Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Sistema Colaborativo de Cache para Stream de Vídeos na Internet

Ronie Rodrigo de Athaydes
Tobias Astoni Sena

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof. MSc. Marcos Fagundes Caetano

Brasília
2015

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Computação — Licenciatura

Coordenador: Prof. Dr. Wilson Henrique Veneziano

Banca examinadora composta por:

Prof. MSc. Marcos Fagundes Caetano (Orientador) — CIC/UnB
Prof. Dr. Eduardo Adilio Pelinson Alchieri — CIC/UnB
Prof. MSc. João José Costa Gondim — CIC/UnB

CIP — Catalogação Internacional na Publicação

Athaydes, Ronie Rodrigo de.

Um Sistema Colaborativo de Cache para Stream de Vídeos na Internet / Ronie Rodrigo de Athaydes, Tobias Astoni Sena. Brasília : UnB, 2015. 119 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2015.

1. Cache, 2. Cache de Vídeos, 3. Cache Colaborativo, 4. Cache Distribuído, 5. Redes Locais, 6. YouTube

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Sistema Colaborativo de Cache para Stream de Vídeos na Internet

Ronie Rodrigo de Athaydes
Tobias Astoni Sena

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof. MSc. Marcos Fagundes Caetano (Orientador)
CIC/UnB

Prof. Dr. Eduardo Adilio Pelinson Alchieri
CIC/UnB

Prof. MSc. João José Costa Gondim
CIC/UnB

Prof. Dr. Wilson Henrique Veneziano
Coordenador do Curso de Computação — Licenciatura

Brasília, 16 de março de 2015

Dedicatória

Dedico aos meus pais, Fabiana Rosa de França e Luiz Carlos Athaydes, à minha irmã Juliana Cristina de França Athaydes e ao meu segundo pai Milton Giacomini Pagliuso Filho pelo apoio ao longo desta difícil jornada.

Ao meu primo Rafael de França Menezes pelo incentivo na mudança para esta cidade e à minha tia Flaviana Rosa de França Menezes pela hospitalidade e abrigo desde que aqui cheguei.

À Vitória Martins da Silva pela paciência, carinho e incentivo durante todo o tempo que dediquei a este trabalho.

À toda minha família.

Ronie Rodrigo de Athaydes

Dedico aos meus pais, Sebastiana Angela Astoni Sena e Francisco de Sena Campos, pelo apoio incondicional durante toda essa jornada acadêmica.

À toda minha família, por sempre apoiar meus estudos ao longo desses anos.

À Mykaella Sales Sousa, pelo carinho e paciência durante todo esse período.

Tobias Astoni Sena

Agradecimentos

Agradecemos, primeiramente, ao nosso orientador Marcos Fagundes Caetano, pelo tempo dedicado, pela orientação e pelo conhecimento transmitido durante a concepção deste projeto.

Aos professores Eduardo Adilio Pelinson Alchieri e João José Costa Gondim pela disponibilidade e interesse em fazer parte da banca avaliadora desta monografia.

À Universidade de Brasília pela qualidade de ensino e pela estrutura que nos proporcionaram chegar até aqui, especialmente aos responsáveis pelo COMNET, que nos forneceram um excelente espaço de estudo.

Aos amigos que estivem sempre presentes durante esta longa jornada, especialmente aos amigos Luciano e Matheus, que colaboraram diretamente na construção deste trabalho.

Resumo

Atualmente, uma parcela significativa do tráfego IP é proveniente de *stream* de vídeos. Um estudo realizado pela Cisco aponta que, em 2018, 79% de todo o tráfego IP irá transportar aplicações de *stream* de vídeos [13]. Como diversos usuários, dentro de uma rede local, podem possuir interesse nos mesmos vídeos, é gerada uma repetição desnecessária de requisições que saem da rede.

O uso de *cache* mostra-se como uma abordagem para diminuir a quantidade de dados transferidos de forma repetida, diminuir a taxa de uso da largura de banda de saída da rede, aumentar o uso da infra-estrutura de rede interna e diminuir a latência na recuperação dos vídeos.

Este trabalho apresenta os conceitos básicos de um sistema de *cache* e faz uma breve revisão da literatura com base nos trabalhos correlatos. Diferentemente dos modelos referenciados, que propõem uma abordagem centralizada, este trabalho propõe um modelo de *cache* colaborativo distribuído para *stream* de vídeos, que tem como objetivo reduzir o consumo da banda de saída da rede. Este modelo utiliza políticas colaborativas que permitem o compartilhamento do *cache* de cada usuário entre os demais membros da rede.

Os resultados apresentados por este trabalho mostram que a implementação do modelo proposto é uma alternativa viável e eficaz para economizar a banda de saída da rede e melhorar a utilização de seus recursos.

Palavras-chave: Cache, Cache de Vídeos, Cache Colaborativo, Cache Distribuído, Redes Locais, YouTube

Abstract

Currently, a significant portion of IP traffic comes from stream videos. A study by Cisco points out that in 2018, 79% of all IP traffic will be video stream applications [13]. As many users, inside a local network, may have interest in the same videos, an unnecessary repetition of outbound requests is made.

Caching has proven itself as an approach to reduce the amount of repeated data, decreasing output bandwidth, increasing internal network infrastructure and reducing latency in video recovery.

This work presents the basic concepts of a caching system and a brief literature review on the basis of related work. Unlike the referenced models that propose a centralized approach, here a collaborative cache distributed model to stream videos is proposed. This aims to reduce the amount of outgoing network bandwidth. This model uses collaborative policies that allow the cache sharing between all users of the network.

The results presented in this work show that the implementation of the proposed model is a viable and effective alternative to save outgoing network bandwidth and improve resources use.

Keywords: Cache, Videos Cache, Collaborative Cache, Distributed Cache, Local Area Networks, YouTube

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

API *Application Program Interface*. 41

CDN *Content Delivery Network*. 19

CE *Caching Elements*. 13

DNS *Domain Name System*. 19

FIFO *First-in, first-out*. 7, 8

HD *Hard Disk*. 26

HTML *HyperText Markup Language*. 16, 19, 20, 33

HTTP *Hypertext Transfer Protocol*. 11, 12, 21, 23, 41

IP *Internet Protocol*. 1, 15, 19, 46

LFU *Least Frequently Used*. 7–9

LRU *Least Recently Used*. 7, 8

PC *Personal Computer*. 23

RAM *Random Access Memory*. 6

RAND *Random*. 7, 9

SRB *Shared Running Buffer*. 16

TCP *Transmission Control Protocol*. 46

TTL *Time To Live*. 9–12

UnB Universidade de Brasília. 1, 18, 20, 21, 23, 47

URL *Uniform Resource Locator*. 19–21, 27, 29, 41

Capítulo 1

Introdução

Nas últimas décadas, os arquivos multimídia se tornaram grandes responsáveis pelo tráfego de dados na Internet [33]. Porém, nos últimos 15 anos, especificamente os vídeos têm assumido o papel de liderança nesse quesito. Atualmente, 66% de todo o tráfego IP (do inglês, *Internet Protocol*) é proveniente de arquivos de vídeos [12]. Esse valor é fortemente influenciado pelas redes sociais, pelo aumento do número de usuários na Internet e pelo aumento da disponibilidade de banda de rede ao consumidor final. A alta demanda por *stream* de vídeos de *sites* como o YouTube, tornou-se reponsável por grande parte do tráfego de dados em redes de médio e grande porte. A exemplo da rede da UnB, onde 64% do tráfego da borda da rede é proveniente de `octet-stream` e vídeos. Os estudos [6] [18] [45] mostram que uma parcela significativa deste tráfego é proveniente do *stream* dos mesmos vídeos, gerando, assim, um desperdício da infraestrutura da rede. Neste cenário, uma medida para racionalizar o uso da rede torna-se necessária.

Este trabalho propõe um modelo de *cache* colaborativo como alternativa para um melhor uso da infraestrutura da rede. Para mostrar sua viabilidade, é implementado um sistema de *cache* que funciona de forma distribuída e transparente para o usuário.

1.1 Problema

A maior parte do tráfego na borda das redes é proveniente de *stream* de vídeos. Tendo em vista que parte destas requisições são redundantes, há um desperdício na utilização dos recursos de rede, que podem ser melhor aproveitados para evitar que a crescente demanda sobrecarregue a largura de banda de saída destas redes.

1.2 Hipótese

O uso de uma política colaborativa de *cache* para vídeos pode reduzir a quantidade de requisições enviadas para fora de uma rede e com isso diminuir a taxa de uso da largura de banda externa.

1.3 Objetivos

1.3.1 Objetivo geral

O principal objetivo deste trabalho é aumentar a taxa de uso da rede local e reduzir a taxa de uso da largura de banda de saída da rede, em relação ao tráfego de vídeos, através de um sistema de *cache* colaborativo.

1.3.2 Objetivos específicos

Os seguintes objetivos específicos foram listados como meios para alcançar o objetivo principal deste trabalho:

- Estudar protocolos de *cache* de vídeos;
- Estudar o funcionamento de protocolos colaborativo de *cache*;
- Estudar os problemas envolvidos no *cache* de vídeos;
- Estudar os problemas envolvidos na criação e gerenciamento de um grupo colaborativo na rede;
- Entender o funcionamento do sistema de requisição e recuperação de vídeos do YouTube; e
- Modelar e implementar uma solução colaborativa para *cache* de vídeos do YouTube.

1.4 Metodologia

A metodologia utilizada para a realização deste trabalho tem como base as seguintes etapas:

1. Estudo de artigos científicos sobre *cache*, *cache* de vídeos e *cache* colaborativo;
2. Estudo de políticas de substituição de *cache*;
3. Estudo de algorítmos de gerenciamento de grupos colaborativos em uma rede local;
4. Estudo e análise do funcionamento do modelo de requisição e recuperação de vídeos do Youtube;
5. Modelagem e implementação de um sistema de *cache* colaborativo específico para vídeos do YoutTube de forma interativa e incremental, seguindo as etapas:
   (a) Software com a função de um *proxy*, capaz de intermediar as requisições de um navegador para vídeos para o YouTube;
   (b) Software capaz de armazenar os vídeos visualizados e serví-los futuramente a medida que forem requisitados novamente, semelhante a um *cache* local;
   (c) Software que implementa uma política de substituiçao, a fim de otimizar o uso do espaço destinado ao *cache*;

(d) Software capaz de compartilhar o *cache* local de vídeos colaborativamente com os demais usuários de um mesmo grupo na rede;

6. Testes e avaliações da eficiência do modelo proposto;
7. Escrita da monografia de graduação; e
8. Defesa da monografia para a banca avaliadora.

1.5 Estrutura do trabalho

Esta monografia está dividida em três partes, sendo a primeira uma apresentação do referencial teórico em relação a *cache*, *cache* de vídeos e *cache* colaborativo. A segunda parte tem como objetivo apresentar o modelo proposto, o sistema implementado para verificar sua viabilidade, o ambiente de testes criado para a simulação e analisar os resultados obtidos. E, por último, a terceira parte abrange a conclusão e os possíveis trabalhos futuros. Deste modo, a monografia foi dividida em 6 capítulos, como segue:

Capítulo 1
Faz uma breve introdução ao tema, apresentando o problema, a hipótese, os objetivos e a metodologia;

Capítulo 2
Apresenta os conceitos básicos de *cache*, alguns dos algorítmos mais conhecidos para políticas de substituição e introduz o conceito de *cache* colaborativo;

Capítulo 3
Apresenta trabalhos na área de *cache* de vídeos e *cache* colaborativo que foram utilizados como embasamento teórico, além de explicar o funcionamento do YouTube, plataforma escolhida como estudo de caso;

Capítulo 4
Apresenta o modelo de *cache* colaborativo proposto e detalha o funcionamento do sistema implementado para testar sua viabilidade;

Capítulo 5
Descreve o ambiente de simulação, os testes que foram realizados e faz uma análise dos resultados obtidos; e

Capítulo 6
Conclui o trabalho e apresenta algumas sugestões para trabalhos futuros.

Capítulo 2

Sistema de Cache

Este capítulo faz uma breve revisão teórica dos principais elementos e conceitos básicos necessários para a compreensão dessa monografia. Alguns cenários serão criados para ilustrar os conceitos e fundamentar as definições propostas. A Seção 2.1 apresenta os conceitos básicos relativos às técnicas de *caching*. A Seção 2.2 descreve os principais algorítmos de substituição, bem como seu funcionamento, vantagens e desvantagens. A Seção 2.3 discorre sobre a consistência do *cache*. A Seção 2.4 aborda a aplicação de sistema de *cache* no contexto da *Web*. E, por fim, a Seção 2.5 apresenta o modelo de *cache* colaborativo e suas vantagens.

Para ilustrar o funcionamento de um sistema de *cache*, suponha que um usuário, chamado cliente, esteja navegando na *Web*. O seu navegador encontra-se requisitando diversos objetos armazenados em bases de dados de máquinas remotas, chamadas servidores. Essa arquitetura é conhecida na literatura como **cliente-servidor** [40]. Suponha, ainda, que a aplicação do cliente requisitou os objetos A, B, A, C, A, nesta ordem, conforme a Figura 2.1.

Neste caso, foram realizadas cinco requisições para o servidor, das quais três foram para buscar o mesmo objeto A. Uma forma de evitar a repetição de requisições desnecessárias é armazenar uma cópia desse objeto no próprio cliente. Mantendo o objeto mais proximo do cliente, em requisições futuras, será mais vantajoso buscá-lo localmente do que no servidor. Esse local disponível para armazenamento dos dados é chamado de *cache*.

Agora, supondo que o dispositivo da Figura 2.1 possua uma área de *cache* local, repetindo a sequência de requisições do cenário anterior (A, B, A, C e A), traria um resultado mais vantajoso, pois duas requisições seriam atendidas pelo *cache*, conforme a Figura 2.2.

Nesse exemplo, a utilização de um *cache* local no cliente dispensou duas requisições do dado A ao servidor. Essa dispensa gerou uma economia de banda para buscar esse dado no servidor e, ainda, uma economia de tempo para transferência do objeto, visto que é mais rápido recuperá-lo localmente. Em contrapartida, foi necessário alocar um espaço no dispositivo do cliente para armazenar o objeto e houve um gasto de tempo para verificar, a cada requisição, se o objeto encontrava-se no *cache* local.

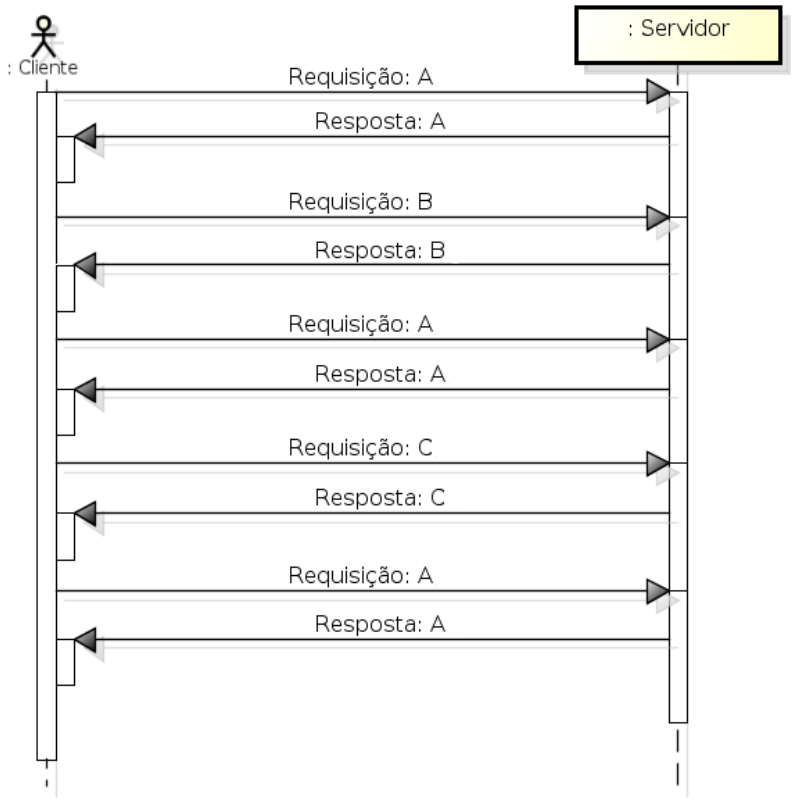

Figura 2.1: Sequência de requisições para um servidor.

2.1 Conceitos Básicos em um Sistema de *Cache*

Nesta seção serão apresentados alguns conceitos fundamentais para compreensão de um sistema de *cache*. Além de apresentar os conceitos básicos, acerto de *cache* e falha de *cache*, serão detalhadas as características da falha e o que é feito quando o espaço de armazenamento do *cache* é totalmente preenchido.

2.1.1 Acerto de *Cache* e Falha de *Cache*

Existem dois conceitos que são basilares em um sistema de *cache*: o acerto de *cache* e a falha de *cache*. Quando um determinado objeto é requisitado pela aplicação do dispositivo, e ele encontra-se armazenado no *cache*, ocorre um acerto de *cache*. De modo contrário, se o objeto requisitado não está armazenado, há uma falha de *cache*. Na Figura 2.2 estão presentes essas duas situações. Quando ocorre uma requisição de um objeto, e o dispositivo não o encontra no seu *cache*, tem-se uma falha de *cache*. Agora, nas duas oportunidades que o objeto A já estava no *cache* e não foi necessário buscá-lo no servidor, bastando recuperá-lo localmente, tem-se um exemplo acerto de *cache*.

2.1.2 Características de uma Falha de *Cache*

Embora os conceitos de acerto de *cache* e falha de *cache* sejam bem triviais, eles são a essência de todo sistema de *caching*. Uma boa política de *cache* tem como objetivo

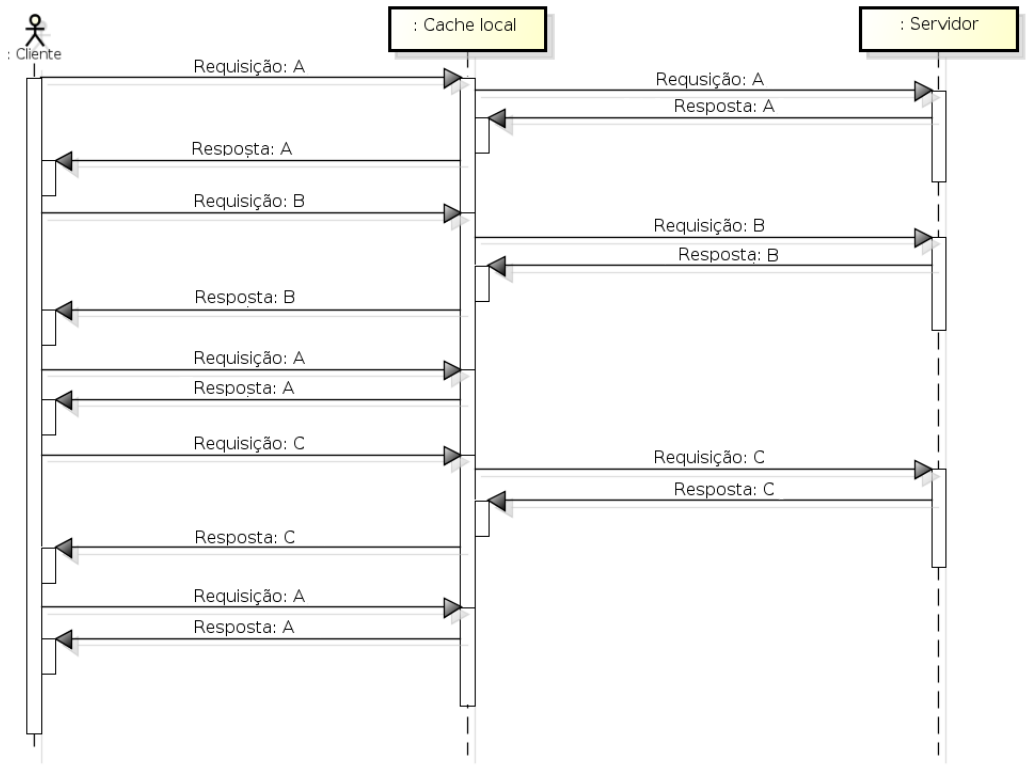

Figura 2.2: Sequência de requisições para um servidor, tendo um *cache* no dispositivo.

evitar, na medida do possível, uma falha de *cache*, isso porque essa operação pode ser extremamente custosa. Os processadores, por exemplo, mantêm um espaço interno dedicado ao *caching*. O tempo de acesso à esse espaço é da ordem de 2ns e, caso ocorra uma falha, será necessário acionar a memória RAM (do inglês, *Random Access Memory*), cujo o tempo de acesso é da ordem de 10ns (quatro vezes maior) [39].

Já no uso de *cache* na Internet, uma falha representa uma requição enviada ao servidor, onde o tempo de acesso pode ser da ordem de segundos. Neste caso, uma sequência de falhas do *cache* pode afetar o desempenho da aplicação e gerar um prejuízo à experiência do usuário que a esteja utilizando. Portanto, é necessário evitar ao máximo uma falha de *cache*.

2.1.3 Substituição do *Cache*

Na figura 2.2, o dispositivo conseguiu armazenar todos os objetos únicos que foram requisitados, ficando, assim, disponíveis para futuras requisições. No entanto, um dos principais problemas enfrentados por um sistema de *cache* é a disponibilidade de espaço físico para armazenamento de objetos que poderão ser utilizados no futuro. No contexto de *cache* na *Web*, não há como um computador armazenar todas as páginas da Internet, pois estima-se que a Internet ultrapasse os 600 EB de conteúdo disponível [22].

Quando a área total destinada para o armazenamento de objetos encontra-se completamente utilizada, e há a necessidade de armazenar um novo objeto, é necessário escolher uma vítima para ser substituída. Esse processo de escolha de uma vítima adequada para a

substituição é alvo de diversos estudos na área computacional. Diversos algorítmos foram propostos para essa finalidade e, a seguir, alguns deles serão apresentados.

2.2 Algorítmos de Substituição

São inúmeros os algorítmos propostos para a substituição dos objetos no *cache* [32], tendo em vista que a escolha de uma vítima ideal não é uma tarefa simples, devido as peculiaridades de cada sistema de *cache*. Não existem algorítmo perfeito, ou seja, aquele que consegue prever todos os objetos que serão utilizados no futuro. Cada proposta foi pensada para otimizar um ponto específico e se encaixa melhor em determinada situação. As políticas existentes levam em consideração várias características dos objetos, para determinar quais serão descartados para dar lugar aos novos que chegarão [25], dentre elas:

Temporalidade
O tempo decorrido desde a última referência ao objeto.

Frequência
O número de requisições ao objeto.

Tamanho
O tamanho do objeto.

Custo de recuperação
O custo para recuperar o objeto do seu servidor de origem.

Tempo de modificação
O tempo decorrido desde a última modificação do objeto.

Tempo de expiração
O momento em que o objeto se torna desatualizado e passível de substituição.

Essas métricas podem ser utilizadas no processo de escolha da vítima para substituição. Muitas das propostas presentes na literatura utilizam as quatro primeiras [32]. A seguir, serão descritos alguns dos principais algorítmos de substituição que se apoiam nessas características [32] [35]: FIFO (do inglês, *First-in, first-out*), LRU (do inglês, *Least Recently Used*), LFU (do inglês, *Least Frequently Used*) e o algorítmo RAND (do inglês, *Random*).

2.2.1 FIFO

O algorítmo de substituição FIFO leva em consideração o momento de entrada dos objetos no *cache*, no qual o primeiro elemento a entrar será o primeiro a ser escolhido como vítima no momento em que ocorrer uma substituição. Na Tabela 2.1, temos a evolução de um *cache* de tamanho dois com a sequência de requisições A, B, A, C, A, B e C, onde o símbolo * indica qual objeto está no *cache* há mais tempo. Observa-se que na requisição 3, A é recuperado do *cache*. Em 4, C é obtido do servidor, e precisa ser armazenado em

Requisições	A	B	A	C	A	B	C
Evolução do Cache	A*	A*	A*	C	C*	B	B*
		B	B	B*	A	A*	C
	1	2	3	4	5	6	7

Tabela 2.1: Algoritmo de substituição FIFO aplicado em um *cache* de tamanho dois a partir da sequência de requisições: A, B, A, C, A, B e C.

cache. A política FIFO irá selecionar A como vítima, pois esteve armazenado por mais tempo, desde a requisição 1.

A vantagem desse algoritmo é justamente a baixa complexidade para gerenciá-lo, exigindo menos processamento e melhorando o tempo de resposta. Dessa forma, o objeto selecionado como vítima estará sempre no início da fila, facilitando o processo de substituição. Apesar desse algoritmo ser intuitivo, na prática a taxa de acerto de *cache* é menos satisfatória que o LRU [11], sendo assim, pouco utilizado em sua forma mais pura.

2.2.2 LRU

O algorítmo LRU é uma das políticas mais utilizadas em diversas áreas do gerenciamento de *cache* [35]. A característica de temporalidade do objeto é utilizada no momento da escolha da vítima para substituição, onde os objetos menos usados recentemente são os primeiros a serem descartados. Na Tabela 2.2, temos a evulução de um *cache* de tamanho dois com a sequência de requisições A, B, A, C, A, B e C, onde o símbolo ***** indica qual objeto foi referenciado há mais tempo. Observa-se que na requisição 3, A é recuperado do *cache* e tem o seu tempo de referência atualizado. Assim, quando C é requisitado posteriormente, B é selecionado como vítima pelo algorítmo LRU, ao contrário do que foi visto acima, na política FIFO.

Requisições	A	B	A	C	A	B	C
Evolução do Cache	A*	A*	A	A*	A	A*	C
		B	B*	C	C*	B	B*
	1	2	3	4	5	6	7

Tabela 2.2: Algoritmo de substituição LRU aplicado em um *cache* de tamanho dois a partir da sequência de requisições: A, B, A, C, A, B e C.

2.2.3 LFU

O algoritmo LFU descarta os objetos com a menor frequência de uso [35]. A implementação deste algoritmo consiste em atribuir um contador para cada objeto que é armazenado no *cache*. Cada vez que é feita uma referência ao objeto, o contador é incrementado por um. Quando houver a necessidade de substituição, o objeto com o menor valor em seu contador será escolhido como vítima e removido do *cache*. Na Tabela 2.3, temos a evulução de um *cache* de tamanho dois com a sequência de requisições A, B, A, C, A, B e C, onde o número sobrescrito indica quantas vezes o objeto foi referenciado. Observa-se nessa política que toda vez que um objeto é requisitado, e encontra-se

em *cache*, seu contador de referências é atualizado. Assim, ao chegar na requisição 7, A continha 3 referências, contra apenas 1 de B, sendo este, portanto, escolhido como vítima pelo LFU.

Requisições	A	B	A	C	A	B	C
Evolução do Cache	A^1	A^1 B^1	A^2 B^1	A^2 C^1	A^3 C^1	A^3 B^1	A^3 C^1
	1	2	3	4	5	6	7

Tabela 2.3: Algoritmo de substituição LFU aplicado em um *cache* de tamanho dois a partir da sequência de requisições: A, B, A, C, A, B e C.

2.2.4 LFU-*Aging*

Se um determinado objeto for requisitado inúmeras vezes num curto período de tempo, e posteriormente ficar sem ser acessado, ele poderá continuar permanentemente no *cache* caso a política LFU esteja sendo aplicada. O algorítmo LFU-*Aging* tenta resolver essa situação, adicionando um parâmetro limiar, de valor arbritrário. Quando a média das frequências de visualização ultrapassar o valor estipulado no parâmetro limiar, todas as frequências dos objetos em *cache* são divididas por dois, evitando que um determinado objeto não mais utilizado fique perpetuado no *cache* [35].

2.2.5 RAND

O algoritmo de substituição RAND adota uma política simples, que não exige qualquer parâmetro adicional para realizar a escolha da vítima, a qual é selecionada aleatoriamente.

2.3 Consistência do *cache*

Como foi visto, existem diversos benefícios associados ao uso de *cache*. Em contrapartida, a utilização de um *cache* pode gerar inconsistências entre os objetos armazenados e os originais. Isso ocorre se o objeto original for modificado depois que as cópias forem recuperadas, o que as torna imediatamente desatualizadas. Esse problema é conhecido como inconsistência ou incoerência do *cache* [30]. Um solução inicial seria verificar a cada requisição se o objeto foi atualizado, porém essa é uma operação custosa, devido principalmente ao aumento do uso da banda, carga no servidor e tempo de resposta (latência). Então, o problema consiste na implementação de um mecanismo que traga perdas mínimas de performance [37]. Para isso, existem algumas técnicas de verificação de consistência, duas delas destacadas a seguir [20] [30].

2.3.1 TTL

O TTL (do inglês, *Time To Live*) é uma estimativa de tempo de vida de um dado, para determinar por quanto tempo ele permanecerá atualizado no *cache*. Quando há uma requisição de um objeto, e seu tempo não expirou, ele é fornecido sem a necessidade de

consultar o servidor de origem para determinar sua consistência. Por outro lado, se o tempo de vida do dado expirou, é feita uma requisição ao servidor de origem. Caso o objeto tenha sido modificado, é retornado o dado atualizado, e, caso ainda não o tenha, é apenas informado para o *cache* um status atualizando o tempo de vida. O grande desafio na utilização do TTL é determinar um valor apropriado de tempo de expiração. Normalmente, é atribuído um intervalo relativamente curto, o que leva a algumas atualizações desnecessárias do objeto. O campo TTL é muito útil em informações com tempo de vida conhecido, como por exemplo jornais *online* que são atualizados diariamente.

2.3.2 *Polling* de Cliente

É uma técnica em que os clientes verificam periodicamente se um objeto no *cache* ainda está válido. É baseado nas suposições de que os objetos novos são modificados com mais frequência do que os objetos antigos. O *cache* associa um campo TTL que é expresso por uma porcentagem, conhecida como limiar. Um objeto se torna inválido quando o tempo desde a última validação excede o tempo do limiar de atualização do objeto. Por exemplo, considerando um dado armazenado em *cache* cuja idade é de 30 dias, e que sua validade foi testada na data de ontem. Se o limiar estiver configurado para 10%, então o objeto deve ser marcado como inválido após 3 dias. Assim, caso haja uma requisição desse objeto nos próximos 3 dias, o dado será recuperado do *cache* sem a necessidade de verificação com o servidor de origem. Após essa data, o objeto será marcado como inválido e a próxima requisição será buscada do servidor de origem com a consequente atualização do dado no *cache*.

2.3.3 Protocolo de Invalidação pelo Servidor

Essa técnica é utilizada quando há a necessidade de manter uma consistência mais precisa no *cache*. Nesse método, cada vez que o objeto é atualizado, o servidor de origem notifica os *caches* que suas cópias se tornaram inválidas. A desvantagem dessa abordagem é o custo computacional envolvido. O servidor precisa guardar um registro de todos os *caches* que armazenam seus objetos e saber tratar com clientes (*caches*) que estiverem indisponíveis, pois caso não consiga ser notificado de imediato o servidor deve continuar tentando, uma vez que o *cache* só conseguirá invalidar o objeto após ser notificado pelo servidor. Esse método requer alterações no servidor, enquanto os outros citados acima podem ser implementados diretamente no cliente.

2.3.4 Coerência Fraca e Forte

Caso o *cache* determine um procedimento heurístico para decidir se a resposta em *cache* ainda é válida, sem consultar o servidor de origem quando haja um acerto de *cache*, essa diretriz é chamada de coerência fraca [25]. De modo contrário, se toda vez que há um acerto do *cache* o dispositivo realiza um pedido de revalidação no servidor de origem, ou o servidor notifica o cliente a cada atualização do objeto, então essa diretriz é chamada de coerência forte. Dependendo de cada uso do *cache*, e do conjunto típico de respostas em *cache*, uma técnica pode ser mais apropriada que outra. Nas abordagens

mencionadas, TTL e *Polling* de Cliente são exemplos de técnicas de coerência fraca, enquanto o Protocolo de Invalidação pelo Servidor é um exemplo de coerência forte.

2.4 *Web Caching*

Utilizar *Web caching* significa armazenar cópias de documentos muito acessados em locais mais próximos do cliente, o que diminui a carga na rede, como as requisições feitas a servidores que possuam dados muito populares [30]. O crescimento exponencial do uso da *Web* resultou no aumento do uso de banda da rede, gerando uma sobrecarga em sua capacidade. Fez, ainda, com que páginas *Web* consideradas populares, devido a grande quantidade de requisições, tivessem dificuldades de ser entregues. Essa combinação levou ao crescimento do tempo de latência para entrega de dados. Assim, realizar o *cache* na *Web* ajuda a aliviar esses problemas [28]. Em termos de redes institucionais de grande porte, como uma universidade, por exemplo, utilizar o *Web caching* reduz o tráfego na rede e, consequentemente, a necessidade de realizar melhorias constantes em sua infraestrutura [26]. O uso de técnicas de *caching* tem sido bem aceito como método viável para [1]:

- Aliviar as crescentes necessidades de largura de banda;
- Aumentar a velocidade da entrega de dados;
- Reduzir a carga do servidor; e
- Diminuir a latência de acesso do cliente.

Existem várias abordagens de *cache* na Web, das quais as mais utilizadas são: *Cache* local no cliente e *Proxy Caching* (muito utilizado nos roteadores de saída de grandes redes institucionais) [45].

2.4.1 *Cache* local no cliente

O *cache* local no cliente é realizado no próprio dispositivo do usuário. Uma das formas mais praticadas é a implementada pelos navegadores. Estes armazenam algumas informações de páginas que foram visitadas pelo dispositivo, de modo que carreguem mais rapidamente em visitas futuras. Quando um servidor retorna uma resposta para o navegador, ele também emite alguns cabeçalhos HTTP (do inglês, *Hypertext Transfer Protocol*), que descrevem diversos tipos de parâmetros, entre eles algumas diretrizes de armazenamento em *cache*. Dois conteúdos são fundamentais para o sistema utilizado pelos navegadores, que é o `Cache-Control` (tendo como valor um TTL), e o `Etag` (que possui um valor criptografado).

Quando o usuário faz a requisição de um conteúdo da Internet, o navegador procura em seu *cache* se esse dado encontra-se armazenado. Caso esteja, verifica se ele possui o TTL válido. No caso afirmativo, não será necessário realizar o *download* da informação do servidor da Internet. Em caso negativo, ele poderia simplesmente realizar uma nova solicitação e buscar a informação completa. Todavia esse é um processo ineficiente, pois se o objeto não foi atualizado na origem ele estaria realizando o *download* de uma cópia idêntica a que possui em *cache*. Para resolver esse problema, é utilizado o valor do *Etag*,

que fará uma correspondência com o objeto na origem. Se o procedimento verificar que o objeto não foi alterado, é atualizado o TTL da informação presente no *cache*, e então utilizado sem necessidade de realizar outra transferência do servidor de origem [19].

2.4.2 *Proxy Caching*

Um servidor *proxy* é um programa de aplicação que recebe requisições de objetos de um conjunto de clientes, repassa essas requisições ao servidor apropriado e envia o objeto requisitado de volta ao cliente [16]. Como ele faz isso de forma transparente ao cliente que fez a requisição, pode-se dizer que um *proxy* é um computador que se faz passar por um outro computador. Os *proxies*, além de funcionar como tradutores para protocolos desconhecidos [38], podem executar as funções de autenticação, filtragem de conteúdo e *cache* [30].

No contexto de *Web caching*, um servidor *proxy* intercepta as requisições HTTP dos clientes, e se encontra os objetos em *cache*, retorna-os para o usuário. Caso não possua, vai até o servidor requisitado pelo usuário, adquire o objeto, transfere ou não o objeto para o *cache*, e então retorna-o para o usuário. Normalmente, os servidores *proxy* encontram-se localizados na extremidade da rede, de forma que possa servir um grande número de usuários. O uso de *proxy* cache resulta em economia de largura de banda, melhora o tempo de resposta e aumenta a disponibilidade de dados e objetos estáticos *Web* [5].

2.5 *Cache* colaborativo

Como visto, adotar um sistema de *cache* traz diversos benefícios ao usuário, como a diminuição da latência, economia no consumo de largura de banda, redução na carga dos servidores e entre outros. Todavia, um *cache* local individual nem sempre traz os benefícios esperados, pois fica limitado ao espaço físico e uso de apenas um dispositivo, o que reduz as chances de obtenção de um acerto de *cache*. No esquema de *cache* colaborativo, diversos usuários mantêm uma integração de seus *caches*, de forma que um objeto armazenado em um dispositivo pode ser obtido por qualquer usuário do grupo cooperativo. Essa integração cria a ilusão de um único *cache* de tamanho muito maior. Isso traz vantagens, pois antes do dispositivo buscar o objeto no servidor de origem, ele busca na sua rede colaborativa, o que aumenta a chance de encontrar o objeto requisitado [24].

Neste modelo, cada dispositivo mantém suas políticas individuais de gerenciamento do *cache*. Quando ocorre um *cache-miss* local, é feito uma requisição a todos do grupo colaborativo, solicitando o objeto não encontrado localmente. Caso encontre em alguém da rede, este encaminhará o objeto para o dispositivo solicitante. Caso não encontre, ocorre um *cache-miss* global, situação em que a requisição será enviada diretamente ao servidor. Obter um dado entre os vizinhos exige um custo maior do que reavê-lo localmente, entretando o custo pode ser bem menor do que submeter uma requisição ao servidor de origem [7] [10].

A seguir, serão vistos alguns modelos de sistema de *cache* colaborativo.

2.5.1 Modelos Teóricos

Com o intuito de criar um sistema colaborativo de *cache* de vídeos em redes de grandes áreas, Tavanapong *et al.* [41] propuseram o *Video Caching Network*. O propósito do modelo é criar uma estrutura hierarquizada de *cache*, de forma a armazenar os vídeos próximo de quem está consumindo a informação. A estrutura consiste num determinado número de CE (do inglês, *Caching Elements*), assim chamado por eles, que fariam uma rota de entrega do vídeo entre o cliente e o servidor. Esses CEs são softwares executando em computadores com espaços para armazenar o *cache* de vídeos. Tais estruturas conseguem comunicar entre si, formando grupos colaborativos, dependendo da localização do cliente que realizou a requisição. Assim, essa rede com estrutura para formar o *cache* do vídeo estaria mais próxima do usuário, provendo as requisições sem a necessidade de acesso ao servidor e com latência reduzida. O autor mostra algumas desvantagens do sistema de *cache* centralizado para redes de grande área. Primeiramente, há uma latência ocorrida devido à grande quantidade de interações sofridas pelo coordenador central. Segundo, o coordenador pode facilmente tornar-se o gargalo da rede ou ocorrer uma falha que paralisaria o sistema. E por último, a requisição realizada poderia estar distante do coordenador, o que resultaria em aumento do tempo de carregamento. Na proposta apresentada, os grupos de CE são criados dinamicamente, dependendo da localização e comportamento do usuário solicitante. Cada membro do grupo seria capaz de se comunicar com outros dois grupos adjascentes, e rapidamente adaptar-se a novas configurações. Assim, conseguiriam prover de forma mais eficiente às requisições.

No contexto de redes locais, Cobârzan *et al.* [14] propuseram um modelo para criação dinâmica de *proxy-cache*, conforme a demanda de crescimento da rede. Assim, a rede iniciaria com apenas uma máquina recebendo as requisições e realizando as operações de *cache*. Porém, ao atingir o limite de espaço de armazenamento do servidor, ao invés de simplesmente realizar a política de substituição, o sistema pode optar por realizar a criação de um novo servidor *proxy* na rede. Deste modo, passará a dividir as requisições e aumentará o limite de armazenamento de *cache* na rede. Para definir se é mais interessante proceder a criação de um novo servidor *proxy*, ou realizar a política de substituição, o programa realiza alguns cálculos em diversas variáveis pré-determinadas, a fim de tentar manter sempre constante o tempo de resposta, balanceamento da carga, disponibilidade e robustez de todo o sistema. A Figura 2.3 mostra uma rede inicial que teve dois nós transformados em servidor *proxy*.

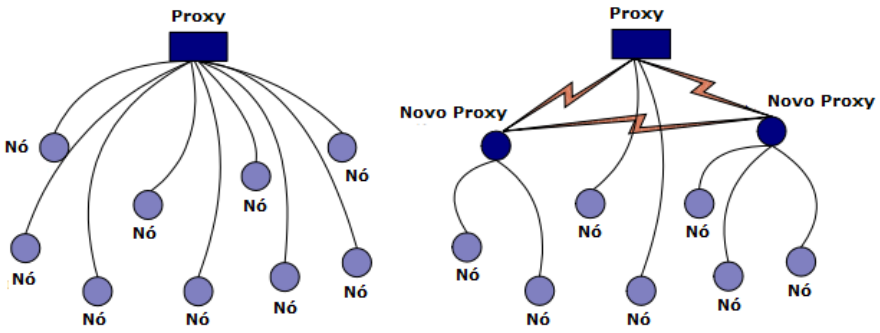

Figura 2.3: Transformação de dois nós da rede em servidores *proxy* [14].

Outra proposta no contexto de redes locais foi realizada por Acharya *et al.* [2]. Eles criaram um sistema de *cache* colaborativo chamado *MiddleMan*. Na solução proposta, cada máquina é responsável por armazenar uma quantidade de *cache*, e um servidor central é responsável por realizar o gerenciamento de todas as requisições e das políticas de substituição da rede. Assim, quando uma máquina solicita um vídeo da Internet, ela manda uma requisição ao computador central, o qual verifica se algum dispositivo da rede tem o vídeo armazenado. Caso encontre, passa o caminho do *cache* do vídeo. Em caso negativo, o cliente realiza a transferência da Internet. Esse modelo apresenta como ponto fraco a alta dependência do computador central, que, caso apresente uma falha, compromete o funcionamento de todo o sistema. A vantagem do modelo é que muitas máquinas possuem um espaço de *cache*, o que aumenta as chances de ocorrer um acerto de *cache* nas requisições realizadas dentro da rede.

2.6 Considerações Finais

Esse capítulo trouxe o conceito de *cache*, algumas definições essenciais e a importância de seu uso. Mostrou que o armazenamento de informações em *cache* reduz a latência das requisições realizadas, diminui a quantidade de carga nos servidores e ajuda a reduzir o consumo da largura de banda da rede. Também foi visto, que a adoção de um sistema de *cache* acarreta decisões sobre políticas de substituição do conteúdo armazenado. Além disso, foi exposto que cada algorítmo de substituição impacta diretamente no desempenho global do sistema, tendo em vista que cada um se encaixa melhor em determinada situação. No próximo capítulo serão abordados o gerenciamento de *cache* específicos para vídeos e o funcionamento de um dos sites de vídeos mais acessados na atualidade, o YouTube [3].

Capítulo 3

Cache de Vídeos

Neste capítulo serão apresentadas algumas particularidades envolvendo o *cache* de vídeos. A Seção 3.1 mostra alguns estudos realizados com a proposta de criação de *cache* de vídeos. A Seção 3.2 apresenta o YouTube, dá uma visão geral do seu funcionamento, de algumas particularidades e mostra a representatividade de seu tráfego em redes universitárias.

Primeiramente, é preciso notar o impacto gerado por vídeos na infraestrutura de rede. Por possuirem um tamanho bem superior aos demais tipos de dados, a transmissão de vídeos na Internet representa um percentual significativo em todo tráfego da rede mundial. Estimativas realizadas pela Cisco System [13] apontam que no ano de 2013, 66% de todo tráfego IP mundial correspondeu a *stream* de vídeos. Estima-se que este valor crescerá para 79% em 2018, sendo o tráfego Global IP estimado em 132 EB de informação por mês. Isso mostra a necessidade de encontrar uma solução para amenizar o impacto dessa grande quantidade de dados e melhorar o uso dos recursos, tendo em vista que a infraestrutura da rede não cresce na mesma taxa que o consumo.

Neste cenário, o uso de políticas de *cache* torna-se um solução viável para melhorar a utilização dos recursos de rede. Porém, arquivos de vídeo, em particular, possuem característica distintas dos demais. Como possuem tamanhos elevados, consomem grande parte do espaço disponível de um *cache*, diminuindo, assim, sua diversidade. Por outro lado, em caso de um acerto de *cache*, haverá uma economia significativa do uso da banda de saída da rede, melhorará o seu tempo de resposta e, ainda, será transparente para o usuário. É importante notar, ainda, que pelo fato dos arquivos de vídeo serem imutáveis, podem ser armazenados em *cache* sem necessidade de uma política de consistência.

A seguir, serão apresentados alguns trabalhos que propuseram um sistema de *cache* para vídeos, tentando aproveitar os benefícios e minimizar a desvantagem citada anteriormente.

3.1 Trabalhos Correlatos

Para a criação do modelo desenvolvido, foi realizada uma ampla pesquisa na literatura e em publicações científicas. O *cache* de vídeos tem sido objeto de estudos nos últimos 20 anos, desde o seu crescimento significativo na Internet. Com o passar dos anos, diferentes abordagens para criação de um sistema de *cache* de vídeo tem sido proposto, devido a dinamicidade das formas de interação com o usuário. A seguir, será mostrado algumas

propostas voltadas para situações de execução do vídeo. Nelas, os autores tentaram otimizar a experiência do usuário durante uma visualização, principalmente na redução do tempo de latência inicial.

Liu [27] pesquisou e classificou diversos algoritmos e políticas de gerenciamento de *cache* em vídeo, destacando as suas características proeminentes e os desafios encontrados por cada uma delas. O fluxo de vídeos na rede possui características singulares que precisam ser trabalhadas em políticas e estruturas de *cache*, que são: tamanho do arquivo, uso intensivo de banda da rede e alta interatividade. Isso faz com que tais mídias necessitem de um tratamento diferenciado em sistemas de *cache*, pois suas características se diferenciam de objetos HTML (do inglês, *HyperText Markup Language*) comuns. De acordo com a porção do vídeo para deixar em *cache*, será mostrado três categorias trazidas pelo autor: *sliding-interval caching, prefix caching e segment caching.*

3.1.1 *Sliding-Interval Caching*

Tewari *et al.* [42] e Chen *et al.* [8] propuseram o algoritmo *Sliding-Interval Caching*. Esse algoritmo trata o *cache* para um intervalo do vídeo quando há duas requisições sequenciais. Desse modo, a primeira requisição busca o objeto diretamente do servidor e, incrementalmente, o armazena no *proxy cache*. A segunda requisição, ao invés de buscar o objeto diretamente no servidor, poderá acessar a porção desejada no *cache*, liberando-o imediatamente após a visualização. Quando duas solicitações chegam num intervalo próximo, haverá a necessidade de pouco armazenamento de *cache*, e a segunda requisição terá todo o objeto buscado diretamente no *proxy*. No caso de vários acessos sequenciais ao vídeo, os intervalos podem ser agrupados de forma que o *cache* será liberado apenas ao final da última requisição. Como mostra a Figura 3.1, `r1` representa a primeira requisição e `r2` a segunda, de forma que o intervalo de vídeo entre elas é armazenado no *cache*. Assim, `r1` encontra-se buscando diretamente do servidor de origem, enquanto `r2` busca do *cache*, que é liberado após sua utilização.

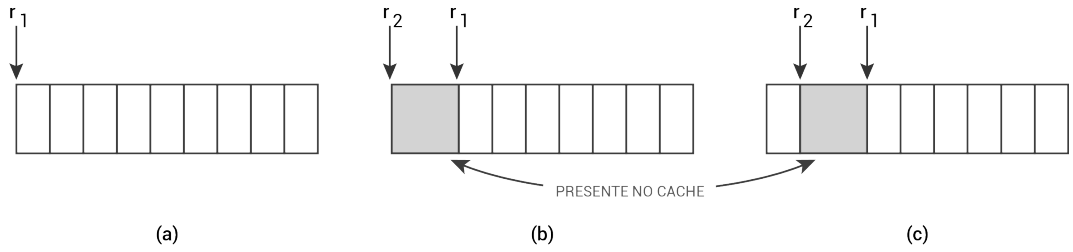

Figura 3.1: Uma ilustração do *Sliding-Interval Caching*. O objeto consiste de nove partes, cada uma sendo entregue pelo *proxy* ao cliente em uma unidade de tempo. `r1` e `r2` chegam no tempo 0 e 2, respectivamente. Para servir `r2`, apenas duas partes precisam estar no *cache* em cada instante de tempo. (a) Tempo 0: `r1` chega; (b) Tempo 1-2: partes 1 e 2 são acessadas por `r1` e armazenadas em *cache*; `r2` chega; (c) Tempo 2-3: parte 3 é acessada por `r1` e armazenada em *cache*; parte 1 é acessada por `r2` e liberada [27].

Devido às constantes quedas nos preços das memórias, e seu consequente aumento nos servidores, Chen *et al.* [9] propuseram o algorítmo SRB (do inglês, *Shared Running Buffer*). Nele, seria possível alocar espaços de leitura e escrita na memória (*buffers*) de

forma a evitar um excessivo uso do disco rígido, tornando o algorítmo mais eficiente. A efetividade desse algorítmo diminui na medida em que aumemnta o intervalo de acesso entre as requisições. Caso o intervalo seja superior ao tempo de exibição do vídeo, o algorítmo acaba realizando o *cache* do objeto por completo.

3.1.2 *Prefix Caching*

Sen *et al.* [36] propuseram um algoritmo para o armazenamento das partes iniciais dos vídeos acessados, o *Prefix Caching*. Havendo uma requisição, a parte já armazenada é imediatamente distribuída ao cliente, enquanto busca do servidor o restante das partes do vídeo. Este algorítmo foi pensado no intuito de reduzir o atraso na reprodução do vídeo, tendo em vista que enquanto o usuário recebe a parte inicial diretamente do servidor *proxy*, há um tempo maior para obter o restante do servidor de origem. Na Figura 3.2 é possível notar que apenas as frações iniciais do vídeo estão armazenadas em *cache*, sendo o restante buscado do servidor.

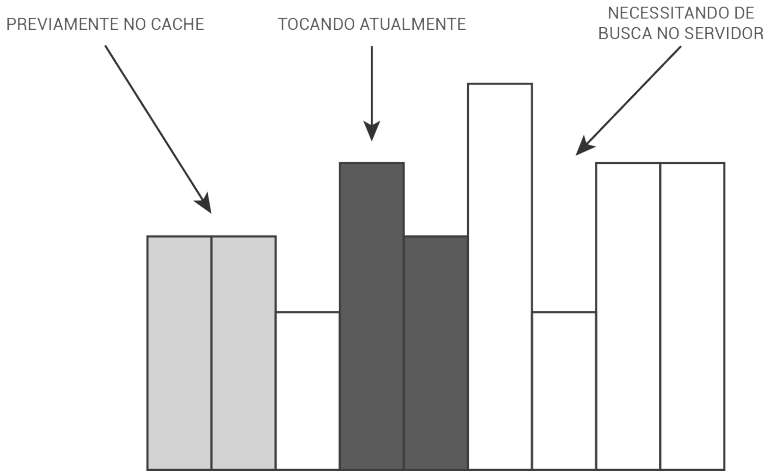

Figura 3.2: Uma ilustração de um instante do funcionamento do *Prefix Caching* [27].

3.1.3 *Segment Caching*

Já Chen *et al.* [8], Wu *et al.* [43], Miao *et al.* [29] e Fabmi *et al* [15] propuseram o algoritmo de *Segment Caching*, no qual o vídeo é subdividido em partes e, as mais importantes, armazenadas em *cache*. A determinação da importância de um segmento é obtida pela relação entre sua frequência de visualização e a distância para o início do vídeo, sendo que quanto mais visto é o vídeo, e quanto mais próximo do início, mais importante é manter esse segmento em *cache*. Uma funcionalidade desse algoritmo é a possibilidade de acesso aleatório, adiantar e retardar o conteúdo nas partes em que estão armazenadas em *cache*. Dessa forma, o usuário pode ter uma visualização geral do vídeo antes de decidir se realmente tem interesse em baixá-lo por completo. Na Figura 3.3 é possível notar que apenas dois segmentos do vídeo encontram-se armazenadas em *cache*.

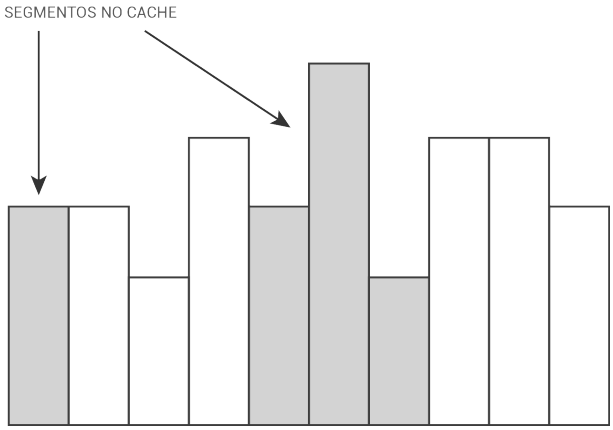

Figura 3.3: Uma ilustração do funcionamento do *Segment Caching* [27].

3.2 Estudo de Caso

O modelo desenvolvido realiza o *cache* colaborativo de vídeos, e como estudo de caso foi escolhido o YouTube. Essa escolha foi decidida tendo em vista a importância atual que o *site* apresenta. Portanto, a seguir será explicado algumas das principais características dessa plataforma, e será mostrado a grande influência que apresenta no tráfego da rede, inclusive da Universidade de Brasília (UnB).

3.2.1 Visão Geral

O YouTube é um site de compartilhamento de vídeos, criado em 2005, que hoje é o 4^o site mais acessado no Brasil e o 3^o mais acessado no mundo, sendo que 3,3% de todo o seu tráfego é proveniente do Brasil [3]. Recebe a cada minuto cerca de 100 horas de vídeos e todo mês é acessado por mais de 1 bilhão de usuários únicos [44]. Devido à sua expressividade, o YouTube tem sido alvo de diversos estudos acadêmicos. Dentre eles, serão apresentados os que possuem mais relevância para este trabalho.

Braun *et al.* [6], monitoraram cerca de 80 mil dispositivos na cidade de Munique, Alemanha, por um mês. Durante o período observado, foram transferidos quase 4 milhões de vídeos, consumindo um pouco mais que 40 TB. Foi constatado que:

- 40% dos vídeos foram requisitados mais de uma vez;
- A maioria deles foram requisitados dez ou menos vezes;
- Alguns vídeos foram vistos centenas ou até milhares de vezes; e
- 70% do tráfego foi gerado por vídeos requisitados mais de uma vez;

Finamore *et al.* [17], em uma pesquisa realizada em campus universitários de dois países da Europa e nos Estados Unidos, relataram que:

- 40% dos vídeos visualizados em computadores pessoais possuem 3 minutos ou menos e 5% deles mais que 10 minutos;

- Entre 40% e 50% dos vídeos visualizados em dispositivos móveis possuem 3 minutos ou menos e 5% deles mais que 10 minutos.

Plissonneau *et al.* [31] trouxeram comparativos entre a qualidade do vídeo requisitado pelo usuário e o tempo de visualização antes de abortá-lo. Segundo os autores:

- 28% dos vídeos com mais de 3 minutos de duração e visualizados com alta resolução foram transferidos por completo;
- 11% dos vídeos visualizados com baixa resolução foram transferidos por completo;
- Na maioria dos vídeos de baixa resolução, a transferência é interrompida antes de completar 20% de seu total; e
- Mais da metade dos vídeos de alta resolução são transferidos por completo e, quando há uma interrupção, não acontece no começo do vídeo.

Na próxima seção será descrito o funcionamento do YouTube para provimento do serviço de visualização de vídeos.

3.2.2 Funcionamento

Para conseguir atender aos acessos globais de forma satisfatória, o YouTube adota a tecnologia CDN (do inglês, *Content Delivery Network*). CDN é uma rede de servidores que armazenam conteúdos e entregam os dados de forma inteligente aos usuários, baseados em sua localidade. Quando um usuário requisita um conteúdo, esta requisição é redirecionada para o servidor CDN mais próximo, que procederá o envio dos dados [21].

Ao submeter uma requisição para um vídeo do YouTube, primeiramente o usuário recebe o HTML da página, os conteúdos para sua construção e o *player* responsável pela reprodução do vídeo na página. Nessa hora, recebe também o nome dos servidores que contêm o arquivo de vídeo solicitado. Feito isso, o navegador do usuário solicita o IP do servidor através do DNS (do inglês, *Domain Name System*), o qual será direcionado ao servidor mais próximos e cujo o tempo de acesso possuir a menor latência. Por último, irá receber os arquivos de vídeos e visualizá-los no *player*. A transferência de um vídeo é realizada através de várias requisições. Cada uma das requisições corresponde à um trecho do vídeo, que é delimitado pelo parâmetro `range`.

3.2.3 Parâmetros das URLs

Os vídeos do YouTube possuem um identificador único, `video_id`, que pode ser acessado diretamente pela URL (do inglês, *Uniform Resource Locator*). O acesso é feito da seguinte forma:

```
http://www.youtube.com/watch?v=<video_id>
```

Durante as requisições de vídeo, Ameigeiras *et al.* [4] identificaram que as URLs eram compostas com a expressão `videoplayback`, tendo a seguinte sintaxe

```
http://<servidor>.youtube.com/videoplayback
```

Na ocasião, os autores identificaram também outros sete diferentes parâmetros, que serão apresentados a seguir:

1. `sparams`: Lista de parâmetros incluídos na requisição, separados por vírgula;
2. `id`: Identificador único do vídeo;
3. `algorithm`: Algorítmo que o servidor pode utilizar para realizar o fluxo de transferência do vídeo;
4. `factor`: Fator de velocidade, expressa o fator da taxa de codificação de um vídeo;
5. `burst`: Duração do vídeo que o servidor irá enviar para o *buffer* inicial medido em segundos;
6. `begin`: Tempo de início de reprodução expresso em milissegundos; e
7. `itag`: Código de formato de vídeo, equivalente a `fmt` (parâmetro de URL não documentado).

Uma característica que dificulta o *cache* de vídeos do YouTube é o fato de sua URL ser dinâmica. Isso significa que a cada requisição a um determinado vídeo, parâmetros diferentes na URL serão passados ao servidor. Isso dificulta, para os sistemas de *cache*, a identificação de qual vídeo fora requisitado. Porém, na primeira requisição a um vídeo, o YouTube realiza um mapeamento entre o parâmetro `video_id`, que é único, e o parâmetro `cpn`, que muda a cada requisição de vídeo. Nas requisições seguintes, que são destinadas à transferência de trechos do vídeo, não há nenhum parâmetro identificador. Entretanto, com o valor do `cpn`, é possível saber qual vídeo está sendo requisitado, e assim é possível realizar o seu *cache*.

Na seção seguinte, será mostrado o impacto do uso YouTube nas redes de algumas unidersidades, bem como na UnB, objetivando entender o comportamento dos usuários neste contexto.

3.2.4 O Uso do YouTube em Universidades

Alguns estudos de caso foram realizados sobre o uso do YouTube em universidades. Serão referenciados três trabalhos, cada qual mostrando diversas características do volume de dados trafegados nas redes dessas universidades. Dessa forma, ficará evidente a representatividade do YouTube em relação ao tráfego total das redes em questão.

Gill *et al.* [18], em uma pesquisa no campus da Universidade de Calgary, Canadá, entre Janeiro e Abril de 2007, constatou que embora apenas 3% das requisições HTML para o YouTube tenham sido de vídeo, essa quantidade foi responsável por 99% do total de bytes transferidos. Segundo o próprio autor, das requisições observadas, 50% delas já tinham sido previamente visualizadas, ou seja, foi gasto uma grande quantidade de recursos para realizar uma requisição que outrora já fora realizada.

Outra análise sobre o YouTube foi realizada por Zink *et al.* [45], onde através do rastreamento e monitoramento do tráfego da rede do campus da Universidade de Massachusetts, Estados Unidos, descreveram, na perspectiva do cliente, como funcionam as requisições a vídeos do YouTube. Além disso, fizeram um estudo sobre as características desse tráfego e a sua análise constatou que:

- nenhuma forte correlação foi observada entre a popularidade global e a popularidade local dos vídeos;
- nem a escala de tempo, nem população de usuários têm impacto na distribuição local de popularidade dos vídeos; e
- vídeos de interesse local têm uma alta popularidade local.

Karam [23] apresentou em seu trabalho algumas das características da rede da UnB, e fez um monitoramento do tráfego de uma parte do sistema, por um período de quatro dias. A rede dessa Universidade possui aproximadamente 13 mil dispositivos, incluindo os computadores, impressoras, dispositivos móveis e entre outros.

A Figura 3.4 mostra o tráfego na rede durante o período estudado. O autor destacou as seguintes informações a respeito:

- Mais de 64% do tráfego *Web* interceptados referiram-se a `octet-stream` e vídeos;
- Em média 68,5% do tráfego *Web* interceptado possuíam extensões consideradas dinâmicas, entretanto, apenas 2% destas receberam hits;
- Em média 55% do tráfego *Web* teve como destino sites de vídeos;
- A maior taxa de acertos no *cache* alcançada em um único dia foi de 10%, porém, representou apenas 1,95% de todo o tráfego *Web* daquele dia;
- A maior parte dos objetos em cache são imagens e com tamanhos inferiores a 100 bytes;
- Em nenhum momento o cache chegou a 15% de sua capacidade, desta forma o espaço em cache não foi um limitador para cache de objetos.

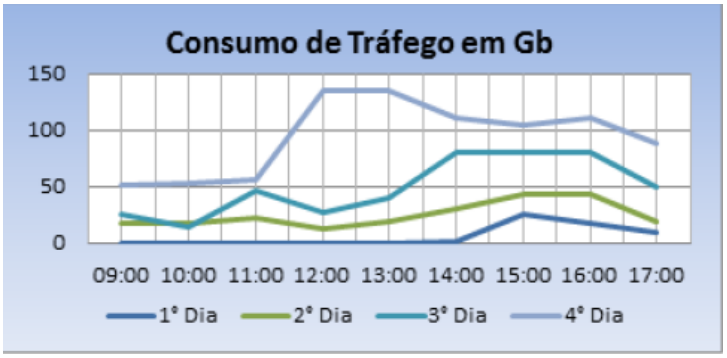

Figura 3.4: Tráfego *Web* em GB por segundo durante o período analisado [23].

Durante a análise dos *logs* capturados, constatou que o consumo do tráfego HTTP da rede é inerentemente dinâmico, e obteve menos de 2% de acerto de *cache*. Destacou, ainda, que o sistema de *proxy Web cache* institucional convencional não consegue tratar as URLs dinâmicas do YouTube.

3.3 Considerações Finais

Esse capítulo mostrou a representatividade do tráfego de arquivos de vídeo na rede. Também, foram apresentados alguns estudos na área de *cache* de vídeos, que buscam otimizar a utilização dos recursos de rede neste contexto.

Apesar dos estudos realizarem suas propostas com sucesso, foi visto que todos evitam manter o vídeo inteiro em *cache*. Isto deve-se ao fato do espaço disponível para *cache* ser limitado, como visto anteriormente na Subseção 2.1.3. Deste modo, buscam armazenar em *cache* os trechos do vídeo que consideram ter maior probabilidade de serem requisitados no futuro.

Neste cenário, a utilização de uma política colaborativa de *cache* minimiza o impacto decorrente do tamanho dos vídeos. Pelo fato dos dispositivos compartilharem seus *caches*, haverá um aumento da disponibilidade de espaço de armazenamento entre eles. Este foi o fator motivador da escolha desse tipo de política para ser adotada neste trabalho.

Mostrou, ainda, que o YouTube será utilizado como estudo de caso deste trabalho, tendo em vista sua representatividade nos tráfegos das redes e o grande número de estudos acadêmicos voltados a esta plataforma.

No capítulo seguinte, será apresendo o sistema de *cache* colaborativo desenvolvido. Além de detalhar seus módulos internos, serão explicados o funcionamento individual de cada um.

Capítulo 4

Protocolo Colaborativo e Arquitetura do Sistema Proposto

Este capítulo apresentará o sistema desenvolvido e o cenário que motivou sua construção. Na Seção 4.1 são mostrados os benefícios do uso de *cache* em uma rede local. A Seção 4.2 apresenta a motivação da escolha de um modelo colaborativo para ser desenvolvido. Ainda, explica o funcionamento geral do sistema, detalha cada módulo existente e a relação entre eles.

4.1 *Cache* em Redes Locais

Na Figura 4.1 é mostrado o cenário de uma rede local que não possui um sistema de *cache* de vídeos. Nela estão presentes diversos dispositivos de acesso à rede, como PCs (do inglês, *Personal Computers*) e *laptops*, que podem entrar e sair a qualquer momento. Neste contexto, quando há uma requisição (HTTP) para um vídeo, estes dispositivos buscarão a informação diretamente dos servidores do YouTube, e consumirão essa informação de forma individual.

Uma das maneiras de evitar que todas as requisições sejam enviadas diretamente para a Internet, é utilizar um sistema de *cache* na rede feito por intermédio de um único servidor. Na maioria dos casos, o servidor *proxy* assume essa função. A Figura 4.2 mostra a situação de uma rede local que apresenta um servidor *proxy* realizando o *cache*. Dessa forma, sempre que houver uma requisição de vídeo, o dispositivo o buscará primeiramente no *proxy*. Havendo um acerto de *cache*, não haverá a necessidade de transferir os dados da Internet. Porém, como foi visto, os arquivos de vídeo possuem tamanhos elevados, o que diminui a diversidade do *cache*. Além disso, há uma limitação da capacidade de processamento, pois apenas um servidor seria responsável pelo tratamento de todas as requisições feitas à Internet. No caso da UnB, em horários de pico, o tráfego de rede passando pelo *proxy* se aproxima de 150 GB por segundo [23]. Neste contexto, o *cache* realizado pelo servidor *proxy* é vantajoso pois consegue reduzir a quantidade de requisições que são enviadas para fora da rede e diminuir o tempo de recuperação da informação. Em contrapartida, é muito custoso para um único servidor gerenciar um sistema de *cache* e cumprir suas atividades habituais com esse volume de dados. Além disso, torna-se um ponto único de falha e dificulta a escalabilidade do sistema.

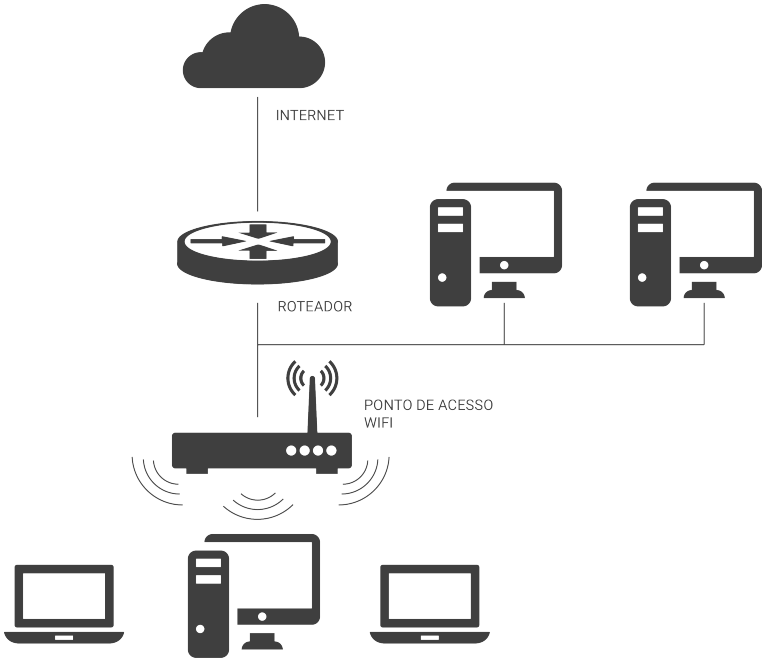

Figura 4.1: Exemplo de uma rede local que não possui um sistema de *cache*, onde todas as requisições são enviadas diretamente para a Internet.

Um sistema de *cache* colaborativo pode suprir a demanda de espaço e diminuir a sobrecarga de processamento no servidor *proxy*, sendo um sistema escalável e tolerante a falhas.

A Figura 4.3 mostra uma rede local com o modelo de *cache* colaborativo proposto, onde há comunicação entre os membros do grupo colaborativo e cada um deles contribui com um espaço de armazenamento de *cache*. Quando algum membro do grupo requisita um vídeo para a Internet, a informação é buscada primeiramente no *cache* do dispositivo requisitante, e, em caso de falha de *cache*, em todos os dispositivos que estejam colaborando. Quanto maior a quantidade de dispositivos colaborando, maior será a quantidade de *cache* disponível. Isto aumentará a probabilidade de acontecer um acerto de *cache*, reduzindo a quantidade de requisições enviadas para fora da rede. Outro benefício do modelo é que a saída da rede de qualquer dispositivo que esteja colaborando não ocasiona falhas no sistema.

Para mostrar a viabilidade do modelo proposto, foi implementado um sistema *cache* colaborativo distribuído. A seguir serão apresentados os elementos que o compõe, sua integração e seu funcionamento.

4.2 Descrição do Sistema

O sistema desenvolvido visa, principalmente, diminuir o tráfego de saída da rede, com consequente diminuição do consumo de banda. Funciona de maneira transparente para a aplicação, onde os dispositivos podem entrar e sair da rede sem que haja influência no funcionamento do sistema. Diferentemente dos trabalhos apresentados na Seção 2.5.1, foi desenvolvido um sistema de *cache* colaborativo, diretamente nos clientes, sem a ne-

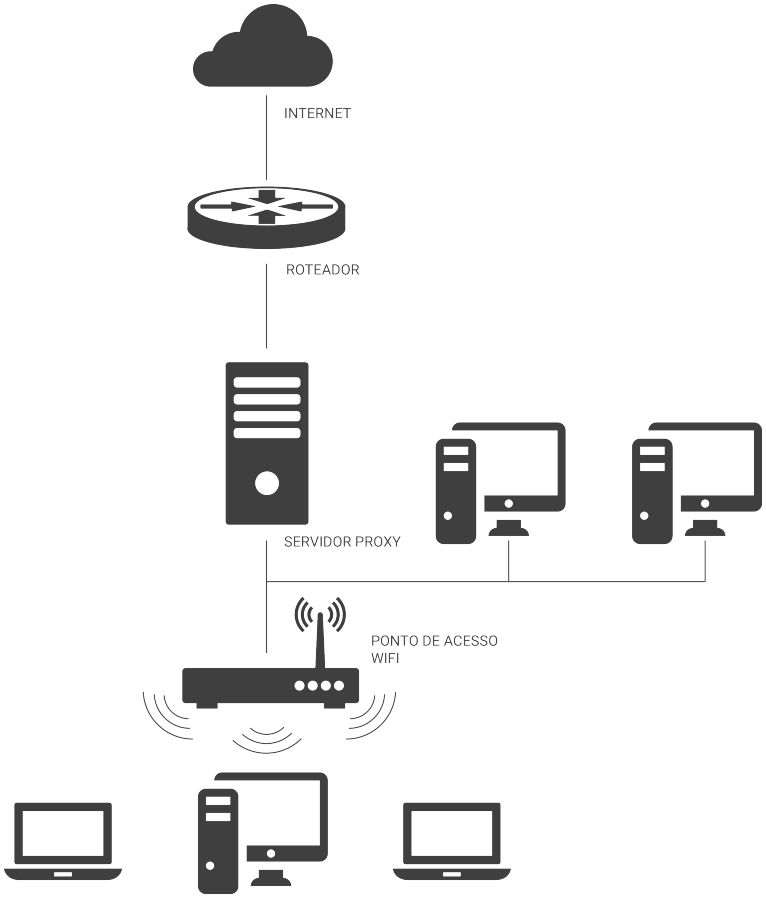

Figura 4.2: Exemplo de uma rede local com um servidor *proxy* centralizando as requisições e realizando *cache*.

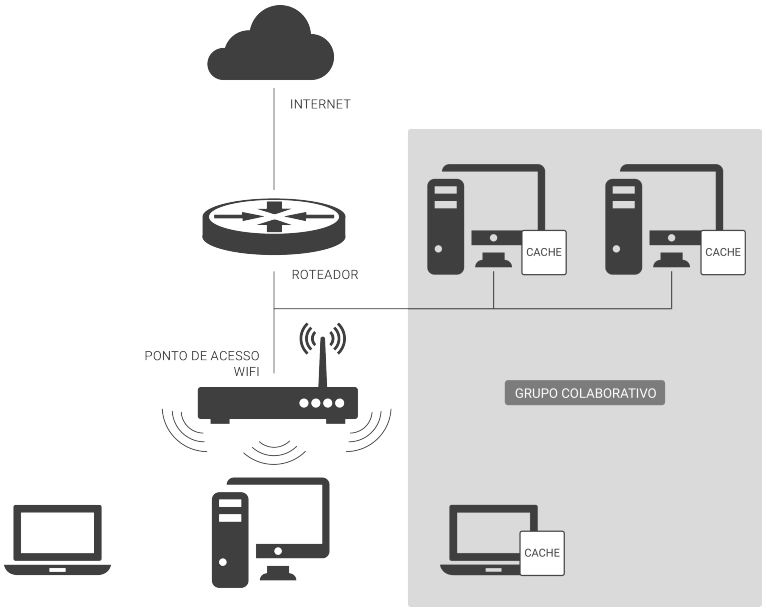

Figura 4.3: Exemplo de uma rede local utilizando o modelo proposto, onde há comunicação entre os membros do grupo colorativo que realizam o *cache* de forma distribuída.

cessidade de um servidor *proxy* centralizado intermediando todas as requisições. Com isso, busca-se diminuir a concentração de carga nestes servidores, pois o sistema será responsável por gerenciar o *cache* em cada usuário, conforme visto na Figura 4.3.

De modo geral, cada usuário mantém um *cache* local que é consultado a cada requisição. Além disso, em caso de falha, a requisição será enviada aos demais usuários da rede antes de ser enviada à Internet.

4.2.1 Arquitetura do Sistema

No modelo proposto, cada computador conectado à rede que esteja participando do sistema de *cache* colaborativo, se associará a um grupo criado e gerenciado pelo sistema desenvolvido, através da biblioteca JGroups. Todos os clientes destinarão um espaço de seu HD (do inglês, *Hard Disk*) para contribuir com o *cache* de vídeos. Deste modo, não há prejuízo ao cliente, e o espaço, embora pequeno, somado às outras máquinas da rede, trará uma quantidade significativa de espaço de armazenamento.

Ao ser realizada uma requisição para o YouTube, um *plug-in* instalado no navegador do usuário redireciona as requisições, de forma transparente, para o sistema desenvolvido, onde primeiramente o vídeo é buscado em seu *cache* local. Caso não seja encontrado, é enviada uma mensagem no grupo para saber se algum outro usuário possui tal vídeo armazenado em seu *cache*. Caso possua, o vídeo é transferido entre eles. Apenas existindo as duas negativas é que o vídeo será transferido diretamente da Internet.

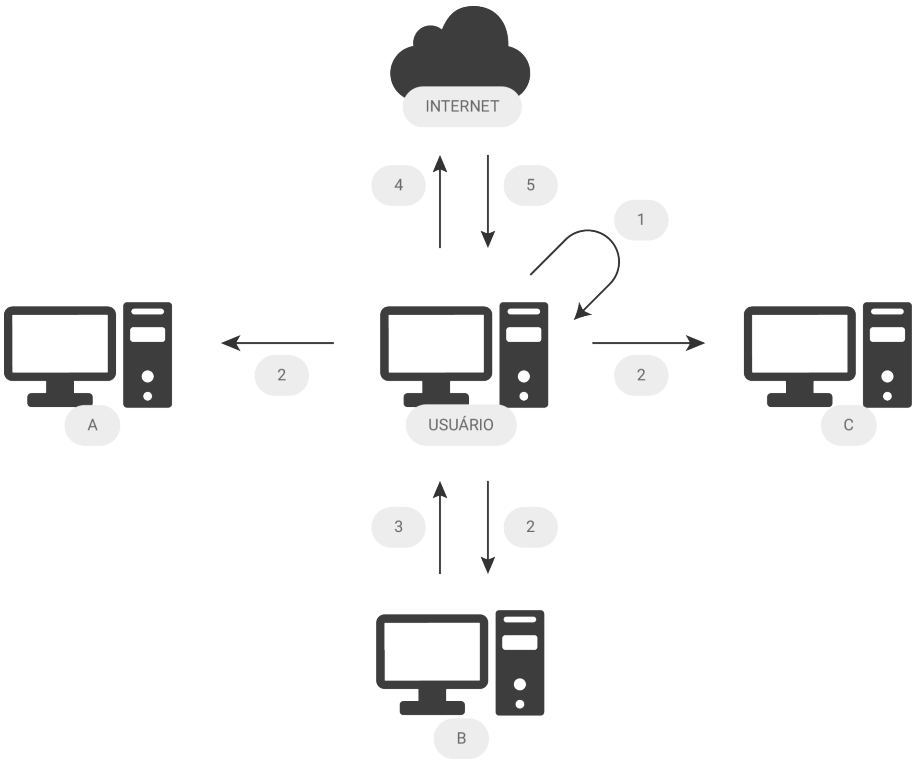

Figura 4.4: Visão geral de todas as requisições e mensagens que podem ser enviadas durante a visualização de um vídeo.

A Figura 4.4 mostra uma visão geral do funcionamento do sistema desenvolvido. O item (1) representa uma requisição de vídeo ao YouTube interceptada pelo *plug-in* do navegador do usuário e redirecionada para o sistema desenvolvido. Primeiramente, a requisição é buscada no *cache* local do usuário. Caso haja um acerto, o vídeo é retornado e apresentado ao usuário. Caso contrário, é enviada uma mensagem ao grupo, representado pelo item (2), para consultar se algum outro usuário possui o vídeo em questão armazenado em seu *cache*. Supondo que o usuário (B) possua o vídeo, este é retornado para o requisitante, como mostra o item (3). Não havendo o vídeo em questão em nenhum outro usuário da rede, a requisição, então, é feita para os servidores do YouTube através da Internet, como indica o item (4). Em (5) está a resposta do YouTube para a requição realizada.

É importante salientar que apenas as requisições de vídeos são inteceptadas e tratadas pelo programa. As demais requisições seguem o fluxo normal de processamento, ou seja, são requisitadas diretamente para os servidores do YouTube.

A seguir, será explicado mais detalhadamente o funcionamento do sistema proposto, dividindo-o em duas partes. A primeira, que é responsável pela interceptação das requisições do navegador e gerência do *cache* de cada usuário, será entitulada como **módulo local**. E a outra, que tem como função gerenciar a troca de mensagens e a transferência de dados entre os usuários na rede, será referenciada como **módulo colaborativo**.

4.2.2 Módulo Local

O módulo local do sistema é responsável pela recuperação do vídeo caso haja um acerto de *cache*. Através de um *plug-in* instalado no navegador do usuário, são interceptados dois tipos de requisições que seriam originalmenente encaminhadas para o YouTube, uma requisição de mapeamento e outra requisição de parte do vídeo, como visto na seção 3.2.2.

Conforme visto na Figura 4.5, quando o navegador faz uma requisição ao YouTube, esta requisição é redirecionada para o módulo local (1), que permanece constantantemente escutando na porta 7512. Como o sistema intercepta dois tipos de requisições, são necessários tratamentos diferentes, realizados da seguinte forma:

Requisição de mapeamento

Esta é a primeira requisição recebida pelo módulo local após a solicitação de um vídeo ao YouTube. Neste momento são extraídos da URL os parâmetros `video_id` e `cpn` (2). Esses parâmetros são armazenados em um mapeamento interno (3), para identificação única dos vídeos. É importante salientar que o parâmetro `video_id` é o identificador único de cada vídeo e só é encontrado nesta requisição, onde é relacionado com o parâmetro `cpn`. A partir desse momento, somente o parâmetro `cpn` é utilizado nas requisições subsequentes de trechos do vídeo.

Requisição de trechos do vídeo

Todas as demais requisições interceptadas são referentes a trechos do vídeo. São extraídos das URLs dessas requisições os parâmetros `cpn` e `range` (4). O primeiro será utilizado para recuperar o identificador único do vídeo no mapeamento citado acima (5), e o segundo identifica um determinado trecho do vídeo. Assim, todas as informações necessárias estarão disponíveis para a identificação única de um trecho de um vídeo.

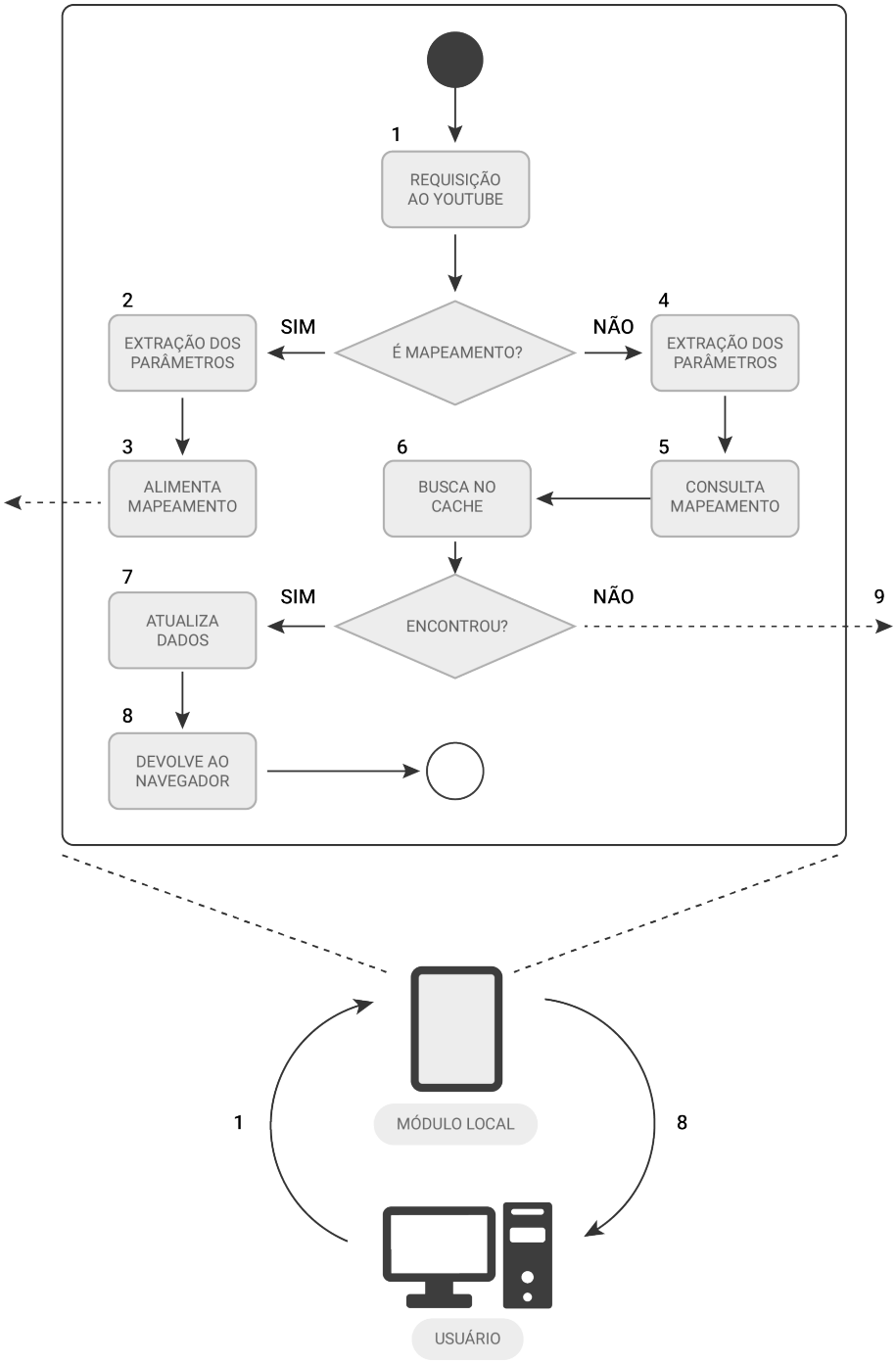

Figura 4.5: Detalhe do comportamento do módulo local.

Após a descoberta dos parâmetros `video_id`, `cpn` e `range` o módulo local consulta no *cache* (6) o trecho requisitado. Caso o encontre, serão atualizadas as informações de quantidade de requisições e data da última requisição (7), necessárias para as políticas de substituição. Feito isso, o vídeo é devolvido ao navegador (8) para a visualização, sem necessidade de buscá-lo em um dos servidores do YouTube. Caso contrário, a requisição será encaminhada para o módulo colaborativo (9).

4.2.3 Módulo Colaborativo

O módulo colaborativo do sistema é responsável pela gerência do grupo, troca de mensagens e transferência dos arquivos de vídeo entre os usuários. Para gerenciar o grupo e realizar a troca de mensagens entre os usuários, foi utilizada a biblioteca JGroups, cujas características encontram-se no Apêndice A.

Funcionamento

No módulo colaborativo do sistema, quando um usuário faz uma requisição de um determinado trecho do vídeo ao YouTube, e esse vídeo não foi encontrado em seu *cache* local, é realizada uma busca em todos os membros da rede, como mostrado nas Figuras 4.6 e 4.7. Para isso, o usuário envia uma mensagem para o grupo, através do JGroups (1), buscando o trecho desejado. Aqueles que o tiverem, enviarão uma resposta positiva ao usuário requisitante (2 e 3), que, então, selecionará o primeiro que o responder. Será aberto um *socket* que será conectado na porta 7513 do usuário escolhido para a transferência do arquivo solicitado (4). Em seguida, serão atualizadas as informações de quantidade de requisições e data da última requisição. Feito isso, o vídeo é devolvido ao navegador para a visualização, sem necessidade de buscá-lo de um dos servidores do YouTube. Todas as buscas realizadas pelos usuários no grupo são limitadas por controladores de tempo, para que não haja demora na transferência dos dados, deixando imperceptível ao usuário que o vídeo não está sendo transferido de um servidor do YouTube.

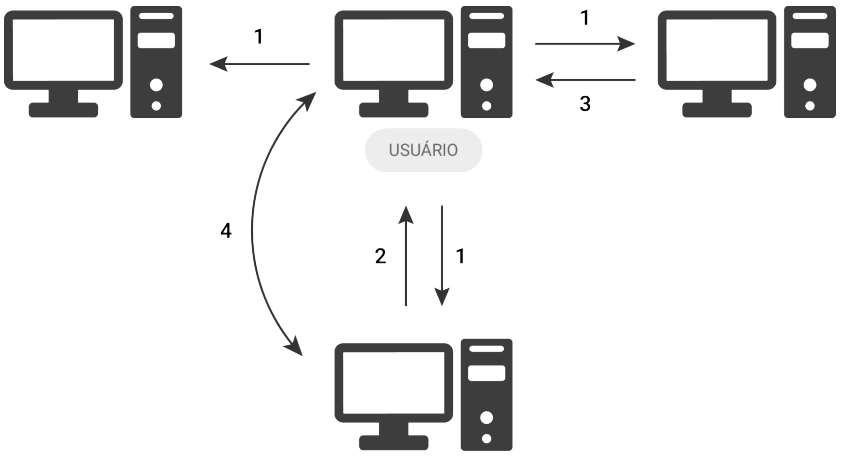

Figura 4.6: Detalhe do comportamento do módulo colaborativo.

4.3 Considerações Finais

Neste capítulo foram apresentadas as justificativas para a criação do modelo de *cache* colaborativo proposto, bem como os elementos implementados para sua viabilização. Foi visto que o módulo local é responsável pelo gerenciamento do *cache* local do usuário. Dentre suas responsabilidades estão tratar as requisições feitas ao YouTube, extrair os parâmetros necessários das URLs dinâmicas para identificação única de um trecho do vídeo e fazer a devidas consultas e atualizações no *cache* local. Também foi visto que

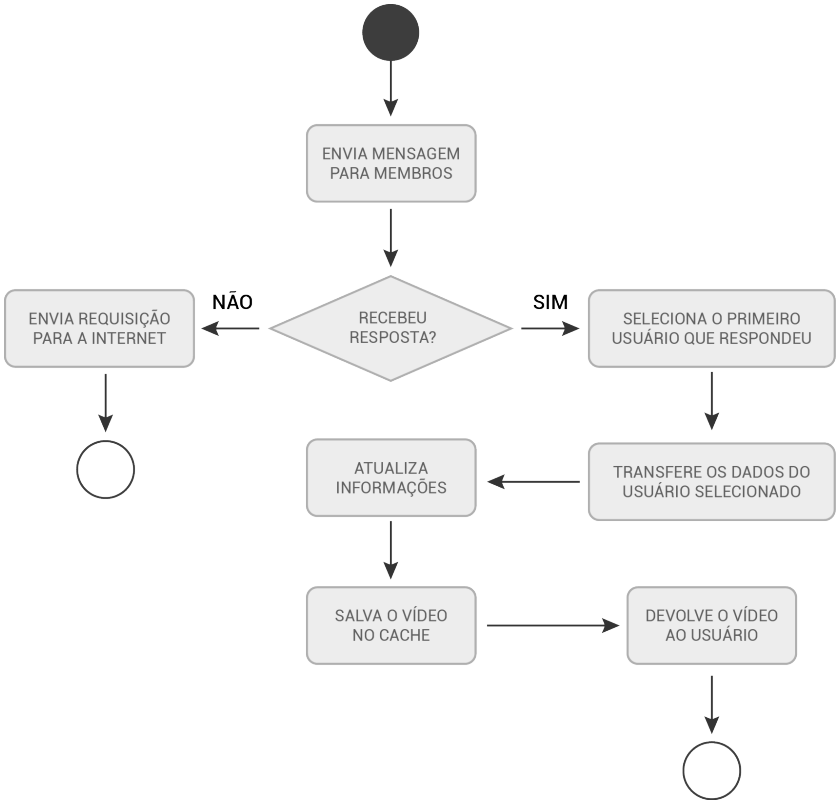

Figura 4.7: Fluxograma detalhado do comportamento do módulo colaborativo.

quando o módulo local se depara com uma falha de *cache* a requisição é enviada para o módulo colaborativo. Este módulo é responsável pela comunicação entre os usuários da rede, gerência dos tempos de espera e transferência dos trechos de vídeo em caso de acertos de *cache*. No próximo capítulo será abordado o ambiente de testes criado e os resultados obtidos.

Capítulo 5

Resultados Experimentais

Este capítulo apresenta o ambiente de testes e os resultados obtidos na simulação do sistema desenvolvido. Na Seção 5.1, são detalhados os programas desenvolvidos para auxiliar na simulação do ambiente de testes. Será mostrada, ainda, a topologia do sistema de *cache* da rede e seu funcionamento em cada usuário. Na Seção 5.2, serão apresentadas todas as simulações realizadas e os resultados obtidos em cada uma.

5.1 Ambiente de Testes

Para obter um ambiente de testes controlado e eficiente, foram desenvolvidos outros dois programas auxiliares: um robô de requisições e um simulador do YouTube. Com isso, foi possível automatizar o processo de testes e gerar iterações suficientes para determinar os padrões de comportamentos e validar o modelo de *cache* colaborativo proposto.

5.1.1 Programas Auxiliares

A seguir, serão apresentados o objetivo e as justificativas para a criação de cada programa auxiliar, além de explicar detalhes do funcionamento de ambos.

Robô de Requisições

Este programa foi criado com o intuito de auxiliar na execução dos testes. Tem como função realizar requisições, simulando o comportamento de um usuário enquanto solicita vídeos ao YouTube. A justificativa de sua criação deu-se aos seguintes pontos:

1. **Parametrização dos testes**:
   Durante a realização dos testes, a escolha do vídeo é feita considerando dois parâmetros: a aleatoriedade e o interesse entre os usuários, como será visto na Subseção 5.1.2. Deste modo, tal escolha é padronizada quando é realizada por um programa auxiliar.
2. **Automatização das requisições**:
   Como será visto na Seção 5.2, serão feitas 10000 requisições de trechos de vídeo ao YouTube em cada simulação. Seria inviável para um usuário comum realizar este procedimento manualmente, sendo necessário a automatização do processo.

Este programa recebe como entrada uma coleção de arquivos que contêm todas as requisições enviadas ao YouTube, necessárias para o *cache* de um vídeo. A cada iteração, este programa sorteia, aleatoriamente, com distribuição uniforme (conforme a classe `Random` do Java), um desses arquivos, simulando a escolha do usuário a um vídeo do YouTube. Para cada requisição presente no arquivo escolhido, este programa a encaminha para o programa de *cache*, simulando a intercepção do *proxy* do navegador, conforme o item (1) da Figura 4.5.

Simulador do YouTube

Este programa também foi criado com o intuito de auxiliar na execução dos testes. Tem como função receber as requisições que seriam originalmente enviadas ao YouTube e responder enviando o trecho de vídeo solicitado. A justificativa de sua criação deve-se aos seguintes pontos:

1. **Expiração das requisições**:
   Como o YouTube implementa, sistematicamente, mecanismos para evitar o *cache* de seus vídeos, foi preciso entender como funciona o seu protocolo de requisições em dado momento. Um desses mecanismos é realizado através de um parâmetro chamado `expire`. Como será visto mais adiante na Subseção 5.1.2, este parâmetro fornece um tempo de validade da requisição, ou seja, passado este período esta requisição não será mais aceita pelos servidores do YouTube. Uma vez que as requisições foram armazenadas para serem utilizadas pelo robô de requisições, haveria pouco tempo hábil para ser executada uma simulação antes que as requisições expirassem. Assim, foi criado este programa para simular o funcionamento do YouTube, aceitando as requisições com o tempo de expiração vencido no momento da simulação. A partir disto, foi realizado um *cache* do universo de vídeos que seria utilizado na etapa de teste e implementado o modelo colaborativo proposto com base nesse protocolo.
2. **Otimização do Tempo**:
   Durante a visualização de um vídeo do YouTube, há um controle sobre as requisições que são enviadas. Para realizar uma transferência completa do vídeo é preciso aguardar praticamente toda a sua visualização. Conforme mencionado, serão feitas 10000 requisições de trechos de vídeo, e tendo cada um desses trechos a duração de 15 segundos, isso representa um total aproximado de 42 horas para cada simulação. Para otimizar o tempo de simulação, este programa responde às requisições sem praticar esse tempo de retardo.

Este programa recebe as requisições que seriam originalmente enviadas ao YouTube. Para isso, possui armazenado todos os vídeos que venham a ser requisitados na simulação. Ao receber um requisição, o trecho do vídeo é localizado e transferido para o requisitante.

A Figura 5.1 apresenta a comunicação entre o sistema de *cache* e os programas auxiliares. Em (1) há uma requisição enviada pelo robô para o sistema de *cache*, de forma análoga à interceptação e redirecionamento das requisições feitas pelo *proxy* do navegador quando o usuário requisita um vídeo ao YouTube. Em caso de falha de *cache*, o vídeo será requisitado ao simulador do YouTube (2), e será transferido ao sistema de *cache* (3).

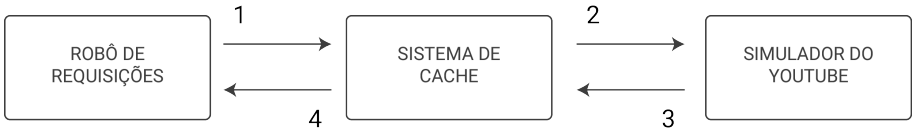

Figura 5.1: Comunicação entre o sistema de *cache* e os programas auxiliares desenvolvidos.

Após esse procedimento, é enviada uma mensagem de confirmação ao robô (4), que assim estará apto a enviar a próxima requisição.

5.1.2 Topologia dos Testes

Para a realização dos testes foi necessário determinar os tipos de vídeos existentes na rede, o espaço disponível para *cache* nos dispositivos dos usuários e a porcentagem de interesse em cada vídeo entre eles. A seguir, estes itens serão apresentados.

Características de um Vídeo

Um vídeo do YouTube é composto por diversos trechos, que são lidos através de um reprodutor contido na página HTML. O usuário, ao solicitar o vídeo, realiza diversas requisições, sendo uma para cada trecho. A Figura 5.2 mostra algumas das requisições interceptadas durante a visualização de um vídeo. Os parâmetros `video_id`, `cpn`, `range` e `expire` foram destacados. É possível notar que o `video_id`, identificador único do vídeo, é encontrado apenas na primeira requisição. Nas requisições subsequentes não é encontrado, mas é possível recuperá-lo através do parâmetro `cpn`, como visto na Subseção 3.2.3.

```
http://www.youtube.com/ptracking?cpn=1M6IjRRK-epFYf68&plid=AAUIFlT-Unp2b2j8&pltype=contentugc&ei=i4RqVN7
_KML58gapqoA4&ptk=youtube_multi&video_id=-3kJJzekFjw&oid=M9mVQuR0e1QFBtqFPK1ktA.Ta2MYghl4Z0lHK1qcf671g

http://r11---sn-gpv7ene7.googlevideo.com/videoplayback?c=web&clen=1816188&cpn=1M6IjRRK-epFYf68&cver=as3&
dur=114.311&expire=1416288491&fexp=907259%2C916602%2C917000%2C919330%2C927622%2C932404%2C938645%2C943909
%2C946602%2C947209%2C947215%2C948124%2C952302%2C952605%2C952901%2C953912%2C957103%2C957105%2C957201%2C95
8303&gir=yes&id=o-ANTebVd-hRtsY5SpmCfQFH-v7_bq8f2l-7SKMPdSeDS0&initcwndbps=2097500&ip=164.41.209.49&ipbi
ts=0&itag=140&keepalive=yes&key=yt5&lmt=1391874816248446&mime=audio%2Fmp4&mm=31&ms=au&mt=1416266756&mv=m
&range=0-237567&ratebypass=yes&signature=1E8B0B882CDBF889DD6505596A7E182476EBF092.7B49E15754428E5E6E9F1C
212DEE4B435C9AD2C9&source=youtube&sparams=clen%2Cdur%2Cgir%2Cid%2Cinitcwndbps%2Cip%2Cipbits%2Citag%2Ckee
palive%2Clmt%2Cmime%2Cmm%2Cms%2Cmv%2Csource%2Cupn%2Cexpire&sver=3&upn=hBcjbZYLbz4

http://r11---sn-gpv7ene7.googlevideo.com/videoplayback?c=web&clen=9628963&cpn=1M6IjRRK-epFYf68&cver=as3&
dur=114.182&expire=1416288491&fexp=907259%2C916602%2C917000%2C919330%2C927622%2C932404%2C938645%2C943909
%2C946602%2C947209%2C947215%2C948124%2C952302%2C952605%2C952901%2C953912%2C957103%2C957105%2C957201%2C95
8303&gir=yes&id=o-ANTebVd-hRtsY5SpmCfQFH-v7_bq8f2l-7SKMPdSeDS0&initcwndbps=2097500&ip=164.41.209.49&ipbi
ts=0&itag=135&keepalive=yes&key=yt5&lmt=1391874838266819&mime=video%2Fmp4&mm=31&ms=au&mt=1416266756&mv=m
&range=0-2068479&ratebypass=yes&signature=02632ED9D5D5D38675BD9D647A5EC9237A366443.629BA5334F70B24BEBAFA
C34BB729ED00422688B&source=youtube&sparams=clen%2Cdur%2Cgir%2Cid%2Cinitcwndbps%2Cip%2Cipbits%2Citag%2Cke
epalive%2Clmt%2Cmime%2Cmm%2Cms%2Cmv%2Csource%2Cupn%2Cexpire&sver=3&upn=hBcjbZYLbz4

http://r11---sn-gpv7ene7.googlevideo.com/videoplayback?c=web&clen=1816188&cpn=1M6IjRRK-epFYf68&cver=as3&
dur=114.311&expire=1416288491&fexp=907259%2C916602%2C917000%2C919330%2C927622%2C932404%2C938645%2C943909
%2C946602%2C947209%2C947215%2C948124%2C952302%2C952605%2C952901%2C953912%2C957103%2C957105%2C957201%2C95
8303&gir=yes&id=o-ANTebVd-hRtsY5SpmCfQFH-v7_bq8f2l-7SKMPdSeDS0&initcwndbps=2097500&ip=164.41.209.49&ipbi
ts=0&itag=140&keepalive=yes&key=yt5&lmt=1391874816248446&mime=audio%2Fmp4&mm=31&ms=au&mt=1416266756&mv=m
&range=237568-475135&ratebypass=yes&signature=1E8B0B882CDBF889DD6505596A7E182476EBF092.7B49E15754428E5E6
E9F1C212DEE4B435C9AD2C9&source=youtube&sparams=clen%2Cdur%2Cgir%2Cid%2Cinitcwndbps%2Cip%2Cipbits%2Citag%
2Ckeepalive%2Clmt%2Cmime%2Cmm%2Cms%2Cmv%2Csource%2Cupn%2Cexpire&sver=3&upn=hBcjbZYLbz4

http://r11---sn-gpv7ene7.googlevideo.com/videoplayback?c=web&clen=9628963&cpn=1M6IjRRK-epFYf68&cver=as3&
dur=114.182&expire=1416288491&fexp=907259%2C916602%2C917000%2C919330%2C927622%2C932404%2C938645%2C943909
%2C946602%2C947209%2C947215%2C948124%2C952302%2C952605%2C952901%2C953912%2C957103%2C957105%2C957201%2C95
8303&gir=yes&id=o-ANTebVd-hRtsY5SpmCfQFH-v7_bq8f2l-7SKMPdSeDS0&initcwndbps=2097500&ip=164.41.209.49&ipbi
ts=0&itag=135&keepalive=yes&key=yt5&lmt=1391874838266819&mime=video%2Fmp4&mm=31&ms=au&mt=1416266756&mv=m
&range=2068480-4136959&ratebypass=yes&signature=02632ED9D5D5D38675BD9D647A5EC9237A366443.629BA5334F70B24
BEBAFAC34BB729ED00422688B&source=youtube&sparams=clen%2Cdur%2Cgir%2Cid%2Cinitcwndbps%2Cip%2Cipbits%2Cita
g%2Ckeepalive%2Clmt%2Cmime%2Cmm%2Cms%2Cmv%2Csource%2Cupn%2Cexpire&sver=3&upn=hBcjbZYLbz4
```

Figura 5.2: Sequência de requisições de trechos de um vídeo ao YouTube.

Para a realização dos testes, optou-se pela utilização de vídeos idênticos. Foi escolhido um vídeo de 2 MB e replicado em quantidade suficiente para realização das simulações. Em seguida, foram alterados os valores dos parâmetros `video_id` e `cpn` para que fossem tratados como vídeos diferentes pelo sistema de *cache*. Definiu-se, também, que cada dispositivo terá um universo de 100 vídeos disponíveis para requisição.

Definições de Interesse

Nas simulações realizadas, cada usuário terá uma coleção de 100 vídeos que poderão ser requisitados. Uma parte destes vídeos também poderão ser requisitados pelos demais membros do grupo, os quais serão chamados de vídeos de interesse comum. Ou seja, caso a porcentagem de interesse seja fixada em 25%, dentre os 100 vídeos, haverão 25 que poderão ser requisitados por todos os demais usuários. Assim, quanto maior a porcentagem de interesse, maior a colaboratividade entre os usuários do grupo. Neste trabalho serão utilizados os interesses 25%, 50%, 75% e 100%.

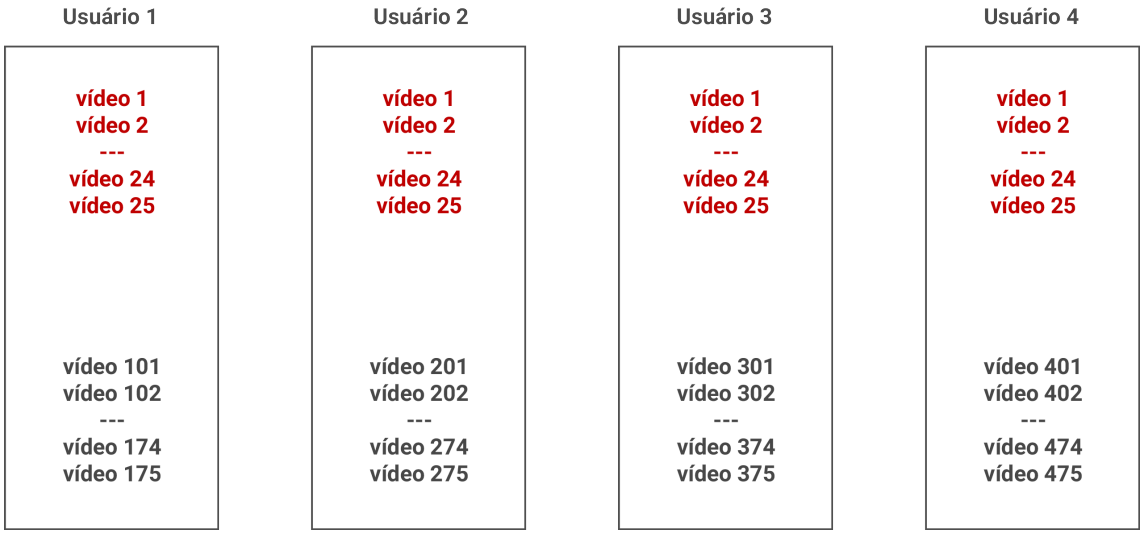

Figura 5.3: Universo de vídeos distribuído em 4 usuários que participam do grupo colaborativo com 25% de interesse.

A Figura 5.3 mostra a coleção de vídeos passíveis de requisição para cada usuário em uma simulação com quatro membros no grupo e 25% de interesse entre os membros do grupo. É possível notar que 25 vídeos são iguais para todos os usuários. Os demais 75 vídeos, de cada usuário, são diferentes entre si e podem ser requisitados apenas pelo usuário que o contém.

Tamanho do Cache

Cada usuário do grupo possui um espaço disponível para *cache*. Nas simulações realizadas, foi definido o tamanho de 50 MB, tendo em vista que seria necessário, nas baterias de testes, simular os seguintes cenários: tamanho total do *cache* colaborativo ser menor, igual e maior que o tamanho do universo de vídeos.

Política de Substituição

Para a realização das baterias de testes, foi utilizada a política de substituição LRU, tendo em vista que esta política consegue obter, na média, os melhores resultados [35].

Consistência de *cache*

Os arquivos de vídeo possuem características únicas, como o fato de serem imutáveis. Uma vez que os arquivos de vídeo são criados e enviados para o YouTube, eles não podem mais ser modificados. Desde modo, não é necessário que haja uma preocupação com a consistência do *cache* armazenado. Neste contexto, as simulações realizadas não terão uma política para verificação da consistência do *cache*.

5.2 Avaliação dos Resultados

Nesta seção serão apresentados e analisados os resultados obtidos nas simulações realizadas. Todos os testes foram realizados com cada usuário fazendo 10000 requisições, e os resultados apresentados representam a média dos valores obtidos entre os membros do grupo em cada simulação.

As simulações ocorreram numa rede local, com máquinas da arquitetura Intel Core i7-3770 CPU 3.40GHz, 8 GB de memória, sistema operacional Windows 7 Professional 64-bits.

5.2.1 Verificação do Funcionamento do Modelo Proposto

Neste tópico, as simulações realizadas tem o objetivo de comprovar o funcionamento do modelo de *cache* colaborativo proposto na medida em que aumenta o interesse entre os usuários do grupo. Note que, em todos os resultados das simulações apresentadas a seguir, o percentual de acerto de *cache* local permanece em torno de 25%. Como mencionado acima, isto é consequência do tamanho do *cache* ser fixo e armazenar em torno de 25% dos vídeos de interesse de cada usuário. Note, também, que em todos os gráficos que seguem, o eixo x representa o aumento do interesse dos usuários nos mesmos vídeos, enquanto o eixo y indica o valor obtido em cada parâmetro avaliado, ou seja, percentual de falhas, acertos locais e acertos colaborativos.

A Figura 5.4 apresenta as simulações realizadas com 2 usuários colaborando seu *cache*. O objetivo desta primeira bateria de testes foi verificar se a política colaborativa de *cache* funciona da maneira esperada. Como pode ser observado, a medida que aumenta o interesse mútuo entre os usuários, aumenta também a quantidade de acertos de *cache* colaborativo. Este comportamento condiz com o esperado. No melhor caso, ou seja, com 100% de interesse entre os usuários, esta política obteve, além dos acertos de *cache* local, praticamente a mesma quantidade de acertos colaborativos. Ou seja, esta política obteve praticamente o dobro de acertos de *cache* em relação à política individual apresentada. Este comportamento ocorre pois o espaço de armazenamento nos demais membros do grupo é igual ao espaço de armazenamento local.

A Figura 5.5 apresenta as simulações realizadas com 4 usuários colaborando seu *cache*. O objetivo desta segunda bateria de testes foi analisar o comportamento do modelo

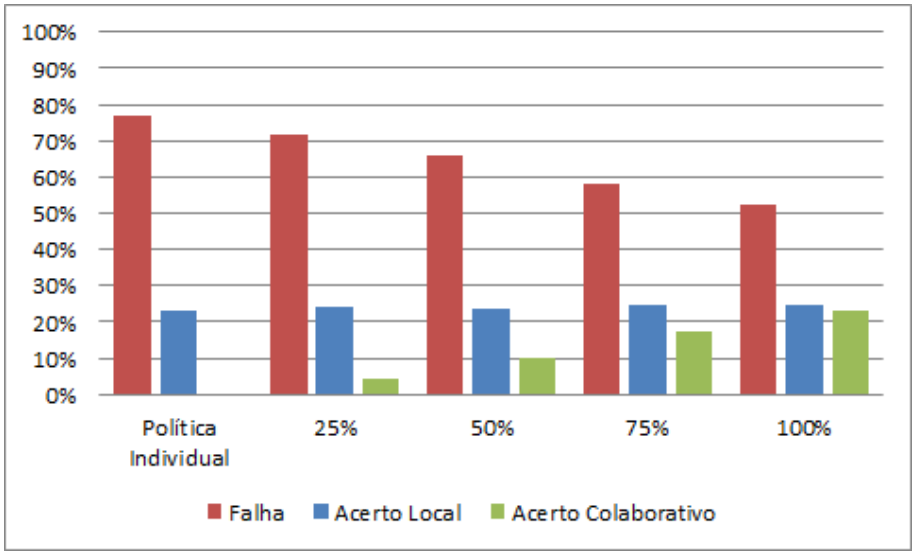

Figura 5.4: Resultados das simulações realizadas com 2 usuários colaborando, variando entre política de *cache* individual e colaborativa de 25% a 100% de interesse.

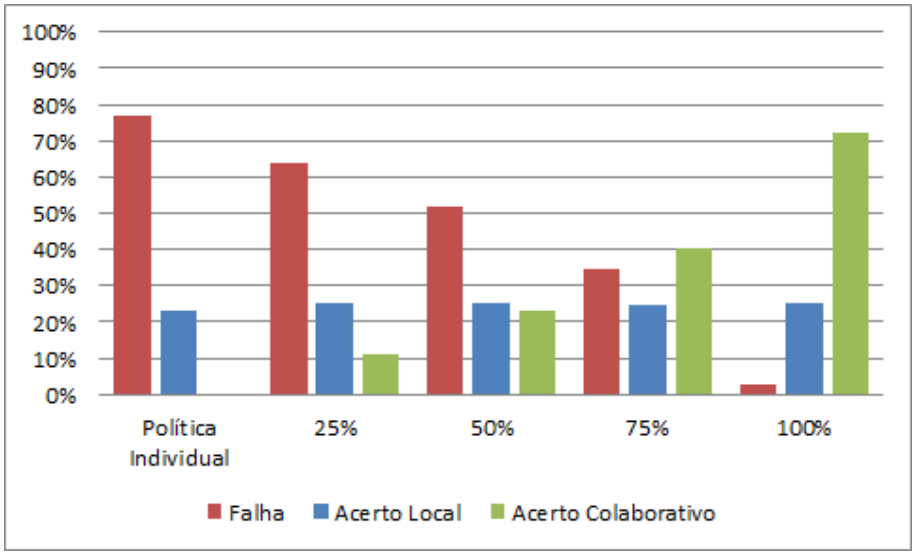

Figura 5.5: Resultados das simulações realizadas com 4 usuários colaborando, variando entre política de *cache* individual e colaborativa de 25% a 100% de interesse.

colaborativo proposto quando, no melhor caso, o tamanho total do *cache* colaborativo equipara-se ao tamanho do universo de vídeos. Como pode ser observado, a medida que aumenta o interesse mútuo entre os usuários, aumenta também a quantidade de acertos de *cache* colaborativo. Nesta bateria de testes, a quantidade de acertos colaborativos se aproximou da quantidade de acertos locais quando o interesse comum entre os usuários foi fixado em 50%, diferentemente da anterior.

Observa-se, também, que na simulação com 100% de interesse entre os usuários, o tamanho total do *cache* colaborativo atingiu o tamanho do universo de vídeos. Consequentemente, é esperado que o modelo proposto consiga atender à todas as requisições realizadas. Este comportamento fica evidente ao verificar que o total de acertos de *ca-*

che (local e colaborativo) atinge 97,48%. A pequena quantidade de falhas é justificada pelo tempo levado para cada usuário popular seu *cache*. Além disso, após os *caches* dos usuários terem sido completamente populados, ocorreram algumas falhas de *cache*. Isto deve-se ao sistema estar trabalhando em seu limite de capacidade de armazenamento, enquanto qualquer atraso que gere uma falha de *cache* ocasionará uma replicação do dado no *cache* e, consequentemente, o universo de vídeos não estará totalmente armazenado.

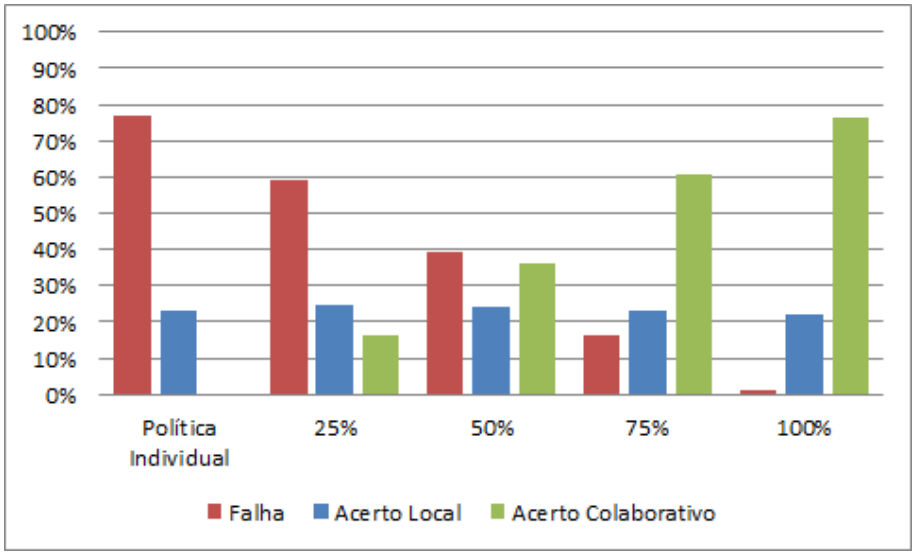

Figura 5.6: Resultados das simulações realizadas com 8 usuários colaborando, variando entre política de *cache* individual e colaborativa de 25% a 100% de interesse.

A Figura 5.6 apresenta as simulações realizadas com 8 usuários colaborando seu *cache*. O objetivo desta terceira bateria de testes foi analisar o comportamento do modelo de *cache* proposto quando o tamanho total do *cache* colaborativo ultrapassa o tamanho do universo de vídeos. Como pode ser observado, a medida que aumenta o interesse mútuo entre os usuários, aumenta também a quantidade de acertos de *cache* colaborativo. Nesta bateria de testes, a quantidade de acertos colaborativos ultrapassou a quantidade de acertos locais quando o interesse comum entre os usuários foi fixado em 50%, diferentemente da anterior. Além disto, nota-se que quando fixado o interesse em 75%, a quantidade de acertos colaborativos é expressivamente maior que a quantidade de acertos locais.

Observa-se, também, que na simulação com 100% de interesse entre os usuários, o tamanho total do *cache* colaborativo foi superior ao tamanho do universo de vídeos, e é possível notar que a quantidade de falhas de *cache* foi inferior ao da bateria anterior. Isto deve-se ao fato do *cache* ter sido populado mais rapidamente e, mais que isto, havia espaço disponível para a recuperação da situação de replicação citada na bateria anterior.

5.2.2 Verificação da Eficiência do Modelo Proposto

O objetivo desta quarta bateria de testes foi comprovar a hipótese de que o modelo de *cache* colaborativo reduz a quantidade de requisições enviadas para fora da rede, comparado com uma política de *cache* individual.

A Figura 5.7 apresenta as simulações realizadas com 2, 4 e 8 usuários colaborando seu *cache*. Estas simulações foram realizadas com o percentual de interesse fixado em 75%. O

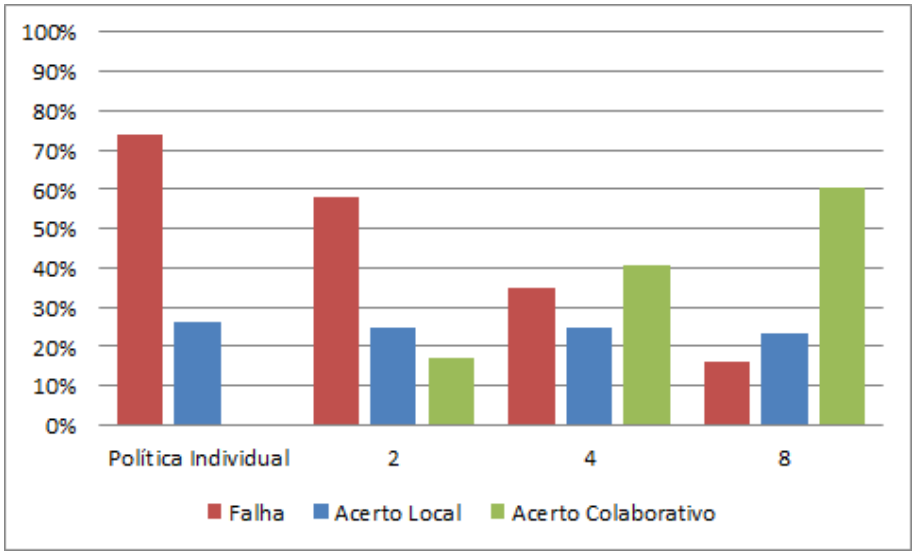

Figura 5.7: Resultados das simulações realizadas com 2, 4 e 8 usuários, mantido o percentual de interesse em 75%.

eixo x representa a quantidade de usuários em cada simulação, enquanto o eixo y indica o valor obtido em cada parâmetro avaliado, ou seja, percentual de falhas, acertos locais e acertos colaborativos.

Nota-se que na simulação com uma política individual de *cache*, 76,7% resultaram em falha de *cache*. Como consequência, 23,3% das requisições resultaram em acerto de *cache* local. O acerto de *cache* local tende a ser constante em todas as simulações, independentemente da quantidade de usuários no grupo. Isto é explicado pelo fato do tamanho do *cache* em cada usuário ser constante (50 MB), podendo armazenar em média 25% dos vídeos de seu interesse.

Já na simulação com 2 usuários no grupo, 58,16% das requisições resultaram em falha de *cache*, ou seja, foram enviadas ao servidor do YouTube. Já com 4 usuários, foram buscados no servidor externo 34,7% das requisições. E, por fim, com 8 usuários, foi de 16,09% o percentual da falha de *cache*.

Nestes testes, observa-se o rápido declínio da quantidade de requisições que saíram da rede. Saindo do modelo individual para o colaborativo com 2 usuários, houve uma queda de 24,17% na quantidade de falhas de *cache*. Ao aumentar de 2 para 4 a quantidade de usuários no grupo, houve uma queda de 40,33%. Finalmente, passando de 4 para 8 usuários, houve uma queda de 53,63%. Deste modo, 8 usuarios no grupo geraram uma redução de 79,02% da quantidade de requisições ao servidor do YouTube em relação à política de *cache* individual. Estes resultados comprovam a hipótese levantada no início deste trabalho.

5.2.3 Verificação do Tempo Médio por Requisição no Modelo Proposto

O objetivo desta subseção é comparar o tempo médio de requisição entre o modelo de *cache* colaborativo proposto, uma política individual de *cache* e um dispositivo sem qualquer sistema de *cache*. As fórmulas e os valores utilizados para o cáculo dos tempos

médios foram extraídos das baterias de testes aprentadas anteriormente, e encontram-se no Apêndice B.

A Figura 5.8 apresenta o tempo médio gasto por requisição, calculado para o modelo de *cache* colaborativo proposto, para uma política de *cache* individual e para um usuário que faz suas requisições sem nenhum sistema de *cache*.

Note que, nas baterias de testes realizadas, a política de *cache* individual obteve um menor tempo médio por requisição comparado com a estimativa para cenário em que nenhuma política de *cache* é aplicada. Esse comportamento é justificado, pois a alta taxa de acertos de *cache* nesta política melhorou o tempo médio por requisição.

É importante observar que o tempo médio por requisição, no pior caso do modelo colaborativo proposto, é pior que o tempo médio da política individual. Isto deve-se ao fato da falha no modelo proposto ser mais custosa, pois envolve também o atraso na espera por reposta de algum membro do grupo antes de a requisição ser enviada ao servido.

Porém, conforme o esperado, a medida que os acertos de *cache* foram aumentando, o tempo médio por requisição foi diminuindo. Na Figura 5.8, como existem 4 usuários no grupo colaborativo, esse aumento dos acertos de *cache* está relacionado ao aumento do percentual de interesse entre eles. Ainda, é visto que o tempo médio por requisição na política individual é 7 vezes o tempo médio por requisição no melhor caso apresentado.

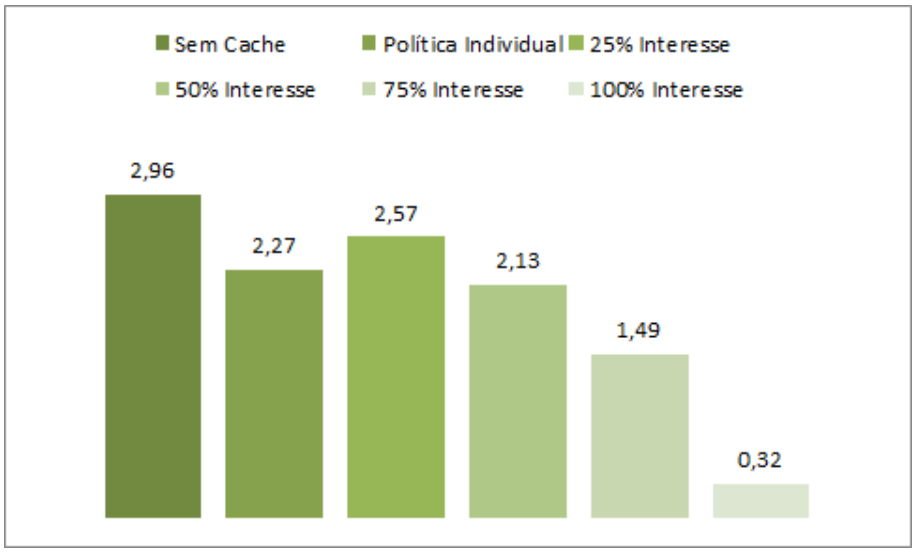

Figura 5.8: Resultados das estimativas de tempo médio por requisição, em segundos, com 4 usuários colaborando, variando entre política de cache individual e colaborativa de 25% a 100% de interesse.

5.3 Considerações Finais

Neste capítulo foram apresentados os resultados obtidos com a realização de quatro baterias de testes. O objetivo destas baterias foi a demonstração do funcionamento do modelo colaborativo proposto e avaliação dos resultados obtidos. Os seguintes pontos foram verificados: validação do funcionamento e eficiência do modelo. Por fim, foi apresentado uma comparação do tempo médio por requisição entre o modelo proposto, uma

política individual e um usuário que faz suas requisições sem nenhum sistema de *cache*. No próximo capítulo será realizado o fechamento do trabalhos.

Capítulo 6

Considerações Finais

Neste capítulo serão apresentadas as considerações finais do trabalho, assim como as limitações e dificuldades encontradas. Por fim, serão apresentados sugestões de trabalhos futuros.

6.1 Conclusão

Neste trabalho foi abordada a crescente demanda por vídeos na Internet. Foi verificado que nas redes de grande porte, como em universidades, a maior parte do tráfego da banda de saída da rede é proveniente de requisições de vídeos. Como visto, uma parcela significativa deste volume de tráfego é destinado a vídeos visualizados mais de uma vez em um curto período de tempo. Ou seja, um sistema de *cache* pode ser uma alternativa para reduzir o volume de dados que trafegam na banda de saída da rede. Tendo como objetivo diminuir as requisições enviadas para fora da rede, este trabalho propôs um modelo de *cache* colaborativo.

Diferentemente dos trabalhos referenciados na bibliografia apresentada, este trabalho propõe que o *cache* seja armazenado em cada usuário que participe do grupo colaborativo. Ainda, caso haja uma falha de *cache* local, é possível recuperar os objetos nos demais membros do grupo sem a necessidade de uma centralização das requisições.

Foi realizada a implementação deste modelo proposto, de forma transparente para a aplicação e que não necessita de qualquer intervenção do usuário. Conforme foi visto, o sistema se encarrega da formação dos grupos colaborativos e realiza a política de substituição quando há um esgotamento do espaço de armazenamento do *cache*. Para efeito de comparação ao sistema proposto, foi implementada uma política individual de *cache*.

Foram executadas 4 baterias de testes como forma de validação da proposta, sendo que cada bateria possuía, em média, 4 simulações distintas. Os resultados finais foram obtidos a partir da média dos resultados entre os membros do grupo. Os seguintes aspectos foram avaliados: funcionamento do sistema colaborativo, eficiência do sistema de *cache* e verificação do tempo de resposta das requisições. Os principais resultados obtidos foram: para um grupo contendo 8 usuários colaborando, a uma taxa de 75% de interesse comum, houve uma redução de 79,02% da quantidade de requisições enviadas para fora da rede, se comparado com a política individual de *cache*; quando analisado o tempo de resposta por requisição, o modelo colaborativo obteve o tempo médio 7 vezes menor que o tempo médio por requisição da política individual.

6.2 Dificuldades Encontradas

Detre as principais dificuldades enfrentadas durante o desenvolvimento deste trabalho, podemos citar:

- Falta de documentação específica quanto ao funcionamento das requisições aos servidores do YouTube. Assim, tudo o que foi necessário para entender como funciona a comunicação com estes servidores foi descoberto através de monitoramento e análise do tráfego HTTP com o YouTube;
- O YouTube trabalha com URLs dinâmicas, o que dificulta a realizacão de *cache* do seu conteúdo. Gastou-se um tempo significativo para descobrir como realizar o mapeamento dos vídeos que estão sendo requisitados;
- Há uma constante mudança no funcionamento do YouTube, o que gerou necessidade de alterações na implementação do modelo proposto. Este foi um dos pontos mais críticos durante o desenvolvimento;
- Recentemente, o YouTube forçou a utilização de HTTPS (do inglês, *Hypertext Transfer Protocol Secure*) nas requisições encaminhadas para os seus servidores. A API (do inglês, *Application Program Interface*) do YouTube que foi utilizada como base para o desenvolvimento do sistema é anterior a esta mudança. Deste modo, o sistema desenvolvido não irá funcionar se as requisições estiverem sob este protocolo; e
- Dificuldade em realizar a simulação do modelo em ambientes virtuais. Foram gastos, aproximadamente, 6 semanas realizando os testes em um ambiente virtual para que houvesse uma maior automação das baterias de testes. Todavia, foram constatados alguns problemas de comunicação entre os membros do grupo no ambiente virtual. Desta forma, todas as baterias de testes foram realizadas em laboratórios com máquinas físicas;

6.3 Trabalhos Futuros

A partir deste trabalho, foi possível encontrar os seguintes pontos a serem melhorados:

- Otimização do módulo colaborativo do sistema. Embora os resultados do tempo de requisição tenham sido satisfatórios, foi estabelecido um alto tempo de espera pelas respostas dos outros membros do grupo. Um mecanismo de adaptação do tempo máximo de espera de acordo com o desempenho do sistema de *cache* em determinado momento, poderia reduzir consideravelmente o valor médio por requisição.
- Verificar o comportamento do sistema com um aumento significativo do número de membros colaborando. Como a troca de mensagens no grupo é realizada pela biblioteca JGroups, é preciso analisar o comportamento desta ferramenta em situações com alta carga de requisições; e
- Realizar a implantação do modelo criado em um ambiente real de funcionamento, como redes de universidades, por exemplo.

Referências

[1] Y. Abdelmalek, A. Abd El Al, and T. Saadawi. Collaborative content caching algorithms in mobile ad hoc networks environment. In *Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management*, pages 311–314. IEEE, 2009. 11

[2] S. Acharya and B. C. Smith. Middleman: A video caching proxy server. *Proc. of NOSSDAY*, 2000. 13

[3] Alexa. Statistics. Disponível em: http://www.alexa.com/topsites/countries/BR. Acessado em 27/10/2014. 14, 18

[4] P. Ameigeiras, JJJ. Ramos-Munoz, J. Navarro-Ortiz, and J.M. Lopez-Soler. Analysis and modelling of youtube traffic. *Transactions on Emerging Telecommunications Technologies*, 23(4):360–377, 2012. 19

[5] G. Barish and K. Obraczka. World wide web caching: Trends and techniques. *IEEE Communications Magazine*, 38:178–184, 2000. 12

[6] L. Braun, A. Klein, G. Carle, H. Reiser, and J. Eisl. Analyzing caching benefits for youtube traffic in edge networks — a measurement-based evaluation. *2012 IEEE Network Operations and Management Symposium*, pages 311–318, 2012. 1, 18

[7] M. Caetano. Uma abordagem colaborativa de cache em redes ad hoc. Master's thesis, Universidade de Brasília, 2008. 12

[8] S. Chen, B. Shen, and S. Wee. Designs of high quality streaming proxy systems. *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, 3:1512–1521, 2004. 16, 17

[9] S. Chen, B. Shen, Y. Yan, S. Basu, and X Zhang. Srb: Shared running buffers in proxy to exploit memory locality of multiple streaming media sessions. *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2004. 16

[10] Chi-Yin Chow, Hong Va Leong, and Alvin T. S. Chan. GroCoca: group-based peer-to-peer cooperative caching in mobile environment. *IEEE Journal on Selected Areas in Communications*, 25(1):179–191, 2007. 12

[11] Marek Chrobak and John Noga. Lru is better than fifo. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 78–81, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics. 8

[12] Cisco. Cisco visual networking index: Forecast and methodology, 2013–2018. Disponível em: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html?CAMPAIGN=VNI+2014&COUNTRY_SITE=us&POSITION=PR&REFERRING_SITE=Press+release&CREATIVE=PR+to+VNI+WP. Acessado em 02/10/2014. 1

[13] Cisco. Statistics. Disponível em: http://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1426270. Acessado em 23/02/2015. iii, iv, 15

[14] C. Cobârzan and L. Boszormányi. Dynamic proxy-cache multiplication inside lans. *Euro-Par 2005 - Lecture Notes in Computer Science*, 3648:890–900, 2005. vii, 13

[15] H. Fabmi, M. Latif, S. Sedigh-Ali, A. Ghafoor, P. Liu, and L. H. Hsu. Proxy servers for scalable interactive video support. *Computer*, 34:54–60, 2001. 17

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol - HTTP 1.1. *Network Working Group*, 1999. 12

[17] A. Finamore, M. Mellia, MM. Munafò, R. Torres, and SG. Rao. Youtube everywhere: Impact of device and infrastructure synergies on user experience. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 345–360, New York, NY, USA, 2011. ACM. 18

[18] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 15–28. ACM, 2007. 1, 20

[19] I. Grigorik. Developers. Disponível em: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching. Acessado em 16/02/2015. 11

[20] M. Gwertzman, J. e Seltzer. World-wide web cache consistency. In *In Proceedings of the USENIX Technical Conference San Diego CA*, pages 141–152, 1996. 9

[21] K. Hosanagar, R. Krishnan, M. Smith, and J. Chuang. Optimal pricing of content delivery network (cdn) services. *Proceedings of the 37th Hawaii International Conference on System Sciences*, 2004. 19

[22] Facts Hunt. Total number of websites & size of the internet as of 2013. Disponível em: http://www.factshunt.com/2014/01/total-number-of-websites-size-of.html. Acessado em 04/03/2015. 6

[23] M. M. Karam. Avaliação e implementação de web cache institucional com foco em urls dinâmicas. Master's thesis, Universidade de Brasília, 2014. vii, 21, 23

[24] M. Kasbekar and V. Desai. Distributed collaborative caching for proxy servers. In *Sixth International World Wide Web Conference*, Santa Clara, California, USA, 1997. 12

[25] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison-Wesley Longman Publishing Co., Inc., 2001. 7, 10

[26] James F. Kurose and Keith Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Pearson, 6nd edition, 2013. 11

[27] J. Liu. Streaming media caching. In *Web Content Delivery*, pages 197–214. Springer US, 2005. vii, 16, 17, 18

[28] R. Malpani, J. R. Lorch, and D. Berger. Making world wide web caching servers cooperate. In *Proceedings of the Fourth International World Wide Web Conference*, page 107–117, Boston, MA, December 1995. 11

[29] Z. Miao and A. Ortega. Scalable proxy caching of video under storage constraints. *IEEE Journal on Selected Areas in Communications*, 20:1315–1327, 2002. 17

[30] J. Oliveira. Uso de caches na web - influência das políticas de substituição de objetos. Master's thesis, Universidade de São Paulo, 2004. 9, 11, 12

[31] L. Plissonneau and E. Biersack. A longitudinal view of http video streaming performance. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, pages 203–214, New York, NY, USA, 2012. ACM. 19

[32] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, December 2003. 7

[33] Lucian Popa, Ali Ghodsi, and Ion Stoica. Http as the narrow waist of the future internet. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 6:1–6:6. ACM, 2010. 1

[34] Red Hat. JGroups - A Toolkit for Reliable Messaging. Disponível em: `http://www.jgroups.org/overview.html`. Acessado em 26/02/2015. 47

[35] Sam Romano and Hala ElAarag. A quantitative study of recency and frequency based web cache replacement strategies. In *CNS '08: Proceedings of the 11th communications and networking simulation symposium*, pages 70–78. ACM, 2008. 7, 8, 9, 35

[36] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. *Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, 3:1310–1319, 1999. 17

[37] I. Steinmacher. Técnicas de web caching e prefetching com prioridades. Master's thesis, Universidade Federal do Rio Grande do Sul, 2004. 9

[38] A. S. Tanenbaum. *Redes de Computadores*. Editora Campus, 1997. 12

[39] A.S. Tanenbaum. *Modern Operating Systems*. Pearson Education, Inc., 3nd edition, 2008. 6

[40] A.S. Tanenbaum and D. Wetherall. *Computer Networks*. Pearson Prentice Hall, 5nd edition, 2011. 4

[41] W. Tavanapong, M. Tran, J. Zhou, and S. Krishnamohan. Video caching network for on-demand video streaming. *GLOBECOM '02*, pages 1723–1727, 2002. 12

[42] R. Tewari, H. Vin, A. Dan, and D. Sitaram. Resource-based caching for web servers. In *Conference on Multimedia Computing and Networking*, pages 191–204. SPIE/ACM, 1998. 16

[43] K. Wu, P. Yu, and J. Wolf. Segment-based proxy caching of multimedia streams. In *Proceedings of the 10th international conference on World Wide Web*, pages 36–44. ACM, 2001. 17

[44] YouTube. Statistics. Disponível em: `http://www.youtube.com/yt/press/statistics.html`. Acessado em 27/10/2014. 18

[45] M. Zink, K. Suh, Y. Gu, and J. Kurose. Characteristics of YouTube network traffic at a campus network – measurements, models, and implications. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 53:501–514, 2009. 1, 11, 20

Apêndice A

JGroups

O JGroups é uma ferramenta de comunicação em grupo, confiável e inteiramente escrita em Java. É baseada em IP *multicast* (embora o TCP também possa ser utilizada como transporte), mas o complementa com confiabilidade e gerência de membros [34].

Confiabilidade inlcui, entre outras coisas:

- Transmissão sem perda de uma mensagem para todos os destinatários (com retransmissão das mensagens não entregues);
- Fragmentação de grandes mensagens em outras menores, e sua reconstrução no lado do destinatário;
- Ordenação de mensagens: mensagens `m1` e `m2` enviadas por `P` serão recebidas por todos os destinatários na mesma ordem; e
- Atomicidade: a mensagem será recebida por todos os destinatários, ou por nenhum.

Gerência de membros inclui:

- Conhecimento de quem são os membros de um grupo; e
- Notificação quando um novo membro entra, quando um membro existente sai ou quando um membro existente falhou.

A tabela A.1 compara o JGroups com outros protocolos de transporte.

	Não confiável	Confiável
Um para um *(unicast)*	UDP	TCP
Um para muitos *(multicast)*	IP *multicast*	JGroups

Tabela A.1: Classificação dos protocolos de comunicação frente ao JGroups.

Apêndice B

Fórmulas para o Cálculo do Atraso Médio

A seguir, estão presentes todas as constantes e fórmulas utilizadas para o cálculo dos atrasos médios por requisição.

B.1 Constantes Utilizadas

Os valores definidos abaixo foram determinados através de medições realizadas no dia 21/2/2015 utilizando a rede UNB Wireless do campus Darcy Ribeiro da UnB.

Taxa de Transferência (TT): 67 KB/s

Tamanho de um trecho do vídeo (TV): 196,6 KB

Tempo de reposta do servidor do YouTube (TR): 24 ms

Atraso para recuperação local (AL): 1,75 ms

Estes valor serão utilizados nos casos a seguir.

B.2 Sem um Sistema de *Cache*

A Fórmula B.1 foi utilizada para calcular o tempo médio por requisição quando não há nenhuma interferência de um sistema de *cache*.

$$\text{Atraso médio} = \frac{TV}{TT} + TR \tag{B.1}$$

B.3 Política Individual de *Cache*

A Fórmula B.2 foi utilizada para calcular o tempo médio por requisição em um sistema que utiliza a política individual de *cache*. Além dos valores definidos na Seção B.1, foram utilizados os seguintes valores, obtidos através das baterias de testes realizadas.

Taxa de acerto local (TL): 23,3%

Taxa de falha (TF): 76,7%

$$\text{Atraso médio} = (TL \times AL) + (\frac{TF \times TV}{TT}) + (TR \times TF) \tag{B.2}$$

B.4 Modelo Colaborativo de *Cache* Proposto

A Fórmula B.3 foi utilizada para calcular o tempo médio por requisição no sistema de *cache* colaborativo proposto. Além dos valores definidos na Seção B.1, foram utilizados os valores presentes na Tabela B.1, obtidos através das baterias de testes realizadas.

	Porcentagem de Interesse			
	25%	50%	75%	100%
Taxa de Acerto Local (TL)	25,16%	25,29%	24,84%	25,08%
Taxa de Acerto Colaborativo (TC)	11,05%	23%	40,46%	72,4%
Taxa de falha (TF)	63,77%	52%	34,69%	2,5%
Atraso da Falha (AF)	1014 ms	1016 ms	1019 ms	1022 ms
Atraso do Acerto Colaborativo (AC)	297 ms	306 ms	276 ms	299 ms

Tabela B.1: Valores obtidos nas baterias de testes usando o modelo de *cache* colaborativo proposto.

$$\text{Atraso médio} = TF \times (AF + TR + \frac{TV}{TT}) + (TL \times AL) + (TC \times AL) \tag{B.3}$$