também conhecidos como quadros P, requerem a decodificação de um quadro anterior (Um quadro I ou outro quadro P). Existem também quadros especiais para os quais tanto a informação de quadros anteriores como a dos quadros seguintes são utilizadas, resultando em uma economia de bits ainda maior. Estes quadros, por usarem a informação nas duas "direções temporais", são denominados quadros bi-preditivos, ou B.

O vídeo geralmente é codificado no que costuma ser chamado de GOP pela sigla em inglês do termo *Group Of Pictures* (Grupo de quadros). Cada GOP contém geralmente um quadro I, seguido por uma alternância de quadros do tipo P e B. Um GOP geralmente tem entre 12 e 15 quadros.

Fica claro então que a perda de alguma informação durante a transmissão do vídeo terá um impacto dependente do tipo de quadro que foi perdido. Se o quadro perdido for um quadro do tipo I, o GOP inteiro pode ser descartado, já que não existe forma de decodificar os quadros seguintes.

2.5 Protocolos de transmissão

Muitas técnicas de transmissão multimídia envolvem o uso do UDP (User Datagram Protocol)[8] como protocolo de camada de transporte, já que este oferece um atraso mínimo na transmissão. A desvantagem é que o UDP por ser um serviço que não garante a entrega dos pacotes, exige que a aplicação implemente mecanismos de

confiabilidade na transmissão, não encontrados na camada transporte. Por outro lado, o TCP (Transmission Control Protocol)[9], tem a vantagem de oferecer garantias quanto à entrega ordenada dos pacotes e controle de fluxo, mas ao custo de um atraso maior em meios com altas taxas de erros de transmissão.

O padrão aberto de transmissão multimídia mais popular talvez seja o RTP (Real-time Transport Protocol)[10]. O RTP é um protocolo de nível de aplicação que oferece um serviço de entrega fim-a-fim ideal para conteúdo na forma de *stream*. Ele é bastante flexível e pode ser usado tanto com TCP como UDP no nível de transporte. Além disso também oferece suporte para transferência a múltiplos destinos usando *multicast*[11], caso a rede ofereça tal serviço.

O RTP é normalmente utilizado em conjunto com o protocolo de sinalização conhecido como RTCP (Real-time Transport Control Protocol). A função do RTCP é a de prover um informe periódico sobre o estado da rede aos participantes da transmissão. Este informe contém informação sobre o número de pacotes perdidos, o identificador do último pacote recebido, o *jitter* da rede, entre outras coisas. Apesar disso, o uso do protocolo de sinalização não é obrigatório e algumas aplicações decidem não utiliza-lo.

Outro protocolo utilizado em conjunto com o RTP é o RTSP (Real-time Streaming Protocol)[12]. O RTSP opera de forma paralela ao RTP e provê suporte para operações de videocassete tais como inicio, pausa, *fast-forward* e de gravação.

Apesar destes padrões abertos oferecerem uma excelente solução técnica, diferentes forças econômicas e/ou decisões de estratégia corporativa levaram a várias das principais companhias comerciais a desenvolver e adotar tecnologias proprietárias para a transmissão de conteúdo multimídia.

Uma dessas tecnologias por exemplo é o protocolo RTMP (Real Time Messaging Protocol), desenvolvido para o stream de audio, vídeo e dados através da internet. Inicialmente o protocolo era uma tecnologia proprietária, mas a especificação foi aberta em 2009[13] e diversas implementações independentes existem na atualidade.

2.6 Classificação

2.6.1 Streaming

Todos os protocolos mencionados até agora podem ser classificados como protocolos de streaming. A característica comum a todos eles é que o cliente que deseja reproduzir o conteúdo estabelece uma conexão com o servidor e este então procede a escrever um fluxo contínuo de dados. Em alguns casos, a taxa de escrita pode ser controlada por meio de algum canal paralelo como o RTSP, mas o fato é que em todos os casos existe um fluxo contínuo de dados indo do servidor para o cliente.

2.6.2 HTTP progressivo

Uma alternativa aos protocolos de *streaming* é o download progressivo, também conhecido como *pseudostreaming*. Neste esquema utiliza-se o próprio protocolo HTTP para a entrega do material, mas em vez de esperar a finalização da transmissão, o conteúdo é decodificado e exibido assim que possível.

Uma vantagem desta modalidade de entrega é a simplicidade. Além disso como o conteúdo está sendo entregue via protocolo HTTP padrão, é praticamente garantido que ele não irá ser bloqueado por um *firewall*. O YouTube por exemplo, inicialmente utilizava somente HTTP progressivo para transmitir mais de 100 milhões de vídeos diariamente [14].

No entanto download progressivo não oferece toda a flexibilidade e as funcionalidades do *streaming*. Não é trivial, por exemplo, fazer uma transmissão ao vivo usando somente pseudostreaming. E por não ter sido projetado para *streaming* multimídia, o uso do HTTP sobre TCP trata a transmissão como uma simples transferência de arquivo, ocupando toda a largura de banda disponível, em vez de enviar os dados a uma taxa próxima à de reprodução, como um protocolo de *streaming* faria. Isso pode parecer uma solução ótima para o cliente que está sendo atendido, mas pode gerar problemas no servidor e na rede, que pode ficar congestionada por um periodo de tempo, provocando atrasos para outros clientes.

Além disso, se existirem múltiplas cópias do mesmo conteúdo com diferentes taxas de codificação, a escolha da alternativa tem que ser feita antes do inicio da transmissão e não pode ser alterada depois. Se por alguma variação das condições da rede, a vazão de dados ficar abaixo da taxa de reprodução por um período prolongado de tempo, o usuário pode experimentar pausas inesperadas na reprodução devido ao esgotamento do *buffer* local.

Por isso, o HTTP progressivo não é considerado um streaming real. Ele é visto mais como uma adaptação rudimentar do HTTP a um uso para o qual ele não foi projetado.

2.6.3 HTTP Adaptativo

A última tendência na indústria tem sido o desenvolvimento de protocolos adaptativos baseados em HTTP. Estes mecanismos são híbridos entre o streaming e o download progressivo e portanto combinam as vantagens das duas tecnologias.

No HTTP adaptativo o cliente solicita primeiramente um arquivo de texto com a descrição do conteúdo a ser exibido. Este conteúdo encontra-se dividido em fragmentos de curta duração no servidor, podendo existir em mais de uma representação. Cada representação é uma versão da mesma mídia, mas com diferentes resoluções, taxas de codificação ou níveis de qualidade.

Desta forma o cliente, de posse desta descrição pode solicitar os fragmentos via pedidos HTTP normais. Cada fragmento é uma unidade independente e pode ser exibido assim que a transmissão for finalizada.

Normalmente o cliente solicita mais de um fragmento inicialmente de forma a construir um buffer local, e compensar os efeitos nocivos do jitter. Além disso, o cliente tem também a possibilidade de escolher sempre o melhor nível de qualidade dependendo de vários parâmetros da rede como latência, medições de banda disponíveis, assim como também da resolução disponível e do uso da CPU.

2.6.4 Modelos de Transmissão: Push vs. Pull

Uma diferença entre o HTTP progressivo e todas as outras formas de *streaming* e pseudostreaming antes descritas tem a ver com o modo de interação entre o cliente e o servidor. No caso do *streaming* apesar de ser o cliente quem inicia o contato, uma vez que a conexão foi estabelecida, ele assume um papel passivo de consumidor de dados. No caso da transmissão via HTTP adaptativo, no entanto, é o cliente quem constantemente solicita novos fragmentos de vídeo, especificando assim bem claramente a sua posição temporal dentro da mídia.

Dizemos então que enquanto todos os demais modos de transmissão de vídeo operam em um modo push, as técnicas de HTTP adaptativo operam no modo pull. Esta pequena diferença é bastante relevante na hora de escolher e avaliar quais estruturas de dados um servidor de cache deve utilizar.

Como já foi mencionado, no caso do streaming em modo pull é o cliente quem fica encarregado de gerenciar as diferentes taxas. Ao remover esta parte da complexidade do servidor, ele oferece uma solução mais simples, barata e escalável. Além disso, por operar sobre HTTP, um simples servidor web pode ser adaptado fácilmente para servir mídia em modo pull sem a necessidade de um servidor especializado que implemente RTSP ou algum outro protocolo específico para stream.

Apesar de reduzir a complexidade e os custos iniciais no lado do servidor, evitando a necessidade de instalar e configurar um servidor multimídia dedicado, o *streaming* em modo *pull* é geralmente menos eficiente no que diz respeito à transmissão. A razão é que deve existir um pedido indo do cliente para o servidor para cada fragmento, coisa que no modo *push* é desnecessário. Além disso, os mecanismos em modo *push* podem oferecer suporte a distribuição em *multicast*. O modo *multicast* permite ao servidor enviar um pacote uma única vez para um grupo de clientes. A rede se encarrega de duplicar tal pacote da maneira mais eficiente. As opções em modo pull precisam de um canal único para cada cliente individual.

Por outro lado, o suporte para o multicast na internet é baixo, e a redundância no envio de pacotes no modo pull pode ser fortemente atenuada usando uma rede

de distribuição de conteúdo e técnicas de cache.

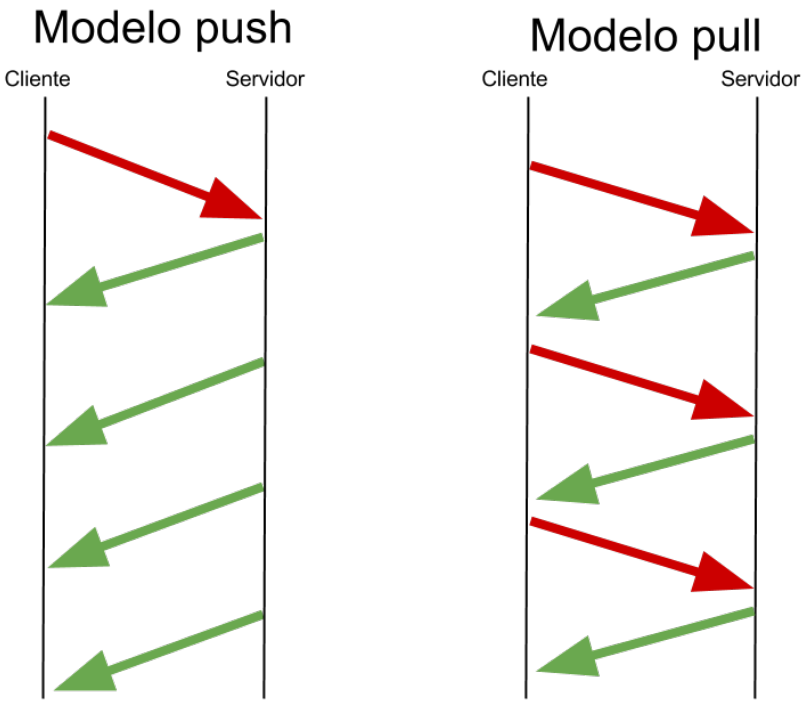

Figura 2.2: Ilustração da diferença de modos de operação entre a versão clásica baseada puramente em fluxos de vídeo e a transmissão via HTTP

Capítulo 3

Cache de Vídeo

Como visto em capítulos anteriores, a maior parte dos sistemas de distribuição de vídeo baseiam-se no modelo cliente-servidor. Um problema tipicamente relacionado a este modelo tem a ver com a escalabilidade em relação ao número de clientes. Devido ao alto consumo de largura de banda da rede de transmissão, este problema é ainda mais acentuado quando o conteúdo é vídeo.

Várias soluções têm sido propostas na literatura para resolver este problema. O uso de *multicast* é uma possibilidade. Como já foi visto, com o *multicast*, vários clientes podem compartilhar o mesmo fluxo de vídeo saindo de um mesmo servidor. Em [15] o autor apresenta uma técnica de *multicast* para VoD. Mas o problema para a aplicação desta técnica fora das redes privadas é, como já foi comentado, o limitado suporte oferecido pela internet para tráfego *multicast*.

Outra alternativa consiste no uso de servidores cache localizados em posições geográficas mais próximas aos clientes. No entanto, devido ao seu grande tamanho, não é prático armazenar a mídia contínua em um servidor cache como um documento único como seria feito com um arquivo de imagem ou texto. A solução adotada em vários trabalhos [16][17] consiste armazenar a mídia em fragmentos.

Em [18] é apresentada a proposta de cache de prefixo. A idéia aplicada nesta técnica é armazenar apenas a parte inicial da mídia.

Ao guardar apenas o prefixo de cada vídeo em um proxy perto do usuário temos uma solução escalável para garantir um baixo tempo de buferização do início do vídeo. E ao mesmo tempo que o cliente recebe e exibe o prefixo, o proxy pode estar entrando em contato com o servidor de origem a fim de obter um novo fluxo que irá dar continuidade ao que já foi iniciado.

Outra técnica apresentada em [19] conhecida como *dynamic caching* consiste em aproveitar a proximidade temporal de dois clientes assistindo uma mesma mídia. Se dois clientes iniciarem a reprodução em um intervalo de tempo suficientemente curto, é possível guardar parte do material repassado para o primeiro cliente em um buffer local dentro de um proxy com a intenção de alimentar o segundo cliente.

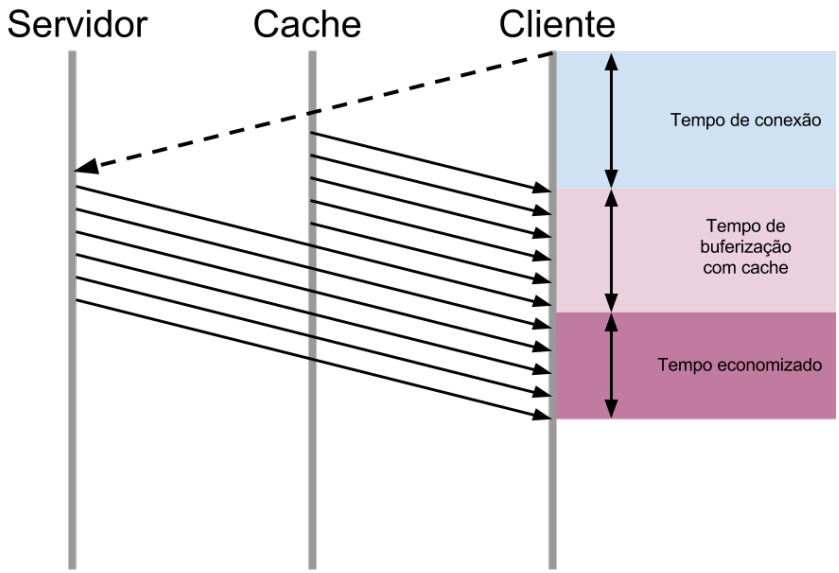

Figura 3.1: Diagrama mostrando um exemplo de *prefix caching*. Os tempos estão fora de escala, já que o tempo de estabelecimento da conexão seria bem menor.

Desta maneira aliviamos a carga do servidor que precisa somente prover um único fluxo para o proxy. Ao mesmo tempo esta técnica é eficiente no uso da memória, já que é preciso somente guardar uma quantidade de mídia proporcional à distância entre os clientes.

3.1 Memória Cooperativa Distribuída

Em vista da grande demanda de recursos gerada pelo *streaming* de vídeo e agregando várias das técnicas discutidas acima, a Memória Cooperativa Distribuída (MCD)[15] propõe explorar os recursos dos eventuais servidores *proxies* e dos próprios clientes dentro de uma rede local e/ou regional de alta velocidade.

O sistema proposto conta com um ou mais servidores centrais de conteúdo, um nó gerenciador que possui informação completa do estado dos clientes e os próprios clientes. A ideia é que utilizando técnicas chaining[20], patching[21] e multicast cada novo cliente possa ser redirecionado para um outro cliente (um par) que esteja com o conteúdo necessário disponível ainda em memória.

A agregação de todos os buffers de memória dos clientes então formam um grande repositório de memória compartilhado, mas distribuído, que pode ser explorado para garantir uma melhor qualidade de serviço para o cliente.

Ao se conectar à rede então um novo usuário entra em contato com o gerenciador, que mantém um registro do estado da memória de cada um dos participantes do

sistema. A função do gerenciador é encontrar a melhor fonte de conteúdo para o cliente, de preferência um servidor proxy ou um outro nó da rede.

Somente no caso de não haver um par em tais condições o novo cliente seria redirecionado para o servidor central, iniciando assim um fluxo único.

3.2 Memória Cooperativa Colapsada

A Memória Cooperativa Colapsada (MCC)[22] foi desenvolvida tentando adaptar as técnicas de reúso de fluxo de vídeo implementadas na MCD para redes geograficamente mais amplas. A idéia seria então colapsar a memória dos *buffers* dos usuários para dentro de um ou mais servidores *proxies* colocados estrategicamente nas bordas da rede. Quando mapeados dessa forma, os *buffers* dos clientes podem se sobrepor uns aos outros. Isso gera uma economia de fluxos, pois é necessário somente um fluxo para atender a dois ou mais clientes assistindo trechos suficientemente próximos do vídeo. Dentro da terminología do sistema este servidor proxy é denominado Distribuidor.

3.3 Estruturas de dados e funcionamento

A implementação original da MCC opera com streams de dados no modo push e adota várias técnicas eficientes para a economia dos recursos da rede. A principal estratégia implementada para aliviar a carga ao servidor de origem é relacionada com o *dynamic caching*, mencionado previamente.

3.3.1 Anel duplo

A memória dos clientes, agora colapsada no distribuidor, precisa ser mapeada em uma estrutura de dados versátil que seja capaz de organizar o envio e sincronizar todos os clientes. Desta maneira garantindo que todos recebam o vídeo na mesma taxa.

Com essa finalidade, uma estrutura de anel duplo de vetores foi desenvolvida e utilizada. A figura 3.2 ilustra esta estrutura. Um dos aneis guarda o conteúdo de vídeo enquanto o outro é simplesmente um array de ponteiros para localizações dentro do vídeo. O anel opera de maneira síncrona, atualizando o ponteiro de todos os clientes registrados de maneira sequencial. Esta operação recebe o nome de "giro", e faz com que efetivamente todos os clientes "avançem"na reprodução do vídeo periodicamente.

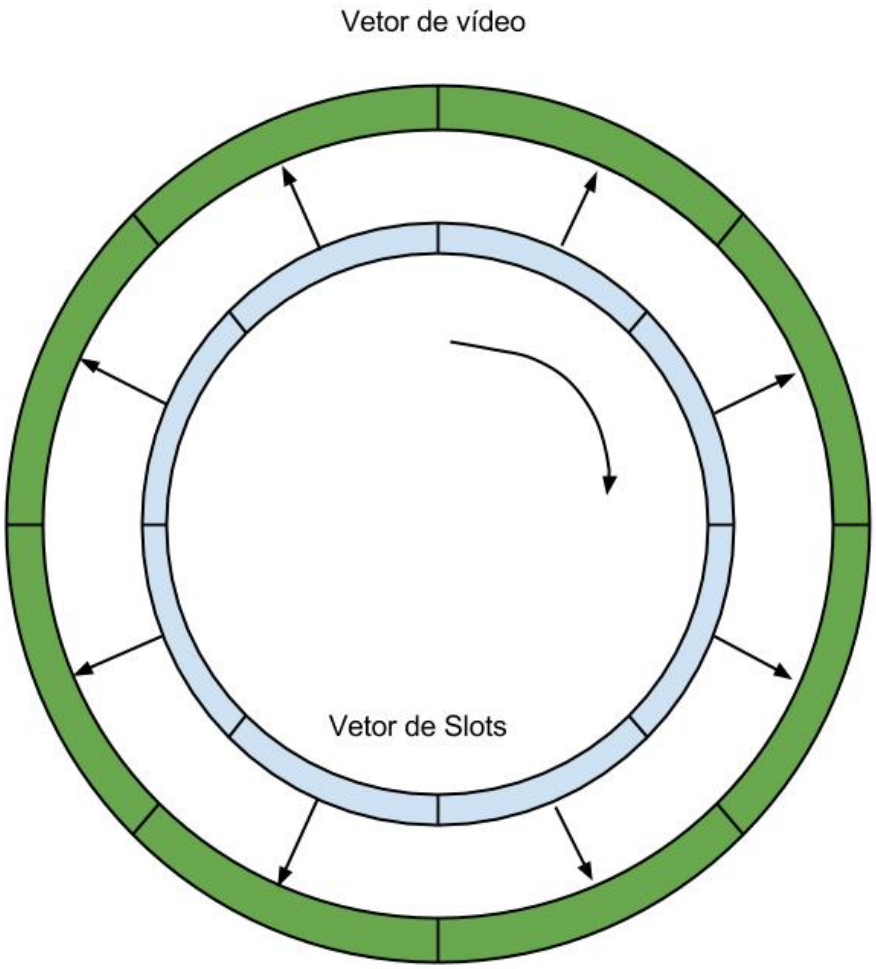

Figura 3.2: Anel duplo utilizado na primeira versão da MCC

3.3.2 Buffer do cliente

O buffer de cada cliente é mapeado dentro do anel de slots, a estrutura circular interna que contém os ponteiros para o anel de vídeo. Cada vez que um cliente novo for adicionado ao distribuidor, o anel interno de ponteiros é atualizado para refletir a presença deste cliente. De maneira similar, as posições do vetor de vídeo correspondentes são preenchidas com os blocos indicados, de forma a implementar um esquema similar à cache de prefixo, discutida anteriormente.

O buffer criado desta forma então recebe o nome de SlotBuffer.

Se dois ou mais clientes estiverem solicitando fragmentos de vídeo em trechos temporalmente muito próximos, o sistema irá automaticamente colapsar os dois SlotBuffers criando uma estrutura maior, desta maneira economizando largura banda entre o distribuidor e o servidor de origem.

O sistema reconhece uma escala de prioridades para diferentes tipos de Slots.

Slots que pertencem a um SlotBuffer são idealmente o mapeamento do buffer do cliente no distribuidor, o que quer dizer que existe uma alta probabilidade que eles sejam solicitados em curto prazo. Por isso o sistema irá atribuir uma prioridade 3; a maior para estes Slots. Para todo cliente aceito no sistema, será criado um SlotBuffer e um número fixo de Slots à frente do fragmento solicitado irá ter a sua prioridade alterada para a máxima. Slots de prioridade 3 não podem ser removidos se for preciso liberar mais memória. Caso isto ocorra o ingresso de novos clientes será bloqueado. A ideia por trás desta estratégia é garantir a qualidade de serviço dos clientes que já estão registrados. Além dos Slots dentro do SlotBuffers, temos outros dois tipos de prioridade para os Slots: prioridade média e baixa, ou 2 e 1 respectivamente. O caso dos Slots de prioridade média será explicado na próxima sub-seção. Os Slots de prioridade média podem ou não ter conteúdo. Nas primeiras versões do sistema quando a prioridade de um Slot era rebaixada para 1, o conteúdo era automáticamente eliminado. Por isso os trabalhos anteriores não tratam de nenhuma estratégia para a remoção de conteúdo deste tipo de Slots. Porém uma versão na qual o conteúdo é eliminado à medida que a demanda por memória ir crescendo também vai ser explorado.

3.3.3 Slots de ligação

Outra entidade de importância para o funcionamento do sistema são os Slots de Ligação, ou LinkSlots. O sistema irá criar uma cadeia de Slots de prioridade média, conhecidos como Link Slots sempre que dois clientes estiverem assistindo trechos o suficientemente próximos do vídeo, como justificativa para manter os blocos em memória, porém não suficientemente próximos para ser colapsados.

Em caso de existir a necessidade de liberar memória (para a inclusão de um novo cliente por exemplo) o sistema irá avaliar a posição e comprimento de cada uma das cadeias de Link Slots presentes, se existirem, para achar a melhor cadeia a ser removida.

A heurística desenvolvida para decidir qual cadeia de Link Slots deverá ser eliminada é formulada com mais detalhes em [15], mas basicamente leva em consideração o custo que a remoção de uma cadeia de Link Slots irá trazer para o sistema, supondo que o cliente que iria fazer uso dela continuasse no sistema.

Para calcular esse custo levamos em consideração duas propriedades da cadeia: o seu tamanho em bytes, e a distância do primeiro byte da cadeia de LinkSlots até o fim do vídeo. Uma terceira variável a ser considerada é o número de bytes que precisam ser removidos.

O cálculo do custo então fica resumido pela seguinte expressão:

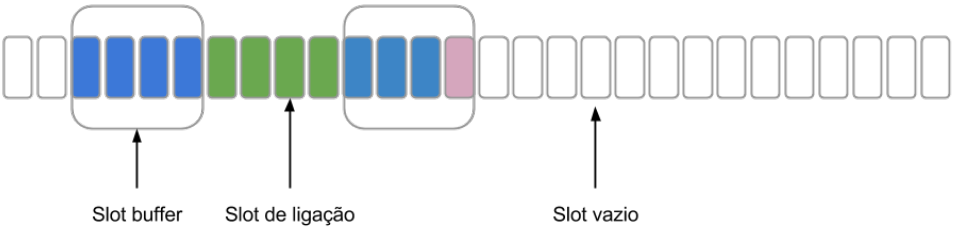

Figura 3.3: Diferentes tipos de Slots. A cor rosa dentro do Slot do Slotbuffer à direita indica um Slot que ja teve sua prioridade aumentada, mas cujo conteúdo ainda está sendo trazido do servidor de origem

$$D(d, f, t) = \frac{d}{min(t, f)}$$

O numerador d é a distância do primeiro byte da sequência de LinkSlots até o final do vetor de vídeo. t é o tamanho em bytes da sequência de LinkSlots e f é o número de bytes a se liberar.

Diferentemente dos trabalhos anteriores, que mediam estas distâncias em números inteiros de Slots, para este trabalho tivemos que fazer as medições em bytes, já que os fragmentos de vídeo não possuem todos um mesmo tamanho.

Capítulo 4

Adaptação da MCC para a modalidade pull

4.1 Visão

Neste capítulo apresentamos o Dynamo, a implementação da MCC para a distribuição de vídeo em modelo *client pull*. Implementações anteriores da memória cooperativa colapsada para distribuição multimídia previam o uso de vários componentes interagindo entre si; um gerenciador, mapeando o conteúdo e redirecionando os clientes para o servidor de origem ou para um ou mais distribuidores. O Distribuidor, opera como um servidor cache e é o componente que o Dynamo está implementando.

Dentre os componentes do sistema, o Distribuidor cumpre um papel central para o foco deste trabalho, já que é onde o conteúdo é armazenado e onde as regras de substituição são aplicadas.

Uma questão importante no entanto é a de investigar a eficiência do algoritmo proposto para a MCC operando basicamente no distribuidor, mas agora para o modelo *pull*. Esta avaliação foi feita ao comparar várias métricas do seu desempenho com as de uma outra alternativa de cache normalmente utilizada para conteúdo web.

Para este trabalho quisemos não somente adaptar o distribuidor e o seu algoritmo aos métodos de distribuição de vídeo no modo pull, mas também torná-lo um elemento modular dentro de uma potencial rede de distribuição de conteúdo mais ampla.

Nesta visão, o distribuidor seria um elemento focado específicamente no armazenamento em memória de um vídeo específico. Várias instâncias do mesmo processo poderiam ser criadas, talvez dentro de uma máquina virtual ou container[23, 24] em máquina física ou na nuvem a fim de cachear conteúdo de vários vídeos. Dentro deste ambiente, cada instância teria limitações ajustáveis de uso de memória, CPU e largura de banda.

4.2 Alterações

Um resultado imediatamente evidente da mudança no modo de streaming é a redução da complexidade da implementação do distribuidor, já que como foi discutido em 2.6.4, neste modelo é o cliente quem toma as decisões de alteração de taxa de codificação e proativamente solicita o conteúdo desejado. Isso efetivamente remove algumas responsabilidades, como o controle da taxa de transmissão, do lado do servidor e os transfere para o cliente.

Como consequência direta dessa troca de papeis, foi preciso introduzir algumas alterações nas estruturas de dados originais e/ou suas interações. Como era esperado no entanto, algumas outras estruturas foram mantidas quase que intactas. Em geral, pode-se dizer que a idéia principal do algoritmo, que tem a ver com o gerenciamento da memória cache e a sua classificação em fragmentos de alta, média e baixa prioridade, se manteve basicamente idêntica.

A estrutura de anel duplo por exemplo teve a sua operação, antes síncrona, alterada para melhor refletir a independência dos clientes entre si. Como cada cliente agora atualiza a sua posição na mídia usando um pedido HTTP convencional, ele tem a sua posição dentro do anel atualizada de maneira independente dos outros. Uma possível adaptação da estrutura de anéis seria por exemplo fazer com que cada cliente tivesse o seu próprio anel, e a cada pedido HTTP, o anel específico daquele cliente seja atualizado. A estrutura então ficaria como a mostrada na 4.1

No entanto, após uma série de análises, ficou claro que tentar manter o modo de operação do "giro"do anel para o modo *pull* na realidade seria não somente ineficiente, mas na realidade desnecessário. Isso porque o modo de operação da estrutura de anéis duplos utilizada em outras implementações fundamentalmente existe para resolver um problema que na transmissão via HTTP dinâmico encontra-se resolvido, e tem a ver com a posição e a sincronia entre os vários clientes.

Em lugar disso, foi decidido manter um vetor de Slots que irá ser mapeado diretamente ao vetor de vídeo, tal como na implementação original, mas sem o conceito de "giro". O vetor de vídeo na realidade é uma abstração implementada como um pool de memória e os SlotBuffers são estruturas que possuem ponteiros indicando a posição atual do cliente dentro do vetor de Slots e apontando para o vetor de vídeo. A figura 4.2 ilustra este esquema de funcionamento.

É importante observar que o cliente pode também a qualquer momento fazer uma pausa ou emitir eventos de interatividade, como "pular"para um ponto adiante ou atrás no vídeo. Tais eventos seriam vistos desde o lado do distribuidor como uma alteração na sequência de pedidos HTTP enviados pelo cliente. No entanto isso não representa um problema. Se o cliente mantiver uma conexão persistente o distribuidor pode mapear diretamente esse deslocamento. Caso uma conexão

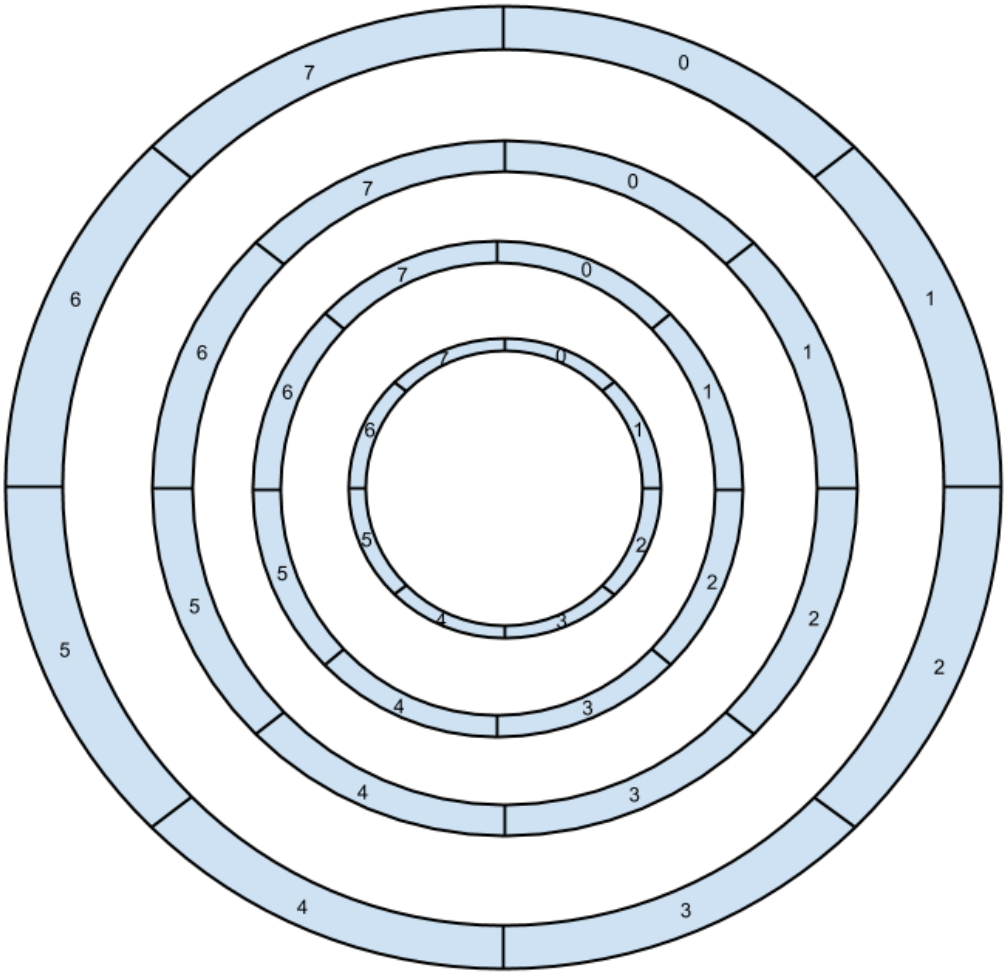

Figura 4.1: Possível adaptação do esquema de aneis para a MCC via HTTP dinâmico

persistente não seja utilizada, desde a perspectiva do distribuidor o evento será interpretado como a saída de um cliente seguido da entrada de outro em algum ponto aleatório da mídia.

A implementação original da MCC dividia o stream em um número de unidades fixas de tempo denominadas "slots", cada um com uma duração de uns poucos segundos. Da mesma forma as implementações de HTTP adaptativo segmentam o vídeo em unidades fixas de tempo. O paralelo entre estas duas implementações então foi muito claro, e os "slots"de tempo nesta nova implementação correspondem a um fragmento de vídeo solicitado via protocolo HTTP. No contexto deste trabalho, as palavras "flot", "fragmento"e "segmento"são usadas da mesma forma para nos referir a esta divisão da midia.

Uma das principais funcionalidades a ser mantidas é o mapeamento do *buffer* do cliente dentro do distribuidor. A diferença de algumas implementações anteriores que possuíam um cliente próprio,[25] o sistema descrito neste trabalho foi projetado de maneira a operar com qualquer player web. O mais comúm no entanto, é que o cliente use um player Flash e/ou soluções em HTML5. Dessa forma, é impossível

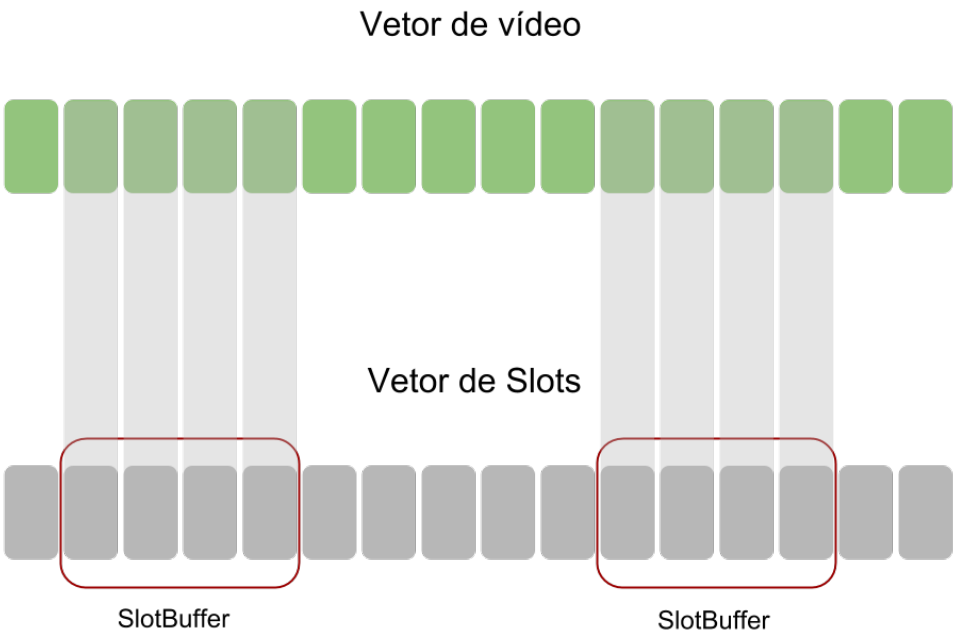

Figura 4.2: Possível adaptação do esquema de aneis para a MCC via HTTP dinâmico

para o distribuidor saber exatamente qual é o conteúdo do *buffer* do cliente.

Uma solução de compromisso então foi adotada. Decidiu-se replicar uma região representativa do buffer do cliente dentro do distribuidor. Geralmente espera-se que o cliente guarde em buffer uns segundos de vídeo. Na terminología original introduzida pelos trabalhos anteriores, nos referimos a um Slot como uma divisão temporal arbitrária geralmente equivalente a um ou dois segundos. Para este trabalho decidiu mapear diretamente cada fragmento de vídeo HTTP a um Slot.

Nos trabalhos anteriores, o comprimento deste *buffer* em número de Slots tinha sido fixado em um número ótimo de 4. Porém, devido as características do vídeo de testes que usamos, especialmente à maior taxa de bits por segundo devido ao uso de um vídeo em HD (720p), para este trabalho foi encontrado um tamanho ótimo de 2. Mais adiante explicamos como chegamos a este resultado e os outros valores que foram testados.

Capítulo 5

Análise Experimental

Neste capítulo descrevemos o ambiente experimental, assim como as características do vídeo a ser usado nos experimentos. Na continuação descrevemos com mais detalhes cada um dos experimentos realizados. Finalmente discutimos com mais cuidado alguns resultados inicialmente não esperados e concluímos com uma discussão geral dos resultados.

Existem atualmente no mercado várias especificações para a transmissão de vídeo de modo adaptativo via HTTP, dependendo do tipo de especificação utilizada para a transmissão do vídeo via HTTP, o arquivo resultante pode ser fisicamente dividido ou não. Para este trabalho foi decidido usar a implementação da Adobe, mais conhecida como "HTTP Dynamic Streaming". No caso desta especificação em particular o vídeo é guardado em um arquivo com extensão F4F no servidor de origem. Um arquivo F4F nada mais é do que um F4V internamente fragmentado[26]. Também é necessária a instalação de um *plugin* no servidor de maneira a que este possa corretamente encontrar e servir os fragmentos de mídia solicitados.

Um resultado importante deste processo é que os fragmentos resultantes não necessariamente possuem o mesmo tamanho e/ou duração. Este fato tem um grande impacto sobre o desempenho final do sistema.

Para se ter uma idéia mais clara da variação de tamanho dos blocos, a figura 5.1 apresenta no gráfico de cima o tamanho de cada bloco, sendo os blocos ordenados de acordo com a sua posição temporal no vídeo. O gráfico de baixo é um histograma de frequências mostrando dois picos; um em torno a 1MB e o outro de 2.6 MB.

As implementações anteriores do algoritmo de memória cooperativa trabalhavam com um fluxo contínuo de vídeo e a divisão deste fluxo em blocos era simplesmente um procedimento operacional feito pelo proxy e cujo efeito era transparente, tanto para o cliente final como para o servidor de origem. Desta maneira, nestas implementações o vídeo podía ser temporariamente segmentado em blocos do mesmo tamanho enquanto estivesse no proxy.

Este procedimento facilitava todos os mecanismos de gerência de memória, já que

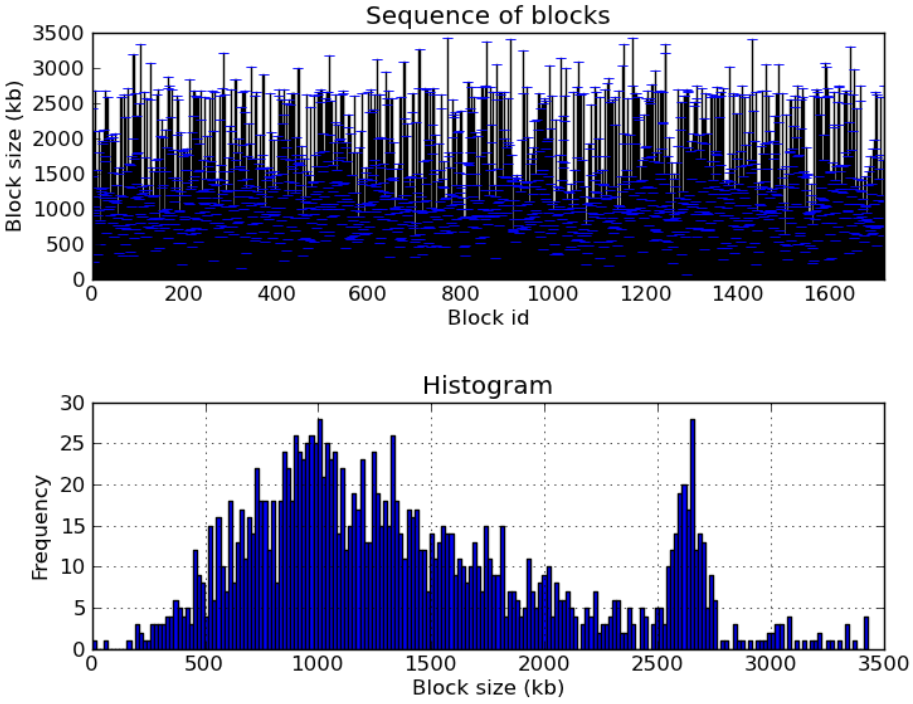

Figura 5.1: Tamanho dos fragmentos ordenados de acordo com a sua posição temporal no vídeo e histograma de tamanhos.

as estruturas de dados criadas e até mesmo o próprio algoritmo em sua formulação inicial trabalhavam com fragmentos de tamanho fixo.

No caso da transmissão de vídeo dinâmico via HTTP, a segmentação do vídeo é um procedimento inerente da técnica e independente do proxy. A princípio este fato pareceu estar completamente alinhado com a aplicação da técnica; mas assim que o problema foi analisado com mais detalhe começaram a surgir alguns problemas relativos à implementação. A primeira tem a ver com o pool de memória.

Um *pool* de memória é uma técnica conhecida como uma alternativa ao uso dos operadores *malloc* e *new* para a obtenção de memória dinamicamente. É desejável evitar o uso excessivo destas funções, já que elas não oferecem garantia sobre o tempo de execução. Além disso, como estas implementações oferecem a possibilidade de alocar blocos de memória de tamanho arbitrário, elas são afetadas pela fragmentação externa da memória, o que pode reduzir o desempenho do sistema originando eventos de falha de alocação.

O problema então surge no momento de criar um *pool* de memória para um conjunto de blocos que apresenta semelhante variação de tamanhos. Ficou claro que se o tamanho dos blocos fossem fixados de acordo com o máximo tamanho, teríamos muita fragmentação interna, o que se traduz como desperdício de memória.

A alternativa foi então escolher um tamanho de 1 MB para os blocos do *pool*. Todo fragmento que precisasse de até 1MB seria rápidamente alocado a uma posi-

ção dentro do *pool*. Para qualquer fragmento que precisasse de mais do que 1MB, utilizamos o operador *new* e alocamos dinamicamente o número exato de bytes que aquele fragmento precisasse.

5.1 Métricas

Vazão: Em uma rede de transmissão de dados, a vazão ou *throughput* é a medida da quantidade de informação transferida com sucesso por unidade de tempo. Ela é geralmente expressa em bits por segundo.

Taxa de acertos ou *Hit rate*: Definimos a taxa de acertos (Ta) como sendo a razão entre o número de pedidos atendidos pela cache e o numero total de pedidos recebidos.

$$Ta = \frac{Pa}{Pt}$$

Onde: Pa representa o número de pedidos atendidos pela cache e Pt o número total de pedidos recebidos.

Impacto no servidor: Para avaliar o impacto gerado no servidor de origem, foi desenvolvido um *script* simples que monitora a quantidade de dados, em bytes, transferidos pela interface de rede ao longo do tempo. Mais especificamente, esta rotina cria um registro com a quantidade total de bytes transferidos desde o inicio do experimento, que tem uma nova entrada sendo adicionada a a cada 10 segundos.

Tanto no caso da taxa de acertos, como nas medições do impacto no servidor, os experimentos foram repetidos pelo menos 5 vezes. Em alguns casos indicados foram feitas até 9 repetições. Em todos os casos, os diversos gráficos resultantes foram combinados em um gráfico médio, mostrando também os limites do desvio padrão usando a distribuição t de Student [27].

5.2 Metodología

Os experimentos foram realizados utilizando 3 máquinas idênticas, cada uma com a seguinte configuração:

Processador: Intel Xeon Quad-Core CPU X3430 @ 2.40GHz Memória: 16 GB RAM

As 3 máquinas estão interligadas em uma topologia estrela a um switch 3Com Gigabit 8 portas modelo 3CGSU08.

O throughput máximo da rede foi medido com a ajuda da ferramenta Iperf[28] e apresentou um valor de aproximádamente 950 Mbps.

O vídeo utilizado para os testes é um trecho de um longa metragem com uma

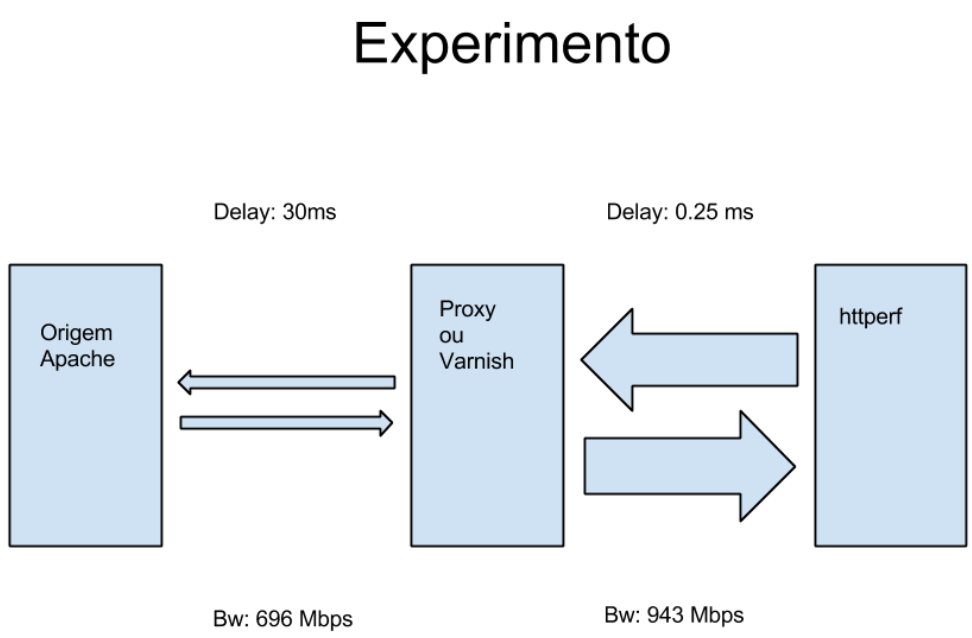

Figura 5.2: Esquema ilustrativo do ambiente experimental mostrando as diferenças de largura de banda disponíveis nos experimentos

duração de aproximadamente 6.6 min e com uma taxa de 1980 kbps. Ele foi fragmentado utilizando a ferramenta f4fpackager,[29] possuindo cada fragmento aproximadamente 2 segundos de duração.

A seguir foram feitos dois testes iniciais. O primeiro com o objetivo de avaliar a eficiência da implementação, focou-se em medir o throughput total do sistema em comparação a outros softwares que poderiam ser utilizados de maneira equivalente a um proxy normal. No segundo e posteriores testes, medimos tanto a taxa de acertos na memória como o volume de dados solicitados do servidor de origem.

5.3 Teste de vazão

O objetivo primordial dos testes preliminares de medição de vazão é de verificar a ausência de falhas graves na implementação do algoritmo que possam causar uma perda considerável de eficiência observada por meio de uma queda na vazão de dados. Além disso, é de se esperar que por guardar os dados em memória e não fazer uso intensivo do acesso à disco, a implementação do Dynamo obtenha uma certa vantagem.

Analizando os dados nos certificamos da confiabilidade da implementação. A vazão obtida com o Dynamo é compatível com as expectativas e muito similar aos

resultados apresentados em [30].

Os sistemas de cache web utilizados para a comparação são os seguintes:

- Apache HTTP Server: Popular servidor HTTP utilizado por aproximadamente 65% dos sites na internet [31].
- Apache Traffic Server: Servidor de cache muito popular por ser rápido, escalável e fácilmente extensível por meio de plugins.
- Varnish: Servidor de cache cujo principal diferencial consiste no fato de utilizar ao máximo possível a memória do sistema. Como política de remoção, o Varnish utiliza um LRU (Least Recently Used) modificado com a introdução de uma duração mínima na memória.
- Dynamo: Implementação da Memória Cooperativa Colapsada para protocolo HTTP dinâmico. Também utiliza a memória do sistema e tem um algoritmo próprio de gestão de memória baseado na categorização de diferentes segmentos.

Na figura 5.3 é possível perceber como o desempenho em termos de vazão tanto para Dynamo como para o Varnish parece aumentar linearmente em função do aumento da taxa de pedidos por segundo.

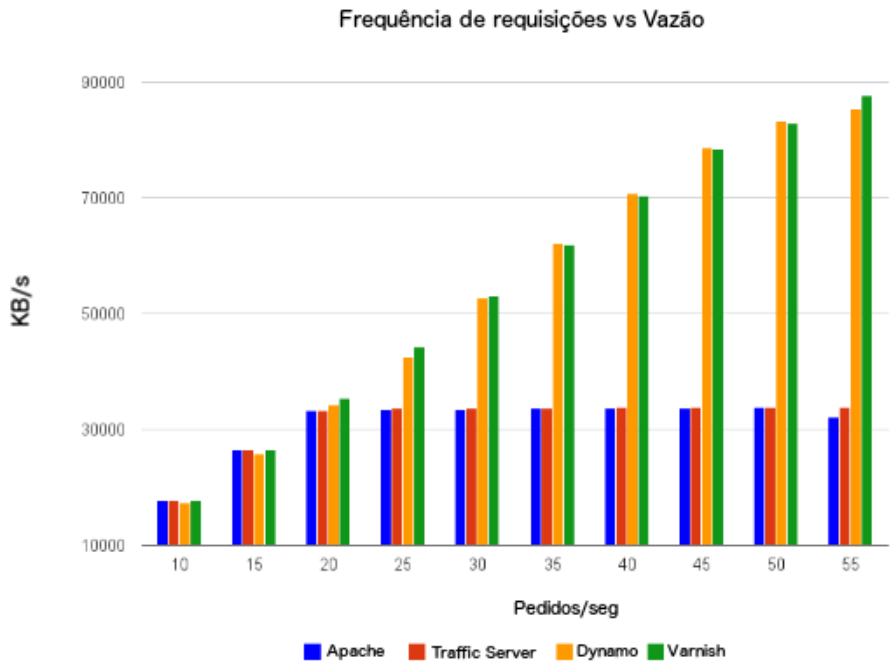

Figura 5.3: Comparação de tempo de transferência entre diferentes programas que podem vir a ser utilizados como proxies HTTP.

Tanto o Apache HTTP Server, como o Traffic Server por outro lado, parecem ter problemas em superar um nível de vazão de dados acima dos 30 mil kilobytes

por segundo. Como explicado em [30], isso deve-se ao fato de termos um gargalo no acesso ao disco para este tipo de carga de testes. Dynamo, assim como Varnish, não fazem acessos ao disco, portanto eles estão livres da latência extra e podem oferecer uma vazão superior.

5.4 Análise da Taxa de acerto

Dando continuação à avaliação de Dynamo, escolhemos Varnish[32] como plataforma de referência, já que apresentou o melhor desempenho no teste de vazão e ele é o único que implementa uma cache em memória.

5.4.1 Perfil geral dos experimentos

Todos os testes tem uma duração de 10.000 segundos (ou 2,778 horas).

Análises empíricas do comportamento dos usuários de um sistema VoD de grande porte mostram que uma modelagem desse comportamento pode ser uma tarefa bastante complexa [33].

Com o objetivo de simplificar a modelagem do comportamento dos clientes, optou-se por representar os 3 principais tipos de comportamento. Em primeiro lugar consideramos o conjunto de usuários que assistem ao vídeo de forma sequencial e até o final. Do ponto de vista do algoritmo da MCC este grupo representa o melhor comportamento de todos, já que a sua previsibilidade maximiza a quantidade de acertos na memória. O segundo tipo de clientes são aqueles que da mesma forma que os da primeira categoria assistem ao vídeo de forma completamente sequencial, mas não chegam a finalizar a exibição. Finalmente, o terceiro conjunto de usuários fazem pedidos sequenciais, mas eventualmente geram operações de interatividade "pulando"para algum trecho distante do vídeo, gerando assim um evento de *cache miss*.

Temos então uma combinação de clientes com e sem interatividade. Consideramos um cliente sem interatividade como aquele que assiste o vídeo de inicio ao fim sem interrupções. Já os clientes com interatividade podem fazer pausas, avanços e recuos na reprodução do vídeo.

A proporção de cada tipo de clientes poderia ter sido igual para cada grupo, mas com o objetivo de estressar o sistema ao máximo, foi decidido elevar o percentual de clientes com interatividade para o 50%.

A distribuição dos clientes nestes grupos então é a seguinte:

- 25 % - Assistem o vídeo sequencialmente até o final
- 25 % - Assistem o vídeo sequencialmente até a metade

- 50 % - Assistem o vídeo com interatividade.

Para simplificar o modelo da interatividade, foi definido que os clientes que seguem este padrão irão começar a reprodução em um lugar pre-definido aleatóriamente e irão ter até 10 eventos de busca, ou seja um salto para um fragmento na frente ou atras da posição atual.

Os clientes não podem entrar todos ao mesmo tempo, já que isso não iria provocar um estresse para o algoritmo de gerência da memória. Em lugar disso, queremos que os clientes entrem aos poucos, de forma que no regime permanente tenhamos uma distribuição uniforme de clientes ao longo da duração da mídia.

Também não seria realista fazer com que os clientes venham entrando a intervalos fixos de tempo. Então em lugar disso, o intervalo de entrada entre clientes é definido como uma variável aleatória com uma distribuição exponencial. Foram testados vários valores de média para esta distribuição, mas nos experimentos que vamos a mostrar a seguir estaremos usando um valor médio de 20 segundos, a menos que seja explicitado diferente.

O experimento pode ser dividido em 3 fases bem diferenciadas. A primeira fase corresponde à entrada dos clientes. Chamamos esta fase de transiente inicial. Enquanto os clientes vão entrando e nenhum deles ainda termina a exibição da mídia, temos um valor crescente para o número de clientes.

A segunda fase começa assim que o primeiro cliente deixa o sistema. Com isso, temos um número aproximadamente constante de clientes assistindo ao vídeo. Chamamos este período do regime permanente.

Finalmente quando os clientes deixam de entrar no sistema, o número de clientes ativos começa a decrescer. Denominamos esta fase como o transiente final. O experimento é concluído assim que o último cliente termina a sua lista de solicitações e/ou registra um evento de erro.

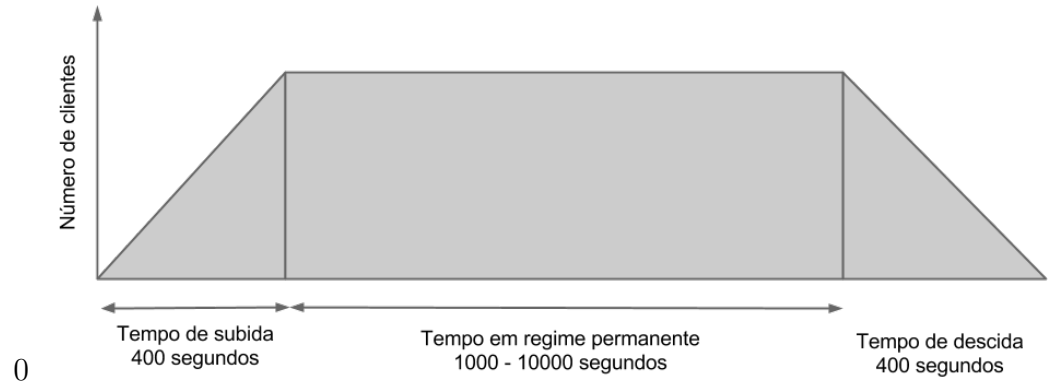

Figura 5.4: Perfil do experimento

Para todos os gráficos, foi indicado o tempo correspondente ao final do transiente inicial com uma linha vertical vermelha.

Em todas as figuras onde exista, uma área sombreada da mesma cor que a linha principal indica a região contida entre um desvio padrão da média das diferentes medições, que no caso é indicada pela linha escura. A menos que seja indicado o contrário, o valor N de número de experimentos é 5.

5.4.2 Comparação de Dynamo com Varnish

No primeiro experimento, foi comparada a taxa de acertos em função do tamanho da cache para Varnish e Dynamo. Os valores escolhidos para o tamanho da cache em memória estão em proporção ao tamanho da mídia, e correspondem a 50%, 20% e 5% do vídeo.

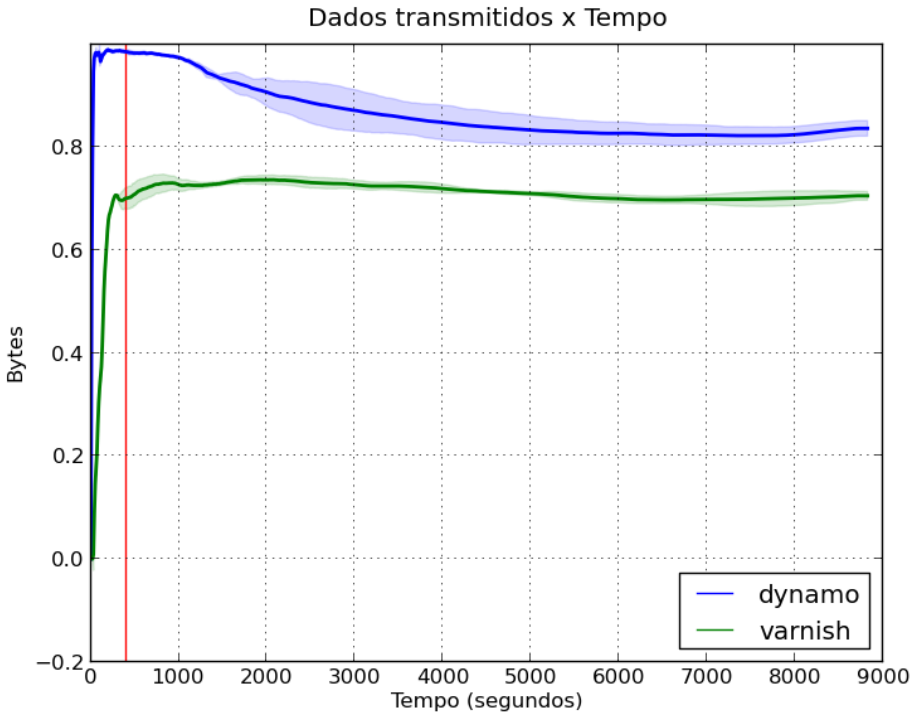

Figura 5.5: Evolução da taxa de acertos ao longo do experimento com 50% da memória

Nas figuras 5.5, 5.6 e 5.7 a curva azul corresponde à taxa de acertos acumulada de Dynamo e a curva verde é o mesmo valor para Varnish. As comparações com Varnish mostram um desempenho superior para Dynamo em todas as condições. Apesar da taxa de acertos de Dynamo ter uma maior variabilidade à medida que o tamanho da cache decresce, ela sempre se mantém acima do valor obtido por Varnish para o mesmo tamanho de cache.

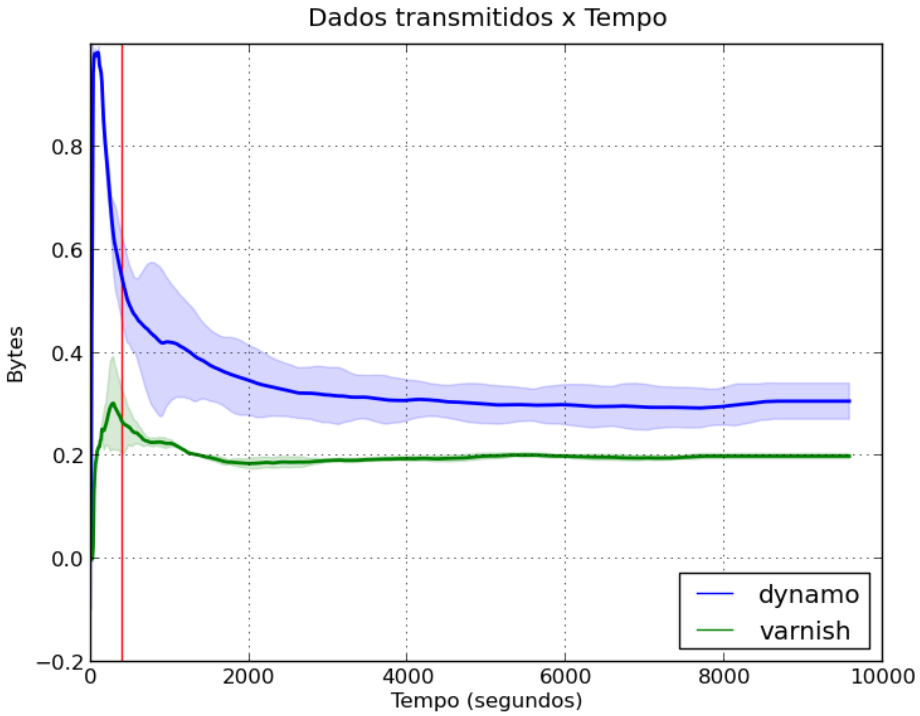

Figura 5.6: Evolução da taxa de acertos ao longo do experimento com 10% da memória

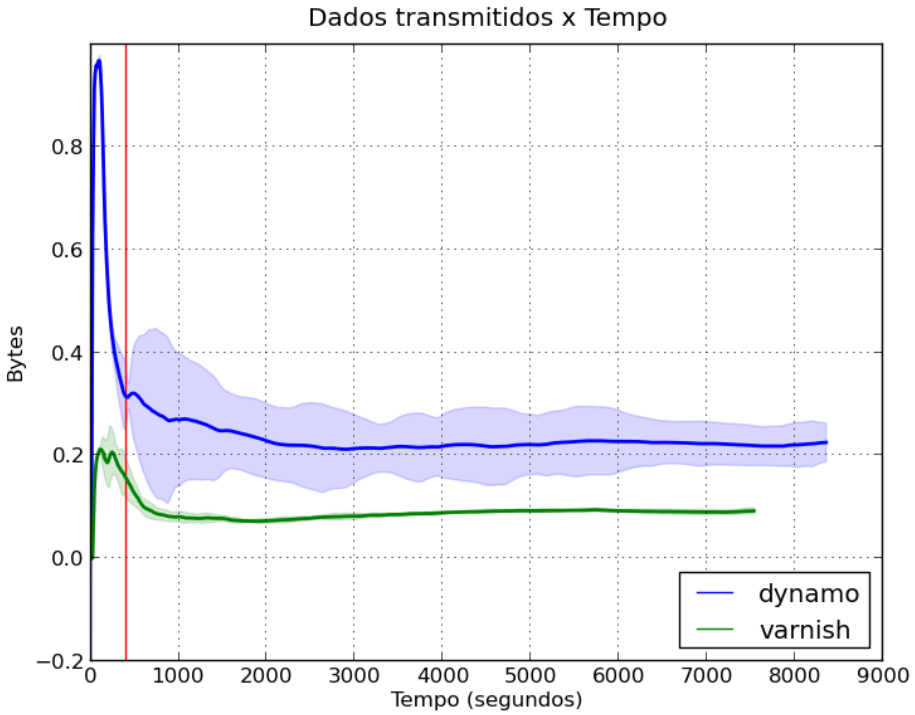

Figura 5.7: Evolução da taxa de acertos ao longo do experimento com 5% da memória

Atribuimos essa variabilidade aos clientes do subgrupo de interatividade, que podem saltar para posições com vídeo já cacheado ou não. Devido à pseudo-randomicidade existente no padrão de interatividade, uma vez que um cliente de um grupo faz um pulo para uma região do vídeo presente na memória, todos os demais clientes do mesmo grupo irão cair na mesma região, o que incrementa o valor da taxa de acertos. Para valores menores de cache, a demanda por memória é maior e por essa razão, algumas vezes os clientes iterativos não encontram o segmento de vídeo em cache após um salto.

Como o httperf gera um padrão diferente de tempos de entrada para os clientes em cada rodada, o valor plotado do hit-rate apresenta incrementos e decrementos distintos em cada execução.

5.4.3 Comparação da taxa de hits do Dynamo para diferentes tamanhos de cache

Para o segundo experimento voltamos a alterar o tamanho da cache em relação ao tamanho da mídia, mas dessa vez exploramos mais alguns valores intermediários e plotamos os resultados em um só gráfico.

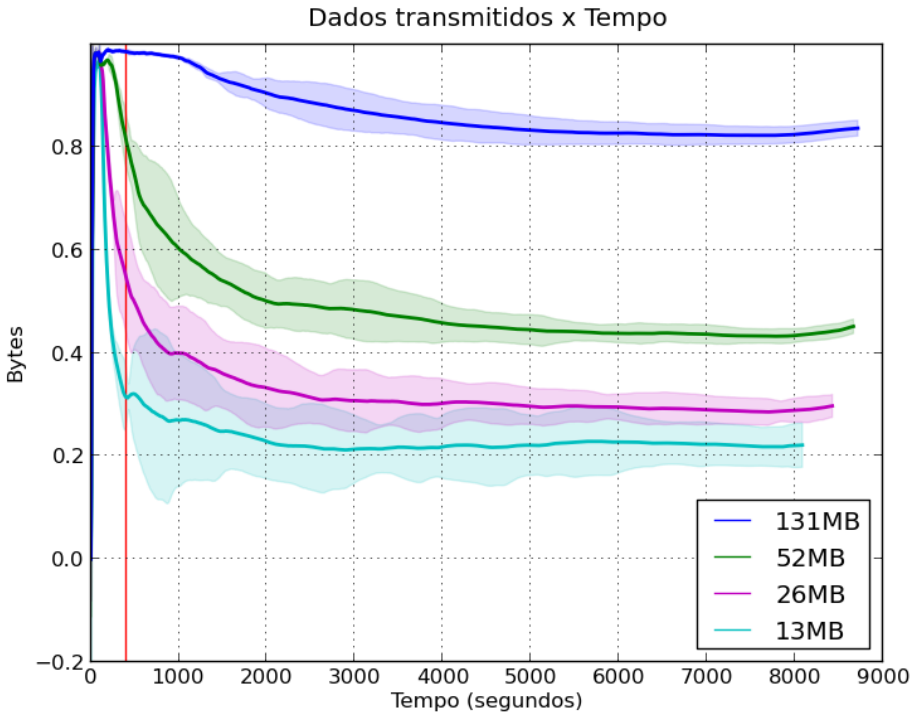

Figura 5.8: Dynamo: evolução da taxa de acertos para diferentes tamanhos de cache

Mais uma vez é possível reparar na figura 5.8 que os resultados das diferentes rodadas são mais parecidos entre si para os casos onde utilizamos mais memória. Como esperado, o resultado também é consideravelmente melhor nesses casos.

É também possível observar que para tamanhos menores de cache, a variação da taxa de acertos sofre alterações consideráveis durante a execução de um experimento. Isso deve ser causado pelo comportamento pseudo-aleatório dos clientes com interatividade, que interferem (às vezes positivamente e às vezes negativamente) com o desempenho do algoritmo.

Pode-se observar também uma leve inclinação para cima de todas as curvas no final do experimento. Isso é claramente o efeito dos clientes deixando de entrar no sistema, o que melhora a qualidade de serviço dos que ainda restam, já que os recursos estão cada vez mais abundantes.

5.4.4 Alteração do tamanho do SlotBuffer

Neste experimento, cujo gráfico pode ser visto em 5.9, foi alterado o tamanho do SlotBuffer. O objetivo é analisar a influência do número de slots, em um SlotBuffer, na taxa de acertos da cache. O tamanho da cache foi fixado em um valor correspondente à 50% da mídia. O objetivo era encontrar alguma diferença consistente no valor final da taxa de acertos para o mesmo experimento, repetido com diferentes tamanhos de SlotBuffer.

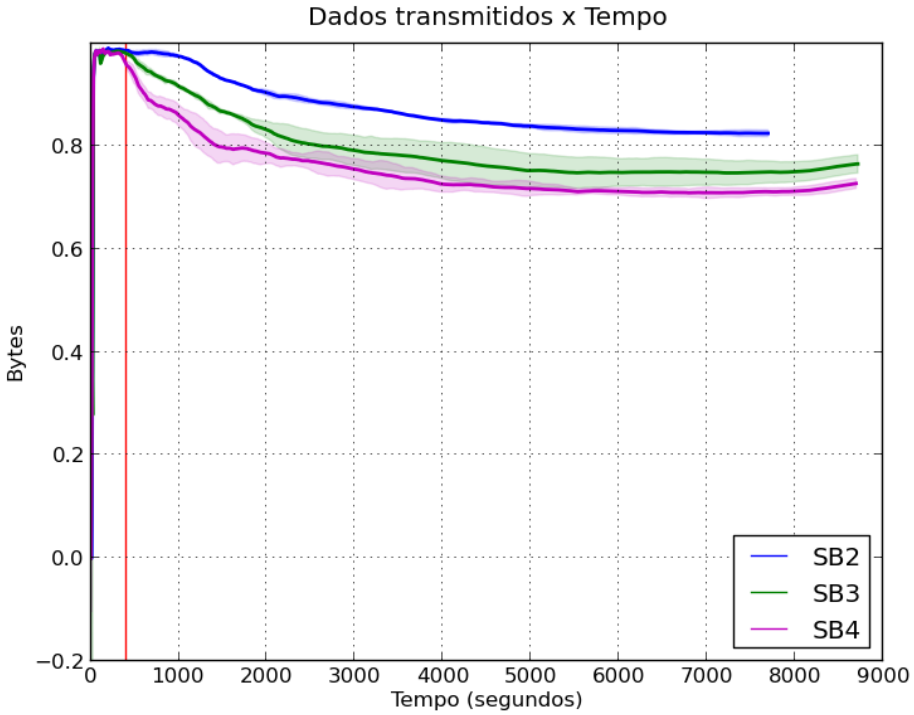

Figura 5.9: Taxa de acertos para diferentes tamanhos de SlotBuffer. Tamanho de cache correspondente a 50% da mídia

Se considerarmos que aumentando o tamanho do SlotBuffer estamos também aumentando o "custo" em memória que cada usuário aceito tem sobre o sistema. Um

SlotBuffer curto ainda provê o benefício para o cliente aceito, que é o de garantir um acerto na memória e evitar o alto custo de trazer o segmento desde o servidor de origem. Ao mesmo tempo, sempre mantém uma quantidade de memória disponível no sistema, e portanto aumentando o número de clientes que podem ser aceitos.

Podemos então concluir que para o vídeo de testes, e com as condições de rede presentes no experimento, o tamanho ideal de SlotBuffer é o de dois fragmentos. Isso claro, pode mudar caso as condições da rede piorem. Em uma rede com um maior *jitter* por exemplo, o ideal poderia ser um SlotBuffer de 3 ou mais fragmentos.

5.4.5 Medição do efeito da criação de cadeias de LinkSlots

Com a intenção de analisar o efeito que a criação de cadeias de Slots de Ligação tem, analisamos duas versões do mesmo experimento para várias sessões. Em primeiro lugar estávamos interessados em saber se a criação de cadeias de LinkSlots tem algum efeito mensurável sobre a taxa de acertos. E em caso de ter, queríamos medir essa diferença.

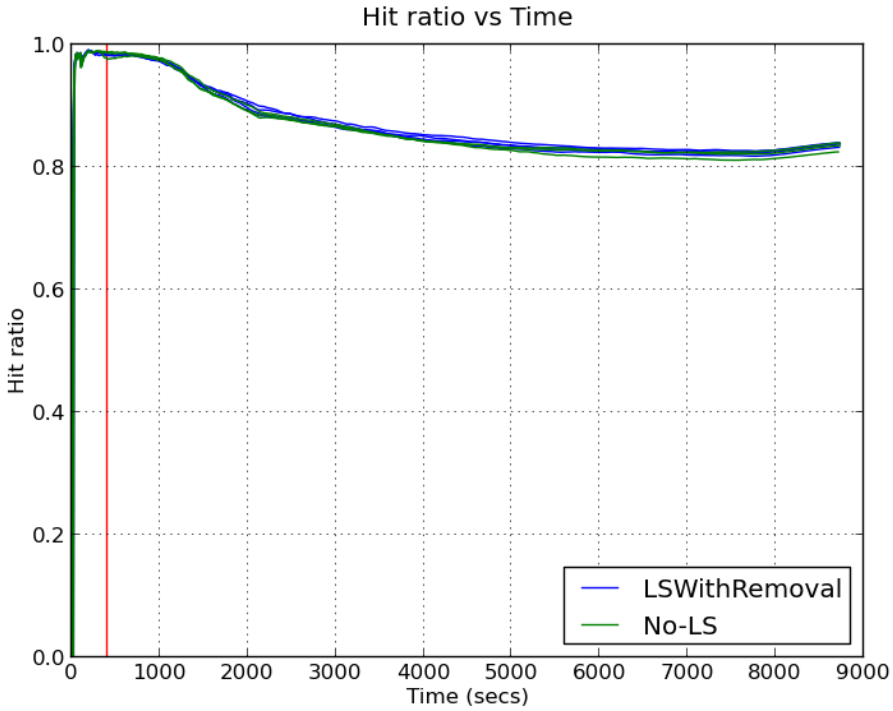

Figura 5.10: Taxa de acertos para a versão com LinkSlots e a versão sem LinkSlots do experimento

Como é possível observar na figura 5.10, as duas versões do experimento não apresentam diferenças significativas no que diz respeito à taxa de acertos. Ambas versões foram executadas com um intervalo médio de chegada de clientes de 20 segundos e o gráfico mostra os traços individuais de cada rodada do experimento.

Este tempo provou ser muito curto, colocando os clientes muito próximos uns dos outros e esgotando a memória do Distribuidor, deixando ele assim sem recursos para gerar cadeias de LinkSlots.

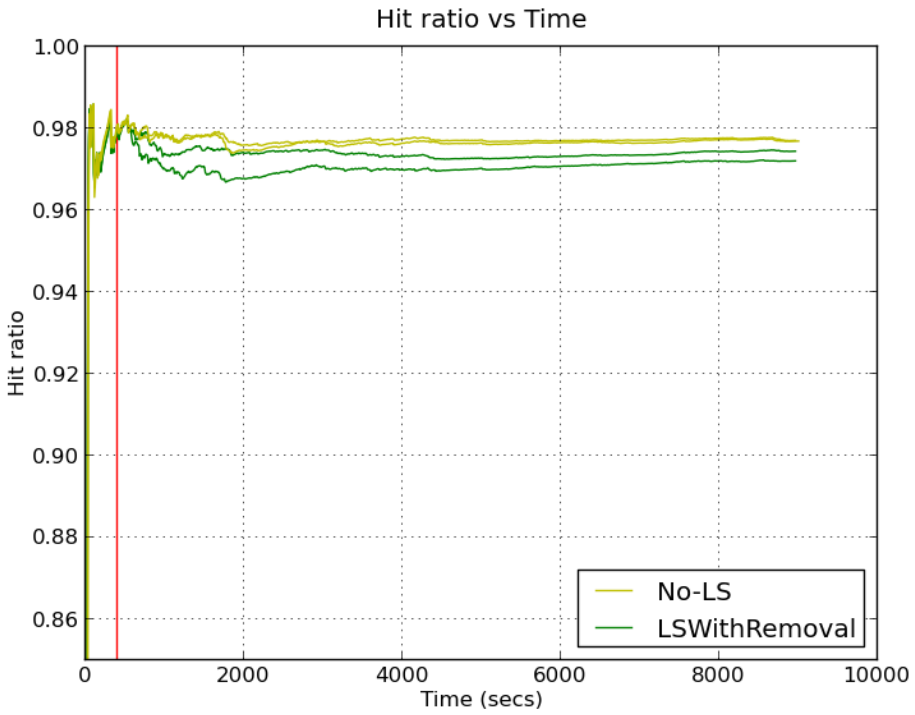

Figura 5.11: Taxa de acertos para a versão com LinkSlots e a versão sem LinkSlots do experimento, com um intervalo entre chegadas médio de 40 segundos.

Um outro experimento foi realizado, aumentando este intervalo médio, para conferir a ocorrência de alguma diferença nos resultados. Duplicando o valor do intervalo de chegada, pelo menos foi possível comprovar a criação de cadeias de LinkSlots por meio de uma ferramenta de visualização de cache desenvolvida em conjunto com o Distribuidor. O resultado pode ser apreciado na figura 5.11. Como era esperado a taxa de acertos foi bem maior. Isso porque o número total de clientes em regime permanente foi menor.

Note que nesta figura a escala vertical teve que ser ajustada para poder apreciar melhor as pequenas diferenças na taxa de acertos acumulada em diferentes rodadas do experimento. E de novo os traços individuais de cada experimento foram plotados.

Apesar disso, não é possível observar uma separação clara na taxa de acertos das duas políticas de cache.

Este resultado foi obtido consistentemente em várias outras alterações dos mesmos testes. Inicialmente isso não foi o esperado, mas como podemos ver a continuação estes resultados são perfeitamente normais se observarmos mais cuidadosamente o funcionamento do algoritmo.

5.4.6 Efeito da Ausência ou Presença de LinkSlots

Se analizarmos um cenário simples onde sería possível criar cadeias de LinkSlots, podemos apreciar melhor qual é o impacto de sua ausência/presença. No caso da figura 5.12 temos dois clientes assistindo trechos do vídeo relativamente próximos entre si. Porém, como estamos rodando a versão sem a possibilidade de criar LinkSlots um segundo fluxo oriundo do servidor deve ser criado. Já na figura 5.13 podemos apreciar que o efeito imediato da criação de uma cadeia de LinkSlots é a economia da largura de banda disponível entre o distribuidor e o servidor de origem.

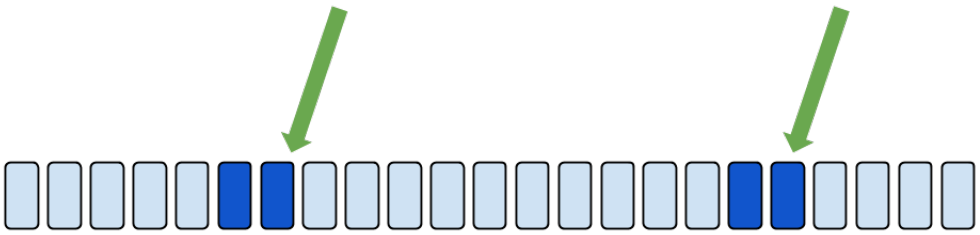

Figura 5.12: Dois SlotBuffers correspondendo a dois clientes muito próximos, mas rodando a versão do algorimo sem cadeias de LinkSlots

É importante reparar, no entanto, que tanto no primeiro como no segundo caso a taxa de acertos para os dois clientes é completamente independente da presença e/ou ausência da cadeia de LinkSlots entre os buffers colapsados dos clientes. Em uma condição de memória excedente então teríamos uma taxa de acerto muito alta para ambos clientes. Isso é principalmente devido à criação do próprio SlotBuffer e ao seu modo de operação, adiantando a busca dos fragmentos que irão ser requeridos.

Em condições de baixa memória, por outro lado, as cadeias de LinkSlots não são criadas pois o algoritmo prioriza a criação de novos SlotBuffers à criação de cadeias de LinkSlots. Neste modo de operação então ambas versões do algoritmo funcionan virtualmente da mesma maneira.

A comparação correta a fazer então seria medir o impacto das duas versões do algoritmo sobre a carga do servidor. O mesmo experimento então foi repetido, mas desta vez medindo-se o fluxo de vídeo enviados do servidor durante um intervalo de 10 segundos.

5.5 Impacto no servidor

Para deixar em evidência a potencial diferença entre os dois resultados, foi decidido operar este experimento somente com um conjunto de clientes sem operações de

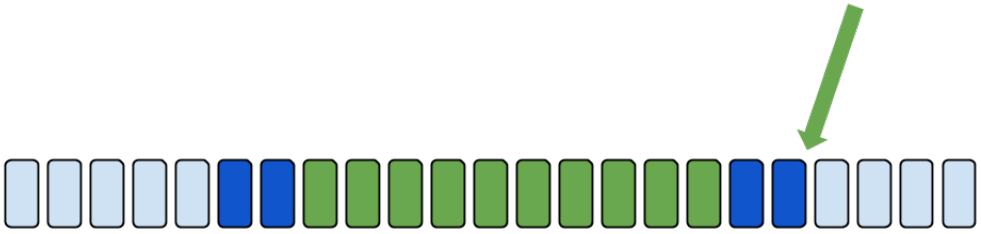

Figura 5.13: Dois SlotBuffers correspondendo a dois clientes muito próximos, rodando a versão do algorimo com cadeias de LinkSlots

interatividade; isso já que elas estariam reduzindo a vida média das cadeias de LinkSlots e forçando a criação de novos fluxos.

Os resultados do experimento são claros, e aparecem na figura 5.14. Nela, o gráfico mostra, no decorrer do tempo, a quantidade acumulativa de conteúdo multimídia, em bytes, transmitida pelo servidor de origem para os clientes. Encontramos um decréscimo de aproximadamente 50% na demanda por segmentos de vídeo pela versão de Dynamo que utiliza as cadeias de LinkSlots, provando desta maneira a sua superioridade em comparação à versão sem cadeias de LinkSlots. A figura mostra os traços individuais dos experimentos. A variação entre rodadas foi tão pequena, que não foi necessário mostrar o desvio padrão.

5.5.1 Alteração do algoritmo

O seguinte questionamento agora tem a ver com o modo de operação do algoritmo no que diz respeito aos Slots de prioridade mais baixa. Nas descrições originais do algoritmo, a operação de avanço de um SlotBuffer mantinha o conteúdo deixado atrás do mesmo, sendo estes Slots de baixa prioridade candidatos para a remoção a qualquer momento, antes mesmo das cadeias de LinkSlots.

Para esta implementação foi proposta uma alteração que consiste em eliminar os Slots em memória imediatamente assim que a posição do SlotBuffer tiver que ser atualizada.

A versão do algoritmo que deixa os Slots de baixa prioridade em memória e removê-los somente quando for necessário foi denominada LS-lazy, enquanto que a versão que faz a remoção imediata foi chamada de LS-strict.

Os resultados, mostrados na figura 5.15 apontaram para um melhor desempenho da versão proposta, ainda que por pouco. A interpretação deste resultado seria que os Slots que não estão claramente destinados a alimentar outro cliente no curto prazo, estariam realmente ocupando um espaço desnecessário na memória. Adiantar

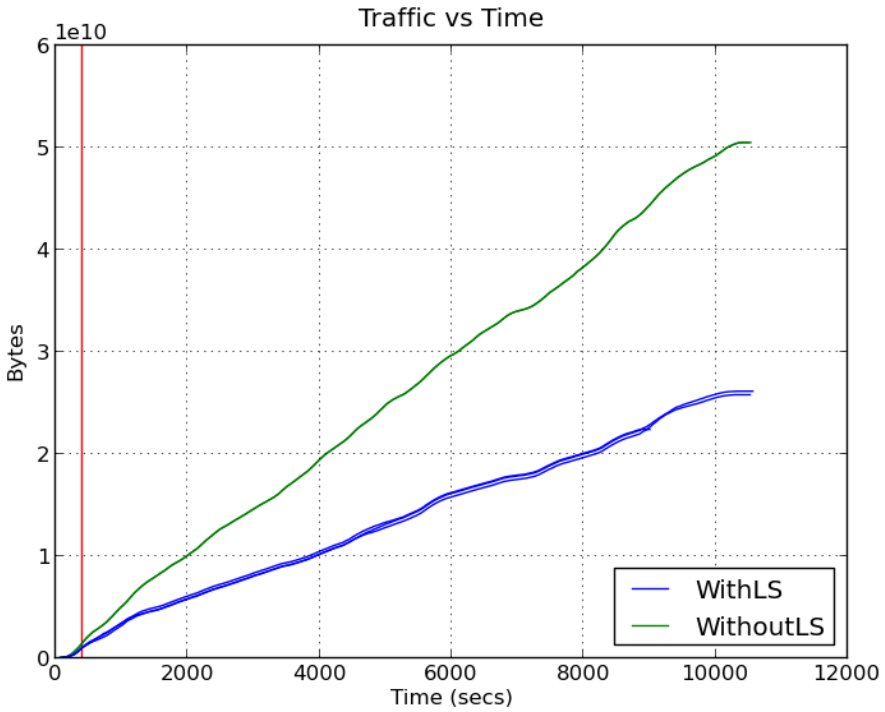

Figura 5.14: Gráfico mostrando o número de bytes transmitidos desde o servidor de origem para o distribuidor.

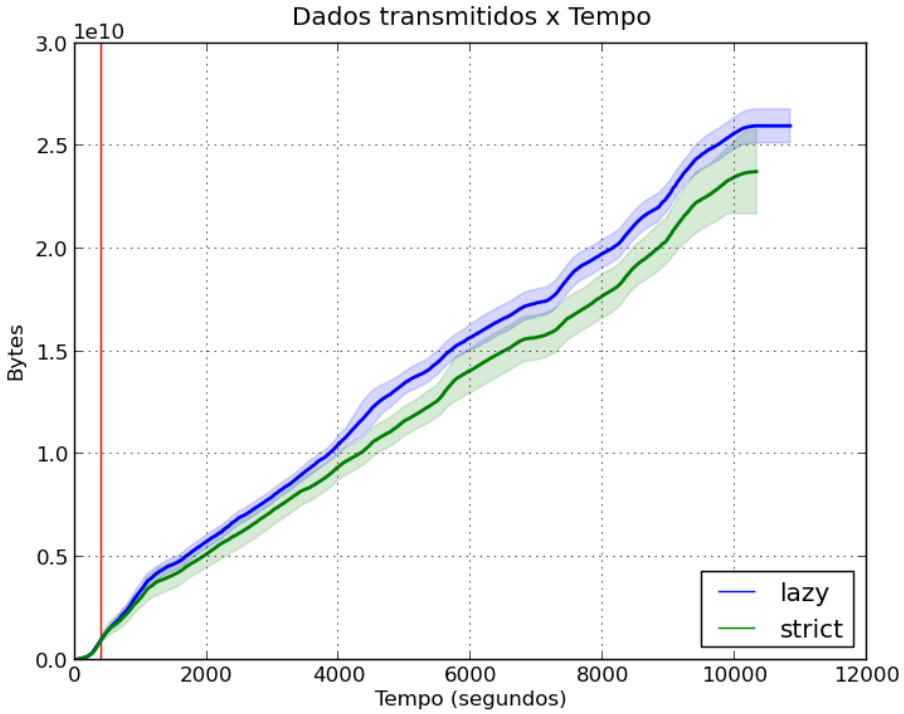

Figura 5.15: Comparação do impacto gerado no servidor para as versões lazy e strict do algoritmo.

a sua eliminação então parece ser a melhor estratégia.

O último teste importante a ser feito agora então é comparar o melhor resultado de Dynamo com o impacto causado por Varnish no servidor. A figura 5.16 mostra o resultado dessa comparação. Nela podemos ver que o desempenho do Varnish na realidade é muito similar ao da versão lazy do algoritmo. Outro detalhe interessante é que até antes dos 2500 segundos aproximadamente, Varnish tem um melhor desempenho, com menos dados sendo transferidos do servidor de origem. Depois disso, no entanto, a curva que descreve a transferência acumulada do servidor para Dynamo fica quase sempre abaixo da curva de Varnish.

A explicação para este fenômeno é simples. Como o algoritmo do distribuidor tira um forte proveito da criação de cadeias de LinkSlots, ele começa em desvantagem. Enquanto os clientes estiverem entrando ao sistema e a cache estiver "fria", menos cadeias de LinkSlots podem ser geradas. Assim que o experimento entra na fase de "regime permanente"é que o algoritmo tem um aproveitamento máximo. Então mesmo começando com uma desvantagem inicial, ele consegue ter um desempenho final melhor. Se o vídeo tivesse uma duração maior, essa diferença só iria crescendo com o tempo.

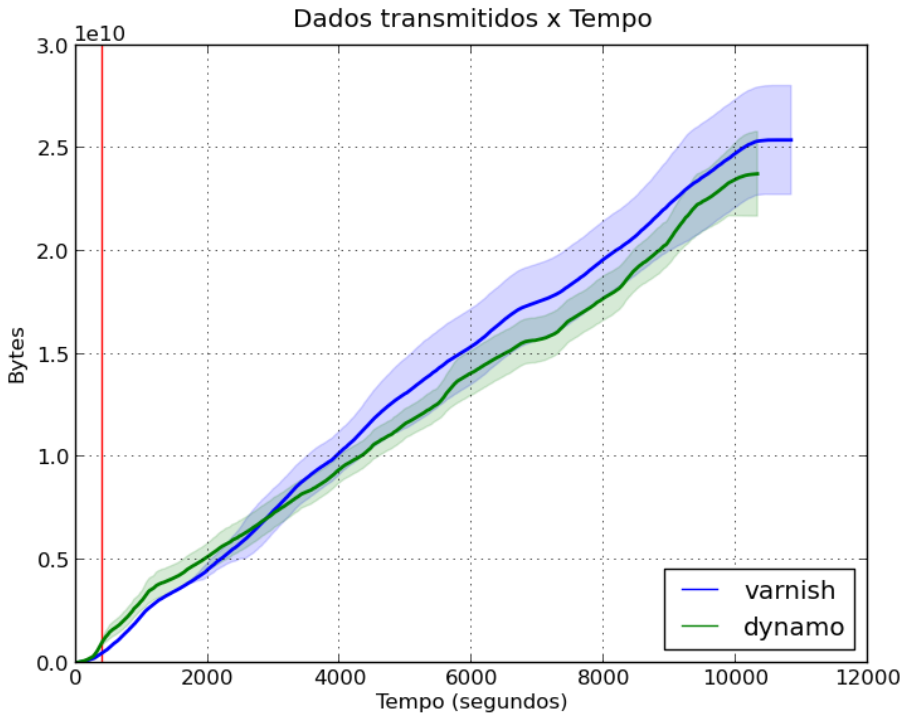

Figura 5.16: Comparação do impacto sobre o servidor entre Varnish e Dynamo na melhor versão do algoritmo (strict).

5.6 Discussão

Nessa seção, o Dynamo foi avaliado em relação aos outros sistemas cache de vídeo conhecidos e analisamos a contribuição de cada técnica MCC para o seu desempenho.

Em primeiro lugar comparamos o desempenho de alguns sistemas de cache em memória com sistemas de cache em disco, basicamente estressando ao máximo ambos com a finalidade de expor a desvantagem que a latência ao acesso à disco apresenta. Como esperado os dois sistemas que fazem uso de cache em memória (Dynamo e Varnish) não apresentaram o limite de vazão experimentado pelos demais, tal como foi mostrado em [30].

A seguir comparamos a taxa de acerto medida por Varnish com aquela medida por Dynamo para diferentes valores percentuais (em relação ao tamanho da mídia completa) de memória cache disponível. Como esperado, a medida que o tamanho da memória cache era reduzido, a taxa de acertos em ambos sistemas cai. Mas a taxa de acertos de Dynamo se mantém sempre acima da taxa de acertos do Varnish. Isso devido principalmente ao *prefix caching* ou *pre-fetch* propostas em MCC. Varnish, sendo uma ferramenta de cache genérica, não tem conhecimento específico sobre a correlação temporal entre os pedidos e portanto é incapaz de antecipar os pedidos HTTP.

O seguinte experimento foi simplesmente uma comparação da taxa de acertos em Dynamo com vários níveis de cache. Com isso foi constatado que com um tamanho de cache equivalente ao 50% da mídia, obtemos um taxa de acerto de um pouco mais de 80%. Um resultado aproximadamente 15% superior ao de Varnish sob as mesmas condições.

Outro resultado interessante foi também a constatação de que o melhor comprimento para o SlotBuffer é o de 2 fragmentos, ao contrário dos 4 utilizados em outros trabalhos. Isso se deve principalmente ao fato de estarmos usando um vídeo com uma maior taxa de bits. Como consequência direta desse fato, agora cada fragmento precisa geralmente de mais espaço em memória. Portanto fazer um *pre-fetch* de vários fragmentos a frente do atual sendo solicitado pelo cliente na realidade implica ter um maior custo em recursos por cliente. Isso sem necessariamente aumentar a taxa de acertos. Ou seja, com um SlotBuffer maior o sistema começa a rejeitar clientes (e com isso registrar eventos de *cache miss*) bem antes do que se usássemos SlotBuffers menores.

Finalmente foi avaliado um aspecto igualmente importante: a quantidade de fluxos de vídeo entre o Distribuidor e o servidor de origem. Um dos principais objetivos da MCC é a redução da carga no servidor de origem. Mais uma vez, por Dynamo ter uma informação mais específica sobre a natureza do conteúdo, é capaz de corretamente priorizar os fragmentos a serem utilizados proximamente por

outros clientes, evitando a sua remoção e assim diminuindo a quantidade de acessos ao servidor de origem.

Porém, este efeito requer a presença de vários clientes relativamente próximos uns dos outros, o que de acordo com o perfil do evento pode não ocorrer, caso o vídeo não seja suficientemente popular.

Além disso a contribuição do uso de LinkSlots foi avaliada, apresentando diferenças significativas de desempenho. A versão de Dynamo, com criação de cadeias de LinkSlots, apresentou uma redução de aproximadamente 50% no tráfego de dados solicitado ao servidor de origem.

Finalmente uma modificação ao algoritmo original foi proposta, na qual os Slots de prioridade mais baixa eram removidos sob demanda. Mas os experimentos confirmaram a superioridade do formato original do algoritmo na qual eles são removidos logo após serem enviados para o cliente, caso não exista uma cadeia de LinkSlots.

Apesar de ter provado que o algoritmo da MCC oferece um melhor desempenho que uma ferramenta de cache web comum, os testes não foram feitos usando um vídeo com múltiplas representações, ou vários níveis de qualidade.

Porém para efeitos práticos, ambos eventos originados pelo cliente: a interatividade via *seek* e troca de representação implicam em um evento de *cache miss*. Então, apesar de não termos feito experimentos com vídeos de múltiplas taxas, é possível argumentar que os resultados não seriam muito diferentes dos obtidos nos experimentos com interatividade.

Capítulo 6

Conclusões

Neste trabalho foi apresentada a adaptação de um algoritmo de cache de vídeo originalmente desenvolvido para operar com fluxos contínuos de dados.

No contexto atual são muito utilizados protocolos de transmissão adaptativos via HTTP. Estes protocolos (dos quais existem várias implementações) à diferença dos métodos tradicionais de *streaming*, trabalham no modo *client pull*. Como consequência direta disso, existe uma maior participação do cliente e uma redução na complexidade do lado do servidor. Isso de certa maneira facilita o projeto de um Distribuidor, simplificando o modo de operação de certas estruturas, como por exemplo, o anel duplo introduzido para mídia contínua.

As avaliações de desempenho do sistema de cache focaram-se em dois segmentos diferentes, a conexão Cliente-Distribuidor e a Distribuidor-Servidor. Para a interface com o cliente, estávamos interessados em garantir na maior medida do possível uma boa qualidade de serviço. Isso se traduz diretamente em um baixo tempo de buferização e menos eventos de esgotamento do buffer. Assumindo sempre que a latência entre o cliente e o Distribuidor seja menor que a latência entre o cliente e o servidor, o tempo de buferização baixo vai ser garantido pelo *prefix caching*. Da mesma maneira, os eventos de esgotamento de buffer seriam mais raros se pudermos garantir uma maior taxa de acertos no Distribuidor. Desta maneira foram projetados vários experimentos com a finalidade de medir a taxa de acerto oferecida pelo Dynamo, e comparar este resultado com o Varnish, uma popular ferramenta de cache baseada em memória, para conteúdo web genérico. O Dynamo apresentou uma melhoria na taxa de acertos de até 15% em comparação com o Varnish.

Por outro lado, na interface entre o Distribuidor e o servidor de origem, queremos minimizar o fluxo proveniente do servidor de origem, aliviando assim a carga de pedidos ao mesmo. Por este lado, varias configurações de Dynamo foram testadas. Dynamo, na melhor de suas configurações, conseguiu uma redução de até 8% da carga de dados saindo do servidor de origem. Esse ganho pode aumentar, caso a duração do evento - atualmente de 6 minutos - for maior.

Em trabalhos futuros talvez seja interessante alterar alguns outros parâmetros secundarios do algoritmo, como por exemplo o comprimento máximo da cadeia de LinkSlots, e fazer testes com vídeos de menor qualidade com a finalidade de observar se o tamanho de SlotBuffer ótimo continua sendo igual.

Espera-se que o tamanho do SlotBuffer ótimo seja dependente tanto das características do vídeo como do nível de *jitter* existente na rede. O tamanho do SlotBuffer poderia então ser dinâmicamente ajustado de acordo com as medições desses dois parâmetros.

Referências Bibliográficas

[1] *Brief history of the internet.* In: Report, 2012. Disponível em: <http://www.internetsociety.org/brief-history-internet>.

[2] WATKINSON, J. *The MPEG handbook.* 2 ed. 70, Blanchard Rd. Suite 402, Burlington, MA, 01803, Focal Press, 2013.

[3] *Cisco Visual Networking Index: Forecast and Methodology, 2013–2018.* In: Report, Cisco, 2014. Disponível em: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf>. Acesso em: 10/09/2014.

[4] BROXTON, T., INTERIAN, Y., VAVER, J., et al. "Catching a Viral Video". In: *IEEE SIASP@ICDM 2010*, 2010.

[5] JOHN L. HENNESSY, D. A. P. *Computer Architecture A Quantitative Approach.* 4 ed. 500 Sansome Street, Suite 400, San Francisco, CA 94111, Morgan Kaufmann Publishers, 2014.

[6] KAMAT, S. P. "Fault-Tolerant Architecture for an MPEG-4 Based Video Decoder Driver". In: *Embedded Systems Letters, IEEE*, 2012.

[7] *Understanding Delay in Packet Voice Networks.* In: Report, Cisco, 2006. Disponível em: <http://www.cisco.com/c/en/us/support/docs/voice/voice-quality/5125-delay-details.pdf>. Acesso em: 9/9/2014.

[8] POSTEL, J. "RFC 768: User Datagram Protocol". ago. 1980. Disponível em: <ftp://ftp.internic.net/rfc/rfc768.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc768.txt>. Acesso em: 18/11/2014. Status: STANDARD.

[9] POSTEL, J. "RFC 793: Transmission Control Protocol". set. 1981. Disponível em: <ftp://ftp.internic.net/rfc/rfc793.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc793.txt>. Acesso em: 28/11/2014. Status: STANDARD.

[10] AUDIO-VIDEO TRANSPORT WORKING GROUP, SCHULZRINNE, H., CASNER, S., et al. "RFC 1889: RTP: A Transport Protocol for Real-Time Applications". jan. 1996. Disponível em: <ftp://ftp.internic.net/rfc/rfc1889.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc1889.txt>. Acesso em: 28/11/2014. Status: PROPOSED STANDARD.

[11] DEERING, S. E. "RFC 1112: Host extensions for IP multicasting". ago. 1989. Disponível em: <ftp://ftp.internic.net/rfc/rfc1054.txt,ftp://ftp.internic.net/rfc/rfc1112.txt,ftp://ftp.internic.net/rfc/rfc2236.txt,ftp://ftp.internic.net/rfc/rfc988.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc1054.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc1112.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc2236.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc988.txt>. Acesso em: 28/11/2014.

[12] SCHULZRINNE, H., RAO, A., LANPHIER, R. "RFC 2326: Real Time Streaming Protocol (RTSP)". abr. 1998. Disponível em: <ftp://ftp.internic.net/rfc/rfc2326.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc2326.txt>. Status: PROPOSED STANDARD.

[13] *Real-Time Messaging Protocol (RTMP) specification*. In: Report, Adobe Systems Incorporated, 2009. Disponível em: <http://www.adobe.com/devnet/rtmp.html>. Acesso em: 10/9/2014.

[14] PHILLIPA GILL, MARTIN ARLITT, Z. L. A. M. "YouTube Traffic Characterization: A View From the Edge", *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, out. 2007.

[15] E. ISHIKAWA, C. L. A. *Memória Cooperativa Distribuída: Uma solução escalável para Sistemas de Vídeo sob Demanda interativos*. In: Report, COPPE/UFRJ, 2000. Disponível em: <http://www.cos.ufrj.br/index.php?option=com_publicacao&task=visualizar&id=541>.

[16] KUN-LUNG WU, PHILIP S. YU, J. L. W. "Segment-based proxy caching of multimedia streams", *Proceedings of the 10th international conference on World Wide Web*, pp. 36–44, maio 2001.

[17] ANNA SATSIOU, M. P. "Efficient caching of video content to an architecture of proxies according to a frequency-based cache management policy", *Proceedings of the 2nd international workshop on Advanced architectures and algorithms for internet delivery and applications*, out. 2006.

[18] SEN, S., R. J. T. D. "Proxy Prefix Caching for Multimedia Streams". In: *Proceedings IEEE Infocom'99*, pp. 1310 – 1319, New York, NY, mar. 1999.

[19] MARKUS HOFMANN, L. B. "Content Networking. Architecture, Protocols and Practice". In: Johnson, K. (Ed.), *Content Networking*, 1 ed., cap. 4, 500 Sansome Street, Suite 400, San Francisco, CA 94111, Morgan Kaufmann Publishers, 2005.

[20] S. SHEU, K. A. HUA, W. T. "Chaining: A generalized Batching Technique for Video-On-Demand Systems". In: *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, Otawa, Canada, 1997.

[21] K. A. HUA, Y. CAI, S. S. "Patching: A Multicast Technique for True Video-On-Demand Services". In: *Proceedings of the sixth ACM international conference on Multimedia*, set. 1998.

[22] ISHIKAWA, E. *Memória Cooperativa para Distribuição de Vídeo sob Demanda*. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2003.

[23] *Containers—Not Virtual Machines—Are the Future Cloud*. In: Report, 2013. Disponível em: <`http://www.linuxjournal.com/content/containers%E2%80%94not-virtual-machines%E2%80%94are-future-cloud`>.

[24] TURNBULL, J. *The docker book*. 1 ed. 500 Sansome Street, Suite 400, San Francisco, CA 94111, 2014.

[25] DE PINHO, L. B. *Implementação e avaliação de um sistema de vídeo sob demanda baseado em cache de vídeo cooperativa*. M.Sc. dissertation, COPPE/UFRJ, Berkeley, California, USA, 2002.

[26] *Adobe Flash Video File Format Specification Version 10.1*. In: Report, Adobe Systems Incorporated, 2010. Disponível em: <`http://download.macromedia.com/f4v/video_file_format_spec_v10_1.pdf`>. Acesso em: 10/9/2014.

[27] HOGG, R. V.; CRAIG, A. T. *Introduction to Mathematical Statistics*. New York, Macmillan, 1978.

[28] *What is Iperf?* Web site, 2014. Disponível em: <`https://iperf.fr`>. Acesso em: 5/09/2014.

[29] "File Packager Reference". Disponível em: <`http://help.adobe.com/en_US/HTTPStreaming/1.0/Using/`

WSaeac10ab694095a12a9a3a7d12823cda643-7ffc.html>. Acesso em: 8/09/2014.

[30] J. SUMMERS, T. BRECHT, D. E. B. W. "Methodologies for Generating HTTP Streaming Video Workloads to Evaluate Web Server Performance". In: *Proceedings of the 5th Annual International Systems and Storage Conference*, 2012.

[31] *Usage statistics and market share of Apache for websites.* In: Report, 2013. Disponível em: <http://w3techs.com/technologies/details/ws-apache/all/all>. Acesso em: 5/09/2014.

[32] "Varnish Cache". Disponível em: <https://www.varnish-cache.org>. Acesso em: 24/11/2014.

[33] HONGLIANG YU, DONGDONG ZHENG, B. Y. Z. W. Z. "Understanding User Behavior in Large-Scale Video-on-Demand Systems". In: *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, 2006.