Aliviando a Carga de Servidores VoD Através de Cache de Vídeo Cooperativa[1]

Leonardo Bidese de Pinho, Edison Ishikawa e Claudio Luis de Amorim
{leopinho, edsoni, amorim}@cos.ufrj.br
COPPE / Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil 21941-972

Resumo

Na era da informação, facilitar o acesso a conteúdo multimídia é um dos mais desafiadores campos de pesquisa. Nesta área de estudo, **Vídeo sob Demanda (VoD)** recebe atenção especial. Este artigo descreve a implementação de um novo tipo de sistema VoD escalável baseado em conceitos de **Cache de Vídeo Cooperativa (CVC)** no qual, diferentemente dos sistemas convencionais, a largura de banda do servidor não limita o número de clientes simultaneamente atendidos. Os resultados preliminares de desempenho do protótipo são analisados e comparados com os de um sistema convencional, demonstrando economia de até 84% da taxa de utilização do servidor.

Abstract

In the information age one of the most challenging research fields is to make easy the access to multimedia content. In this area of study **Video on Demand (VoD)** gets special attention. This article describes the implementation of a scalable VoD system based on concepts of **Cooperative Video Caching (CVC)** in which, differently from conventional systems, the server bandwidth does not limit the number of simultaneous clients. Preliminary performance results of a CVC-based VoD prototype are analyzed and compared to that of a conventional system, demonstrating that savings up to 84% of the server's utilization rate could be achieved.

Palavras-chave: vídeo sob demanda, sistemas distribuídos multimídia, cache cooperativa

1 Introdução

Atualmente, a crescente busca por informação multimídia torna essencial o desenvolvimento de novas formas mais eficientes de distribuir conteúdo multimídia. Na perspectiva do cliente que busca o conteúdo, seja este de notícias, entretenimento, educacional, entre outros, o ideal é ter a possibilidade de acessá-lo instantaneamente, em qualquer momento. Vários esforços de pesquisas foram realizados na área denominada Mídia Contínua sob Demanda (CMoD), englobando técnicas de armazenamento, processamento, acesso e distribuição de informação [14].

Em particular, trabalhos pertencentes ao escopo de transmissão de CMoD, com maior ênfase em vídeo - **Vídeo sob Demanda (VoD)**, salientam o fato de a largura de banda do servidor ser o

[1]Este trabalho foi financiado pela CAPES e FINEP

principal fator que inibe o atendimento de grandes quantidades de clientes. Para contornar esta limitação, diversas técnicas foram propostas nos últimos anos com o intuito de proporcionar um acesso mais eficiente ao servidor. Antes de abordá-las, convém salientar que os sistemas convencionais de VoD utilizam fluxos *unicast* (um fluxo de vídeo partindo do servidor atendendo a um único cliente). Dentre suas principais características, as mais significativas são o suporte a operações de videocassete (VCR) e o atendimento imediato do cliente, ou seja, o cliente começa a assistir o conteúdo praticamente no momento da solicitação. Entretanto, como este tipo de transmissão exige grande largura de banda (Ex.: Vídeo MPEG-1[2] gera fluxo médio de 1,5 Mbps), a mesma fica esgotada com um número reduzido de clientes.

Para um melhor aproveitamento da largura de banda, técnicas de *broadcast* (único fluxo disponível para todos clientes), onde fluxos são gerados em intervalos pré-estabelecidos, foram desenvolvidas. Esta abordagem é conhecida como Near-VoD, visto que ocorre uma latência significativa no começo da exibição do vídeo solicitado, modificando o conceito de VoD puro.

A fim de agregar os benefícios dessas técnicas, novos trabalhos foram propostos para diminuir o uso da banda e ao mesmo tempo gerar uma latência mínima na exibição. Dentre eles, salientam-se *Batching* [6], *PiggyBacking* [9], *Stream Tapping/**Patching*** [11, 5] e ***Chaining*** [17].

No *Batching*, requisições relativas a um determinado conteúdo são enfileiradas até que uma certa quantidade seja atingida, a partir da qual a transmissão se dá através de *multicast* (único fluxo enviado a um grupo de clientes), economizando banda, mas gerando uma latência possivelmente grande nos primeiros clientes que solicitaram o conteúdo, privilegiando os últimos.

Adotando uma abordagem distinta, o *PiggyBacking* consiste em fazer com que dois fluxos existentes com um mesmo conteúdo, porém defasados, sejam sincronizados, possibilitando a eliminação de um deles. Isto é feito acelerando a taxa de transmissão do mais novo em até 5% (fato imperceptível ao ser humano) e retardando o mais antigo na mesma proporção, fazendo-os entrar em sincronia.

Já *Stream Tapping* e ***Patching***, duas técnicas semelhantes, partem do princípio de que os clientes têm capacidade de receber pelo menos o dobro do fluxo a ser exibido. Quando um cliente é o primeiro a pedir ao servidor um conteúdo, ele recebe um fluxo *multicast* completo. Já os clientes posteriores, ao fazerem suas solicitações, passam a fazer parte do grupo de receptores do primeiro fluxo, sendo o mesmo armazenado em um buffer. Ao mesmo tempo, um segundo fluxo - **remendo** - é criado, contendo a parte compreendida entre o início do vídeo e a parte já bufferizada. Com isso, o cliente ocupa banda adicional do servidor por um período relativamente curto.

Apesar de obterem bons resultados, essas técnicas desconsideram o fato do cliente estar bufferizando - realmente ou potencialmente, dependendo da técnica - o conteúdo, devido a necessidade de existir pelo menos um buffer mínimo introduzido para eliminar o jitter da rede de transmissão. Considerando esta característica, surge a oportunidade de migrar o sistema para o modelo peer-to-peer, solucionando o problema do gargalo existente no acesso ao servidor, intrínseco ao modelo cliente-servidor. O ***Chaining*** aproveita esta característica, gerando um encadeamento de buffers (B), tornando os clientes provedores de novos clientes, o que alivia a carga no servidor. A desvantagem de ***Chaining*** está na impossibilidade de novos clientes reutilizarem o fluxo de encadeamento, quando a parte inicial do conteúdo já tiver sido descartada.

[2]Motion Picture Experts Group - 1 (International Standart 11172)

Baseada em uma abordagem híbrida entre ***Chaining*** e ***Patching***, com ênfase no modelo peer-to-peer, foi desenvolvida a técnica denominada **Cache de Vídeo Cooperativa (CVC)** [12, 13]. Essencialmente, CVC consiste em tratar os buffers locais dos clientes como componentes de uma memória global passível de suprir conteúdo para os demais. Com isso, os clientes tornam-se provedores preferenciais de novos fluxos *multicast*, reduzindo a necessidade de acesso ao servidor, sendo que remendos, provenientes sempre que possível de outros clientes, e não do servidor, são enviados para repor seqüências ausentes (a Figura 1 apresenta diferenças entre as três últimas técnicas abordadas). Resultados preliminares obtidos através de simulação indicam que a CVC tem um grande potencial para melhorar o desempenho e a escalabilidade de sistemas VoD.

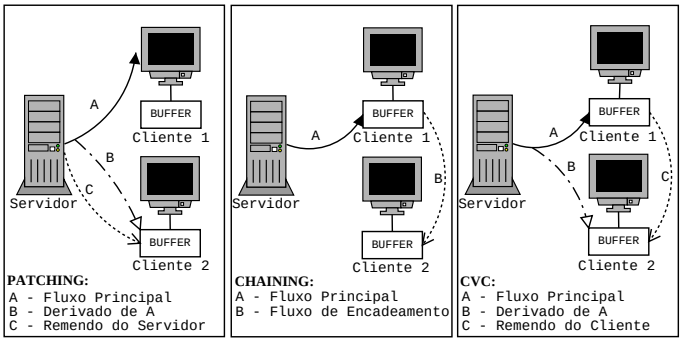

Figura 1: Diferenças entre as técnicas

Este trabalho tem como objetivo principal investigar aspectos práticos de implementação da CVC, bem como o comportamento resultante da introdução de seus conceitos em sistemas convencionais. Para isso, encontra-se em fase de desenvolvimento no Laboratório de Computação Paralela (LCP) da COPPE/UFRJ, o protótipo denominado **Sistema CVC (CVCS)**.

Suas principais características de implementação e funcionamento, bem como resultados preliminares de desempenho, são abordados no decorrer deste texto, cujo foco está voltado para a viabilidade de desenvolvimento e a economia de banda que se pode obter em servidores convencionais usando conceitos de CVC, atentando para a influência da relação existente entre a capacidade dos buffers locais e o intervalo entre chegadas de clientes ao sistema na ocupação de canais do servidor.

O protótipo foi desenvolvido para ambientes de rede fechados, com estrutura homogênea. Isto ocorre em meios empresariais, acadêmicos e similares, onde existe largura de banda simétrica, e abundante o suficiente para que as técnicas de streaming de mídia contínua sejam efetivas. No que tange a interatividade VCR, implementou-se o recurso de pausa/recomeço, visto que os clientes contam com pelo menos esta funcionalidade em um sistema VoD [1].

A organização deste artigo é a seguinte. Na seção 2, a técnica CVC é brevemente descrita. O protótipo desenvolvido - CVC System - é abordado na seção 3, enfatizando aspectos de implementação, descrição dos componentes e a forma como os mesmo interagem entre si. Na seção 4 é apresentada a metodologia experimental adotada, contendo a descrição do ambiente experimental e da carga de trabalho, além da análise dos resultados obtidos. Por fim, a seção 5 expõe as conclusões e trabalhos futuros.

2 Cache de Vídeo Cooperativa (CVC)

Esta seção descreve as principais características da técnica CVC, as quais serviram como base para a criação do protótipo.

A CVC gerencia os buffers dos clientes de forma global, tendo como unidade de acesso o Grupo de Quadros (GoF). Este grupo representa um conjunto autocontido de quadros definido no padrão MPEG, dotado de um *timestamp*, que permite um acesso aleatório aos mesmos.

O controle dos buffers é feito com 5 ponteiros. Os ponteiros denominados *write* e *read*, indicam, respectivamente, as posições de armazenamento e exibição do próximo GoF. O *begin* aponta para o GoF de *timestamp* mais antigo, sendo atualizado quando ocorre descarte de GoF. Por fim, *low* e *high* determinam os níveis de ocupação mínimo e máximo do buffer. Quando estes são atingidos, geram alertas para o gerente da CVC, para que este execute as ações necessárias para evitar *undeflow/overflow*.

O algoritmo básico da CVC segue os seguintes passos. Ao chegar a primeira requisição, o servidor inicia um novo fluxo para o cliente. Tão logo o nível mínimo do cliente seja atingido, ele começa a exibição. Quando uma segunda requisição chegar, o gerente da CVC procura em sua tabela por um cliente, chamado provedor, que possua a parte inicial do vídeo em seu buffer local, o qual é capaz de armazenar *tb* segundos.

Caso a segunda requisição tenha sido feita até *tb* segundos depois de algum cliente ter recebido o primeiro GoF, este pode ser o provedor. Neste caso, o cliente é inserido no grupo de *multicast* servido por este provedor. Como a parte inicial se perdeu, outro cliente do grupo, denominado colaborador, proverá este trecho. Se nenhum provedor estiver disponível, então o último cliente que recebeu a parte inicial ao menos *tb* segundos antes implementa um encadeamento entre seu buffer e o do requisitante.

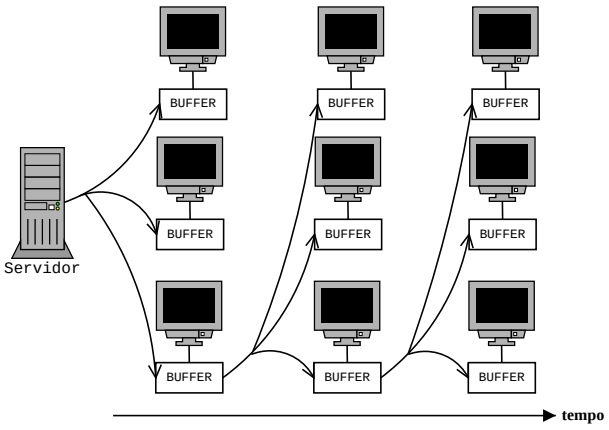

Figura 2: Árvore de encadeamentos gerada pela CVC

Conforme os clientes vão chegando, vai sendo gerada uma árvore de encadeamento (Figura 2). Como são estabelecidos fluxos em intervalos de tempos defasados de acordo com a capacidade do buffer, surge a oportunidade de oferecimento de funções VCR aos clientes, da seguinte forma: a) Caso a operação solicitada seja de avanço, o gerente percorre a cadeia em direção ao servidor em busca do novo provedor que está transmitindo blocos adiantados; b) Se

for requisitado um retrocesso, basta ao gerente percorrer o caminho inverso.

3 Sistema CVC (CVCS)

O sistema foi desenvolvido em Linux, usando prioritariamente a linguagem C padrão (ANSI/ISO C). No entanto, foram necessários alguns trechos de código C++ no cliente, devido ao servidor adotado, descrito na próxima seção, possuir uma API nessa linguagem. Na criação do cliente, utilizou-se *POSIX threads* [3] para permitir sobrepor computação com comunicação.

O protocolo UDP é usado em todas as comunicações, tanto de dados quanto de controle. Nas de dados, UDP foi adotado porque ele funciona diretamente sobre *IP Multicast* [7] e pelo seu pequeno *overhead*. Nas de controle, UDP facilitou a implementação do gerente da CVC, serializando os acessos a sua estrutura de forma a evitar os custos de criação de seções críticas. Uma interface gráfica simples para o cliente foi moldada no ambiente de desenvolvimento de interfaces Glade [8], sendo posteriormente desenvolvida diretamente com a biblioteca GTK+ [10].

3.1 Os Componentes do Sistema

O diagrama do Sistema CVC aparece na Figura 3.

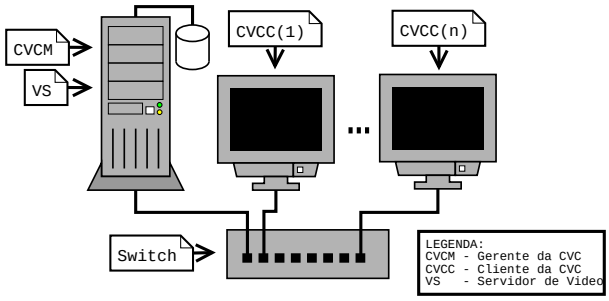

Figura 3: Diagrama do CVCS

3.1.1 *Switch* com *IGMP Snooping*

Com a queda nos custos necessários para sua implantação, a tecnologia de *switch ethernet* vem sendo adotada em diversos ambientes empresariais, acadêmicos e similares. Isto oferece um grande impulso para aplicações multimídia, já que desaparece o problema de colisão inerente ao protocolo CSMA/CD, proporcionando tempos de resposta previsíveis e grande largura de banda. No caso de transmissão de dados através de *IP Multicast*, os *switches* se tornam ainda mais importantes, visto que eles passam a atuar como filtros, de modo que os pacotes sejam enviados apenas para aquelas máquinas que tenham clientes pertencentes ao grupo de recebimento dos mesmos.

Em *switches* de nível 2, o suporte a *multicast* é feito geralmente através da técnica chamada *IGMP Snooping*. Para isso, o *switch* deve monitorar suas portas de forma que, quando cheguem

pacotes IGMP referentes a inclusão/exclusão do host em um grupo *multicast*, ele possa atualizar as entradas de sua tabela de *multicast*, que é usada no encaminhamento dos datagramas endereçados a tais grupos. Muitos dos equipamentos comercializados atualmente possuem esta funcionalidade.

3.1.2 Servidor de Vídeo (VS)

A implementação do protótipo foi feita de forma que, com pequenas alterações, qualquer servidor de vídeo possa ser usado.

- Servidor *Randomized I/O* (RIO)

O protótipo desenvolvido utiliza como servidor de vídeo uma versão modificada [4] do *Randomized I/O* (RIO) [16]. O RIO serve como exemplo de sistema convencional por trabalhar apenas com *unicast*. Dentre suas principais características, cabe destacar que ele trata os objetos (ex.: vídeo) como uma seqüência de blocos de 128KBytes. O armazenamento de tais blocos é feito aleatoriamente de forma que o tempo médio de recuperação dos mesmos seja minimizado. A comunicação do cliente com o servidor se dá através de uma API que permite criar sessões, onde a taxa de bits é definida na inicialização. Após aberta a sessão, o cliente passa a requisitar blocos, um por vez, de acordo com seu número na seqüência. O controle de admissão do sistema é feito através das taxas fixadas na abertura das sessões. Quando a soma das taxas de sessões abertas atinge a largura de banda total, nenhuma outra pode ser iniciada até que outra seja encerrada.

3.1.3 Cliente da CVC (CVCC)

O CVCC funciona basicamente como controlador de um buffer no lado do cliente (Figura 4), tendo como tarefas requisitar blocos para o buffer, enviar os blocos para o software de decodificação/exibição, além de transmitir blocos por *multicast* quando sinalizado pelo gerente da CVC (CVCM).

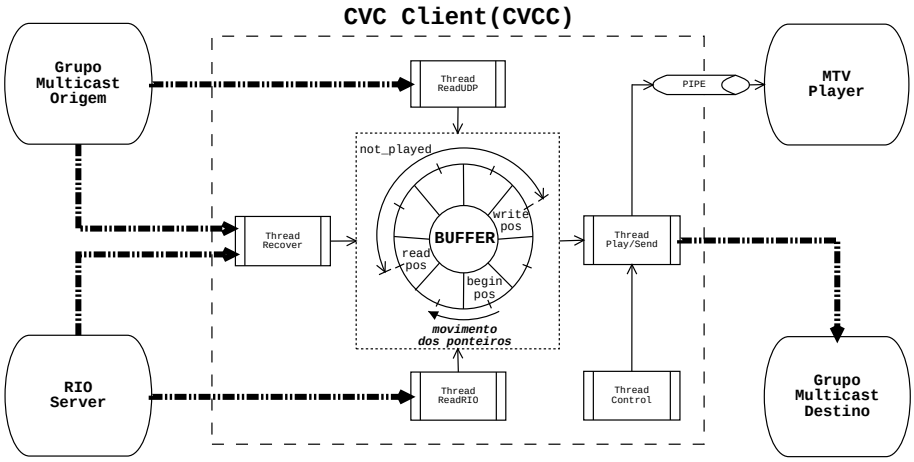

Figura 4: Diagrama do CVCC

- **Buffer do Cliente**

O buffer do cliente é circular e alocado de forma contígua como um vetor, sendo a sua capacidade total, medida em posições, denominada tamanho do buffer (BS). A forma como ele foi concebido difere, em alguns aspectos, da proposta original da CVC, principalmente no que se refere a granularidade de armazenamento e quantidade de ponteiros.

Enquanto na original cada posição do buffer contém um GoF, na implementada cada posição armazena uma estrutura composta pelos campos Block e Buffer. O primeiro funciona como identificador do bloco. O segundo guarda o bloco propriamente dito, cujo tamanho é de 128KBytes. Esta abordagem foi adotada para compatibilizar a granularidade da CVC com a do servidor. Em virtude disto, ocorre também um desvinculamento da CVC em relação ao tipo de vídeo, o que proporciona um suporte direto a qualquer padrão, desde que exista um decodificador capaz de se comunicar com o CVCC através de algum mecanismo de IPC[3].

No que tange a ponteiros, foram implementados os de leitura (*read_pos*) - referente a posição do próximo bloco a ser enviado para exibição, de escrita (*write_pos*) - referente a posição que armazenará o próximo bloco a ser recebido, de envio (*begin_pos*) - aponta para a posição do bloco mais antigo e de remendo (*patch_pos*) - referente a posição onde o bloco de remendo é inserido. O ponteiro de underflow (*low*) foi substituído por um contador (*not_played*) que mantém a quantidade de blocos recebidos e não enviados para exibição.

Dois limites foram introduzidos: Limite de Pré-busca (PL) e Limite de Descarte (DL). O PL define o número mínimo de blocos que devem estar presentes no buffer para que se possa iniciar a exibição. Já o DL determina o momento em que blocos já enviados para exibição, mantidos no buffer para fins de cooperação, começam a ser eliminados, a fim de liberar espaço para o recebimento de novos. Como o pior caso ocorre quando o cliente recebe os blocos de um provedor, onde a taxa de chegada de blocos é, em média, igual a de exibição, a capacidade do buffer fica esgotada no momento em que é exibido o bloco (BS - PL). Devido a variação existente nos tempos de consumo de blocos contíguos e ao jitter da rede, foi introduzida uma margem de folga de 4 blocos (DL = BS - PL - 4). Com isso, no caso geral, o descarte inicia no momento em que o bloco DL é exibido, o que limita a **janela de cooperação**[4] neste valor. Entretanto, no caso do cliente se tornar provedor, o descarte começa imediatamente.

Para controlar o acesso concorrente ao buffer, foi utilizado o algoritmo para buffers limitados usando semáforos descrito em Andrews [2], levemente modificado com a introdução da política de descarte exposta acima.

- **Threads**

Para executar as tarefas, foram criadas cinco threads: ReadRIO, ReadUDP, Play/Send, Control e Recover.

ReadRIO: responsável por, através da respectiva API, abrir uma sessão com o servidor, requisitar os blocos e armazenar os mesmos no buffer.

ReadUDP: tem por função aderir a um grupo de *multicast* determinado pelo CVCM, controlando a chegada dos blocos através de *headers*. Isto ocorre da seguinte forma: antes de enviar

[3]Comunicação entre processos

[4]Intervalo de tempo no qual o cliente pode criar novo *multicast*

os blocos, fragmentados em pacotes de 1KByte, o provedor envia uma mensagem contendo uma estrutura com os seguintes campos: magic number - número pré-definido que representa o início do *header*, video id - identificação do vídeo que está sendo transmitido, block id - identificador do bloco à ser transmitido e, novamente, magic number - representando fim do *header*. Com isso, a thread consegue saber se o bloco recebido é o esperado. Caso ocorra perda de bloco, desbloqueia a thread Recover.

Play/Send: fica bloqueada até que um número mínimo de blocos (minPrefetch) estejam no buffer. Ao atingir este limite, Play/Send cumpre dupla função. A primeira é ler blocos do buffer e enviá-los, através de um *pipe*, para o decodificador. A outra é enviar blocos para um endereço *multicast* definido pelo CVCM conforme sinalização da thread Control. A transmissão se dá de forma síncrona em relação ao envio de blocos para o *pipe*. Desta forma, a taxa de envio é a mesma com que os dados são consumidos, simplificando o controle da transmissão.

Control: fica a espera de uma sinalização do gerente, fazendo com que este cliente se torne provedor. A sinalização é feita com o envio, por parte do gerente, da identificação do primeiro bloco que o cliente deve transmitir. Ao receber o sinal, avisa a Play/Send para começar a enviar.

Recover: funciona atrelada a ReadUDP. Quando ocorre algum problema no fluxo de recebimento de blocos, cabe a Recover contatar o gerente em busca de um novo provedor. Caso exista, a thread ReadUDP é reinicializada, passando a receber o fluxo de um novo grupo *multicast*. Se não existir, realiza novos contatos até que o contador not_played atinja um nível mínimo. Se isso ocorrer, uma nova sessão é aberta com o servidor.

- **MpegTV Player (MTV)**

O decodificador usado no desenvolvimento chama-se MpegTV Player (MTV) [15]. Este software decodifica fluxos MPEG-1, suportando diversas formas de recebimento de fluxo, dentre elas a *stdin* (entrada padrão). Com isso, o CVCC cria, através da chamada de sistema *fork*, um novo processo que, a seguir, realiza a chamada de sistema *exec* para iniciar o MTV com o parâmetro que força a leitura a partir da entrada padrão. Após, o CVCC cria um *pipe*, redirecionando a saída do mesmo para a *stdin*, o que proporciona o mecanismo de comunicação entre os dois processos.

- **Algoritmo de Execução**

O cliente segue os seguintes passos:

1. Contata o gerente, requisitando um provedor e uma entrada na estrutura da CVC

2. Se existir provedor disponível, o gerente retorna três identificadores, referentes a: posição ocupada pelo provedor na estrutura (*pro_pos*), entrada alocada para o novo cliente (*cli_pos*) e primeiro bloco a ser lido (*TempBlock*). Se nenhum provedor puder ser usado, o gerente retorna um valor inválido em *TempBlock*

3. Se *TempBlock* for válido, inicializa as threads Recover e ReadUDP, usando pro_pos como índice para o cálculo do endereço IP do fluxo *multicast* a ser recebido, e criando posteriormente a thread Play/Send. Se *TempBlock* for inválido, recebe então o número total de blocos do vídeo

Status	Estado
0	Entrada livre
1	Recebendo blocos iniciais
2	Livre para prover
3	Já provendo
4	Já descartando mas livre para prover
5	Já descartando e provendo
6	Problemas ao enviar fluxo

Tabela 1: Estados dos clientes no sistema

4. Caso *TempBlock* seja diferente de zero, torna-se necessário criar um fluxo do servidor, o que é feito através da inicialização da thread ReadRIO, passando *TempBlock* como parâmetro. Com isso, a thread fica encarregada de suprir o buffer com a seqüência de blocos compreendida entre zero e *TempBlock*-1, considerada um remendo no caso de estar apenas recuperando blocos iniciais perdidos ao aderir a um fluxo já em andamento

5. Quando o nível do buffer atinge minPrefetch, a thread Play/Send é desbloqueada, começando a exibição

6. Contata novamente o gerente, avisando que começou a exibir e que já pode prover

7. Envia os blocos em seqüência para o *pipe*, 4KB por escrita

8. Caso o gerente sinalize a thread Control, passa a enviar blocos para o endereço IP determinado por *cli_pos*

9. Se o contador *not_played* atingir um nível mínimo, a thread Recover é desbloqueada

3.1.4 Gerente da CVC (CVCM)

O gerente mantém uma estrutura com as informações sobre os clientes ativos no sistema. Baseado nestas informações, ele é capaz de incorporar novos clientes de forma que o sistema tenha um desempenho eficiente, evitando ao máximo o acesso ao servidor.

- **Estrutura de Controle**

A estrutura de controle é um vetor cujas entradas vão sendo ocupadas pelos clientes que chegam no sistema. Nela são agrupadas todas as informações necessárias para que o gerente esteja a par do estado (status) do cliente (Tabela 1), dos blocos atualmente armazenados nos buffers, entre outras.

- **Políticas de Escolha**

Baseado nas informações contidas na estrutura, diferentes políticas podem ser utilizadas na escolha de qual cliente se tornará provedor do novo cliente, recém chegado ao sistema.

Diferentemente do protocolo original, o protótipo oferece diferentes possibilidades que influenciam na escolha do provedor. Dentre elas, pode-se optar por sempre criar um novo fluxo *multicast* primeiro, ou seja, toda vez que houver um cliente que ainda não começou a descartar blocos, um novo fluxo *multicast* será criado para atender novos clientes. Em contra partida, pode-se escolher sempre reutilizar fluxos que ainda sejam passíveis de receber remendo. Outras opções se referem a, quando houver mais de um cliente com o mesmo status, qual deles será usado, entre o mais novo e o mais antigo, de acordo com seus tempos de chegada ao sistema.

Nos experimentos efetuados, o gerente foi configurado para realizar a escolha, conforme a Tabela 1, respeitando a seguinte ordem de precedência:

1. Escolhe o mais novo entre os de status igual a 2
2. Escolhe o mais novo entre os de status igual a 3
3. Libera novo fluxo do servidor

Quando ocorre algum problema no recebimento de blocos do provedor, o cliente gera alertas para o gerente requisitando novo provedor. Ao receber o pedido, o gerente busca em sua base de dados um cliente que satisfaça alguma das seguintes condições, pela ordem:

1. Esteja provendo o bloco imediatamente anterior ao último bloco recebido pelo cliente
2. Possua status igual a 2 e tenha o último bloco em seu buffer
3. Possua status igual a 4 e tenha o último bloco em seu buffer

Os alertas continuam a ser emitidos, em intervalos de um segundo, até que um novo provedor seja encontrado ou not_played atinja 3 blocos. Neste segundo caso, o gerente libera um novo fluxo do servidor.

- **Algoritmo de Execução**

O gerente funciona como um servidor que fica a espera de requisições em uma porta pré-definida (6554). Quando uma mensagem é recebida pelo socket UDP, é realizada a atualização das informações que dependem do fator tempo. Após isso, um *switch*[5] faz o desvio da execução para a função correspondente a chamada recebida, dentre as expostas na próxima seção.

3.2 Interação entre o Gerente e os Clientes

Os clientes se comunicam com o gerente através de uma API, com as funções definidas na Tabela 2.

[5]Estrutura de desvio da linguagem C

search	retorna o ip de um cliente que ainda não descartou blocos
update	atualiza as informações da estrutura da CVC
create_entry	aloca uma entrada para o cliente na estrutura
include_provider	avisa ao gerente que exibição começou e que já pode prover
include_client	busca um provedor e aloca entrada
replace_provider	busca um novo provedor
delete_provider	desaloca entrada na estrutura
pause	avisa o gerente que foi feito pause/resume
start_discard	avisa o gerente que começou a descartar

Tabela 2: API do Gerente da CVC

4 Metodologia Experimental

Esta seção descreve os aspectos relacionados ao ambiente experimental e os testes efetuados para analisar o desempenho do protótipo.

Note que, como o protótipo encontra-se em desenvolvimento, alguns aspectos de CVC ainda não foram implementados, como fluxos *multicast* desde o servidor, geração de *multicast* a partir dos clientes mesmo que a parte inicial do vídeo já tenha sido descartada, a figura do colaborador no envio do remendo e funções de avanço e retrocesso. Futuramente, essas funcionalidades permitirão melhorar ainda mais o desempenho do sistema avaliado nesta seção.

4.1 Ambiente Experimental

O ambiente experimental é composto por 8 máquinas, configuradas de acordo com a Tabela 3, interconectadas por um *switch Fast Ethernet* 3COM SuperStack II 3300. Este equipamento trabalha em nível 2, entretanto, está apto a funcionar como filtro para aplicações *multicast*, pois suporta *IGMP Snooping*. Das máquinas citadas, uma executa o servidor de vídeo e o gerente da CVC. Esta máquina possui um disco rígido Ultra2 SCSI IBM Ultrastar 18ES, com capacidade de armazenar 9.1GBytes e taxa de transferência sustentável variando de 12,7 a 20,2 MBytes/s (o que permite afirmar que a interface de rede é o principal gargalo no lado do servidor). Outra é utilizada para gerar a carga e monitorar o estado do sistema. As seis restantes tem a função de executar instâncias dos clientes.

Processador	Intel Pentium III 650MHz
RAM	512 MBytes
NIC	Intel EtherExpress Pro 10/100
SO	Linux 2.2.14-5.0

Tabela 3: Configuração das máquinas

4.2 Carga de Trabalho

A geração da carga de trabalho foi feita simulando a chegada de clientes no sistema, baseada em um Processo de Poisson, com taxas variando entre 0,5 e 120 chegadas/minuto. Para isso, foi desenvolvido um programa gerador de intervalos entre chegadas (em microsegundos) de n clientes segundo tal processo, o qual escreve os tempos respectivos em linhas consecutivas de um arquivo denominado times.dat. Além deste, foi criado o arquivo hosts.dat contendo os endereços das máquinas configuradas para executar os clientes. A partir disto, a máquina geradora de carga segue o seguinte procedimento:

1. Lê um tempo de times.dat
2. Chama *usleep* com o valor lido como parâmetro a fim de gerar o intervalo entre chegadas
3. Passado o tempo, lê de hosts.dat, segundo uma política round-robin, o endereço da máquina onde o cliente deve ser inicializado
4. Faz uma chamada rsh para o endereço lido, iniciando de fato o cliente
5. Volta para o primeiro passo se todos os intervalos ainda não foram lidos

O vídeo adotado nos testes é formado pelos 3550 segundos iniciais do filme Star Wars IV, codificados em MPEG-1 segundo o padrão NTSC (352 x 240, 29,97 quadros/s). Como o objetivo principal é analisar a contribuição da CVC no desempenho do sistema, a utilização de um único vídeo é suficiente.

Através de medições realizadas na exibição do vídeo, identificou-se um comportamento bastante homogêneo (Figura 5) nos tempos de consumo dos blocos por parte do decodificador, com duração média de 0,69 s/bloco.

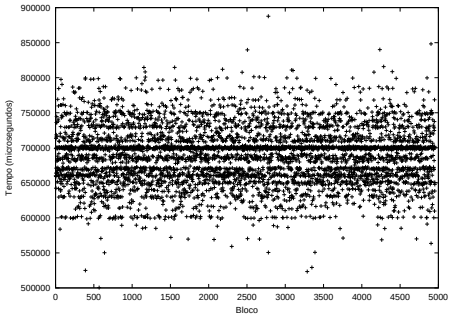

Figura 5: Tempos de consumo dos blocos do vídeo (Tempo Médio = 0,69 s/bloco)

Devido ao programa de decodificação MPEG consumir muita CPU e memória RAM, foi implementado um simulador de decodificador, a fim de liberar recursos indispensáveis para a execução de diversos clientes por máquina. Como o decodificador se comunica com o cliente através de um *pipe*, de onde ele lê 4KBytes por vez, um simulador com comportamento bastante semelhante pôde ser desenvolvido. Com base no tempo médio de consumo dos blocos, o simulador calcula, descontando o *overhead* por ele introduzido, o tempo necessário para exibir

4KBytes de vídeo. Feito isso, ele simula o consumo através de chamadas regulares a *usleep*, passando o tempo calculado como parâmetro.

Os parâmetros utilizados nos experimentos estão resumidos na Tabela 4. Os tamanhos de buffer nela expostos equivalem, respectivamente, a 4, 8 e 16 MBytes.

Duração do Vídeo (s)	3550*
Tempo Médio de Exibição de Bloco (s)	0.69
Tamanhos de Buffer (posições)	32/64/128
Taxas de Chegadas (clientes/min)	0,5 a 120

Tabela 4: Parâmetros dos experimentos (* 7100 s com Taxa de Chegada = 0,5 clientes/min)

4.3 Análise dos Resultados

Nesta seção são discutidos os resultados de desempenho comparativo entre um servidor convencional e o protótipo de servidor com CVC.

Desempenho do Servidor Convencional

Como experimento base, buscou-se determinar a capacidade máxima de atendimento de clientes simultâneos do servidor, isto é, o número de canais lógicos oferecidos pelo mesmo, segundo a abordagem convencional. Inicialmente foi medida a vazão do servidor para atender a um único cliente. O valor obtido foi de 1,7Mbits/s, demonstrando que o servidor possui um *overhead* de 200Kbits/s por fluxo. Executando posteriormente os testes para definir o número total de canais, descobriu-se que o servidor consegue sustentar 56 fluxos simultâneos, com uma ocupação de 96% da banda nominal de 100Mbits. A partir de então, partiu-se para a investigação do efeito da CVC na ocupação do servidor, com 56 clientes simultâneos, dados diferentes tamanhos de buffer e taxas de chegadas. As curvas resultantes destas medições, expostas nas figuras a seguir, foram interpoladas com o método de Bezier [18].

Impacto da Taxa de Chegada de Clientes

A Figura 6 demonstra o efeito da taxa de chegada de clientes na ocupação do servidor estendido com CVC. Note que o melhor desempenho ocorre com buffer de 128 posições e taxas de chegada variando entre 10 e 20 chegadas/min[6]. Nesta configuração, a CVC atinge uma economia de 84% na ocupação dos canais do servidor, visto que apenas 9 são usados.

Quanto menor a taxa de chegada, maior a influência do tamanho do buffer na ocupação do servidor, de forma que quanto maior o buffer, menos canais são usados. No entanto, a economia obtida não varia linearmente com o aumento do buffer, visto que a diferença na ocupação do servidor existente entre as versões do cliente com buffer de 32 e 64 posições é claramente maior do que a medida entre 64 e 128.

Além disso, constata-se que o número de canais ocupados no servidor converge para um certo valor, independentemente do tamanho do buffer. Isto acontece devido a PL, que está fixado

[6]Esses resultados validam os anteriormente simulados em [13]

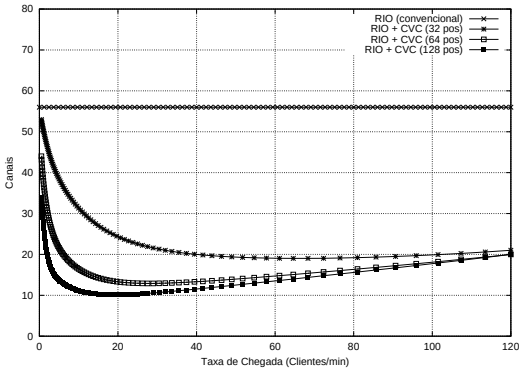

Figura 6: Canais Ocupados no Servidor X Taxa de Chegada

em 16 blocos, equivalentes a aproximadamente 11 segundos. Com taxa de 120 chegadas/min, ou seja, intervalo médio de 0,5 entre chegadas, em torno de 20 clientes são iniciados até que o primeiro possa prover, ocupando pelo menos 20 canais. Ou seja, o **número mínimo de canais ocupados** é igual ao tempo de pré-busca dividido pelo intervalo entre chegadas. Uma possível solução para este problema, diagnosticada durante os testes, consiste em introduzir técnicas de *Batching*, da seguinte forma: caso outros clientes contatem o gerente enquanto o primeiro cliente estiver fazendo a pré-busca, estes são introduzidos no grupo de *multicast* que será provido por ele a partir do momento em que o seu buffer alcançar a ocupação mínima. O único ponto negativo desta abordagem é o aumento na latência de início de exibição dos clientes providos, embora o valor continue extremamente baixo, no máximo 22 segundos.

Outro fator interessante é que, através de testes auxiliares, efetuados com intervalos fixos entre chegadas, foi descoberto que os limites mínimos para a ocorrência de cooperação são 4,6, 1,8 e 0,8 chegadas/min, para os buffers de, respectivamente, 32, 64 e 128 posições. Entretanto, devido ao fato do Processo de Poisson apresentar intervalos entre eventos bastante heterogêneos, ocorre um nível satisfatório de cooperação mesmo com taxas inferiores as descritas acima, o que justifica a economia conseguida nestas condições.

Coeficiente de Cooperação

A Figura 7 mostra a relação existente entre a capacidade do buffer do cliente (em segundos) e o intervalo médio entre chegadas (em segundos), denominada Coeficiente de Cooperação (CC), na ocupação do servidor com CVC.

Teoricamente, um cliente só é capaz de prover conteúdo para um novo cliente que chega ao sistema se o intervalo entre sua chegada e a do novo cliente for menor ou igual a capacidade do seu buffer. Devido a isso, o sistema com CVC tende a se comportar como um sistema convencional quando o CC for menor que 1, e a ocupar apenas o número mínimo de canais para CC maior ou igual a este valor.

Como o Processo de Poisson gera intervalos heterogêneos entre chegadas, a ocupação mínima do servidor é atingida com CC em torno de 15. Entretanto, reduções significativas são alcançadas a partir de CC=4, onde no máximo a metade dos canais do servidor são ocupados. Resta ressaltar que, mesmo com CC em torno de 0,6, pelo menos 10 canais são economizados.

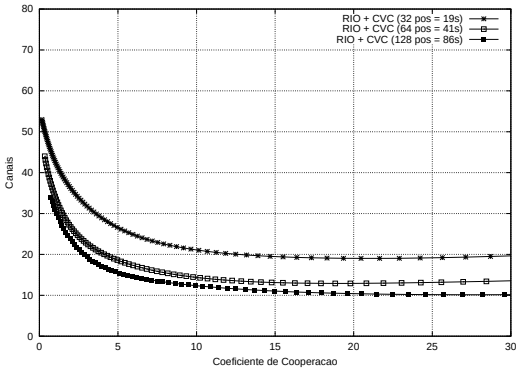

Figura 7: Canais Ocupados no Servidor X Coeficiente de Cooperação

5 Conclusões

Neste trabalho, foram expostos os principais aspectos de implementação e funcionamento do sistema de VoD baseado em Cache de Video Cooperativa (CVC), em desenvolvimento no Laboratório de Computação Paralela (LCP) da COPPE/UFRJ.

Os experimentos preliminares constataram redução de até 84% na ocupação dos canais do servidor, considerando buffers de 16MBytes nos clientes. Além disso, foi comprovado que, mesmo com intervalos médios entre chegadas superiores a capacidade do buffer, um nível significativo de economia é atingido.

Isto posto, pode-se dizer que os resultados obtidos são bastante promissores dado que o protótipo ainda não suporta algumas características da técnica CVC original que possibilitam maior economia. Além disso, possíveis melhorias foram detectadas, capazes de remover a limitação imposta pela pré-busca no número mínimo de canais ocupados no servidor.

Posteriormente, pretende-se também investigar a influência da interação do cliente (Pausa / Recomeço) já implementada, no comportamento do sistema.

6 Agradecimentos

Os autores gostariam de agradecer ao professor Edmundo de Souza e Silva pela disponibilização do servidor RIO e a sua orientada, Adriane Cardozo, pelo suporte oferecido.

Referências

[1] James R. Allen, Blaise L. Heltai, Arthur H. Koenig, Donald F. Snow, and James R. Watson. VCTV: A video-on-demand market test. *AT&T Technical Journal*, 72(1):7–14, 1993.

[2] Gregory R. Andrews. *Concurrent programming: principles and practice*, chapter 4. Benjamin/Cummings, 1991.

[3] David R. Butenhof. *Programming with POSIX Threads*. Addison-Wesley, 1997.

[4] A. Cardoso. Mecanismos para garantir qualidade de serviço de aplicações de vídeo sob demanda. Master's thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2002.

[5] S. Carter and D. Long. Improving video-on-demand server efficiency through stream tapping. In *Sixth International Conference on Computer Communications and Networks*, pages 200–207, 1997.

[6] Asit Dan, Dinkar Sitaram, and Perwez Shahabuddin. Dynamic batching policies for an on-demand video server. *Multimedia Systems*, 4(3):112–121, 1996.

[7] Steve Deering. Host extensions for IP multicasting. *RFC 1112, Network Working Group*, August, 1989.

[8] GLADE GTK+ user interface builder. http://glade.gnome.org.

[9] Leana Golubchik, John C. S. Lui, and Richard R. Muntz. Adaptive Piggybacking: A novel technique for data sharing in video-on-demand storage servers. *Multimedia Systems*, 4(3):140–155, 1996.

[10] GTK+ the gimp toolkit. http://www.gtk.org.

[11] Kien A. Hua, Ying Cai, and Simon Sheu. Patching: A multicast technique for true video-on-demand services. In *ACM Multimedia*, pages 191–200, 1998.

[12] Edison Ishikawa and Claudio Amorim. Cooperative Video Caching for interactive and scalable VoD systems. In *First International Conference on Networking*, volume 2, pages 786–795, 2001.

[13] Edison Ishikawa and Claudio Amorim. Memória Cooperativa Distribuída para sistemas de VoD peer-to-peer. In 19° *Simpósio Brasileiro de Redes de Computadores*, http://www.sbrc2001.ufsc.br/artigos/8184-1834.pdf, 2001.

[14] Jonathan C. L. Liu and David H. C. Du. Continuous media on demand. *IEEE Computer*, 34(9):37–39, 2001.

[15] MpegTV Player. http://www.mpegtv.org.

[16] J.R. Santos and R. Muntz. Design of the RIO (Randomized I/O) storage server. Technical Report 970032, UCLA Computer Science Department, 1997.

[17] Simon Sheu, Kien A. Hua, and Wallapak Tavanapong. Chaining: A generalized batching technique for video-on-demand. In *International Conference on Multimedia Computing and Systems*, pages 110–117, 1997.

[18] T. Williams, C. Kelley, J. Campbell, D. Kotz, and R. Lang. Gnuplot an interactive plotting program, 1990.