UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

BRUNO SILVEIRA NEVES

Proposta de algoritmo de cacheamento para *proxies* VoD e sua avaliação usando um novo conjunto de métricas

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Microeletrônica.

Orientador: Prof. Dr. Altamiro Amadeu Susin

Porto Alegre
2015

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Neves, Bruno Silveira

Proposta de algoritmo de cacheamento para *proxies* VoD e sua avaliação usando um novo conjunto de métricas [manuscrito] / Bruno Silveira Neves. – 2015.

153 f.:il.

Orientador: Altamiro Amadeu Susin.

Tese (Doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica. Porto Alegre, BR – RS, 2015.

1.Vídeo 2.Proxy 3.Algoritmo de cacheamento. I. Susin, Altamiro Amadeu. II. Proposta de algoritmo de cacheamento para *proxies* VoD e sua avaliação usando um novo conjunto de métricas.

AGRADECIMENTOS

Agradeço:

- À Deus, por ter a chance de concluir este curso de doutorado, pela saúde, paz para concentração, proteção, luz, força e coragem para trabalhar e lutar.
- Em especial, ao professor, Altamiro Amadeu Susin, pela amizade e contínua atenção, pela orientação e ensinamentos e pelo incentivo e compreensão, transmitidos de forma bem peculiar.
- Ao Anderson, meu ex orientando do Curso de Graduação em Engenharia da Computação da Unipampa e colaborador de pesquisa, pelas inúmeras contribuições dadas a este e outros trabalhos realizados em parceria nestes últimos anos.
- A minha querida filhinha, Naila, por ter chegado em minha vida durante este curso, abastecendo-me assim de felicidade e orgulho em um momento importante. Agradeço também a ti filha pelo carinho e compreensão por eu não poder estar presente e participativo em todos os momentos de sua vida até então. Só assim, pude honrar, em paralelo a dedicação dada a ti, meus compromissos de estudo e trabalho assumidos antes da tua chegada.
- A minha esposa, Mariléia, pelo companheirismo, amor, carinho, estímulo, apoio, dedicação, paciência e compreensão. Fazer meu doutorado em paralelo ao trabalho, distante das nossas famílias e com dedicação a nossa filha certamente não foi mais simples ou fácil do que exercer as tarefas de mãe, professora e mestranda. Acredito que nossa situação nos motivou mutuamente a batalhar ainda mais pelo nosso aprimoramento.
- Aos meus familiares que, mesmo distantes fisicamente, forneceram apoio psicológico importante para este trabalho. Aos meus pais, Antônio e Sandra, e à minha sogra, Zilda, obrigado por pedir a Deus força e luz para que eu pudesse trabalhar mesmo nos momentos mais difíceis.
- Aos coordenadores e demais dirigentes do PGMicro da UFRGS, pela oportunidade de realizar o curso.• Aos professores Lisandro, Marinho e Bampi, pelo aceite em participar da banca de qualificação, pelas contribuições/sugestões fornecidas para aperfeiçoamento do trabalho e pela aprovação.
- Aos professores que aceitaram o convite para participar da banca examinadora desta tese.

- A todos os professores do curso com os quais tive contato, pela atenção fornecida. Agradeço em especial àqueles com quem tive aulas, pelo empenho em transmitir os conteúdos da melhor forma. O aprendizado será ainda muito útil em futuras questões de interesse pessoal e profissional.
- Aos colegas e amigos que tive e formei, pelo convívio agradável, pelo espírito de fraternidade em todos os momentos e pelas experiências trocadas. Em especial, agradeço aos dois colegas (não me recordo o nome) que me ajudaram em meu apartamento em PoA quando estive doente. Sou muito grato pelo carinho e senso de humanismo de vocês que mesmo sem me conhecer agiram como grandes amigos no momento em que precisei.
- A alguns membros/setores da Unipampa pela compreensão e oportunidade de obter meu afastamento, mesmo que apenas na reta final da conclusão deste curso.
- Enfim, obrigado a todos que, de alguma forma, direta ou indiretamente, contribuíram para a realização desta pesquisa e para a conclusão desta Tese.

Dualidade

“Em sonhos adolescentes
Quisera eu modificar o mundo
Esforcei-me, reuni informações
Hoje sei que falta muito
Embora me sentindo grande
Sou como um grãozinho de areia
No deserto deste universo
Resta-me a condição de pertinência ao planeta terra
E, como um passarinho, posso carregar a água com meu pequeno bico
Para apagar o fogo na floresta
É uma dualidade: às vezes me sinto-me importante, cheio de condições
Mas isso não basta para dizer que não preciso de ninguém
Junto, tenho noção de minha pequenez
Contudo, sempre posso ajudar alguém
O difícil é conciliar essas duas condições
Sei que aí que mora o segredo
Para não me tornar arrogante ou insignificante
Quero ser equilibrado
Pensar alto, com os pés na terra
Gosto de ser essa dualidade
Preciso do outro, também tenho a consciência
Que muitos precisam de mim
O importante é ser um indivíduo completo
E alimentar-me de fraternidade
Pois sei que existo na medida do outro.”

— SANDRA MARA SILVEIRA NEVES

RESUMO

Atualmente, o serviço digital conhecido como Vídeo sob Demanda - Video on Demand (VoD) - está em ascensão e costuma requerer uma quantidade significativa de recursos físicos para a sua implementação. Para reduzir os custos de operacionalização desse serviço, uma das alternativas comumente usada é o emprego de *proxies* que cacheiam as partes mais importantes do acervo, com o objetivo de atender a demanda para esse conteúdo no lugar do servidor primário do sistema VoD. Nesse contexto, para melhorar a eficiência do *proxy*, propõe-se neste trabalho um novo algoritmo de cacheamento que explora o posicionamento dos clientes ativos para determinar a densidade de clientes dentro de uma janela de tempo existente em frente de cada trecho de vídeo. Ao cachear os trechos de vídeo com maior densidade em frente a eles, o algoritmo é capaz de alcançar um alto desempenho, em termos de taxa de acertos para as requisições recebidas pelo *proxy*, durante intervalos de alta carga de trabalho. Para avaliar esta abordagem, o novo algoritmo desenvolvido foi comparado com outros de natureza semelhante, fazendo uso tanto de métricas tradicionais, como a taxa de acertos, como também de métricas físicas, como, por exemplo, o uso de recursos de processamento. Os resultados mostram que o novo algoritmo explora melhor a banda de processamento disponível na arquitetura de base do *proxy* para obter uma taxa de acertos maior em comparação com os algoritmos usados na análise comparativa. Por fim, para dispor das ferramentas necessárias para construir essa análise, produziu-se uma outra contribuição importante nesse trabalho: a implementação de um simulador de proxies VoD que, até onde se sabe, é o primeiro a possibilitar a avaliação do hardware utilizado para implementar essa aplicação.

Palavras-chave: Vídeo. Proxy. Algoritmo de cacheamento

Proposal of Caching Algorithm for VoD *proxy* Implementation and Its Evaluation Including a New Set of Metrics for Efficiency Analysis

ABSTRACT

Today, Video on Demand (VoD) is a digital service on the rise that requires a lot of resources for its implementation. To reduce the costs of running this service, one of the commonly used alternatives is using *proxies* that cache the most important portions of the collection in order to meet the demand for this content in place of the primary server of the VoD system. In this context, to improve the efficiency of *proxy*, we proposed a novel caching algorithm that explores the positioning of the active clients to determine the density of clients inside a time window existing in front of each video chunk. By caching the video chunks with the greater density in front of them, the algorithm is able to achieve high performance, in terms of the hit ratio for the requests received by the *proxy*, during periods of high workload. To better evaluate our approach, we compare it with others of similar nature, using both traditional metrics like hit rate, as well as physical metrics, such as the use of processing resources. The results show that the new algorithm exploits the processing bandwidth available in the underlying architecture of the *proxy* for obtaining a larger hit rate in comparison to the other algorithms used in the comparative analysis. Finally, to dispose of the necessary tools to perform this analysis, we produced another important contribution in this work: the implementation of a VoD *proxy* simulator that, to the best of our knowledge, is the first one to enable the evaluation of the hardware used to implement this application..

Keywords: Video. Proxy. Caching algorithm.

LISTA DE FIGURAS

Figura 1.1 - Sistema VoD básico onde o servidor de vídeo e a espinha dorsal da rede tendem a ficar sobrecarregados com o fornecimento do serviço para os clientes 17
Figura 1.2 - Redução da carga sobre o servidor principal e a rede através da colaboração entre clientes 17
Figura 1.3 - Estratégia para uso de *proxies* VoD 18
Figura 2.1 - Organização do vídeo 23
Figura 2.2 - Cálculo da prioridade de cacheamento no algoritmo CCVC 26
Figura 2.3 - Cálculo da prioridade de cacheamento no algoritmo CC 27
Figura 2.4 - Cálculo da prioridade de cacheamento no algoritmo RTBC 30
Figura 3.1 - Comparativo: segmentos (tamanho fixo) vs. sequências (tamanho variável) 35
Figura 3.2 - Cálculo da prioridade de cacheamento no CARTE 37
Figura 4.1 - Arquitetura do SIMPRO 41
Figura 4.2 - Trecho inicial de um vetor de carga para simulação (arquivo SIV) 42
Figura 4.3 - Rodada de serviço: conjunto de operações executadas pelo *proxy* VoD, em cada unidade de tempo, para servir seus clientes 45
Figura 4.4 - Fluxo de dados em uma rodada de serviço 46
Figura 4.5 - Ação executada pelo gerador de *handler* para clientes ativos 47
Figura 4.6 - Criação de um nó para a árvore de prioridades 48
Figura 4.7 - Processo de obtenção do endereço físico a partir do número do bloco 49
Figura 4.8 - Abastecimento das sequências de vídeo de maior prioridade de cacheamento 51
Figura 4.9 - Fluxo de simulação no modo funcional 54
Figura 4.10 - Fluxo de controle da execução das rodadas de serviço 57
Figura 4.11 - Fluxo de simulação no modo de tempo real 59
Figura 4.12 - Lotes de E/S concentrados para reduzir taxa de falhas na cache 61
Figura 4.13 - Fluxo de simulação no modo de tempo discreto 64
Figura 4.14 - Sintaxe para configuração de uma memória de cache 66
Figura 4.15 - Sintaxe para configuração de um nível de cache unificado (compartilhado) no Simplescalar 67
Figura 4.16 - Sintaxe para configuração da latência de acesso a uma cache no Simplescalar 67
Figura 4.17 - Sintaxe para configuração de preditor de desvios no Simplescalar 68
Figura 4.18 - Sintaxe do Simplescalar para configuração da latência de acesso à memória RAM 69
Figura 4.19 - Sintaxe do Simplescalar para configuração da fila de instruções (*load* e *store*) para acesso à memória RAM 70
Figura 4.20 - Fluxo interno de contagem dos ciclos durante uma rodada de serviço no modo de tempo discreto 71
Figura 4.21 - Validação dos dados produzidos pelo SIMPRO no modo funcional 73
Figura 4.22 - Validação dos dados produzidos pelo SIMPRO nos modos de tempo real e discreto 76
Figura 5.1 - Desempenho dos algoritmos de cacheamento: resultados para taxa de acertos 82
Figura 5.1 - Desempenho dos algoritmos de cacheamento: resultados para taxa de acertos (cont.) 83
Figura 5.1 - Desempenho dos algoritmos de cacheamento: resultados para taxa de acertos (cont.) 84
Figura 5.2 - Desempenho dos algoritmos de cacheamento: resultados para tempo de serviço 86
Figura 5.2 - Desempenho dos algoritmos de cacheamento: resultados para tempo de serviço (cont.) 87
Figura 5.2 - Desempenho dos algoritmos de cacheamento: resultados para tempo de serviço (cont.) 88
Figura 5.3 - Uso de recursos: resultados para tempo de processador 91
Figura 5.3 - Uso de recursos: resultados para tempo de processador (cont.) 92
Figura 5.4 - Uso de recursos: resultados para tempo de memória 93
Figura 5.4 - Uso de recursos: resultados para tempo de memória (cont.) 94
Figura 5.4 - Uso de recursos: resultados para tempo de memória (cont.) 95
Figura 5.5 - Curva de abandono dos clientes VoD: percentual de fechamento cumulativo das sessões para cada instante de exibição do vídeo (baseada em (YU et al., 2006)) 98
Figura 5.6 - Desempenho sob abandono de sessão: taxa de acertos para variação do tamanho dos vídeos e da memória 100

Figura 5.7 - Taxa de acertos para variação do tamanho da janela de tempo 103
Figura 5.7 - Taxa de acertos para variação do tamanho da janela de tempo (cont.)..... 104
Figura 5.8 - Funcionamento do scan fino: (a) estado inicial com a definição dos primeiros valores para L, C e R. (b) e (c) iterações do scan fino com deslocamento de L, C e R. (c) estado final com $L \leq C$ and $C \geq R$..... 108
Figura B.1 - Distribuição de Poisson para os diferentes valores de λ usados neste trabalho. Nos Gráficos da figura, o número total de requisições recebidas pelo *proxy* foi 8000 127
Figura B.1 - Distribuição de Poisson para os diferentes valores de λ usados neste trabalho. Nos Gráficos da figura, o número total de requisições recebidas pelo *proxy* foi 8000 (cont.)..... 128
Figura B.1 - Distribuição de Poisson para os diferentes valores de λ usados neste trabalho. Nos Gráficos da figura, o número total de requisições recebidas pelo *proxy* foi 8000 (cont.)..... 129
Figura B.1 - Distribuição de Poisson para os diferentes valores de λ usados neste trabalho. Nos Gráficos da figura, o número total de requisições recebidas pelo *proxy* foi 8000 (cont.)..... 130
Figura B.2 - Distribuição de Zipf para os diferentes valores de θ usados neste trabalho. Nos Gráficos da figura, NV = 100 e o número total de requisições recebidas pelo *proxy* foi 8000..... 130
Figura B.2 - Distribuição de Zipf para os diferentes valores de θ usados neste trabalho. Nos Gráficos da figura, NV = 100 e o número total de requisições recebidas pelo *proxy* foi 8000 (cont.) 131
Figura B.2 - Distribuição de Zipf para os diferentes valores de θ usados neste trabalho. Nos Gráficos da figura, NV = 100. Sob estas configurações, o número total de requisições recebidas pelo *proxy* foi 8000 (cont.) 132
Figura G.1 - Pasta com arquivos de instalação do Code::Blocks 145
Figura G.2 - Interface do Code:Blocks para instalação e configuração de *plug-ins*: configuração do *plug-in* cbp2make 146
Figura G.3 - Pasta release com códigos binários do cbp2make para diferentes sistemas operacionais 147
Figura G.4 - Visão geral sobre os *targets* do SIMPRO implementados sobre o Code::Blocks..... 148
Figura G.5 - Interface do Code::Blocks para configuração das definições gerais de programa para os *targets* do SIMPRO..... 149
Figura G.6 - Exemplificação da interface para entrada das definições de programa específicas para o *target funcional*_CARTE do SIMPRO..... 150
Figura G.7 - Opções gerais de configuração do compilador para síntese do SIMPRO 150

LISTA DE TABELAS

Tabela 2.1 - Taxonomia para divisão lógica dos vídeos 22
Tabela 4.1 - Conjunto de parâmetros de configuração da carga de trabalho para excecussão das simulações em qualquer modo 53
Tabela 4.2 - Conjunto de parâmetros de configuração do sistema para excecussão das simulações nos modos funcional e de tempo real 55
Tabela 4.3 - Conjunto complementar de parâmetros de configuração do sistema para excecussão das simulações no modo discreto 65
Tabela 4.4 - Parâmetros de carga para validação funcional 74
Tabela 4.5 - Configuração da arquitetura para simulações no modo discreto 76
Tabela 5.1 - Parâmetros utilizados para configuração do SIMPRO 78
Tabela 5.2 - Resultados gerais sobre taxa de acertos e tempo de serviço 90
Tabela 5.3 - Resultados gerais sobre uso de recursos 96
Tabela 5.4 - Resultados gerais para cenários com saída prematura de clientes 99
Tabela G.1 - Descrição do código desenvolvido neste trabalho 144
Tabela G.2 - Descrição do código desenvolvido neste trabalho 151

LISTA DE ABREVIATURAS E SIGLAS

AIR	Abstrator de Interface de Rede
API	Application Program Interface
ARM	Advanced RISC Machine
CARTE	Current demAnd Rather Than futurE
CBR	Constant Bit Rate
CCVC	Collapsed Cooperative Video Cache
DV	Duração dos Vídeos
DS	Duração das Simulações
Gbps	Giga bits por segundo
HD	High-Definition
IPCM	Intel Performance Counter Monitor
ISS	Instruction Set Simulator
JSS	JT Space Scanner
L1	Memória Cache de Nível 1
L2	Memória Cache de Nível 2
L3	Memória Cache de Nível 3
LB	Largura de Banda Máxima para o canal servidor-*proxy*
LFU	Least Frequently Used
LRU	Least Recently Used
Mbps	Mega bits por segundo
MIPS	Microprocessor without Interlocked Pipeline Stages
NC	Número de Clientes
NV	Número de Vídeos
PISA	Portable Instruction Set Architecture
POSIX	Portable Operating System Interface
RAM	Random Access Memory
SIMPRO	Simulador de *proxy*
SIV	Simulation Input Vector
SPARC	Scalable Processor Architecture
TCP	Transmission Control Protocol
TM	Tamanho da Memória
TOE	TCP Offload Engine

TC	Tamanho dos Segmentos
VoD	Video on Demand
VCR	Video Casset Recorder
ZC	Coeficiente Zipf

SUMÁRIO

1 INTRODUÇÃO 14

2 TRAJETÓRIA DOS ALGORITMOS DE CACHEAMENTO 20

2.1 Algoritmo CCVC 22

2.2 Algoritmo CC 26

2.3 Algoritmo RTBC 28

3 ALGORITMO PROPOSTO 32

3.1 Organização de vídeo 33

3.2 Lógica de cacheamento 34

4 AMBIENTE EXPERIMENTAL 39

4.1 Motivação para a construção de um novo simulador 39

4.2 Fluxo de Síntese do Simulador 40

4.3 Fundamentos do Processo de Medição 44

4.4 Fluxo de Simulação 45

4.5 Modos de Simulação 52

4.5.1 Modo Funcional 52

4.5.2 Modo de Tempo Real 58

4.5.3 Modo de Tempo Discreto 63

4.6 Validação dos Dados de Desempenho 72

4.6.1 Validação do Modo Funcional 72

4.6.2 Validação dos Modos de Tempo Real e Discreto 74

5 RESULTADOS E DISCUSSÕES 77

5.1 Parâmetros de Carga e de Funcionamento para a Avaliação do *Proxy* 77

5.2 Avaliação da Taxa de Acertos e Tempo de Serviço para a Execução dos Algoritmos de Cacheamento 81

5.3 Avaliação dos Recursos Computacionais Usados por cada Algoritmo 90

5.4 Impactos Produzidos pela Saída Prematura dos Clientes 97

5.5 Análise da Relação entre o Tamanho da Janela de Tempo e a Carga de Trabalho 101

5.6 Procedimentos para configurar o CARTE 107

4 CONCLUSÃO 110

REFERÊNCIAS 113

APÊNDICE A <FORMATO DO ARQUIVO SIV> 121

APÊNDICE B < CURVAS PARA AS DISTRIBUIÇÕES DE POISSON E ZIPF: DIFERENTES RESULTADOS PARA A VARIAÇÃO DOS PARÂMETROS DE CONTROLE DAS DIS-TRIBUIÇÕES> 127

APÊNDICE C < EXTRATO DE CÓDIGO FONTE: USO DO IPCM PARA COLETA DE DADOS DE DESEMPENHO> 133

APÊNDICE D < SCRIPT PARA INSTALAÇÃO DO SIMPLESCALAR COM *TARGET* PISA> 134

APÊNDICE E < ARQUIVO .CFG PARA CONFIGURAÇÃO DO *TARGET* PISA> 138

APÊNDICE F < SCRIPT PARA CONFIGURAÇÃO E EXECUÇÃO DO SOFTWARE DO *PROXY*> 139

APÊNDICE G < DESCRIÇÃO DO PACOTE DE SOFTWARE E TUTORIAL PARA CONFIGURAÇÃO EM MODO GRÁFICO COM O CODE::BLOCKS> 144

G.1 Code::Blocks e cbp2make - Instalação e Configuração 144

G.2 Configuração do SIMPRO usando o Code::Blocks 147

REFERÊNCIAS 152

1 INTRODUÇÃO

A tecnologia digital ainda é recente se comparada ao tempo em que a tecnologia analógica dominou o segmento da produção de artefatos eletrônicos. Com o surgimento da tecnologia digital, gradativamente, diferentes nichos da eletrônica analógica passaram por uma transição para a então nova tecnologia, que possui inúmeras vantagens sobre a tecnologia anterior, tais como: maior robustez, menor custo e maior qualidade dos serviços oferecidos ao usuário final.

Com o passar dos anos, a evolução digital no campo das redes de comunicação de dados possibilitou que diferentes produtos digitais fossem oferecidos através de uma única mídia, criando assim o que se chama hoje de convergência digital. Entre os benefícios proporcionados por essa convergência à vida cotidiana estão os serviços relacionados à saúde, educação, comércio e entretenimento, entre outros. Um aspecto importante para que estes benefícios possam estar ao alcance do maior número de pessoas é o custo de implantação da infra-estrutura de rede convergente para suporte a todos esses serviços.

Neste contexto, a popularização do acesso à banda larga, que é um dos objetivos estratégicos de maior interesse governamental e também não governamental, tem requerido investimentos na ordem de bilhões para custear todo conjunto de recursos físicos e humanos necessários para disponibilizar e manter toda a infra-estrutura convergente (e o amplo espectro de serviços convergentes que sobre ela são ofertados) (USTELECOM, 2015). Em meio a este cenário, aplicações multimídia, em especial aquelas que lidam com transmissão de vídeo, por operarem com grandes volumes de dados, tendem a consumir significativamente recursos da malha convergente de rede. A exemplo disso, estimativas recentes indicam que a soma de todas as formas de tráfego de vídeo, incluindo TV digital, vídeo sob demanda (Video on Demand - VoD) e transmissões em tempo não real, como compartilhamento de mídia, irá representar 86% de todo o tráfego de dados global sobre a rede em 2016 (CISCO, 2011). A mesma fonte também estima que o fornecimento de vídeo sob demanda representará 54% desse tráfego total de vídeo e o tráfego devido ao VoD irá triplicar até 2016. Isso sugere que a transmissão de VoD irá precisar de um suporte mais adequado para a sua realização, pois, caso contrário, o custo necessário para sustentar o futuro tráfego de vídeo tende a ficar tão alto quanto o investimento realizado para popularização da banda larga.

Do ponto de vista funcional, o objetivo principal desta aplicação é oferecer aos usuários as facilidades de um serviço de vídeo locadora virtual, através do qual os usuários interagem com o sistema para solicitar e receber um fluxo de vídeo que é remetido a partir de

um servidor remoto, conforme ilustra a figura 1.1. Todavia, com esta organização mais básica, a escalabilidade do sistema é baixa, uma vez que cada cliente que se conecta ao sistema para receber um fluxo de vídeo ocupa uma parte da banda de processamento suportada tanto pelo servidor de vídeo principal como pela espinha dorsal da rede.

Frente a essa limitação da organização básica para o serviço VoD, diferentes alternativas tem sido propostas para alavancar a escalabilidade do sistema. Estas alternativas se reúnem em dois grupos principais.

No primeiro deles, o princípio básico de sustentação do serviço de VoD consiste em explorar tanto a largura de banda dos clientes como a suas capacidades de armazenamento de tal modo que, à medida que alguns clientes iniciam o recebimento de fluxos de vídeo a partir de um servidor principal, estes clientes passam a se tornar provedores (servidores de menor porte) para outros clientes que vierem a solicitar, a curto prazo, o mesmo fluxo de vídeo, conforme ilustra a figura 1.3 reduzindo assim a carga sobre a dorsal da rede e sobre o servidor principal. Neste cenário, enquanto o cliente recebe através de um canal lógico o fluxo de vídeo para seu próprio consumo, ele também envia, através de outro canal, os trechos de vídeo disponíveis em sua memória local para outros clientes que acessarão, a seguir, os mesmos trechos.

No segundo grupo, a estratégia usada para amortizar os impactos causados pelo VoD sobre os recursos de rede e, ao mesmo tempo, prover uma melhora da qualidade do serviço consiste em empregar *proxies* próximos às redes locais dos clientes. Cada um desses *proxies* atuam como um servidor secundário armazenando as informações mais acessadas para fornecer essa informação a seus clientes (na área de cobertura do *proxy*) em lugar do servidor de vídeo principal, como mostra a figura 1.2.

O uso de *proxies*, ao mesmo tempo em que evita o congestionamento na rede, também complementa a capacidade de processamento de um servidor de vídeo. Por outro lado, o custo dos *proxies* também se soma ao montante total do custo de implantação do serviço VoD. Consequentemente, a eficiência do *proxy* constitui um parâmetro fundamental para determinar a relação custo-benefício inerente ao serviço de distribuição de vídeo. Sob esta conjuntura, como demonstrado por uma lista extensa de publicações nessa área (GOLREZAEI et al., 2012), (WU; YU; WOLF, 2001), (SUN et al., 2014), (ABEYWICKRAMA; WONG, 2015), (BORST; GUPTA; WALID, 2010), (BEN ABDESSLEM; LINDGREN, 2014), (LING et al., 2014), (POURMIR; RAMANATHAN, 2014), (PARK; CHOI; LEE, 2014), um dos aspectos mais importantes para a eficiência do *proxy* VoD é o algoritmo de cacheamento (também

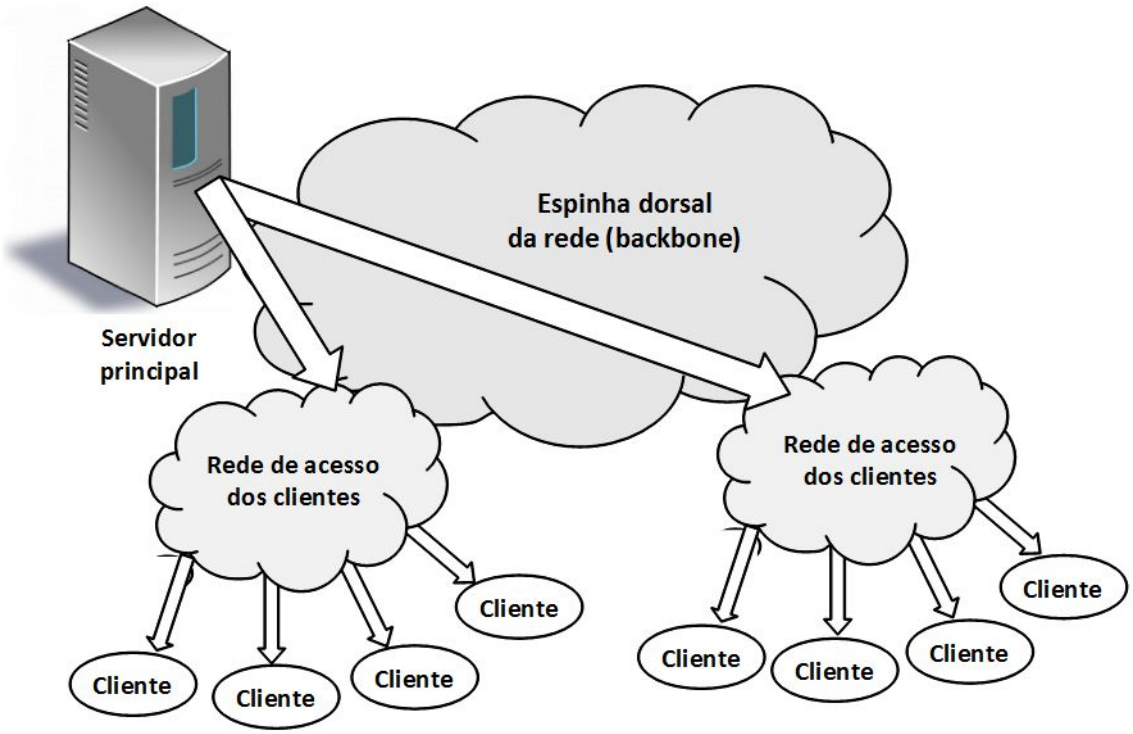
denominado algoritmo de substituição) usado para determinar o conteúdo preferencial a ser mantido na memória do *proxy*.

A ampla maioria dos algoritmos de cacheamento existentes, tais como (LI; CHEN, 2009), (CHIU et al., 2011), (LI; LI; JIANG, 2011), (FAMAEY et al., 2013), (YU et al., 2006) e (YU et al., 2006), usa informações sobre o comportamento dos clientes VoD no passado para estimar o comportamento dos clientes no futuro e, dessa forma, fazer a melhor escolha no que diz respeito à definição do conteúdo a ser mantido na memória para servir esses clientes. No que diz respeito ao comportamento dos clientes VoD são considerados aspectos como:

- O padrão de acesso em cada horário do dia, cada dia da semana, cada semana de um mês, etc.
- O fenômeno conhecido “Multidão de flashs"(“*Flash Crowds*"), sob o efeito dos quais alguns vídeos de um acervo tem sua popularidade subitamente aumentada. Os eventos que comumente contribuem para a causa deste fenômeno são o lançamento de novos vídeos, a divulgação através de propagandas e o histórico de recomendações dos clientes para os itens de um acervo.
- A velocidade com que a popularidade se modifica para cada item de um acervo ao longo do tempo, sendo esta velocidade diferente para itens menos populares e mais populares.
- O desvio provocado pelos clientes sobre o fluxo de reprodução dos vídeos, através da execução de operações do tipo VCR (*Video Cassette Recording*), tais como: “avanço", “retrocesso"e “pausa". Em alguns casos, os instantes de execução (dentro do vídeo) e os respectivos códigos das operações VCR podem ser iguais ou semelhantes ao longo dos acessos realizados por diferentes clientes que assistiram a um mesmo vídeo. Como exemplo, clientes de um filme de ação podem preferir saltar sobre os trechos menos dinâmicos do filme.

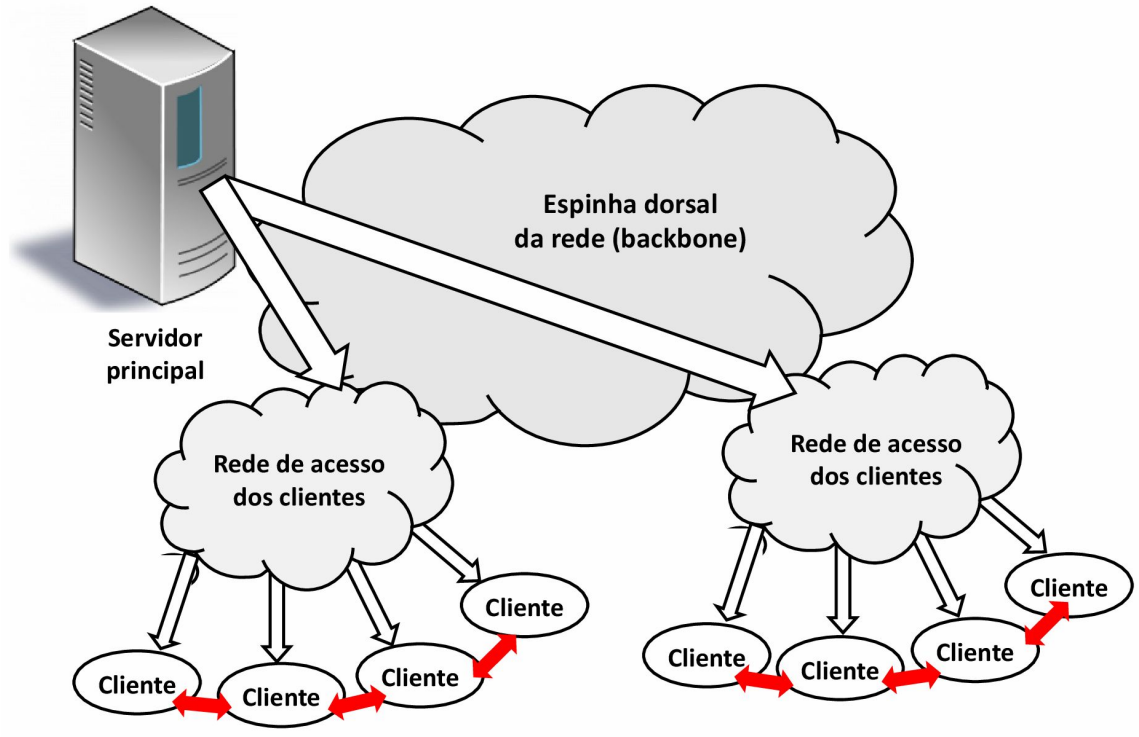
A estratégia de considerar o comportamento passado dos clientes pode, contudo, não ser a melhor alternativa para o *proxy* durante cenários de alta carga de trabalho, uma vez que, sob essas circunstâncias, o algoritmo de cacheamento tende a necessitar de uma estimativa mais sólida e segura para determinar quais trechos de vídeo terão um maior volume de acesso em curto prazo. Assim, durante os períodos críticos de operação, se as tendências de comportamento criadas com base no histórico do padrão de acesso não forem precisas, o *proxy* tende a não contribuir da forma esperada para a redução do consumo de banda na rede e no servidor principal VoD.

Figura 1.1 - Sistema VoD básico onde o servidor de vídeo e a espinha dorsal da rede tendem a ficar sobrecarregados com o fornecimento do serviço para os clientes



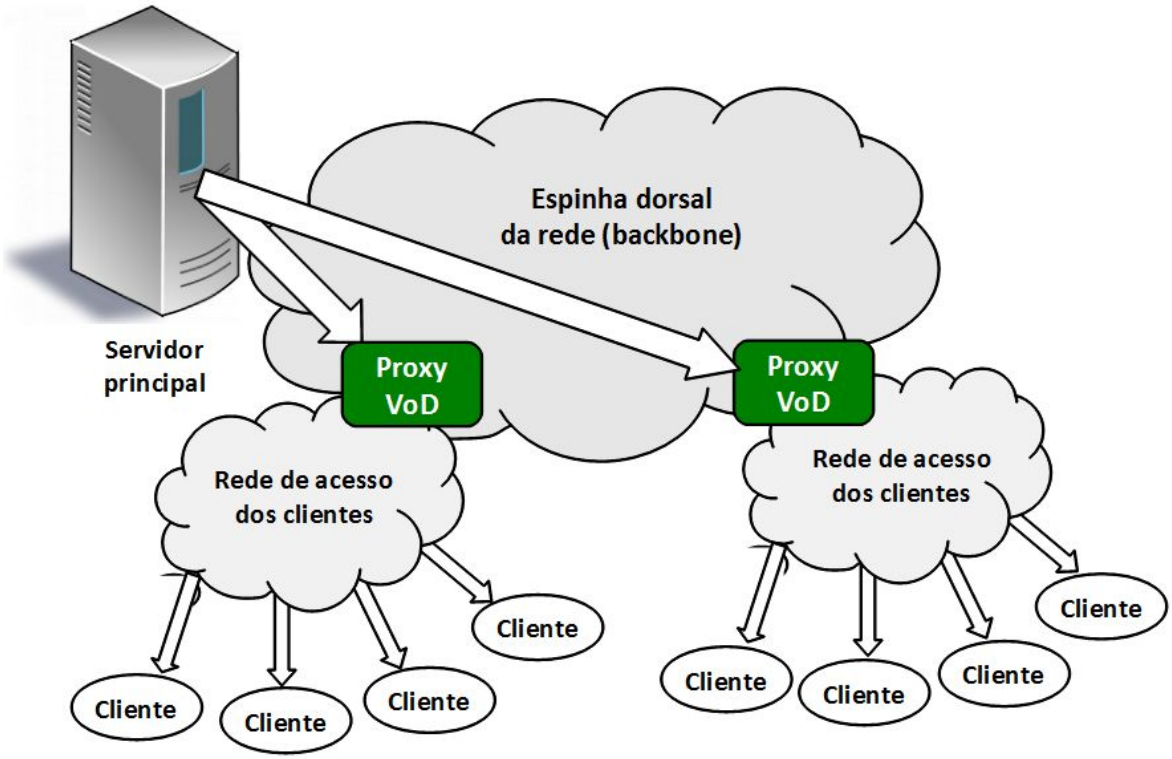
Fonte: do autor (2015).

Figura 1.2 - Redução da carga sobre o servidor principal e a rede através da colaboração entre clientes



Fonte: do autor (2015).

Figura 1.3 - Estratégia para uso de *proxies* VoD



Fonte: do autor (2015).

Por outro lado, alguns trabalhos recentes (HONG; DE VLEESCHAUWER; BACCELLI, 2010; WU et al., 2012) vêm concentrando sua atenção em um novo paradigma (denominado de “olhar à frente" no escopo deste estudo) para guiar as escolhas feitas pelo algoritmo de cacheamento. Levando em consideração a estrutura intrínseca de acesso linear dos fluxos de vídeo, essas novas estratégias propõem que as decisões de cacheamento devam se basear na observação do posicionamento atual dos clientes ativos[1] em cada vídeo, como forma de calcular precisamente o volume da demanda efetiva que esses clientes irão criar para cada trecho de vídeo num futuro próximo, não superior ao tempo que estes clientes levarão para finalizar suas sessões de vídeo. Assim, as escolhas feitas pelo algoritmo de cacheamento são estritamente baseadas nas requisições dos clientes atuais, constituindo, portanto, uma base mais sólida para determinar quais partes de cada vídeo devem ser mantidas na memória do *proxy* de modo a obter uma eficiência maior quando o número de clientes simultâneos é alto.

Todavia, identificou-se um aspecto fundamental para melhorar a lógica de cacheamento empregada por esses algoritmos e, com ênfase nesse aspecto, foi criada uma nova e mais eficiente estratégia de cacheamento, capaz de alcançar uma taxa de acertos

[1] Clientes conectados ao *proxy* para receber um fluxo de vídeo.

significativamente mais alta do que outras propostas que também seguem o paradigma "olhar à frente". Nesse contexto, enquanto as abordagens prévias empregam como um critério de cacheamento, por exemplo, o número total de clientes que irão acessar um determinado trecho de vídeo, a nova estratégia criada leva em consideração a densidade de clientes existentes em uma região limitada à frente[2] de cada trecho, prevenindo assim que os clientes que estão distantes de um dado trecho possam interferir no cálculo da prioridade de cacheamento deste trecho.

Por fim, para avaliar este novo algoritmo proposto, foi também desenvolvido um simulador capaz não apenas de analisar a eficiência usando métricas convencionais, como, por exemplo, a taxa de acertos, mas também de avaliar o uso dos recursos computacionais do hardware de base para o *proxy*. Desta forma, o simulador possibilita determinar como o hardware é capaz de restringir a escalabilidade do sistema, especialmente levando em consideração cenários onde a carga de trabalho é alta.

Assim, para elencar resumidamente, o presente material bibliográfico reúne e apresenta três principais contribuições:

1. A criação de um novo e também mais escalável algoritmo para cacheamento de vídeo, voltado mais especificamente para uso em cenários de alta carga de trabalho de um *proxy* VoD.
2. O desenvolvimento de um simulador capaz de identificar os gargalos físicos da arquitetura de um *proxy* de VoD.
3. Uma avaliação comparativa mais profunda da escalabilidade fornecida por diferentes algoritmos de cacheamento.

O restante desse texto está organizado da seguinte forma. O capítulo 2 apresenta brevemente o contexto da trajetória dos algoritmos de cacheamento de vídeo antes e depois do surgimento do paradigma "olhar à frente". O capítulo 3 descreve a solução para cacheamento de vídeo proposta nesse trabalho. O capítulo 4 apresenta o simulador construído para avaliar o desempenho do novo algoritmo criado. Os resultados da avaliação experimental do algoritmo criado são apresentados e discutidos no capítulo 5. O capítulo 6 conclui esse estudo, destacando as principais descobertas dessa pesquisa e apontando as direções futuras para ela.

[2] Em direção ao início do vídeo.

2 TRAJETÓRIA DOS ALGORITMOS DE CACHEAMENTO

Os algoritmos de cacheamento para vídeo (MA; BARTOS; BHATIA, 2011; PASSARELLA, 2012) têm sido amplamente usados em *proxies* de vídeo sob demanda com o objetivo de fazer a melhor escolha na definição do conteúdo preferencial a ser mantido na memória de vídeo do *proxy*. Nesse cenário, o melhor conteúdo para armazenamento no *proxy* é aquele que permitirá ao *proxy* atender o maior número de requisições geradas pelos clientes do sistema VoD. Quando os clientes não podem ser servidos pelo *proxy*, eles tendem a enfrentar um atraso maior para receber o serviço, uma vez que o conteúdo requisitado precisa ser obtido junto ao servidor principal. De um modo geral, cada acesso ao servidor principal contribui para: (1) reduzir a largura de banda de processamento disponível nesse servidor e (2) causar congestionamento na rede.

Um problema parecido é abordado pelos algoritmos de replicação (PASSARELLA, 2012), (YANG; HAJEK, 2014), (LEE; LEU; CHEN, 2014), (HO; YU; LEE, 2011), (VIJENDRAN; THAVAMANI, 2012), (SAVI et al., 2014), (LO; SU, 2014), (REN et al., 2014), (VO et al., 2014), (BRUNEAU-QUEYREIX et al., 2014), (APPLEGATE et al., 2010) que atuam criando réplicas do conteúdo disponível no servidor principal e as armazenando em servidores substitutos. Porém, ao contrário do que ocorre no contexto dos algoritmos de cacheamento, no contexto da replicação, as políticas usadas para determinar quais objetos de vídeo devem ser replicados e em quais servidores as réplicas devem ser criadas não levam necessariamente em consideração as demandas criadas pelos clientes como o único (ou primeiro) critério de decisão (PATHAN; BUYYA; VAKALI, 2008). Mais comumente, aspectos como o balanceamento de carga entre os servidores substitutos, em geral, guiam a implementação das políticas de replicação (VINAY et al., 2011).

Cabe observar aqui que o motivo da apresentação desta distinção entre as duas classes de algoritmos reside em descrever mais claramente o escopo do presente estudo, o qual foca na criação e na análise de algoritmos de cacheamento de vídeo comumente usados em *proxies* VoD. Adicionalmente, o alvo principal desse estudo é a abordagem centralizada[3] onde é assumido que as decisões feitas pelo algoritmo de cacheamento de um *proxy* VoD não estão baseadas em nenhum tipo de colaboração com outros *proxies* do sistema VoD.

[3] Em oposição à abordagem distribuída também comumente encontrada na revisão de literatura (ZENG; VEERAVALLI; LI, 2011).

Uma vez tornado isso claro, muitos algoritmos de cacheamento empregam o histórico do padrão de acesso dos clientes como uma referência para tentar prever quais partes do acervo serão mais solicitadas futuramente. Assim, os objetos de vídeo mais acessados no passado são armazenados na memória do *proxy* para servir futuras requisições. Contudo, à medida que essa estimativa começa a falhar, a eficiência do *proxy* é reduzida e o algoritmo de cacheamento precisa atualizar o histórico de frequência para definir o novo conteúdo alvo para ser armazenado para servir às futuras requisições.

Um exemplo dessa abordagem de cacheamento é proposta por Li et al. (LI; ZHENG; ZHANG, 2011) que apresenta um modelo de cacheamento baseado em duas variáveis de controle de popularidade: a primeira conta o número de acessos a um objeto de vídeo durante um intervalo de curto prazo e a segunda em um intervalo de longo prazo. Ambos os intervalos são formados pelo tempo compreendido entre o instante atual e um instante no passado. Se o número de acessos observados em qualquer desses intervalos exceder um valor limite para seu respectivo intervalo, o objeto é marcado com uma prioridade de cacheamento mais alta, senão, o objeto recebe uma prioridade de cacheamento menor.

Dessa forma, a preocupação dos autores em analisar a relevância dos objetos de vídeo considerando diferentes intervalos passados induz a uma reflexão sobre a dinamicidade relativa às mudanças na popularidade dos objetos de vídeo. Todavia, a análise da abordagem usada por Li et al. leva a concluir que uma deficiência relacionada a esta estratégia reside no posicionamento dos intervalos de análise nela empregados, uma vez que ambos estão localizados predominantemente no passado. Como consequência, o algoritmo tende a fazer decisões de cacheamento imprecisas, já que as mesmas variações que contribuem para a potencial obtenção de diferenças na contagem de acessos em cada intervalo, também podem fazer com que a demanda atual possa ser diferente daquela prevista por suas estimativas.

Chen et al. (CHEN et al., 2003) propôs um algoritmo no qual o cálculo da popularidade dos objetos de vídeo se baseia em dois fatores principais: duração dos acessos e tempo transcorrido desde o último acesso a um item do acervo. Além disso, os autores propuseram que cada vídeo deveria ocupar uma parte da memória na proporção da sua respectiva popularidade. A análise desta estratégia leva a concluir que o algoritmo apresenta aspectos inexplorados que poderiam melhorar potencialmente a eficiência do *proxy*. A exemplo, o algoritmo poderia experimentar um aumento em sua taxa de acertos se a duração dos acessos (mensurada por ele) fosse, de alguma maneira, correlacionada aos segmentos de vídeo acessados durante esse período. Desta maneira, o algoritmo poderia determinar quais

parcelas de um vídeo costumam ser mais acessadas pelos clientes, de modo que estas parcelas pudessem ser armazenadas com maior prioridade na memória.

Conforme descrevem as próximas seções deste capítulo, mais recentemente, alguns autores desenvolveram algoritmos onde a lógica de cacheamento segue um padrão diferente daquele historicamente empregado em *proxies* de VoD. Um ponto em comum entre estas estratégias é o uso do posicionamento atual dos clientes ativos[4] em cada vídeo como base para o cálculo da prioridade de cacheamento dos trechos de vídeo. Uma vez que a unidade básica de vídeo, para fins de vinculação da prioridade de cacheamento, varia nestas estratégias, a tabela 2.1 apresenta a taxonomia necessária para entendimento do mecanismo de divisão lógica dos trechos de vídeo usado em cada sistema . Esta taxonomia será também referenciada ao longo dos demais capítulos deste texto.

Tabela 2.1 - Taxonomia para divisão lógica dos vídeos

Unidade de vídeo	**Descrição**
Bloco	Quantidade em bytes equivalente a um segundo de vídeo considerando a taxa usada para transmissão do vídeo em formato comprimido. A exemplo, se o vídeo é transmitido a uma taxa de 8Mbps (Mega bits por segundo), o tamanho do bloco é 8Mb (Mega bits).
Segmento	Quantidade em bytes equivalente a um múltiplo do tamanho de um bloco de vídeo. Em outras palavras um segmento é formado por um ou mais blocos de vídeo e, por consequência, uma outra unidade também usada para designar seu tamanho é o segundo.
Sequência	Quantidade de blocos de vídeo existentes entre as posições de acesso de dois ou mais clientes sucessivos que assistem a um mesmo vídeo. Considera-se como posição de acesso o número do bloco de vídeo que está sendo transmitido pelo *proxy* VoD para um determinado cliente no segundo corrente do tempo.

Fonte: do autor (2015).

2.1 Algoritmo CCVC

Ishikawa e Amorim (ISHIKAWA; AMORIM, 2009) apresentam um algoritmo de cacheamento (chamado CCVC - *Collapsed Cooperative Video Caching*), inspirado na política de cacheamento usada em redes peer-to-peer, que trabalha em dois níveis. No primeiro nível, o algoritmo seleciona o vídeo com menor popularidade (menor número de clientes ativos) como vítima. No segundo nível, o algoritmo decide que partes do vídeo selecionado no

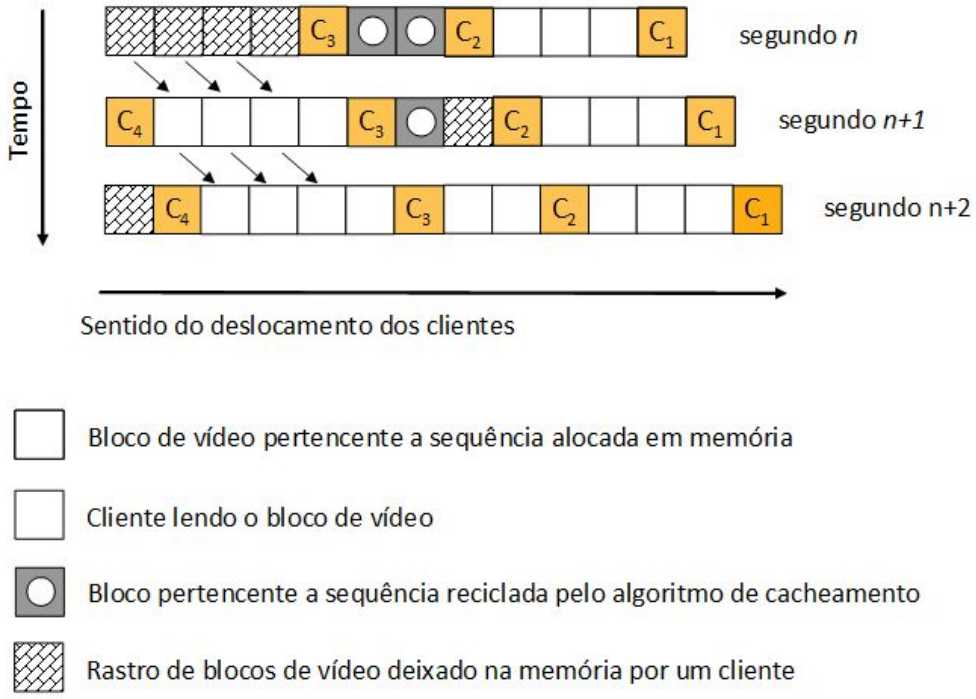
[4] Clientes conectados ao *proxy* para receber um fluxo de vídeo.

primeiro nível devem ser descartadas para liberar espaço para a alocação de conteúdos recebidos do servidor principal.

Um outro aspecto chave da abordagem usada por esses autores é atribuir uma prioridade de cacheamento maior para blocos confinados em sequências de vídeo. Como benefício principal proporcionado pelo uso desta estratégia, os blocos de vídeo trazidos para a memória para servir os clientes mais antigos do sistema (aqueles que limitam pelo lado direito as sequências de vídeo) tendem a permanecer na memória para serem acessados pelos clientes mais novos (os que limitam as sequências pelo lado esquerdo) que assistem ao mesmo vídeo. Essa característica torna o CCVC um dos primeiros algoritmos a considerar os clientes ativos que não acessaram ainda certas partes de um vídeo como um fator para determinar quais parcelas dos vídeos devem ser prioritariamente cacheadas no *proxy*.

A figura 2.1 ilustra o processo de organização do vídeo usado pelo CCVC onde cada quadrado representa um bloco de vídeo e as fileiras horizontais de blocos sucessivos (existem três na figura) correspondem ao mesmo clipe de vídeo. Cada fileira no sentido de cima para baixo descreve o estado dos blocos que pertencem ao vídeo, à medida que o tempo avança em um segundo. Conforme o tempo avança a cada segundo, os clientes se movem em direção ao fim do vídeo (lado direito da figura) e, solidariamente à movimentação dos clientes, as sequências também se deslocam em direção ao final do vídeo.

Figura 2.1 - Organização do vídeo



Fonte: do autor (2015).

Os quadrados que contêm um círculo representam os blocos de vídeo que pertencem a uma sequência de baixa prioridade de cacheamento removida da memória pelo algoritmo de cacheamento[5]. Cada quadrado com numeração no formato C_{δ} representa um bloco de vídeo que está sendo transmitido para o cliente δ. Estes blocos que estão sendo lidos pelos clientes não são elegíveis para descarte pelo algoritmo de cacheamento quando a memória livre se torna escassa. Os quadrados com hachuras indicam uma trilha de blocos deixados na memória por um cliente (ou grupo de clientes), à medida que este avança, a cada segundo (assumindo uma taxa de transmissão de bits constante), para o próximo bloco de vídeo. Os blocos com hachuras constituem o conteúdo preferencial para descarte sempre que o *proxy* não dispõe de mais memória livre para alocação do conteúdo recebido do servidor principal.

Quando a posição de acesso de um cliente limita uma trilha pela esquerda, o algoritmo aumenta a prioridade de cacheamento dos blocos pertencentes à trilha para um valor intermediário (indicado pelos quadrados em cor branca). Assim, explorando o fato de que esses blocos tendem a ser usados por clientes que estão posicionados à esquerda da trilha, o algoritmo previne o acesso à rede para obter novamente esse conteúdo com o servidor principal. Seguindo a taxonomia apresentada na tabela 2.1, cada trilha de blocos lateralmente limitados por dois ou mais clientes foi designada no escopo deste estudo de sequência, uma vez que é uma designação mais simples do que a originalmente usada pelos autores do CCVC (os autores se referiam a essas trilhas como "*Slots* de ligação").

Como é possível concluir pela análise da figura 2.1, diversas sequências podem ser formadas durante o tempo de fornecimento do serviço pelo *proxy*. Nessas circunstâncias, à medida que a demanda aumenta, isto é, o número de clientes a serem servidos pelo *proxy* aumenta, os espaços de memória usados para alocar os blocos de vídeo vindos do servidor principal precisam ser obtidos através de um processo de reciclagem das sequências disponíveis.

A equação 2.1 descreve a função usada pelo algoritmo CCVC para cálculo da prioridade de cacheamento de uma sequência *s*. Na definição da função *F(s)*, f representa a quantidade total de blocos de vídeo que precisam ser descartados para alocar o novo conteúdo proveniente do servidor principal. A variável *t* representa o tamanho da sequência s e a variável *d* informa a distância entre o último bloco pertencente a essa sequência e o final do

[5] As características que determinam a baixa prioridade de cacheamento vinculada a esta sequência não foram detalhadamente descritas nesta figura.

vídeo. Quanto menor o valor de *F(s)* para uma sequência, maior é a sua probabilidade de descarte.

Assim, quanto maior for a distância entre uma sequência e o final do vídeo, menor será a sua probabilidade de descarte. Desta forma, segundo os autores, as sequências mais próximas do início de cada vídeo (prefixo) tendem a ser mantidas na memória, reduzindo, assim, a latência para início do fornecimento de serviço para os novos clientes do sistema. Além disso, os autores também defendem que o prefixo do vídeo tendem a ser mais importante em determinados cenários nos quais os clientes abandonam o vídeo antes do fim, como em (ERMAN et al., 2011). Entretanto, segundo (SHEN; TU; STEINBACH, 2007), esta estratégia só produz o efeito desejado quando os clientes efetivamente realizam acesso a todo o prefixo do vídeo, o que não ocorre em todos os cenários de demanda.

$$F(s) = d / min\{f,t\} \tag{2.1}$$

Conforme também descreve a equação 2.1, o algoritmo CCVC também privilegia o descarte das maiores sequências uma vez que, desta forma, é preciso descartar um menor número de sequências para liberar o espaço necessário em memória para alocar os fluxos recebidos do servidor principal. Assim, segundo os autores, esta estratégia não apenas poupa banda de processamento do *proxy*, mas também reduz a quantidade de fluxos de reposição junto ao servidor principal, caso seja necessário restaurar o conteúdo descartado para servir os clientes cuja posição de acesso precede as sequências removidas da memória.

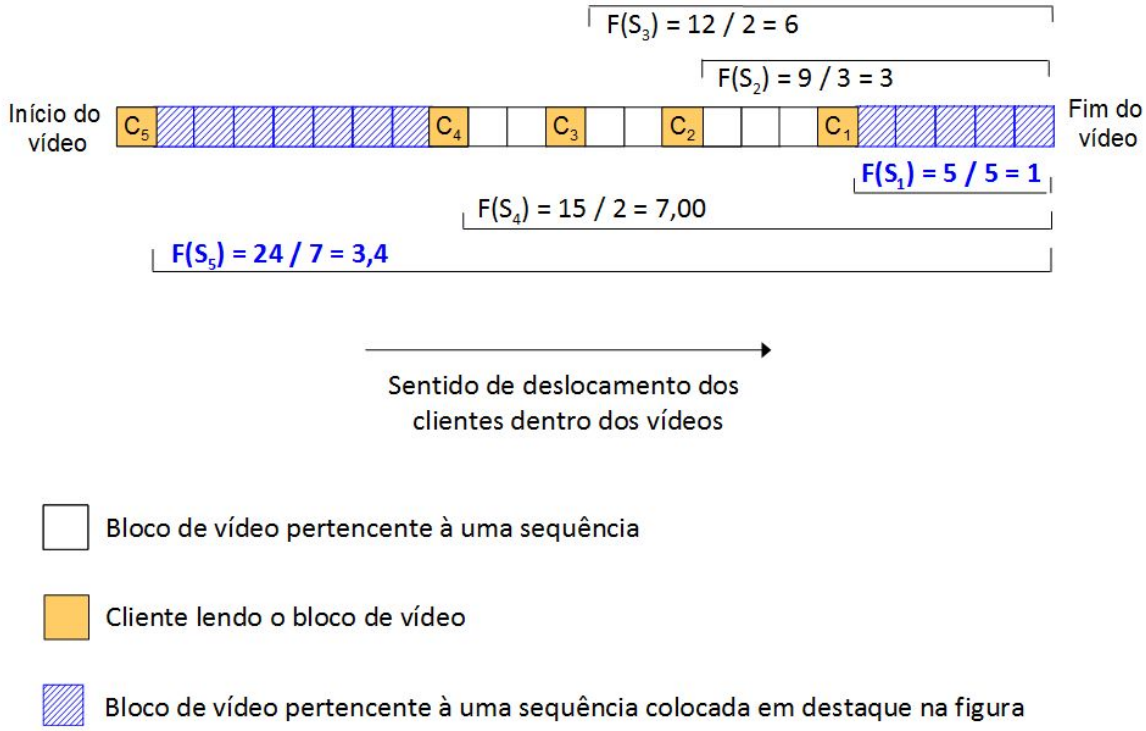
Ainda no que diz respeito à influência do tamanho das sequências sobre o cálculo da prioridade de cacheamento, é importante notar que quando o valor da variável *f* é menor do que *t*, *f* é utilizada para o cálculo da prioridade da sequência *s*, uma vez que, sob estas condições, qualquer sequência é capaz de atender a demanda especificada por *f*.

A figura 2.2 ilustra como é feito o cálculo de *F(s)* para algumas sequências de um vídeo com poucos segundos de duração[6]. Assume-se no contexto desta figura que a demanda *f* é grande o suficiente para que o cálculo das prioridades de cacheamento de todas as sequências presentes na figura leve em consideração o tamanho *t* destas sequências. Em especial, o cálculo da prioridade de cacheamento para as sequências S_1 e S_5, colocadas em destaque com blocos hachurados, demonstra como o algoritmo atribui maior prioridade de

[6] Expressa pela quantidade de blocos que compõem o vídeo.

cacheamento tanto as sequências mais próximas do início do vídeo como as maiores sequências.

Figura 2.2 - Cálculo da prioridade de cacheamento no algoritmo CCVC



Fonte: do autor (2015).

2.2 Algoritmo CC

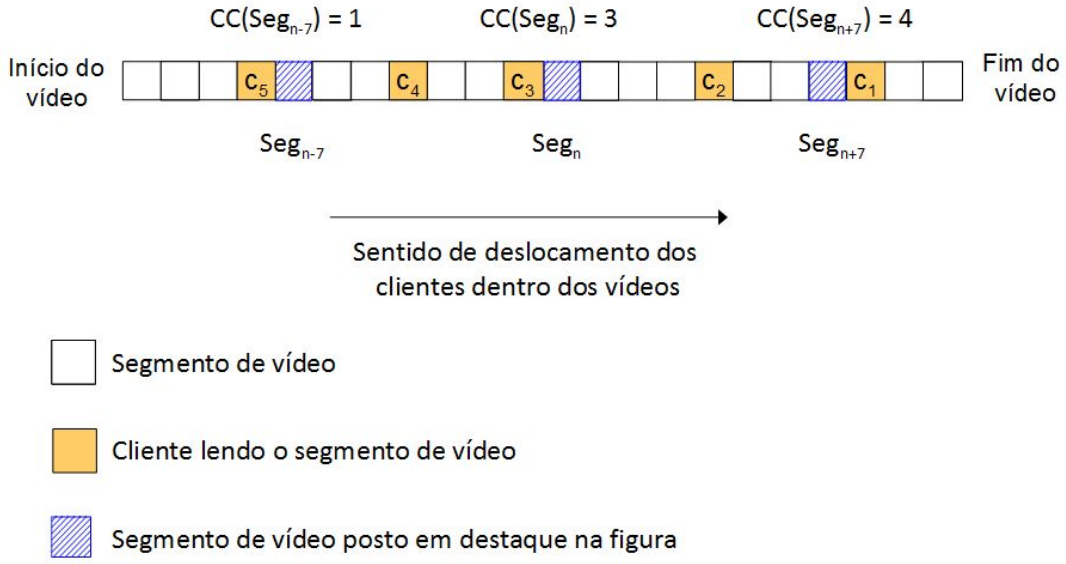
Recentemente, Hong et al. (HONG; DE VLEESCHAUWER; BACCELLI, 2010) propôs um algoritmo chamado CC[7] que, comparado com o CCVC, emprega mais claramente um novo paradigma para decidir quais porções do conteúdo disponível devem ser cacheadas.

No contexto deste algoritmo, os vídeos são organizados em segmentos de tamanho fixo e uma prioridade de cacheamento é atribuída a cada segmento. Para isso, o algoritmo considera o posicionamento atual dos clientes que estão assistindo um determinado item do acervo como uma forma de determinar o número efetivo de acessos a serem realizados no futuro para cada segmento de vídeo pertencente a este item. Somente os clientes que ainda não acessaram um determinado segmento são contabilizados no número de acessos futuros para esse segmento.

[7] Os autores não dão indícios claros se o nome dado ao algoritmo corresponde a uma sigla.

A figura 2.3 ilustra como é feito o cálculo da prioridade de cacheamento para um segmento de número κ, CC(Seg_{κ}) pelo algoritmo CC. Na ilustração, o cálculo da prioridade de cacheamento é demonstrado para três segmentos de vídeo distintos, CC(Seg_{n-7}), CC(Seg_{n}) e CC(Seg_{n+7}), ressaltados com um preenchimento hachurado. O segmento *n-7* possui a sua frente (lado esquerdo) apenas um cliente (C_5). Assumindo que este cliente não irá abandonar sua sessão de vídeo no instante corrente, ele é computado como um acesso futuro para o segmento *n-7*, na prioridade de cacheamento deste segmento, Seg_{n-7}. Como não existem outros clientes à frente do segmento Seg_{n-7}, apenas C_5 forma a prioridade de cacheamento deste segmento, de forma que CC(Seg_{n-7}) = 1.

Figura 2.3 - Cálculo da prioridade de cacheamento no algoritmo CC



Fonte: do autor (2015).

No que diz respeito ao segmento *n*, existem a sua frente três clientes, C_3, C_4 e C_5, que conta para o cálculo da prioridade de cacheamento de dois segmentos, CC(Seg_{n-7}) e CC(Seg_{n}), de modo que, CC(Seg_{n}) = 3. O algoritmo CC considera que quanto maior é o valor de CC(Seg_{κ}), maior é a prioridade de cacheamento do segmento κ, uma vez que se espera que, neste caso, um número maior de clientes realize acesso no futuro a este segmento. Consequentemente, Seg_{n} recebe uma maior prioridade de cacheamento do que Seg_{n-7}. Um raciocínio semelhante leva o segmento Seg_{n+7} a ter a maior prioridade de cacheamento dentre todos os três segmentos colocados em destaque na figura 2.3.

Desta forma, quando existe necessidade de realizar o descarte de segmentos para disponibilizar espaço para alocação dos dados provenientes do servidor principal, os

segmentos com menor valor de CC(Seg_{κ}) são selecionados como vítima pelo algoritmo. Existindo um empate, o algoritmo seleciona para descarte o segmento mais próximo ao início do respectivo vídeo.

Cabe observar que o valor de $CC(Seg_{\kappa})$ é diminuído pelo algoritmo CC em uma unidade toda vez que um cliente acessa um segmento κ. Adicionalmente, os autores descrevem que, se um cliente cancela a visualização de um vídeo antes do final, ou mesmo realiza outras operações do tipo VCR como "retroceder"ou "avançar", os valores de CC(Seg_{κ}) são atualizados em conformidade. Como exemplo, no caso de um usuário selecionar um avanço de capítulo do vídeo que está assistindo, o algoritmo, imediatamente, subtrai uma unidade do valor de CC(Seg_{σ}) para todo $\sigma \leq \tau$, tal que τ é o número do segmento de destino do avanço. Também é importante observar que, embora os autores tenham descrito a estratégia adotada para prover suporte a operações VCR ao algoritmo CC, nenhum estudo de caso experimental foi apresentado com cenários onde ocorre mudança no fluxo de execução dos clientes.

Por fim, um possível aspecto a ser revisto nessa abordagem está relacionado com o fato de que a falta de um limite, com respeito à distância temporal entre as posições de acesso dos clientes e os segmentos alocados na memória do *proxy*, para fins de contabilização do valor da prioridades de cacheamento dos segmentos geridos pelo algoritmo CC, leva os segmentos localizados mais próximos ao final de cada vídeo a terem um maior número potencial de acessos futuros associados a eles. Consequentemente, o algoritmo fornece uma maior prioridade de cacheamento para estes segmentos que tendem a permanecer na memória do *proxy*, mesmo que existam segmentos no começo do vídeo com demanda maior associada a eles no curto prazo.

2.3 Algoritmo RTBC

Wu et al. (WU et al., 2012) propôs um algoritmo que estima o tempo para o acesso a cada segmento do vídeo. Para isso, o algoritmo mede a distância entre cada segmento e o a posição de leitura do cliente mais próximo a esse segmento. Quando o cliente que está mais próximo a um segmento realiza o acesso a este segmento, o sistema automaticamente recalcula a distância entre o segmento e o cliente mais próximo seguinte.

Os autores chamam esta estimativa da distância entre cada segmento e seu cliente mais próximo de *Tempo Exato de Reuso* para o segmento σ, $TER(Seg_{\sigma})$, embora eles não

mencionem as possíveis influências causadas pelo fenômeno da saída prematura dos clientes sobre o cálculo do $TER(Seg_{\sigma})$. Eles também explicam que o valor do TER para o primeiro segmento de cada vídeo precisa ser estimado tendo em vista que, neste caso, não existe um cliente ativo prévio ao segmento. Para situações como esta, o RTBC usa uma previsão de popularidade para o primeiro segmento de cada vídeo, denominada *Tempo de Reutilização Previsto* (TRP), que pode ser calculada em duas modalidades:

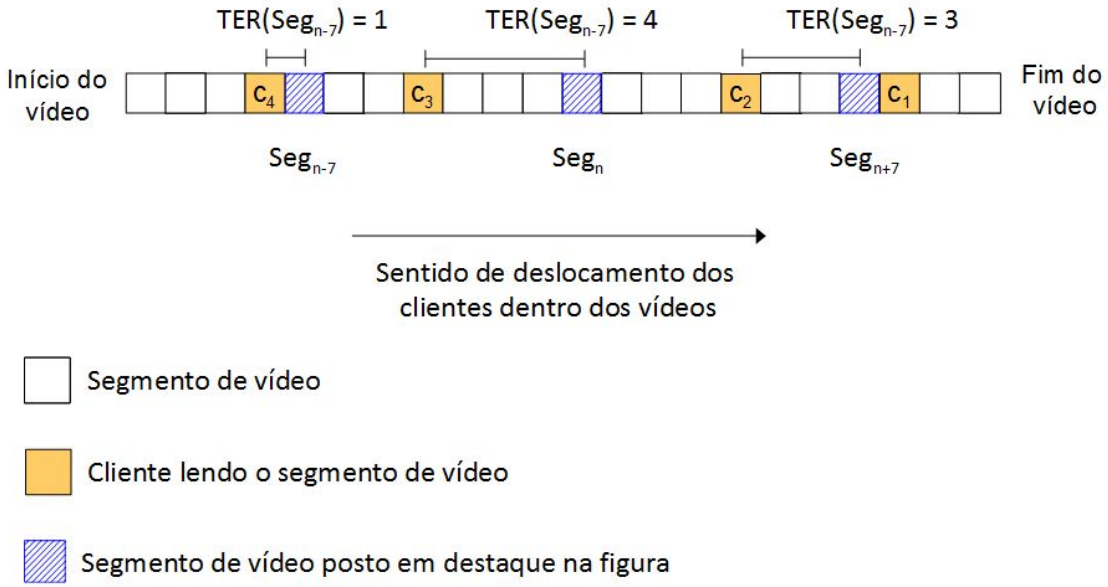
- *Cálculo do TRP_Arb*: consiste em atribuir um alto valor de TRP para os primeiros segmentos de cada vídeo, fazendo com que estes segmentos sejam substituídos antes de qualquer outro tipo de segmento destes vídeos, para os quais seja possível calcular o $TER(Seg_{\sigma})$.
- *Cálculo do TRP_Freq*: consiste em utilizar a frequência de chegada das requisições para cada vídeo como um valor para o TRP do primeiro segmento destes vídeos. Para vídeos que não tenham recebido qualquer requisição dentro de uma janela de tempo de tamanho *W*, o algoritmo RTBC atribui uma prioridade de cacheamento baseada na modalidade de *TRP_Arb*.

Os autores salientam que a principal fraqueza relacionada ao cálculo do *TRP_Arb* reside na alta probabilidade de descarte do prefixo (primeiro segmento) de vídeo com alta popularidade. Assim, poderia ser mais produtivo se o prefixo destes vídeos fosse mantido na memória do *proxy* em vez de segmentos internos (outros que não os primeiros de cada vídeo) de um vídeo de menor popularidade. Os autores mencionam que esta deficiência é mitigada com o uso do *TRP_Freq* para os vídeos cujo intervalo médio de acesso está abaixo do valor de *W*.

A equação 2.2 descreve, resumidamente, como o RTBC calcula a prioridade de cacheamento do segmento σ pertencente ao vídeo *v*, $RT(Seg_{\sigma,v})$, com base no valor de σ e da variável $H(_v)$, que determina, para o vídeo *v*, o número de clientes cujas requisições chegaram até o *proxy* dentro do intervalo de tempo *W*. Se o número do segmento for maior do que zero (quando σ não é o primeiro segmento do vídeo *v*), o cálculo é feito através do tempo exato de reuso, $TER(Seg_{\sigma,v})$, conforme ilustra a figura 2.4. Do contrário, o algoritmo considera o valor de H_v e realiza o cálculo de $RT(Seg_{\sigma,v})$ com base em $TRP\_Freq(Seg_{\sigma,v})$ ou $TRP\_Arb(Seg_{\sigma,v})$.

$$RT(Seg_{\sigma,v}) = \begin{cases} TER(Seg_{\sigma,v}), & \sigma > 0 \\ TRP\_Freq(Seg_{\sigma,v,}) & \sigma = 0 \quad e \quad H_v \neq 0 \\ TRP\_Arb(Seg_{\sigma,v,}) & \sigma = 0 \quad e \quad H_v \neq 0 \end{cases} \tag{2.2}$$

Figura 2.4 - Cálculo da prioridade de cacheamento no algoritmo RTBC



Fonte: do autor (2015).

Quando é necessário descartar segmentos para liberar espaço na memória do *proxy* para alocação do conteúdo egresso do servidor principal, o RTBC constrói uma lista de segmentos formada por um segmento de cada vídeo. O segmento selecionado em cada vídeo é aquele que possui um maior valor para $RT(Seg_{\sigma,v})$. Se houver durante a seleção do segmento em vídeo, o segmento mais próximo do fim do vídeo é selecionado.

Em seguida, o RTBC acessa a lista construída buscando pelo segmento com menor prioridade de cacheamento (maior valor de $RT(Seg_{\sigma,v})$) e o seleciona como vítima para descarte. Se ocorrer um empate, o algoritmo seleciona um subconjunto[8] de segmentos da lista com base no número de clientes ativos nos seus respectivos vídeos. Irão compor este subconjunto os segmentos cujos vídeos possuem um maior número de clientes ativos. Por fim, no desempate final, o RTBC seleciona como vítima a partir do subconjunto aquele segmento, cujo vídeo não recebe uma requisição há mais tempo.

Como uma análise crítica, sob o ponto de vista do presente estudo, a respeito desta estratégia, entende-se que ela compartilha o mesmo problema presente na abordagem de Hong, pois, ao priorizar a preservação na memória dos segmentos que serão acessados em mais curto prazo, não necessariamente estarão sendo preservados os segmentos com maior demanda (concentração de clientes prévios a curto e médio prazo), visto que a densidade de

[8] O tamanho do subconjunto não é informado pelos autores.

clientes pode ser maior próxima aos segmentos cuja distância até o cliente mais próximo é também maior.

A exemplo disso, em um cenário de operação do *proxy* com recursos restritos, especialmente no que diz respeito à quantidade de memória e a largura de banda e rede disponíveis, é possível que a lógica de cacheamento usada pelo RTBC aponte para o armazenamento de um segmento que pertence a um vídeo de baixa popularidade. Todavia, uma curta distância entre este segmento e o cliente mais próximo a ele tende a fazer com que o algoritmo destine recursos para armazená-lo, embora seja provável, neste caso, que nenhum outro cliente (excetuando-se aquele que determinou o cacheamento do segmento) realize acesso novamente a este segmento a curto ou médio prazo.

3 ALGORITMO PROPOSTO

A ideia principal em torno da nova estratégia de cacheamento advém de uma observação mais ampla dos aspectos de escalabilidade para maximizar não somente a eficiência relacionada ao uso da largura de banda de rede disponível, mas também a outros recursos físicos do *proxy* que são frequentemente sujeitos a picos na carga de trabalho para os quais o sistema precisa responder de forma apropriada.

Assim, os diferenciais dessa abordagem surgem da compreensão de que, em cenários de maior demanda, o *proxy* deveria fazer a melhor escolha para emprego dos seus recursos, considerando somente a demanda atual (formada exclusivamente por clientes que já começaram suas sessões de vídeo no sistema) existente dentro de uma janela de tempo de curto prazo de alguns segundos[9] em direção ao futuro. Caso contrário, o uso desses recursos objetivando a qualificação do serviço a mais longo prazo (possíveis clientes) pode reduzir a escalabilidade do *proxy* para sustentar a demanda atual.

De acordo com essa visão, quando o *proxy* trabalha para servir potenciais futuras requisições a mais longo prazo, como ocorre ao cachear os prefixos de vídeo (KHEMMARAT et al., 2011; LI; LI; JIANG, 2011), a taxa de troca do conteúdo armazenado na memória do *proxy* tende a ser menor, visto que, nesses casos, a filosofia de cacheamento não leva em consideração as especificidades inerentes às condições correntes da carga de trabalho. Como consequência, a largura de banda associada a todos os dispositivos físicos usados pelo *proxy* tende a ser subutilizada, uma vez que o sistema despende pouco esforço para movimentar dados com o propósito de atualizar a memória do *proxy*, objetivando capturar o estado de carga atual e sustentar dessa forma a melhor escalabilidade possível para o *proxy*.

Em contrapartida, no contexto da abordagem proposta neste estudo, sob condições de carga de trabalho alta, o algoritmo de cacheamento deve organizar constantemente o fluxo de alocação de dados para maximizar a produtividade do *proxy*. Para esse propósito, é necessário avaliar e selecionar os trechos de vídeo com as maiores demandas, isto é, com maior número de clientes que irão acessá-los a curto prazo, priorizando, em seguida, a alocação desse conteúdo na memória do *proxy*. Na nova estratégia criada, isto é feito através da implementação de uma janela de tempo, à frente a cada trecho de vídeo, com o propósito de estabelecer a densidade de clientes para esse intervalo. Essa densidade é monitorada e

[9] Não superior a 300 segundos de vídeo nos cenários de simulação utilizados neste estudo.

atualizada seguindo as mudanças no estado da carga de trabalho, colocando assim em uso os recursos disponíveis no *proxy*.

As seções a seguir descrevem o algoritmo CARTE (*Current demAnd Rather Than futurE* - demanda atual em vez de futura), desenvolvido para avaliar os impactos no desempenho do *proxy* VoD gerado pelo uso das características de projeto mencionadas acima que brevemente reprisadas são:

- monitorar a densidade de clientes ativos em um futuro de curto prazo;
- reagir às condições de estado de carga atuais atualizando o conteúdo de memória;
- empregar a capacidade de processamento (larguras de banda) do hardware disponível para aumentar a escalabilidade do *proxy*.

Com esse objetivo, a seção 3.1 mostra como o algoritmo divide logicamente os vídeos em trechos menores e descreve a base do mecanismo de cálculo da prioridade de cacheamento para cada tipo de trecho. A seção 3.2 explica mais detalhadamente a lógica de decisão do cacheamento empregado pelo CARTE, apresentando descrições matemáticas sobre como o algoritmo trabalha.

3.1 Organização de vídeo

A estratégia empregada pelo algoritmo desenvolvido neste trabalho consiste em dividir logicamente os vídeos em trechos menores, chamados Sequências, inspirada na *Política de Gerenciamento de Slots de Ligação* usada pelo algoritmo CCVC (descrita na seção 2.1 do capítulo anterior). Como demonstrado ao longo desta seção, a organização do vídeo em sequências permite um uso mais eficiente dos recursos de memória disponíveis se comparado à largamente utilizada divisão clássica dos vídeos em segmentos de mesmo tamanho. Um raciocínio parecido pode ser usado para comparar o uso de sequências com o uso de segmentos de tamanho variáveis, como aqueles propostos em (TU et al., 2009).

Em termos de eficiência, a vantagem principal ligada à filosofia de projeto usando sequências é ilustrada na figura 3.1 que compara a organização lógica do vídeo através de sequências com a organização do vídeo através de segmentos de igual tamanho. Nesta figura, os dois fluxos de blocos de vídeo (na parte de cima e de baixo da figura) novamente correspondem ao mesmo clipe de vídeo. A parte de cima da figura mostra como o clipe é organizado usando segmentos do mesmo tamanho, onde cada segmento contém quatro blocos

de vídeo. A parte de baixo mostra como o mesmo clipe é organizado considerando o uso de sequências criadas a partir da posição de acesso dos clientes.

Na organização do vídeo através de segmentos de tamanho fixo, qualquer segmento que está sendo acessado por um cliente não pode ser removido da memória. Consequentemente, para o clipe de vídeo mostrado na figura, há somente dois segmentos (*n*+1 e *n+4*, totalizando oito blocos de vídeo) que podem ser removidos da memória pelo algoritmo de substituição. Por outro lado, na organização do vídeo através de sequências, somente os blocos de vídeo que estão sendo acessados atualmente pelos clientes precisam ser protegidos da ação do algoritmo de substituição. Como resultado, todas as sequências existentes no clipe de vídeo são elegíveis para propósitos de descarte, totalizando dezessete blocos de vídeo (entre aqueles presentes na figura).

Para descrever outro exemplo, assumindo que o algoritmo de substituição tem atribuído uma prioridade de descarte mais alta para o intervalo entre os últimos dois clientes (no lado esquerdo da figura, para ambas estratégias), o intervalo inteiro pode ser removido da memória somente se o mecanismo de organização dos vídeos emprega sequências. Se os vídeos forem organizados em segmentos do mesmo tamanho, a parte final do intervalo, que corresponde aos dois primeiros blocos com hachuras pertencentes ao segmento *n+2*, não pode ser descartada, apesar de sua baixa probabilidade em receber um acesso no futuro.

Assim, a organização por sequências define mais precisamente as áreas de interesse para a ação do algoritmo de substituição, possibilitando uma melhor escalabilidade para o *proxy*. Por essa razão, adotou-se o uso de sequências como base para a implementação do mecanismo de vinculação de prioridades usado pelo novo algoritmo desenvolvido, cuja lógica de cacheamento é descrita mais detalhadamente na próxima seção.

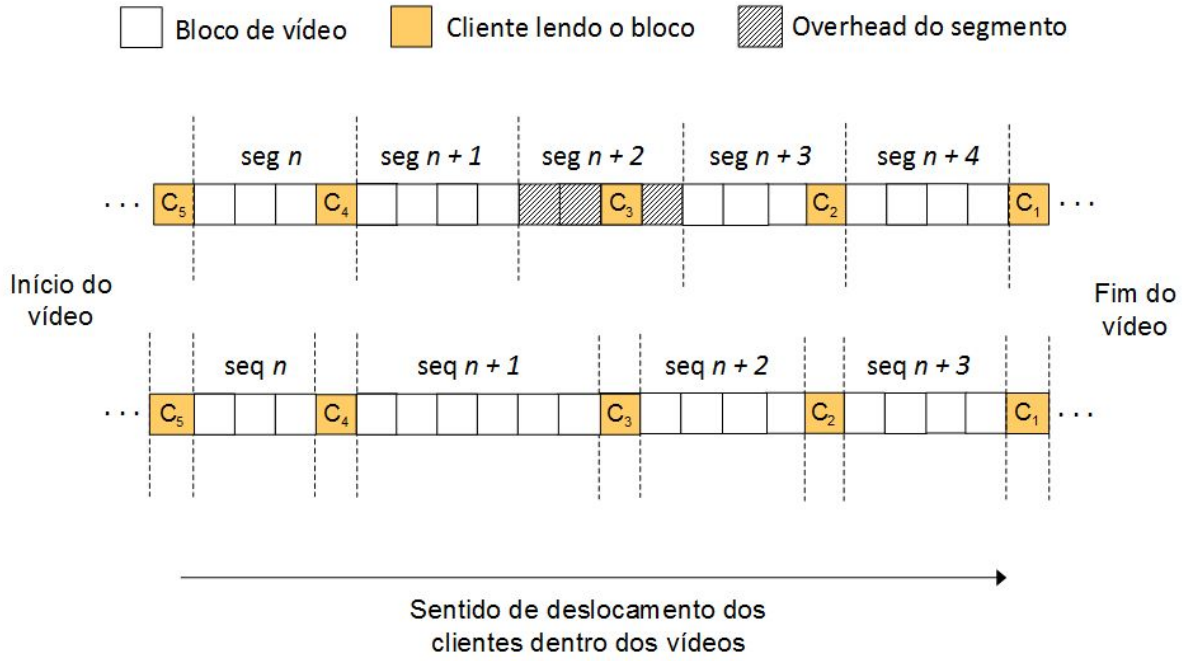
3.2 Lógica de cacheamento

De acordo com a estratégia de organização dos vídeos descrita na seção anterior, a vinculação das prioridades de cacheamento a cada trecho de vídeo, definida no algoritmo desenvolvido neste trabalho, considera que:

- os blocos que estão atualmente sendo lidos por um ou mais clientes não podem ser descartados da memória;

- quando há disponibilidade de espaço livre (sem uso), a memória empregada para alocar o conteúdo vindo do servidor principal deve ser obtida, primariamente, a partir deste espaço;
- quando a memória livre se torna escassa para alocar o conteúdo recebido do servidor principal, o *proxy* tenta descartar uma ou mais trilhas de blocos deixadas na memória como um resultado de entrega de serviço feita anteriormente para um cliente. Se não há nenhum bloco nessa condição, o algoritmo seleciona uma sequência para descarte como forma de liberar espaço suficiente para alocar o conteúdo ingressante.

Figura 3.1 - Comparativo: segmentos (tamanho fixo) vs. sequências (tamanho variável)



Fonte: do autor (2015).

A sequência vítima é escolhida com base na prioridade de cacheamento das sequências disponíveis, que é calculada de forma incremental durante a formação de sequências pela combinação dos seguintes fatores:

- Posição Inicial ($PI_{s,v}$): corresponde à posição atual do primeiro bloco (da esquerda) pertencente à s-ésima sequência do vídeo v, denominada S_v.
- Tamanho da Sequência ($TS_{s,v}$): número de blocos pertencentes à S_v.
- Densidade de Clientes ($DC_{s,v}$): total de clientes presentes em uma janela de K blocos precedentes a S_v que potencialmente acessarão essa sequência.

Dessa forma, a prioridade de cacheamento para cada sequência, $PC_{s,v}$, é definida pela equação 3.1, onde quanto maior o valor para $PC_{s,v}$, maior a probabilidade da sequência S_v ser cacheada na memória do *proxy*.

$$PC_{s,v} = (DC_{s,v} / TS_{s,v}) * X \tag{3.1}$$

$$\text{onde } X = \begin{cases} PI_{s,v}, & PC_{s',v'} = PC_{s'',v''} \\ 1, & PC_{s',v'} \neq PC_{s'',v''} \end{cases}$$

$$\text{onde } DC_{s,v} = \sum_{i=1}^{TC_{v,K,s}} C_{i,v} \tag{3.2}$$

$$\text{tal que } WB_{s,v} \leq L_{i,v} < PI_{s,v}$$

$$\text{e } WB_{s,v} = PI_{s,v} - K \tag{3.3}$$

Na equação 3.2, $C_{i,v}$ é o cliente *i* assistindo o vídeo *v* e $TC_{v,K,s}$ representa todos os clientes que estão assistindo algum dos *K* blocos que precedem a sequência *s* do vídeo *v*. $WB_{s,v}$, presente nas equações 3.2 e 3.3, informa a posição atual da borda esquerda da janela de tempo precedendo essa sequência e $L_{i,v}$ é a localização atual do cliente *i* que assiste ao vídeo *v*.

Como mostra a equação 3.1, o algoritmo prioriza o cacheamento de sequências com densidade de clientes associada a elas (indicada pelo número de clientes existentes dentro da janela de *K* blocos que precede cada sequência), pois essa escolha permite definir que sequências de vídeo podem beneficiar um número maior de clientes no curto prazo. Neste sentido, para possibilitar que o algoritmo consiga responder as flutuações no fluxo de entrada dos clientes, as sequências em formação no prefixo[10] do vídeo que ainda não possuem *K* blocos prévios não são descartadas, uma vez que a densidade de clientes para estas sequências não pode ser definida sem que a janela de tempo esteja completamente constituída.

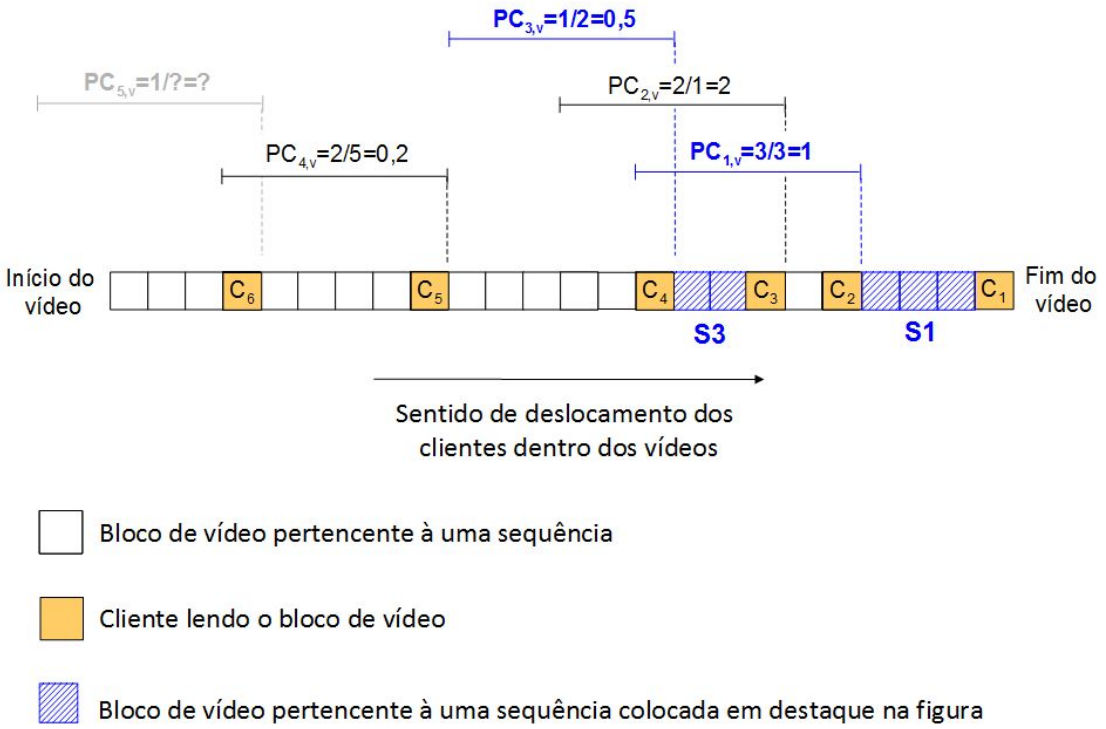
Somado a isso, a equação 3.1 mostra ainda que o algoritmo também favorece a preservação em memória de sequências menores, haja vista que essa escolha proporciona uma maior escalabilidade devido ao consumo mais baixo dos recursos de memória.

[10] Parte inicial do vídeo

Além disso, a variável *X*, também presente na equação 3.1, é usada para tratar a possibilidade de empate entre as prioridades de cacheamento calculadas para duas ou mais sequências. Deste modo, quando ocorre um empate, as sequências próximas ao final de cada vídeo recebem prioridades de cacheamento maiores, com o objetivo de possibilitar não somente que os clientes existentes dentro de cada janela possam acessar essas sequências, mas também os clientes existentes a mais longo prazo.

A figura 3.2 ilustra o cálculo da prioridade de cacheamento realizado pelo CARTE. Nesta figura, existem ao todo cinco sequências de vídeo para as quais o algoritmo deve calcular a prioridade de cacheamento, embora apenas duas destas sequências tenham sido colocadas em destaque (com hachuras) para direcionar a exemplificação.

Figura 3.2 - Cálculo da prioridade de cacheamento no CARTE



Fonte: do autor (2015).

As diferentes linhas horizontais existentes sobre o vídeo representam as janelas de tempo para cada sequência presente na figura. O primeiro bloco encompassado por cada janela sempre corresponde ao bloco que está sob acesso do cliente que limita a respectiva sequência pela esquerda. Sobre as linhas horizontais que representam cada janela de tempo, é apresentado o cálculo da prioridade de cacheamento para a respectiva sequência.

A sequência mais a esquerda na figura, limitada pelos clientes C_6 e C_5, não possui uma prioridade de cacheamento completamente definida, uma vez que sua janela de tempo transcede o início do vídeo. Consequentemente, esta sequência não está sujeita a ação do algoritmo de substituição do *proxy*. A sequência S_1, colocada em destaque na figura, possui três clientes em sua janela de tempo. Dividindo-se esta quantidade pelo tamanho da sequência (três), obtém-se o valor de uma unidade para a respectiva prioridade de cacheamento. No que diz respeito à sequência S_3, também em destaque na figura, existe apenas um cliente em sua janela de tempo. Como o tamanho da sequência é dois, o valor da respectiva prioridade de cacheamento é igual a 0,5. Desta forma, fazendo um comparativo entre as prioridades calculadas para as sequências S_1 e S_3, é possível notar que, embora o tamanho de S_1 seja maior do que o de S_3, a maior densidade de clientes vinculada à S_1 torna sua prioridade de cacheamento maior do que a de S_3.

Por último, tomando como referência a lógica de funcionamento do algoritmo CC (capítulo 2, seção 2.1), o projeto do CARTE prevê o tratamento para operações do tipo VCR, como “pular para frente", “pular para trás" ou “abortar o vídeo". Quando esses eventos acontecem, o algoritmo atualiza a densidade de clientes das janelas de tempo afetadas pelas operações feitas pelos clientes. Assim, quando um cliente, por exemplo, aborta a execução de um filme, o algoritmo subtrai em uma unidade a densidade relacionada à janela onde o cliente estava posicionado no instante em que decidiu abortar a transmissão do vídeo.

Cabe, contudo, observar que a maior parte das simulações realizadas neste trabalho foram implementadas sem a ocorrência de eventos VCR. A razão para o maior interesse na execução de simulações neste formato advém do objetivo de configurar um ambiente comparativo imparcial para avaliar o desempenho do algoritmo proposto neste trabalho, uma vez que a análise sob cenários com operações VCR também não foi o foco durante os estudos realizados pelos autores dos algoritmos selecionados (usando o critério descrito na seção 5) para a análise comparativa com o CARTE.

4 AMBIENTE EXPERIMENTAL

Esta seção apresenta aspectos motivacionais e operacionais relacionados à plataforma construída para avaliar o desempenho do algoritmo CARTE (descrito no capítulo anterior). Com esse intuito, a seção 4.1 descreve as principais dificuldades em torno do uso das ferramentas, até então disponíveis, que poderiam ser utilizadas para avaliação de desempenho de um *proxy* de vídeo, expondo assim os benefícios do ambiente de avaliação desenvolvido no escopo deste estudo. A seção 4.2 apresenta os fluxo de síntese desta ferramenta. A seção 4.3 descreve em linhas gerais o processo de avaliação, expondo os conceitos principais. A seção 4.4 apresenta o fluxo para simulação com o ambiente proposto. A seção 4.5 mostra as alternativas de simulação disponíveis para a aquisição de dados. Por fim, a seção 4.6 descreve a estratégia seguida para validar os dados produzidos pelo simulador.

4.1 Motivação para a construção de um novo simulador

Pelo que se sabe, na maioria das publicações na área de desenvolvimento de *proxies* e algoritmos de cacheamento para VoD, os pesquisadores desenvolveram seus próprios simuladores, como feito em (KAI-CHUN; YU, 2012), ou eles adaptaram simuladores de rede existentes, como em (CARBUNAR et al., 2011), para realizar seus experimentos a fim de evitar o custo e a complexidade inerente ao desenvolvimento de ambientes reais para prototipar seus sistemas.

Além disso, essas duas alternativas têm sido amplamente usadas para avaliar o impacto de novas políticas de cacheamento sob o ponto de vista de métricas convencionais tais como as taxas de acertos, taxas de bloqueio de clientes, ocupação de recursos de rede e atrasos para fornecimento do serviço, entre outros. Acredita-se, porém, que ambas abordagens são incompletas, dado que elas não consideram a análise de aspectos arquiteturais do *proxy*, os quais são também potencialmente decisivos para o desempenho e a escalabilidade do sistema, tais como o consumo de recursos físicos para executar a aplicação.

Por outro lado, simuladores de arquitetura concentram o foco na avaliação do desempenho do hardware subjacente usado para implementar o *proxy* VoD, permitindo que o projetista selecione ou descreva a arquitetura alvo que irá ser usada na simulação. A exemplo disto, os simuladores de conjunto de instruções (*Instruction-Set Simulators*) - ISS (ALMASRI et al., 2011) permitem obter os resultados de desempenho usando o consumo médio

aproximado para a execução de cada instrução da arquitetura alvo. Entre as métricas mais mensuradas por um ISS estão o número de ciclos e o consumo de energia produzidos por uma aplicação.

Outro subtipo dos simuladores em nível de arquitetura é o simulador ciclo-a-ciclo (*Cycle Accurate Simulator*) - CAS (AUSTIN; LARSON; ERNST, 2002). Este modelo de simulador pode obter dados de desempenho com uma maior precisão quando comparado ao ISS, ao custo, geralmente, de um tempo de simulação também maior. A grande precisão dos resultados produzidos advém, como o próprio nome diz, da execução do ciclo-a-ciclo de cada instrução usando uma descrição (conhecida também como modelo) detalhada da arquitetura alvo que permite emular com alta fidelidade o comportamento do hardware.

Dois problemas principais, porém, estão ligados ao uso destas classes de simuladores. Em primeiro lugar, elas são geralmente orientadas a fazer análises de propósito geral e, portanto, não possuem quaisquer facilidades para validação e análise de um *proxy* VoD. Assim, diferentes aspectos necessários para executar essa aplicação, tais como a forma de interação do *proxy* com os clientes e com um conjunto de servidores de vídeo, precisam ser desenvolvidos pelo projetista antes do início da simulação.

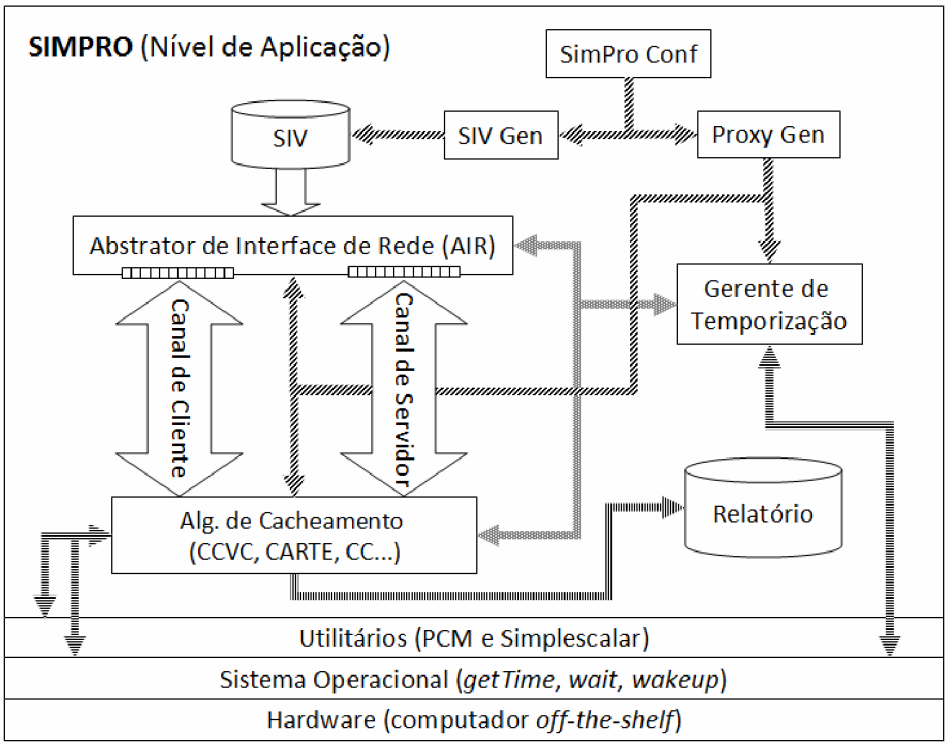
Segundo e mais importante, o tempo de simulação produzido tanto pelo CAS como pelo ISS é superior em várias ordens de magnitude ao tempo necessário para executar a aplicação em tempo real. Sendo assim, uma vez que a execução de um *proxy* VoD pode demandar vários minutos ou até mesmo horas considerando uma simulação em tempo real, a realização das mesmas análises com uso do ISS ou CAS tende para um tempo quase que proibitivo, limitando, portanto, o número de testes passiveis de serem realizados através dessas plataformas.

Diante dessas limitações, tornou-se claro que a ferramenta mais adequada para avaliação de *proxies* VoD deveria combinar as melhores características presentes nos simuladores de arquitetura e de rede. Porém, até onde se conhecia, nenhuma ferramenta desenvolvida possuía essa habilidade. Dessa forma, encontrou-se motivação para desenvolver um novo ambiente que reunisse tais características e, assim, criou-se o SIMPRO – SIMulador de *proxy*, o qual é o centro do fluxo de simulação usado nesse trabalho, apresentado nas seções a seguir.

4.2 Fluxo de Síntese do Simulador

A figura 4.1 ilustra os passos do fluxo de síntese, que começa pela geração de um vetor de carga para simulação (*Simulation Input Vector* - SIV). O elemento básico para construção do SIV é uma solicitação para criação de uma sessão de vídeo. A figura 4.2 mostra o trecho inicial de um vetor de carga (arquivo SIV), onde cada linha (exceto pela primeira que descreve o cabeçalho do arquivo) descreve uma solicitação para abertura de sessão. Cada solicitação especifica o instante de chegada, o identificador do vídeo solicitado e a duração de sessão, sendo este último parâmetro usado para simular cenários com saída prematura de clientes, tendo em vista que este fenômeno tem sido observado em durante alguns contextos de funcionamento de sistemas VoD, como em (ERMAN et al., 2011), (KRISHNAN; SITARAMAN, 2013), (FAN; PANKANTI, 2012), (LAI et al., 2011) e (HWANG et al., 2013). O apêndice A apresenta um extrato de cinco páginas obtido a partir de um vetor de carga utilizado nas simulações realizadas no âmbito deste trabalho.

Figura 4.1 - Arquitetura do SIMPRO



Fonte: do autor (2015).

Cada arquivo SIV é gerado automaticamente, através do painel *SIMPRO Conf*, a partir de um conjunto de parâmetros informados pelo projetista. Além do tempo total de simulação, outros parâmetros usados para configurar a carga de trabalho são: o número máximo de requisições, a quantidade, o tamanho e a taxa de transmissão dos vídeos disponíveis para

acesso e o modelo de distribuição para a dispersão das requisições sobre o acervo. O modelo de distribuição de requisições é baseado no uso do Poisson (ALMEIDA et al., 2001) para definir os intervalos entre as chegadas dos clientes. Quanto maior é o valor do coeficiente de Poisson (λ), maior é o tempo médio entre as chegadas das requisições ao *proxy*. Complementarmente, utilizou-se a distribuição Zipf (DAN; SITARAM; SHAHABUDDIN, 1996) para descrever a associação de cada cliente a um vídeo disponível no acervo. Quanto maior o coeficiente Zipf (θ), conhecido como "skew" (inclinação), maior é a concentração das requisições sobre os vídeos mais populares do acervo, com um decaimento exponencial. Dito de outra forma, quanto mais próximo de zero é o valor de θ, mais homogênea é a probabilidade de acesso aos vídeos.

Figura 4.2 - Trecho inicial de um vetor de carga para simulação (arquivo SIV)

ID: Identificador do vídeo
AS: Instante (segundo) para abertura da sessão do cliente
FS: Instante (segundo) para fechamento da sessão do cliente

-ID-	-AS-	-FS-
095	0.003	1.800
078	0.006	0.240
054	0.009	0.000
019	0.012	2.340
007	0.015	0.960
035	0.016	0.660
024	0.020	1.080
033	0.023	1.380
094	0.024	2.340
056	0.027	0.120
086	0.030	5.400
016	0.034	0.180
079	0.036	0.420
043	0.038	5.400
014	0.041	2.520
054	0.046	0.600
036	0.047	0.420
000	0.049	0.180
…	…	….

Fonte: do autor (2015).

O apêndice B contém diferentes curvas produzidas usando o gerador de cargas do SIMPRO. Os dados exibidos apresentam variações gradativas sobre os parâmetros λ e θ, utilizados neste estudo para modelar a distribuição das requisições sobre os vídeos de um acervo. O próximo passo no fluxo de síntese é configurar a arquitetura do *proxy* que constitui:

- *Algoritmo de Cacheamento*: descrito e/ou selecionado pelo projetista do *proxy*.
- *Gerente de Temporização*: criado para fornecer suporte para a sincronização das operações do *proxy*, tais como o avanço dos clientes para o próximo bloco de vídeo a cada segundo e o controle do tempo de retorno do servidor principal, sendo este último configurado pelo projetista para ser um valor constante ou variante dentro de uma faixa especificada.
- *Abstrator de Interface de Rede* - AIR: Contém os buffers de E/S usados para armazenar o arquivo SIV e os blocos de vídeo que entram e saem do *proxy*. Semelhantemente a uma interface de rede real, o AIR também implementa um mecanismo para emular o envio de interrupções para o processador do *proxy*. Isto foi feito com propósito de reproduzir o custo para a realização de trocas de contexto, entre o algoritmo de cacheamento e as rotinas que realizam as transferências de dados com a memória, sempre que os dados enviados pelo servidor são recebidos pelo *proxy*.
- *Canais de cliente e de servidor*: sintetizados para suportar larguras de banda de acordo com as configurações informadas pelo projetista. Em todos os cenários de simulação utilizados neste estudo (assim como também ocorre em cenários semelhantes descritos na literatura), a largura de banda usada para o canal de cliente é irrestrita, visto que, na maioria dos casos, o cliente previamente se responsabiliza por contratar uma conexão de banda larga adequada aos requisitos de transmissão informados pelo provedor de VoD. A largura de banda usada para configurar o canal de servidor nos experimentos realizados, por outro lado, em geral, é restrita a uma fração da quantidade máxima de clientes ativos estimada para toda uma simulação. Desta forma, pretende-se modelar casos típicos para operação do *proxy* VoD, nos quais a banda disponível para conexão com o servidor é sempre menor do que a banda necessária para servir a todos os clientes na região de cobertura do *proxy*.

Ainda no que diz respeito à síntese do SIMPRO, é importante salientar que, durante o projeto deste simulador, levou-se em consideração que, para os vídeos mais populares de um acervo, a velocidade de memória é um requisito tão importante quanto a capacidade da mesma, especialmente quando o *proxy* trabalha para servir centenas (ou milhares) de clientes simultâneos, cujas requisições estão concentradas em alguns poucos vídeos mais populares do acervo. Conforme descrito em (DHAGE; MESHRAM, 2013; SUMMERS et al., 2012), nessas condições, mesmo que haja bastante espaço no disco para alocar completamente esses vídeos, a sua largura de banda tende a não sustentar a vazão necessária para servir os clientes ativos que estão assistindo a esses vídeos. Por outro lado, como demonstrado por (INTEL, 2005), a

largura de banda alta atualmente sustentada tanto pelos barramentos de E/S como pelas interfaces de rede faz com que nenhum desses componentes conformem um gargalo para a aplicação alvo, mesmo frente a cenários de alta carga de trabalho. Como consequência disto, o SIMPRO foi projetado para avaliar a eficiência do *proxy* com foco exclusivamente sobre o desempenho do processador e da RAM, uma vez que esses dispositivos são, inexoravelmente, os principais elementos capazes de prover, bem como também limitar, a escalabilidade do sistema frente ao aumento da demanda para os vídeos mais populares do sistema.

4.3 Fundamentos do Processo de Medição

Para tornar mais clara a avaliação de desempenho do *proxy* frente a uma determinada carga de trabalho, introduziu-se no escopo desse estudo o conceito de *rodada de serviço* que consiste na execução de um conjunto de tarefas pelo *proxy* para servir cada um dos seus clientes ativos com uma quantidade predeterminada de vídeo[11].

Neste contexto, as operações principais que o sistema precisa desempenhar durante a execução de uma rodada de serviço são:

1. Verificar a presença do conteúdo requisitado na memória e, quando necessário, enviar uma solicitação de conteúdo para o servidor principal.
2. Receber os dados solicitados ao servidor principal e executar o algoritmo de substituição para liberar espaço na memória para armazenar este conteúdo.
3. Admitir novos clientes e fechar as sessões dos clientes quando solicitado.
4. Atualizar as prioridades de cacheamento dos trechos de vídeo.
5. Entregar o respectivo conteúdo para cada cliente.

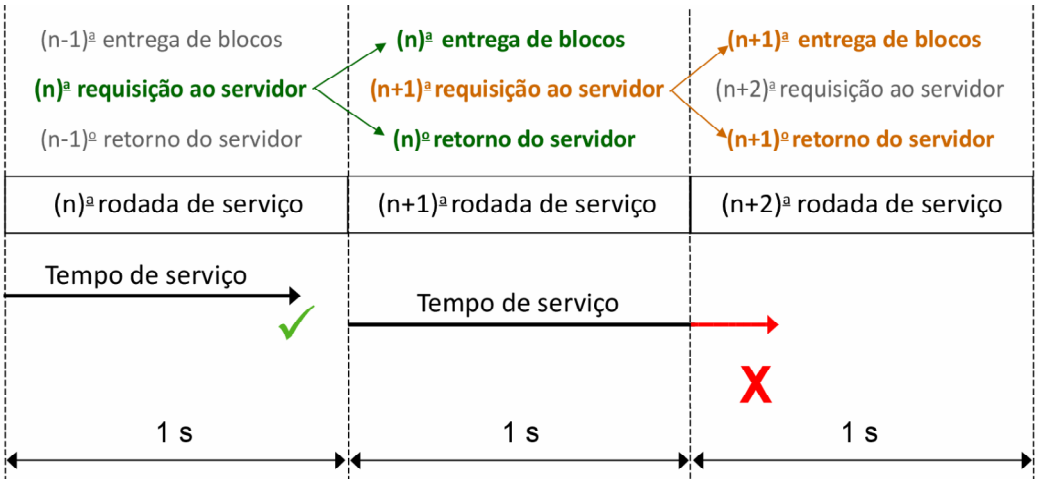
Naturalmente, considerando que a operação 1 foi executada, as operações 2 e 3 têm o objetivo de preparar o *proxy* para a transmissão dos blocos de vídeo para clientes, que ocorrerá nas próximas rodadas de serviço na linha de tempo, como descrito na figura 4.3.

Complementarmente, assumindo que o *proxy* envia o conteúdo para seus clientes usando uma taxa de transmissão constante, igual à (também constante) taxa de codificação do vídeo, um novo bloco de vídeo, contendo um segundo do vídeo, deve ser entregue para cada cliente ativo a cada segundo. Portanto, em um cenário onde a entrega dos blocos de vídeo para os clientes é sempre a última tarefa executada em uma rodada de serviço, considera-se que o *proxy* terá perdido seu prazo para servir parte de seus clientes se a rodada levar um

[11] nos experimentos realizados, foi realizada a entrega de segundo de vídeo para cada cliente.

tempo maior do que um segundo para terminar. Tendo essa condição como referência, adotou-se a rodada de serviço como a unidade de esforço para avaliação do desempenho do *proxy* e, por consequência, todos os resultados produzidos pelo SIMPRO estão relacionados ao desempenho do sistema para executar uma ou mais rodadas de serviço.

Figura 4.3 - Rodada de serviço: conjunto de operações executadas pelo *proxy* VoD, em cada unidade de tempo, para servir seus clientes



Fonte: do autor (2015).

4.4 Fluxo de Simulação

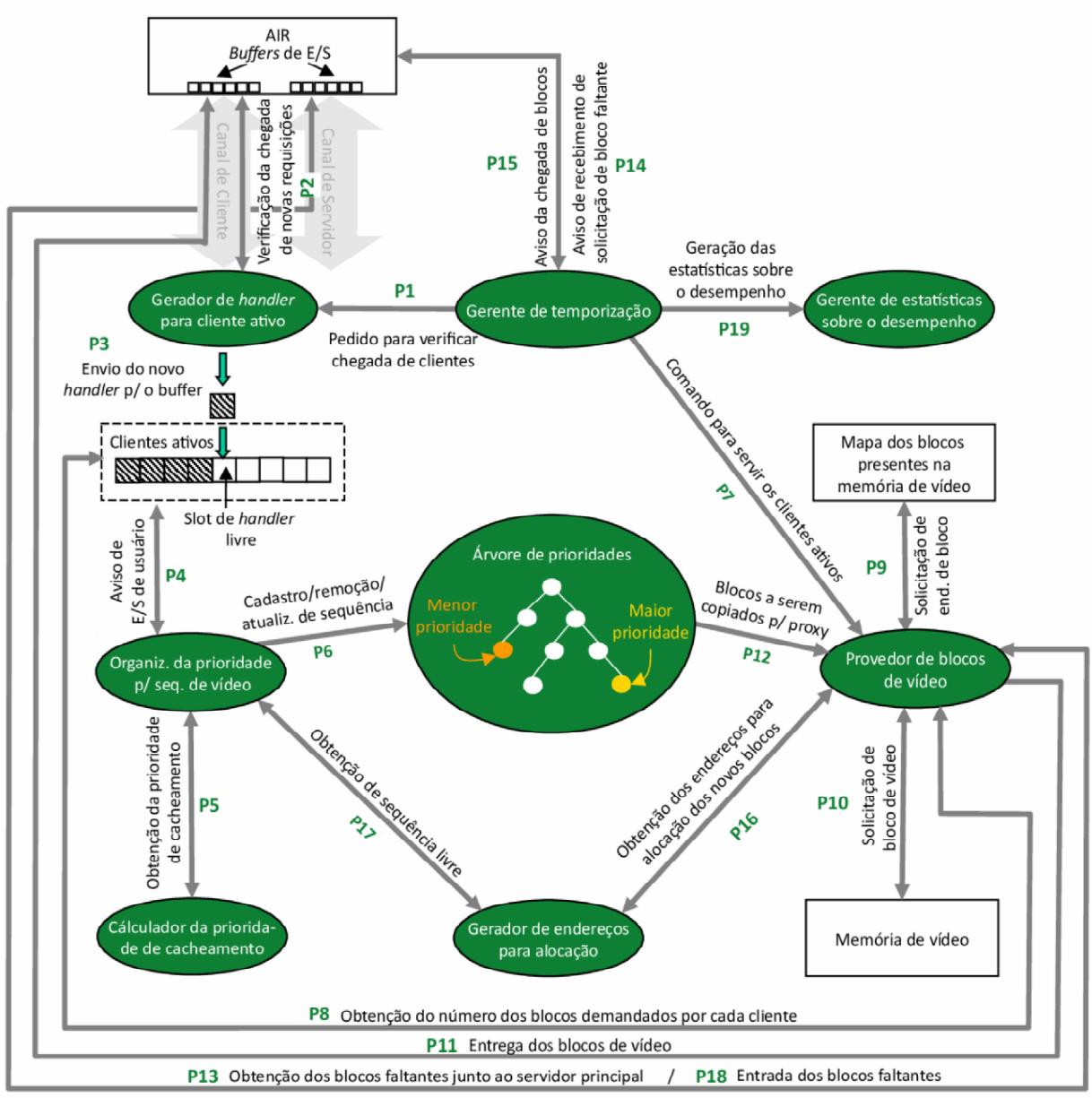
A figura 4.4 ilustra os diferentes passos do fluxo de simulação de uma rodada de serviço através do SIMPRO. No passo 1 (P1), o gerente de temporização inicia a rodada de serviço solicitando ao gerador de *handler* que execute a verificação da chegada de novas requisições aos buffers de E/S do AIR. A chegada de requisições é emulada no passo 2 fazendo com que o gerador de *handler* acesse o arquivo SIV (armazenado no AIR) a partir de um cursor que aponta para a última posição (linha) do SIV acessada durante a simulação.

Conforme mostra a figura 4.5(a), que ilustra o acesso do gerador de *handler* ao arquivo SIV, na primeira rodada de serviço, o gerador usa o cursor para acessar a primeira posição do SIV e verifica se o instante previsto (mensurado em segundos ou, equivalentemente, em rodadas de serviço) para abertura da sessão de vídeo do cliente coincide com o tempo corrente de simulação[12]. Como o resultado da comparação é negativo, o gerador de *handler* não avança para a próxima linha do arquivo SIV. Durante as duas rodadas de serviço subsequentes, o

[12] que, neste caso, equivale ao primeiro segundo ou primeira rodada de serviço da simulação.

mesmo comportamento se repete. Na quarta rodada de serviço, o gerador de *handler*, identifica que o número de rodada de serviço corrente é igual ao número da rodada de serviço prevista para chegada da requisição. Com base nesta condição, o gerador de *handler* executa o passo 3 do fluxo de simulação, criando um *handler* de sessão, para o cliente correspondente, e o armazenando no buffer dos clientes ativos. Em seguida, o cursor de leitura do SIV avança para a próxima entrada do arquivo e verifica novamente o instante de chegada do cliente. Como este instante não corresponde ao número da rodada de serviço corrente, nenhum outro *handler* é criado para a rodada de serviço atual.

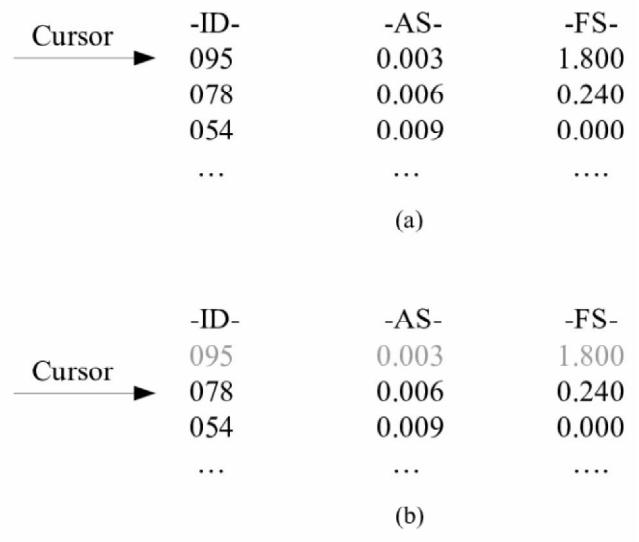
Figura 4.4 - Fluxo de dados em uma rodada de serviço



Fonte: do autor (2015).

Em seguida, no passo 4, assumindo o uso de mecanismo de organização lógica dos trechos de vídeos sequências (conforme explicado na seção 3.1), o organizador de prioridades para sequências de vídeo leva em consideração o(s) cliente(s) entrante(s) na rodada de serviço corrente para criar um novo nó para a árvore de prioridades. Este novo nó conterá informações sobre a sequência que se originou em decorrência da chegada dos clientes na atual rodada de serviço.

Figura 4.5 - Ação executada pelo gerador de *handler* para clientes ativos

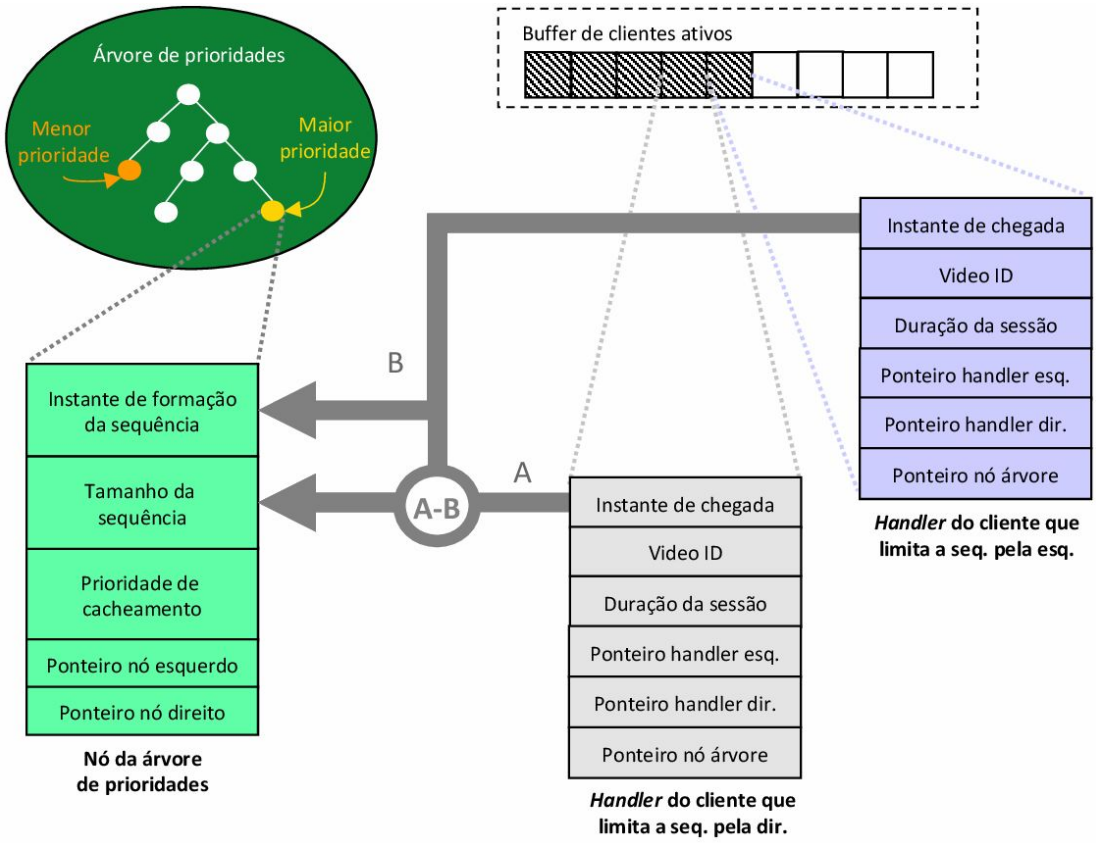


Fonte: do autor (2015).

A figura 4.6 apresenta a estrutura de um nó para a árvore de prioridades. O primeiro campo da estrutura armazena o instante de formação da sequência, o qual é obtido acessando o campo instante de chegada do *handler* vinculado ao último cliente que aderiu ao sistema. A importância do armazenamento do instante de formação da sequência reside no fato de que, quando a sequência é selecionada para descarte, é necessário localizar cada um dos seus blocos de vídeo na memória. Assim, subtraindo o instante de formação da sequência do tempo corrente de simulação, obtém-se a posição (número de bloco) corrente de início da sequência dentro do respectivo vídeo. A partir desta posição, o conjunto completo de blocos a serem desalocados se estende desde o início até o final da sequência, sendo que o número do último bloco (final da sequência) é obtido através da soma do tamanho da sequência (segundo campo do nó da árvore de prioridades) com o número do bloco correspondente à posição de início da mesma. O tamanho da sequência, por sua vez, é obtido subtraindo-se a posição do cliente que limita a sequência pela esquerda da posição do cliente que limita a sequência pela direita, conforme ilustrado na figura 4.6. Por último, cada nó da árvore de prioridades armazena

também a prioridade de cacheamento para a sequência, calculada no passo 5. Uma vez definidos os valores para cada campo do nó, o passo 6 consiste no cadastramento do nó na árvore de acordo com o valor do campo de prioridade, deixando à esquerda os nós com menor prioridade e, por consequência, à direita os nós com maior prioridade de cacheamento.

Figura 4.6 - Criação de um nó para a árvore de prioridades



Fonte: do autor (2015).

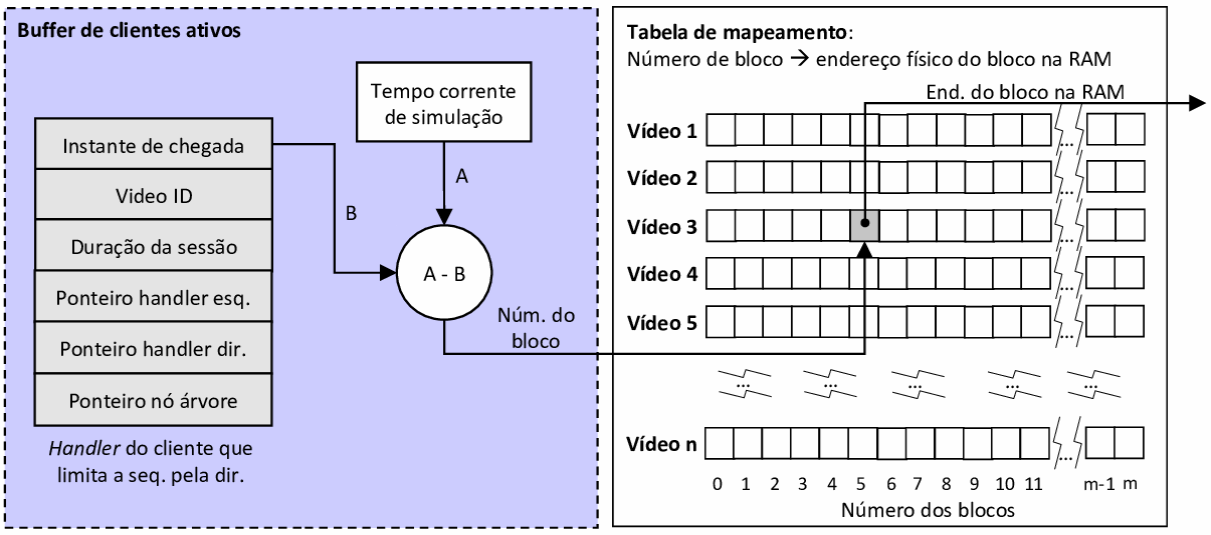
Complementarmente, ainda no contexto do passo 6 do fluxo apresentado na figura 4.4, o Organizador de prioridades também realiza:

- A atualização da prioridade de cacheamento para todas as sequências cuja janela de tempo englobe o(s) novo(s) cliente(s) que aderiram ao *proxy*.
- Remover da árvore de prioridades a sequência afetada pelo encerramento da sessão de vídeo de um cliente.

Uma vez concluído o passo 6, o gerente de temporização executa o passo 7, que consiste no envio de um comando, direcionado ao Provedor de blocos de vídeo, para que seja iniciado o processo de transferência dos blocos de vídeo para os clientes ativos. No passo 8, o provedor se comunica com o Buffer de clientes ativos para obter o número do bloco de vídeo

a ser transmitido para cada cliente. Cada bloco de vídeo possui um número de acordo com sua respectiva posição dentro do vídeo e um vídeo, por sua vez, possui uma quantidade de blocos[13] de acordo com sua duração.

Figura 4.7 - Processo de obtenção do endereço físico a partir do número do bloco



Fonte: do autor (2015).

No passo 9, o Provedor de blocos de vídeo usa o número dos blocos para obter o endereço físico (RAM) dos blocos na memória de vídeo do *proxy*. A figura 4.7 ilustra de forma mais detalhada o processo de obtenção do endereço de memória para cada bloco. A parte da esquerda desta figura mostra como o Buffer de clientes ativos calcula o número do bloco a ser entregue para cada cliente, subtraindo do tempo corrente de simulação o instante de chegada do cliente. Na parte direita da figura, o número do bloco é utilizado para indexar uma das colunas da tabela de mapeamento dos blocos, cuja linha é selecionada com base ID do vídeo vinculado ao cliente. A partir da entrada correspondente à coluna indexada pelo número do bloco, é extraído o endereço físico para acesso ao bloco dentro da memória de vídeo do *proxy*. Se o bloco não estiver presente na memória do *proxy*, o endereço físico armazenado na tabela de mapeamento será inválido (valor igual a -1).

Uma vez obtido o endereço físico para acesso ao bloco desejado, o Provedor de blocos de vídeo executa o passo 10 do fluxo de simulação, descrito na figura 4.4, enviando uma solicitação de bloco para a memória de vídeo hospedada na RAM do *proxy*. No passo 11, o

[13] contendo um segundo de vídeo cada.

Provedor encaminha o bloco lido da memória de vídeo para o AIR, para que seja entregue para o respectivo cliente.

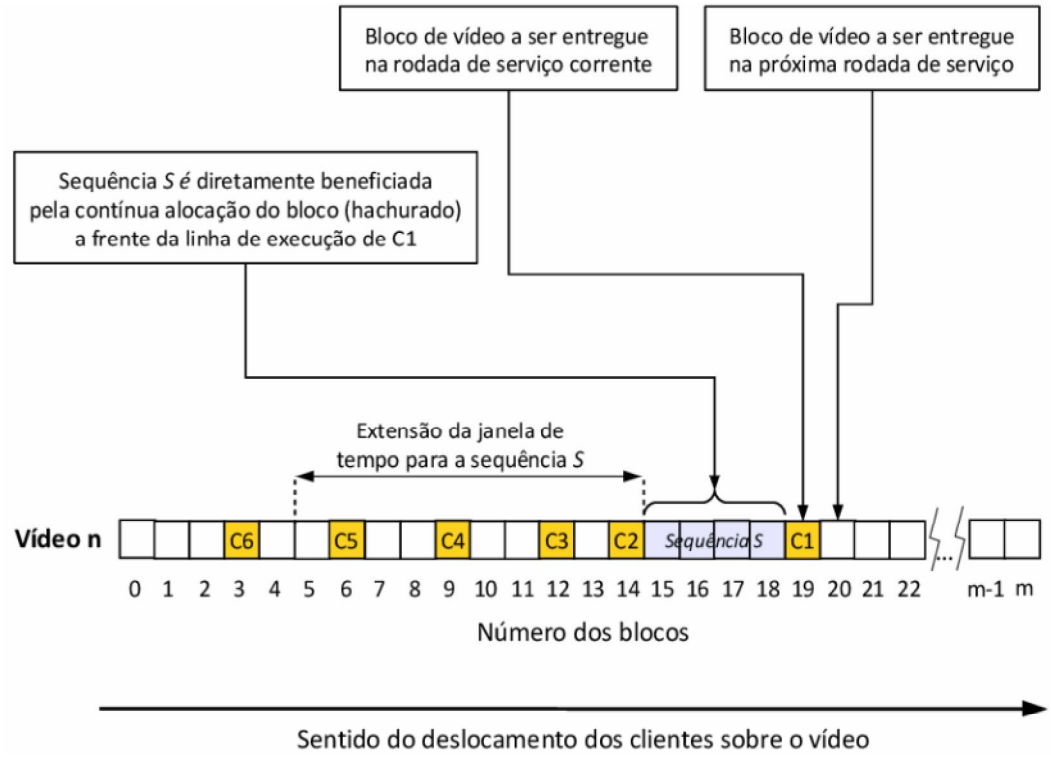
Considerando que no escopo deste presente estudo a entrega dos blocos de vídeo é feita assumindo um fluxo de transmissão CBR (*Constant Bit Rate*) para os clientes e que a latência para retorno de blocos do servidor principal para *proxy* é constante ao longo das simulações realizadas, em termos mínimos, a presença em memória do bloco seguinte à posição corrente de acesso de cada cliente garante a fornecimento contínuo do serviço. Para viabilizar este comportamento, no passo 12, o Provedor de blocos de vídeo acessa a árvore de prioridades para identificar as sequências de vídeo com maior prioridade de cacheamento. Em seguida, no passo 13, o Provedor de blocos encaminha para o AIR um pedido para obtenção dos blocos necessários para abastecer os clientes que estão posicionados à direita das sequências de maior prioridade. Desta forma, as sequencias de vídeo posicionadas à esquerda destes clientes são também diretamente abastecidas, tendendo, com isso, a permanecer à disposição na memória para o acesso dos clientes que estão concentrados dentro das suas respectivas janelas de tempo, conforme ilustra a figura 4.8 para uma sequência de alta prioridade do *proxy*.

A seguir, no passo 14, o AIR informa ao gerente de temporização que recebeu uma solicitação de bloco para ser encaminhada para o servidor principal. O gerente de temporização cadastra o instante de recebimento da notificação enviada pelo AIR e calcula o tempo de retorno do servidor principal. Quando o tempo de retorno expira, o gerente de temporização executa o passo 15, sinalizando para o AIR a chegada dos blocos solicitados ao servidor principal. Depois disso, o provedor de blocos de vídeo executa o passo 16 para obter junto ao Gerador de endereços as posições de memória livres para alocação do conteúdo proveniente do servidor. No passo 17, o Gerador de endereços se comunica com o Organizador de prioridades para obtenção de sequências livres (removidas da árvore de prioridades pelo Organizador). Uma vez que tenha sido alocado o espaço para os blocos de vídeo oriundos do servidor, no passo 18, é realizada a entrada desses blocos na memória de vídeo do *proxy*. Por fim, no passo 19, o Gerente de temporização executa o protocolo de geração das estatísticas de desempenho para a rodada de serviço de acordo com o modo de simulação selecionado pelo usuário do SIMPRO.

É importante salientar que, ao longo de cada simulação, os blocos de vídeo gerenciados pelo *proxy* são todos cópias obtidas a partir de um mesmo bloco extraído de um vídeo real. Assim, o simulador é capaz de imitar os custos decorrentes da entrada de blocos no

proxy, sem que, para isso, exista necessidade de utilizar um disco ou uma interface física (real) de rede.

Figura 4.8 - Abastecimento das sequências de vídeo de maior prioridade de cacheamento



Fonte: do autor (2015).

Adicionalmente, durante o projeto do SIMPRO buscou-se criar representações para os overheads comumente causados por interrupções e chamadas de sistema para acesso à E/S, os quais, segundo (INTEL, 2006) podem contribuir significativamente para o aumento do tempo de execução de aplicações que executam continuamente operações de E/S para transferência de dados via rede. Para isso, foram criadas abstrações para as primitivas de envio (*send*) e recepção (*receive*) de dados, tipicamente presentes no kernel de um SO hospedeiro, no nível de aplicação. Assim, todas as solicitações para transferência de dados através do AIR foram descritas no escopo do simulador fazendo uso destas primitivas.

O acesso às primitivas é inspirado no modelo de vetor de interrupções (*Vectored Interrupt Service Routines* - ISR) (SILBERSCHATZ; GALVIN; GAGNE, 2013), codificando-se a primitiva de envio para a posição 0 do vetor de interrupções e a primitiva de recepção para a posição 1. Neste contexto, tomando como exemplo a circunstância na qual o *Proxy* necessita transmitir um bloco de vídeo, o Provedor de blocos de vídeo, ao executar o passo 11 do fluxo de simulação apresentado na figura 4.4, emitirá um pseudo-trap (dirigido ao

simulador, em vez de para o sistema operacional) passando 0 como argumento para identificar o serviço desejado. Ao receber o pseudo-trap, o simulador atua no lugar do sistema operacional, indexando o vetor de interrupções com o argumento informado pelo *proxy*, a fim de selecionar o *handler* apropriado para tratar a solicitação do Provedor de blocos de vídeo. Depois disso, o simulador suspende o programa principal (algoritmo de cacheamento) e passa o controle do processador para o *handler*, que é responsável por executar a transferência do bloco de vídeo para os buffers de E/S usando o endereço de memória informado através do segundo argumento do pseudo-trap. Uma vez concluída a transferência, o AIR emite uma pseudo-interrupção (implementada nos mesmos moldes do pseudo-trap), fazendo com que o simulador execute uma nova mudança de contexto para devolver o controle à aplicação principal.

Cabe observar que a inspiração para a implementação deste mecanismo para modelagem das chamadas de sistema e interrupções se origina dos estudos apresentados em (SIGOURE, 2013) e (LI; DING; SHEN, 2007), onde os autores demonstram que o tempo para execução dos passos descritos no parágrafo anterior equivale ao custo típico para processamento de uma chamada de sistema.

4.5 Modos de Simulação

Conforme descrito na seção anterior, o último passo do fluxo de execução de uma rodada de serviço é a geração do relatório de desempenho do *proxy*. O sistema de gestão dos relatórios de desempenho do SIMPRO prevê a síntese de relatórios parciais (um relatório por rodada de serviço executada) e um relatório final, com estatísticas gerais (como médias, valores máximos e mínimos e nível de confiança, entre outras medidas) para um intervalo de rodadas de serviço definido pelo usuário do simulador. As métricas de desempenho são mensuradas pelo SIMPRO de acordo com o modo de simulação também selecionado pelo usuário. Três modos de simulação são atualmente sustentados pelo SIMPRO: (1) tempo real, (2) tempo discreto e (3) funcional. Os modos de simulação disponíveis no simulador são apresentados a seguir.

4.5.1 Modo Funcional

No modo funcional (simulação rápida), são obtidos valores para taxa de acertos[14], número de substituições de bloco e largura de banda da rede demandada em cada rodada serviço, bem como outras informações mais detalhadas (se requisitado pelo usuário), tais como dados para depuração do sistema. Além disso, nesse modo, o *proxy* executa todas as suas operações de modo a preservar o estado coerente do sistema ao longo da execução das rodadas de serviço, exceto a movimentação dos blocos de vídeo para dentro e fora da RAM. Como resultado, o tempo de simulação se torna significativamente mais rápido do que aquele obtido através dos modos de tempo real e discreto, permitindo que o projetista possa ter um rápido retorno no que diz respeito às métricas mais comumente usadas para realizar análises de comportamento e de eficiência de um *proxy* VoD.

A figura 4.9 apresenta todo o fluxo de simulação através do modo funcional do SIMPRO. O fluxo inicia através do passo 1 (P1) com a geração do arquivo SIV, que contém a carga de trabalho para a simulação, com base nas configurações informadas pelo usuário do simulador. O conjunto de parâmetros disponíveis para a configuração da carga é exibido na tabela 4.1. Todas as unidades utilizadas para definição dos valores para os parâmetros presentes nesta tabela seguem, em geral, a mesma convenção utilizada em documentos da literatura relacionada à area de desenvolvimento deste presente estudo.

Tabela 4.1 - Conjunto de parâmetros de configuração da carga de trabalho para excecussão das simulações em qualquer modo

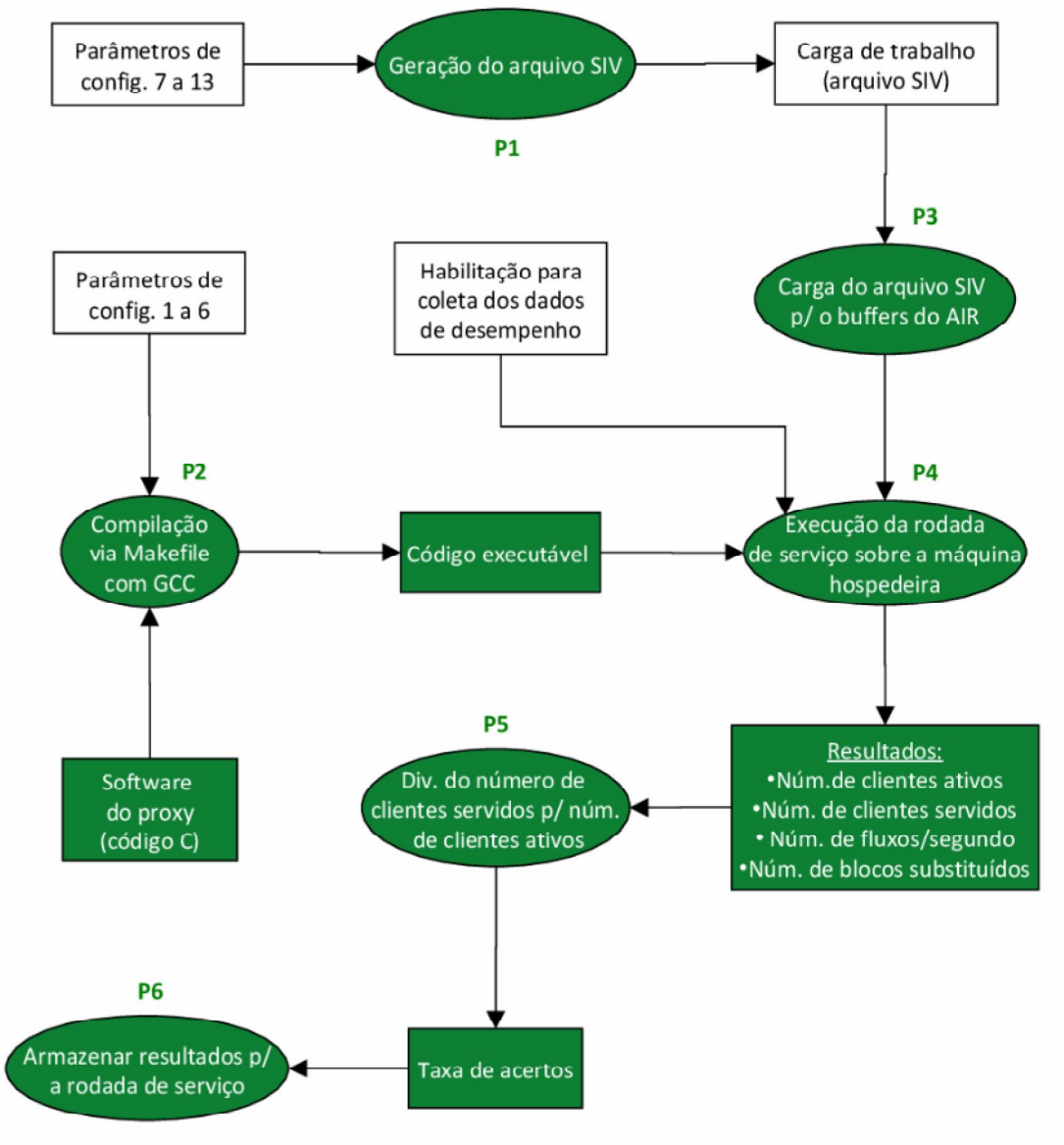
ID	**Nome do parâmetro**
07	Taxa de Transmissão dos Vídeos (Mbps)
08	Duração dos Vídeos (minutos)
09	Número de Vídeos
10	Coeficiente Zipf- θ (inclinação)
11	Intervalo médio entre chegada das Requisições - λ (segundos)
12	Duração da Simulação (minutos)
13	Início do intervalo de análise (minutos)

Fonte: do autor (2015).

Em seguida, no passo 2, ocorre a sínteze do *proxy* com base nas configurações do usuário apresentadas na tabela 4.2. O primeiro parâmetro presente nesta tabela se refere à configuração da largura de banda máxima para o canal de servidor (figura 4.1). A unidade para configuração deste parâmetro é a quantidade de fluxos de vídeo que poderão ser transmitidos, a cada segundo, através do canal que interliga o *proxy* com o servidor principal. Um fluxo de vídeo equivale a um bloco de vídeo que atravessa o canal de servidor na unidade de tempo (segundo).

[14] Correspondente à disponibilidade, na memória do *proxy*, dos blocos de vídeo solicitados.

Figura 4.9 - Fluxo de simulação no modo funcional



Fonte: do autor (2015).

Uma vez que o segundo parâmetro apresentado na tabela 4.2, tamanho da memória de vídeo de *proxy*, é comumente expresso na literatura em GB (Giga Bytes), usou-se a mesma escala no presente trabalho, embora alguns documentos da literatura apresentem a unidade de tamanho da memória como sendo uma fração (percentual) do tamanho total do acervo disponível no servidor principal.

Tabela 4.2 - Conjunto de parâmetros de configuração do sistema para excecussão das simulações nos modos funcional e de tempo real

ID	Nome do parâmetro
01	Max. Largura de Banda no enlace servidor-*proxy* (fluxos por segundo)
02	Tamanho da Memória (GB)
03	Tempo de retorno do Servidor (rodadas de serviço)
04	Algoritmo de cacheamento de vídeo (CARTE, CCVC ou CC)
05	Tamanho da Janela de Tempo (segundos)
06	Tamanho dos Segmentos (segundos de vídeo)

Fonte: do autor (2015).

No que diz respeito ao tempo de retorno do servidor principal, terceiro parâmetro na tabela 4.2, usou-se como unidade de configuração o número de rodadas de serviço por duas razões principais:

1. O uso de valores mais precisos e realistas para configurar o tempo de retorno do servidor principal oferece uma influência muito baixa sobre os resultados produzidos pelo simulador em decorrência do tipo de análise executada neste trabalho, onde o interesse maior é pela análise do comportamento médio do *proxy* durante a execução ao longo de um intervalo de tempo.
2. O número de rodadas de serviço transcorridas provê uma unidade mais genérica para configuração do tempo de retorno do servidor e, consequentemente, mais adequada aos três modos de simulação suportados pelo SIMPRO. No modo funcional, por exemplo, o simulador não executa a E/S dos blocos de vídeo em relação à RAM. Isto faz com que o tempo de execução de uma rodada de serviço seja significativamente menor do que o tempo real necessário para a execução de uma rodada de serviço quando a E/S de blocos ocorre. Consequentemente, qualquer solicitação encaminhada pelo *proxy* ao servidor principal tenderia a ser atendida em uma rodada de serviço diferente daquela onde o retorno ocorreria se a simulação fosse executada em tempo real.

Através do quarto parâmetro presente na tabela 4.2, o usuário do SIMPRO define o algoritmo de cacheamento do *proxy*. O espaço de opções atualmente disponíveis ao usuário é composto pelo algoritmo CARTE (proposto no escopo deste trabalho) e pelos algoritmos CC e CCVC, ambos descritos no capítulo 2 deste documento. O quinto parâmetro da tabela 4.2 é usado para configurar o tamanho da janela de tempo quando o algoritmo selecionado pelo usuário é o CARTE. Por último, o sexto parâmetro define o tamanho dos segmentos de vídeo quando o algoritmo selecionado pelo usuário é o CC. O apêndice G apresenta um script de *shell* para configuração dos parâmetros da tabela 4.2 em modo texto. O apêndice F descreve como é possível, alternativamente, configurar os parâmetros da tabela em modo gráfico

usando as ferramentas *Code::Blocks* (CODE::BLOCKS, 2015) e *cbp2make* (MOLLER, 2015).

Continuando a descrição do fluxo de simulação através do modo funcional, apresentado na figura 4.9, durante o passo 2 ocorre a carga do arquivo SIV para o buffer de E/S do AIR. Na sequência, o passo 3 prevê a execução da simulação sobre a máquina hospedeira. A execução é decomposta em uma quantidade de rodadas de serviço proporcional ao tempo de simulação informado durante a síntese da carga de trabalho.

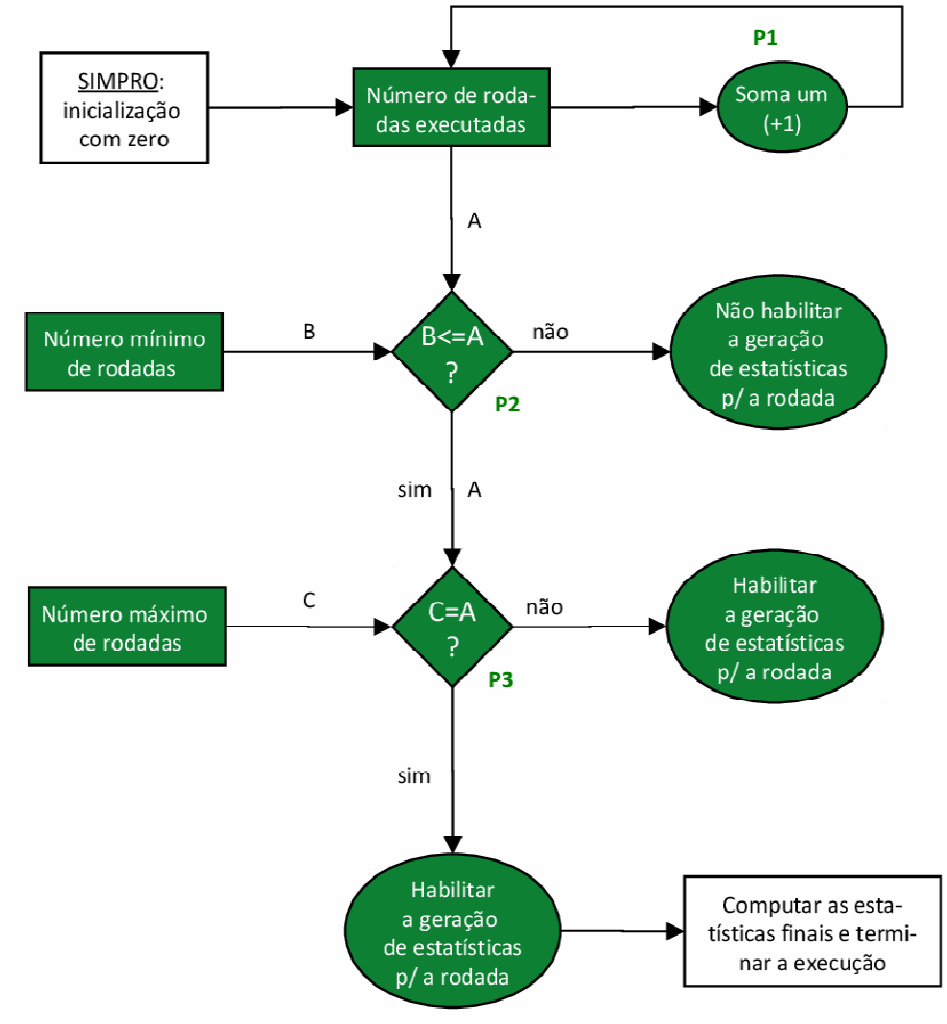
Durante a execução das rodadas de serviço, o sistema somente sintetiza os relatórios de desempenho (um para cada rodada de serviço) se a rodada de serviço a ser executada estiver confinada dentro do intervalo de análise informado através dos parâmetros 12 e 13 apresentados na tabela 4.1. Neste caso, conforme representado na figura 4.9,o simulador enviará para módulo de execução um comando para habilitação da coleta de dados de desempenho. A figura 4.10 descreve como ocorre o fluxo de controle da execução das rodadas de serviço e a emissão do comando para habilitação da coleta de dados sobre o desempenho *proxy* durante a execução de uma rodada de serviço.

Na figura 4.10, o fluxo inicia no passo 1 (P1) com o incremento do contador de rodadas de serviço executadas. Após isto, no passo 2, ocorre a comparação entre o número de rodadas de serviço executadas e o número mínimo de rodada de serviço para início da coleta de dados de desempenho. O número mínimo de rodadas é obtido através da conversão do valor do parâmetro de configuração número 12 (tabela 4.1) para uma escala de segundos, haja vista que cada rodada de serviço equivale ao tempo para transmissão de um segundo de vídeo para cada usuário ativo do sistema.

Se a comparação executada no passo 2 da figura 4.10 for negativa, isto é, se o número de rodadas executas for menor do que o número mínimo de rodadas, o sistema não habilita a coleta dos resultados de desempenho para a rodada de serviço corrente. Do contrário, o fluxo avança para o passo 3 da figura 4.10, onde é realizada a comparação entre o número de rodadas de serviço executadas e o número máximo de rodadas de serviço, usado para delimitar o fim do intervalo de coleta de resultados. O número máximo de rodadas de serviço é obtido através da conversão do valor do parâmetro de configuração número 11 (tabela 4.1) para uma escala de segundos, de forma análoga ao processo de obtenção do número mínimo de rodadas. Se a comparação executada no passo 3 da figura 4.10 for negativa, o sistema habilita a coleta dos resultados de desempenho para a rodada de serviço corrente.

Caso a comparação executada no passo 3 da figura 4.10 seja positiva, o sistema habilita a coleta dos resultados de desempenho para a rodada de serviço corrente e, após isto, computa as estimativas finais para todo o intervalo de análise e encerra a simulação.

Figura 4.10 - Fluxo de controle da execução das rodadas de serviço



Fonte: do autor (2015).

O conjunto de resultados de desempenho originalmente produzidos durante a execução de uma rodada de serviço no modo funcional (passo 4 do fluxo apresentado na figura 4.9) é constituído pelos seguintes itens, também apresentados na figura 4.9:

- Número de clientes ativos
- Número de clientes servidos
- Número de fluxos/segundo no canal de servidor
- Número de blocos de vídeo substituídos pelo algoritmo de cacheamento

No passo 5 do fluxo ilustrado pela figura 4.9, o sistema de relatórios calcula a taxa de acertos para a rodada de serviço corrente, através da divisão do número de clientes servidos pelo número de clientes ativos durante a rodada. Por último, finalizando o fluxo descrito pela figura 4.9, no passo 6, o sistema armazena os dados de desempenho produzidos pela rodada de serviço.

4.5.2 Modo de Tempo Real

No modo de tempo real, o SIMPRO se comunica com o nível físico da máquina hospedeira, usando a API Intel Performance Counter Monitor (IPCM) (WILLHALM, 2012). Esta API explora o hardware dedicado, existente dentro de algumas famílias de processadores Intel, para mensurar o desempenho do software sem interferir no tempo de execução das aplicações. Desta forma, a cada rodada de serviço executada pelo *proxy*, o SIMPRO consegue capturar os dados de desempenho contidos nos registradores dedicados.

Para separar os resultados produzidos em diferentes rodadas, após a coleta dos resultados de desempenho ao final de uma rodada de serviço, os registradores acessados pelo IPCM são zerados.

As métricas mensuradas originalmente no modo de tempo real são:

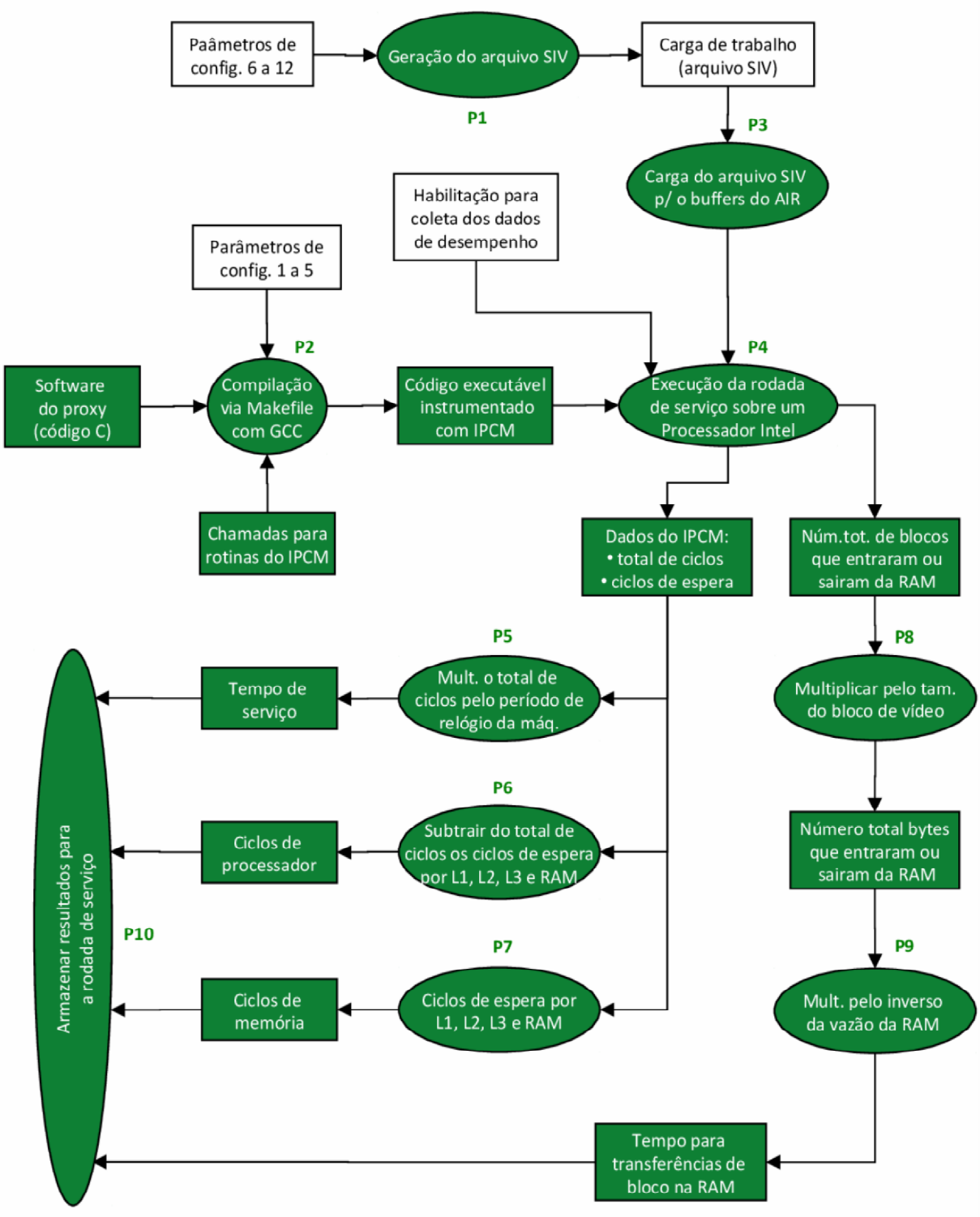
- Número total de ciclos executados na rodada de serviço corrente
- Ciclos em que o processador está em estado de espera para acesso à hierarquia de memória da máquina hospedeira.
- Número total de blocos que entraram ou sairam da memória RAM

A duas primeiras métricas elencadas acima são fornecidas pelo IPCM e a terceira é calculada diretamente pelo *proxy* durante a execução da simulação.

A figura 4.11 apresenta o fluxo de simulação através do modo de tempo real, onde o passo 1 (P1) equivale ao primeiro passo do fluxo de simulação através do modo funcional.

No passo 2, além da aplicação das configurações de sistema informadas pelo usuário, ocorre a incorporação de chamadas para a API do IPCM em meio ao código fonte do *proxy*. No passo 3, ocorre a carga do arquivo SIV (em disco) para o buffer de E/S do AIR. Os ciclos decorrentes da execução do passo 3 não são contabilizados no custo de execução de qualquer uma das rodadas de serviço executadas pelo *proxy*.

Figura 4.11 - Fluxo de simulação no modo de tempo real



Fonte: do autor (2015).

No passo 4, ocorre a execução das rodadas de serviço sobre o hospedeiro. No passo 5, o tempo de serviço (tempo para execução de uma rodada de serviço) é calculado através da multiplicação do número total de ciclos, produzidos durante a execução da rodada, pelo

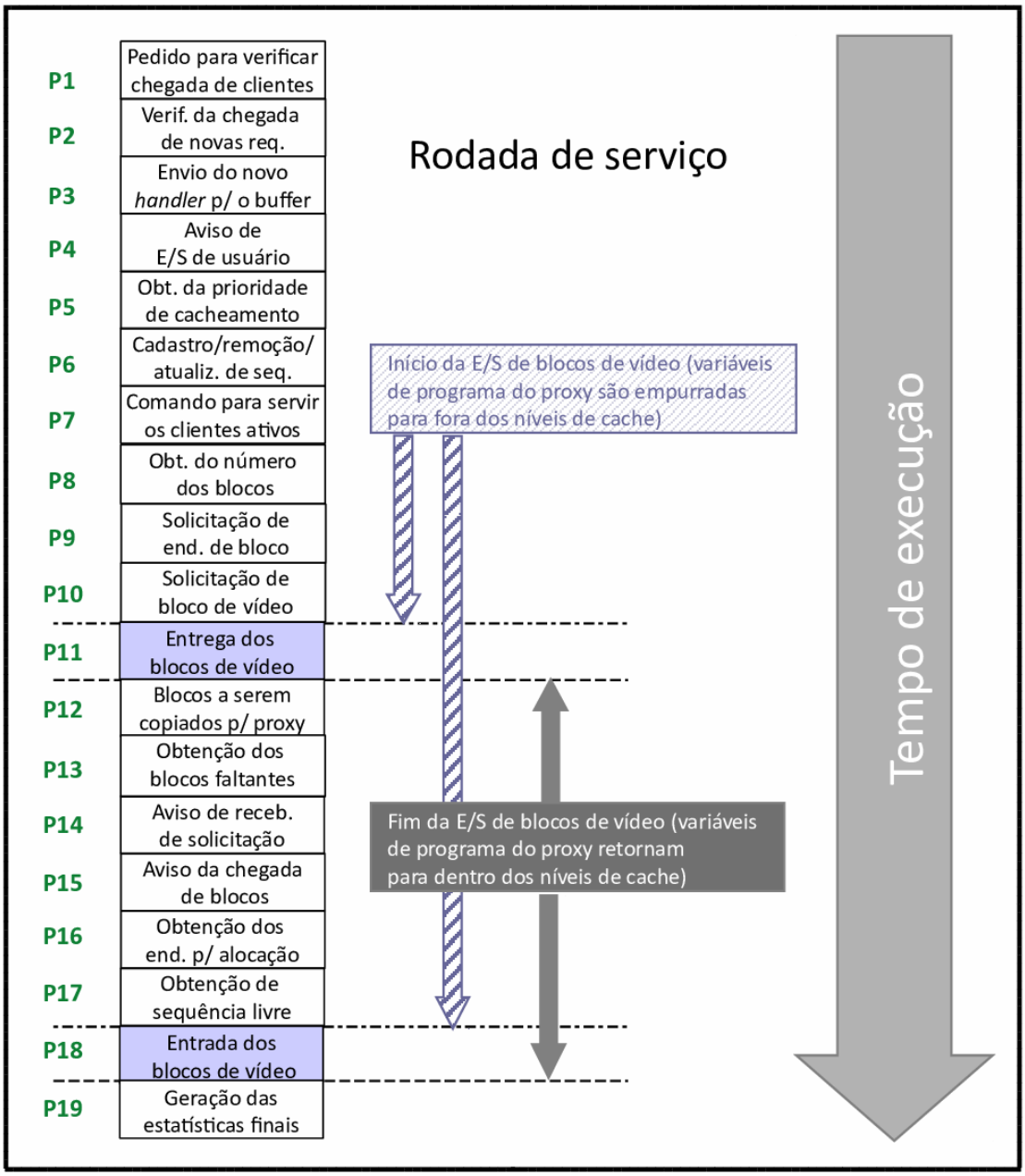
período de relógio do processador. No passo 6, os ciclos consumidos exclusivamente pelo processador são obtidos subtraindo-se do total de ciclos os ciclos que o processador consome em estado espera pela hierarquia de memória. No passo 7, o número de ciclos exclusivos de processador é multiplicado pelo período de relógio do processador para obtenção do tempo exclusivo de processador. Desta forma, o usuário do SIMPRO pode aferir com maior clareza a influência produzida pelo processador sobre o tempo total de execução da rodada de serviço.

No passo 8, os ciclos exclusivos produzidos pelo acesso à hierarquia de memória são obtidos através da soma dos ciclos para acesso a cada nível da hierarquia. O extrato de código fonte apresentado no apêndice C mostra como IPCM coleta os ciclos de espera pela memória através de rotinas que calculam a quantidade de ciclos consumidos para tratar as falhas de acesso a cada nível da hierarquia.

Uma vez que tanto os buffers de E/S do AIR como a memória de vídeo do *proxy* estão alocados na RAM da máquina hospedeira, um aspecto avaliado foi o impacto produzido pelos níveis da hierarquia de memória formados pelas caches do processador. Em um contexto real de funcionamento do *proxy*, a transmissão dos blocos de vídeo entre a memória de vídeo e os buffers da interface de rede precisa ser roteada através dos níveis de cache do processador, uma vez que o processador necessita executar o protocolo de rede (geralmente o TCP/IP) para organizar os blocos após o recebimento (ou antes) do envio via rede. Todavia, se a interface de rede do *proxy* usa recursos dedicados para processamento do protocolo de rede diretamente em hardware, como ocorre, por exemplo, com as placas TOE (*TCP Offload Engine* (BROADCOM, 2009)), a ação desempenhada pelo processador precisa muitas vezes se restringir à execução das instruções necessárias para programar o DMA (*Direct Memory Access* (CORBET; RUBINI; KROAH-HARTMAN, 2005)) para a realização das transferências entre a RAM e a rede.

Para viabilizar a realização do mesmo tipo de análise nos diferentes modos do SIMPRO, os acessos à memória para emular a transferência entre os buffers de E/S do AIR e a memória de vídeo foram implementados usando o processador principal seguindo um modelo de E/S programada (GODSE, 2011). Consequentemente, ao movimentar os blocos de vídeo de uma região para outra da memória ocorre o roteamento dos blocos de vídeo através dos níveis de cache do processador. Estes blocos de vídeo tendem a empurrar as variáveis de programa do *proxy* para fora da cache causando a queda da eficiência do *proxy*.

Figura 4.12 - Lotes de E/S concentrados para reduzir taxa de falhas na cache



Fonte: do autor (2015).

Para tratar esse problema, as transferências dos blocos de vídeo foram todas concentradas em dois lotes de E/S: um para entrega dos blocos de vídeo para os clientes e outro para recebimento dos blocos oriundos do servidor principal. Assim, o processamento das demais etapas inerentes a uma rodada de serviço não é interrompido por transferências esparsas de bloco, de tal modo que as variáveis de programa permanecem na cache durante a execução destas etapas, conforme ilustra a figura 4.12. Depois que cada lote de transferências

de bloco é inteiramente concluído, o *proxy* retorna a execução das demais atividades de uma rodada, causando o retorno das variáveis de programa para os níveis de cache.

Complementarmente, devido à própria natureza vinculada ao padrão de acesso aos blocos de vídeo, realizado pelo algoritmo de cacheamento do *proxy*, e a proporção entre o tamanho dos blocos usados nos experimentos e o tamanho de cada nível de cache, não ocorrem acertos de cache (cache hits) para leitura ou escrita de blocos de vídeo. Este comportamento é desejado, uma vez que, sob esta conjuntura, cada leitura ou escrita de bloco precisa obrigatoriamente ser realizada a partir da memória RAM do hospedeiro, da mesma forma como aconteceria num cenário real de operação do *proxy*.

Dando continuidade à descrição do fluxo de simulação através do modo de tempo real (figura 4.11), no passo 9 os ciclos de espera pela hierarquia de memória são multiplicados pelo período de relógio da máquina hospedeira para obter o tempo exclusivamente produzido pela hierarquia durante a execução de uma rodada de serviço. Assim, com um objetivo semelhante àquele vinculado à execução do passo 7, a execução do passo 9 permite que o usuário do SIMPRO possa aferir com maior clareza a influência produzida pela memória para o tempo total de execução da rodada de serviço.

Como um complemento para as informações produzidas pelo modo de tempo real, o SIMPRO também estima o tempo de memória exclusivo para entrada ou saída de blocos de vídeo em relação à memória RAM. Para isso, primeiramente, no passo 10 da figura 4.11, o volume total de blocos movimentados, produzido pelo passo 4 da mesma figura, é multiplificado pelo tamanho em bytes de um bloco de vídeo. Em seguida, no passo 11, o volume obtido em bytes é multiplicado pela vazão média inversa da RAM do hospedeiro para obter o tempo necessário para transferência dos blocos durante a rodada de serviço.

É importante observar que este resultado não corresponde ao valor de tempo real como o dado produzido pelo passo 9 e, por esse motivo, do ponto de vista taxonômico, poderia ser calculado através do modo funcional do SIMPRO. Todavia, preferiu-se implementar essa estimativa no modo de tempo real para concentrar, tanto quanto possível, as métricas relativas ao tempo de processamento em um mesmo modo. Desta forma, quando o intuito do projetista é avaliar os diferentes tempos de execução produzidos pelo *proxy* em um dado cenário, é necessária a execução de uma única simulação sob a carga desejada. Por último, o passo 12 finaliza o fluxo descrito pela figura 4.11 com armazenamento dos dados de desempenho produzidos pela rodada de serviço.

4.5.3 Modo de Tempo Discreto

No modo discreto (simulação mais lenta), o SIMPRO interage com o Simplescalar (AUSTIN; LARSON; ERNST, 2002) para produzir, por meio de simulações ciclo-a-ciclo, informações adicionais sobre a interface software-hardware, como, por exemplo, um perfil das instruções executadas. Um perfil de instruções pode fornecer informações importantes, como, por exemplo, as instruções mais usadas pelo software, as quais tendem a contribuir para a otimização tanto do software como do hardware usados na implementação do *proxy* VoD.

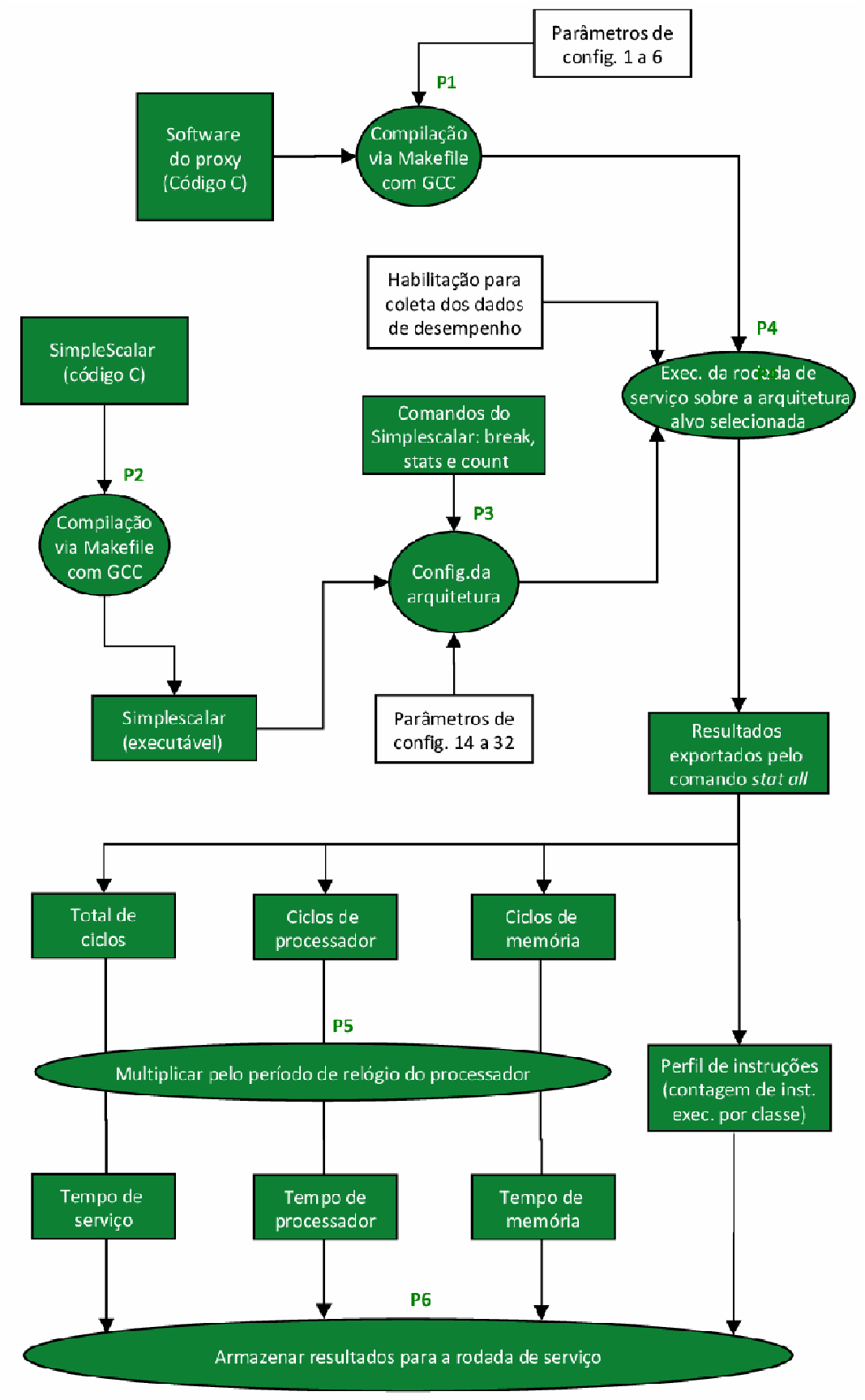
Outro benefício obtido pela criação do modo discreto com o suporte da ferramenta Simplescalar é a possibilidade de avaliar o desempenho do *proxy* usando diferentes arquiteturas como base de processamento. Essa capacidade se torna especialmente importante tendo em vista que a API IPCM, empregada como meio de captura de dados no modo de tempo real, tem sua usabilidade restrita aos processadores Intel.

Em termos práticos, o modo de tempo discreto pode ser usado para avaliar as mesmas métricas providas pelo modo de tempo real, quando os processadores Intel não são o alvo para a avaliação ou quando o processador alvo físico (Intel) não está disponível para executar a simulação no modo de tempo real.

A figura 4.13 apresenta o fluxo de simulação através do modo de tempo discreto do SIMPRO. No primeiro passo (P1), é realizada a compilação do software do *proxy* para execução no ambiente Simplescalar. Nesta etapa, o compilador GCC usado para síntese do software do *proxy* é uma versão disponibilizada para uso com Simplescalar, uma vez que este simulador também permite que o usuário customize a arquitetura alvo com impactos subsequentes no formato e tamanho do conjunto de instruções.

Outra diferença importante entre o passo 1 do fluxo de simulação no modo discreto (figura 4.13) e o passo 2 dos fluxos de simulação no modo funcional (figura 4.9) e no modo de tempo real (figura 4.11) reside na forma como é descrita a inicialização dos buffers de E/S do AIR (seção 4.2) com o conteúdo do arquivo SIV. Enquanto que nos modos funcional e de tempo real essa inicialização é feita prevendo a carga do SIV a partir do disco do hospedeiro, no modo de tempo discreto, essa carga é feita na declaração do AIR, enquanto variável de programa do *proxy*. Assim, usando esta estratégia, não foi necessário implementar um mecanismo para realizar o acesso a este arquivo no disco do hospedeiro.

Figura 4.13 - Fluxo de simulação no modo de tempo discreto



Fonte: do autor (2015).

Em seguida, no passo 2, é feita a compilação do ambiente Simplescalar, usando um arquivo Makefile que possui adaptações em relação ao Makefile original fornecido pelo autores desta ferramenta. As adaptações realizadas tiveram como objetivo instalar complementarmente as bibliotecas *flex-old*, *bison* e *binutils*, necessárias para rodar o Simplescalar sobre o sistema operacional Suse Enterprise Edition, usado para hospedar o SIMPRO nos experimentos realizados. O anexo D apresenta o Makefile modificado usado para compilação do Simplescalar.

Tabela 4.3 - Conjunto complementar de parâmetros de configuração do sistema para excecussão das simulações no modo discreto

ID	Nome do parâmetro
14	Conjunto de instruções (ARM, x86 SPARC,PISA, Alpha ou MIPS)
15	Tam. das caches de nível 1 - separadas para dados e instr. (KB)
16	Tam. da cache nível 2 - unificada para dados e instr. (KB)
17	Tam. da cache nível 3 - unificada para dados e instr. (MB)
18	Tam. de bloco para as caches de nível 1 - separadas para dados e instr. (bytes)
19	Tam. de bloco da cache nível 2 - unificada para dados e instr. (bytes)
20	Tam. de bloco da cache nível 3 - unificada para dados e instr. (bytes)
21	Associatividade para as caches de nível 1 (número de vias)
22	Associatividade da cache nível 2 (número de vias)
23	Associatividade da cache nível 3 (número de vias)
24	Política de substituição para as caches de nível 1 (LRU, LFU ou Random)
25	Política de substituição para a cache nível 2 (LRU, LFU ou Random)
26	Política de substituição para a cache nível 3 (LRU, LFU ou Random)
27	Tempo para acesso ao nível 1 (ciclos)
28	Tempo para acesso ao nível 2 (ciclos)
29	Tempo para acesso ao nível 3 (ciclos)
30	Unidade de predição de desvio (nottaken, taken, bimod, 2lev e comb)
31	Latências para acesso à memória RAM (x-y-y-y, onde x é o tempo para o primeiro acesso e y o tempo para realizar o acesso aos bytes subsequentes)
32	Tam. da fila de requisições load e store da RAM (entradas)

Fonte: do autor (2015).

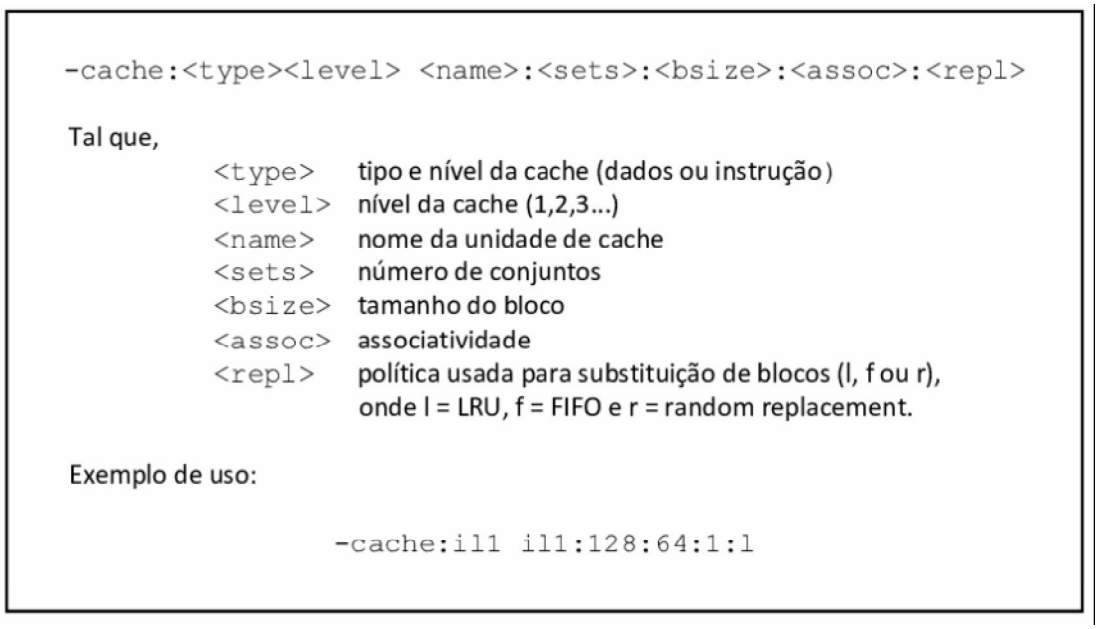
No passo 3, com base no conjunto de parâmetros apresentado na tabela 4.3, é realizada a configuração da arquitetura de base para execução do *proxy*. Os valores para os parâmetros informados pelo usuário são repassados pelo SIMPRO ao Simplescalar através de um arquivo de configuração, cujo formato é exemplificado através do apêndice E deste documento. A carga de um arquivo de configuração é feita no escopo do SIMPRO através de um comando -config <*nome-do-arquivo-de-configuração.cfg*>. Através do parâmetro número 14, presente na tabela 4.3, o usuário seleciona o conjunto de instruções paras as simulações no modo de tempo discreto, podendo optar por arquiteturas ARM, x86, SPARC, PISA, Alpha ou MIPS. Os parâmetros de número 15 a 17 são usados para especificar o tamanho de cada um dos três níveis de cache da arquitetura. Os parâmetros 18 a 20 descrevem o tamanho do bloco de

dados para cada nível. Os parâmetros 21 a 23 descrevem o grau de associatividade usados nos níveis da hierarquia. Os parâmetros 24 a 26 descrevem a política de substituição de blocos em cada nível e os parâmetros 27 a 29 são utilizados para configurar a latência de acessos aos níveis.

Por padrão, o Simplescalar executa uma arquitetura PISA (*Portable Instruction Set Architecture*) com dois níveis de cache unificados, isto é, compartilhadas para dados e código. O uso de mais níveis deve ser explicitamente declarado pelo usuário.

A figura 4.14 ilustra a sintaxe genérica para configurar uma cache no Simplescalar. O exemplo apresentado na parte inferior desta figura descreve como foi configurada a cache nível um de instruções neste trabalho, cujo nome é il1 e possui 128 conjuntos, tal que cada conjunto armazena um único bloco de 64 bytes. A política de substituição nesta cache é um LRU (*Least Recently Used*). Os parâmetros <set>, <bsize>, <assoc> e <repl> presentes na sintaxe descrita pela figura 4.14 foram usados para realizar as configurações dos parâmetros de número 15 a 26 da tabela 4.3.

Figura 4.14 - Sintaxe para configuração de uma memória de cache



Fonte: do autor (2015).

Conforme descrito nesta tabela, os níveis de cache dois e três foram implementados de forma unificada (níveis compartilhados para dados e instruções). A sintaxe usada para implementar os níveis de cache unificados no Simplescalar, é apresentada na figura 4.15. O nível dois unificado de cache foi criado dando à cache de instruções (il2) o mesmo nome da

cache de dados (dl2). Um procedimento análogo foi usado para implementar a unificação das caches de nível três do processador.

Figura 4.15 - Sintaxe para configuração de um nível de cache unificado (compartilhado) no Simplescalar

```
-cache:dl1 dl1:128:64:4:l
-cache:il1 il1:128:64:4:l
-cache:dl2 dl2:512:64:8:l
-cache:il2 dl2
-cache:dl3 dl3:8192:64:16:l
-cache:il3 dl3
```

Fonte: do autor (2015).

A figura 4.16 descreve a sintaxe do Simplescalar para especificação da latência de acesso a cada memória cache do processador. Esta sintaxe foi usada para implementar o tempo de acesso correspondente aos parâmetros 27 a 29 da tabela 4.3.

Figura 4.16 - Sintaxe para configuração da latência de acesso a uma cache no Simplescalar

```
-cache:<name>lat <cycles>
```

Tal que,

<name> nome da cache
<cycles> latência de acesso em ciclos

Exemplo de uso:

```
-cache:dl1lat 4
```

Fonte: do autor (2015).

O exemplo apresentado na parte inferior da figura 4.16, descreve como foi feita a vinculação de uma latência de quatro ciclos para uma cache de dados de nível um usada neste trabalho.

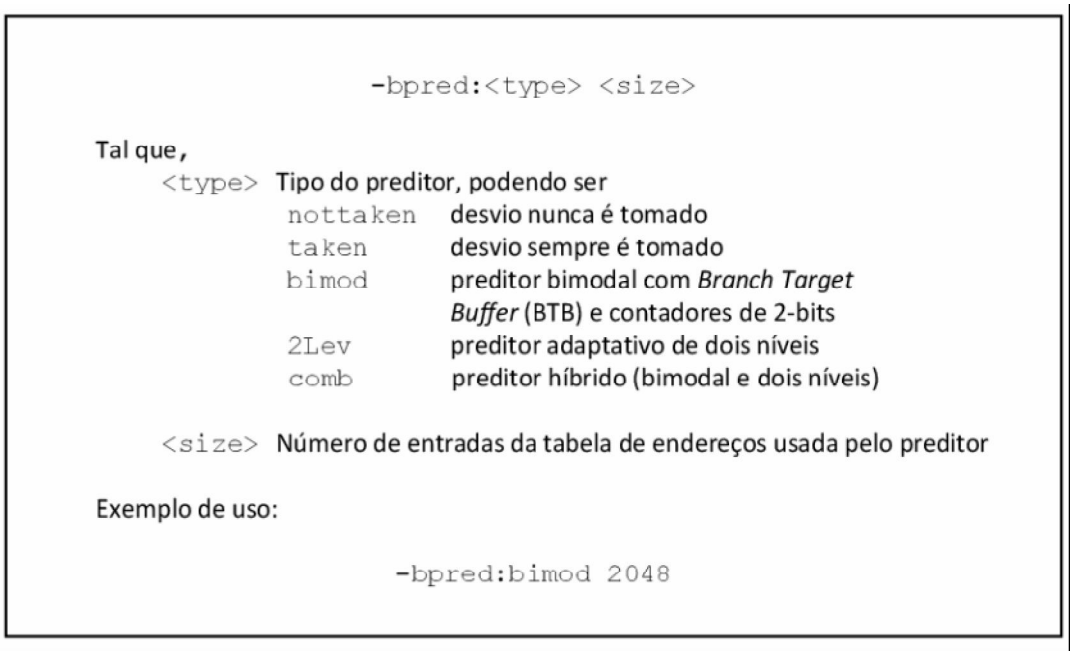
À parte às configurações das caches do processador, o parâmetro de número 30 da tabela 4.3 permite que o projetista especifique as configurações para a unidade de predição de desvios do processador.

Os tipos de preditores suportados pelo Simplescalar são:

- *nottaken*: assume que desvios nunca são tomados.
- *taken*: assume que desvios sempre são tomados.
- *bimod*: preditor bimodal com tabela de endereços.
- *2lev*: preditor em dois níveis com *Branch Target Buffer*.
- *comb*: preditor híbrido (bimodal e dois níveis).

A figura 4.17 apresenta a sintaxe do Simplescalar para especificação das configurações do usuário para o preditor de desvios. A parte inferior da figura exemplifica sintaxe com a declaração de um preditor bimodal com uma tabela de endereço contendo 2048 entradas.

Figura 4.17 - Sintaxe para configuração de preditor de desvios no Simplescalar



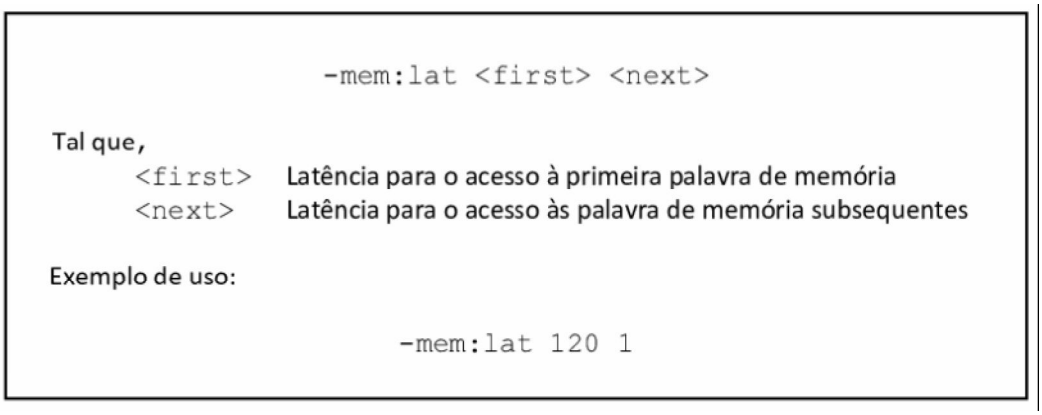
Fonte: do autor (2015).

Retornando à tabela 4.3, o parâmetro de número 32 possibilita a configuração das latências para acesso à memória RAM da arquitetura usando o formato para transferências em modo *burst*, *x-y-y-y*, tal que *x* é a latência para acesso à primeira palavra do bloco de memória[15] e *y* é latência para acesso às demais palavras deste mesmo bloco. A figura 4.18 apresenta a sintaxe do Simplescalar para configuração das latências de acesso à memória RAM. O exemplo na parte inferior desta figura descreve uma RAM, cuja latência para acesso

[15] Existe uma diferença entre bloco de memória e bloco de vídeo. Um bloco de memória possui, tipicamente, poucos bytes de tamanho, enquanto que um bloco de vídeo, possui um tamanho proporcional à taxa de transmissão do vídeo.

à primeira palavra do bloco é 120 ciclos[16] e a latência para acesso às demais palavras deste bloco é um ciclo. É importante observar neste ponto que, em termos de capacidade de armazenamento, a memória RAM da arquitetura é sintetizada pelo Simplescalar com base no tamanho dos dados declarados na aplicação alvo (neste caso, o SIMPRO). Para isso, um requisito fundamental é declaração e subsequente alocação de dados exclusivamente estáticos.

Figura 4.18 - Sintaxe do Simplescalar para configuração da latência de acesso à memória RAM



Fonte: do autor (2015).

Por fim, o último parâmetro apresentado na tabela 4.3, permite que o usuário especifique o tamanho da fila de instruções do tipo *load* ou *store* usado pelo processador para realizar o acesso à RAM. A sintaxe apresentada na figura 4.19 descreve como é feita a configuração do tamanho da fila de acesso à memória. O exemplo apresentado na figura, descreve uma fila de instruções com vinte entradas.

Cabe salientar que outros parâmetros não elencados na tabela 4.3, mas também disponíveis para configuração da arquitetura usada em simulações com o Simplescalar, foram mantidos com suas configurações padrão. Uma descrição completa sobre as configurações padrão usadas para simulações com o Simplescalar está disponível em (BURGER; AUSTIN, 1997).

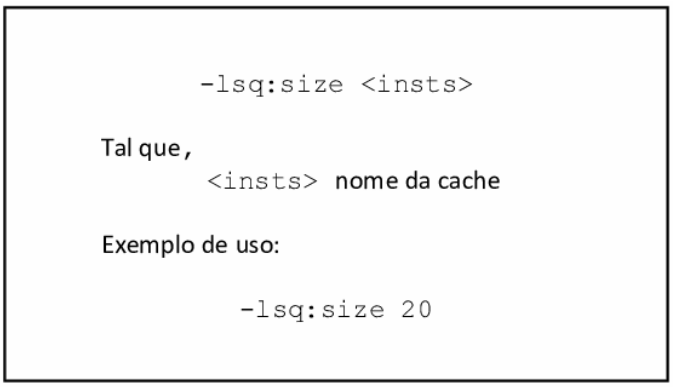
Uma vez realizadas todas as configurações da arquitetura, ainda no passo 3 da figura 4.13 é realizada a configuração do Simplescalar com a execução dos comandos necessários para que o simulador possa coletar os dados de desempenho para cada rodada de serviço. Os comandos executados[17] e suas respectivas ações semânticas para coleta de dados são:

[16] Considerando como referência o período de relógio usado pelo processador.

[17] Através de um shell script (.sh).

- *break prperfdata*: faz o Simplescalar pausar a simulação (e, consequentemente, também na contagem do ciclos) sempre que o *proxy* executa a rotina *prperfdata*, criada especificamente para viabilizar o uso do comando *break*.
- *stats all*: realiza, a cada execução do comando *break*, a exportação dos dados de desempenho da rodada de serviço para a saída padrão do hospedeiro.
- *cont*: o comando cont faz com que a execução do *proxy* seja retomada, após a exportação dos dados, para que o Simplescalar inicie a execução da rodada de serviço seguinte.

Figura 4.19 - Sintaxe do Simplescalar para configuração da fila de instruções (*load* e *store*) para acesso à memória RAM



Fonte: do autor (2015).

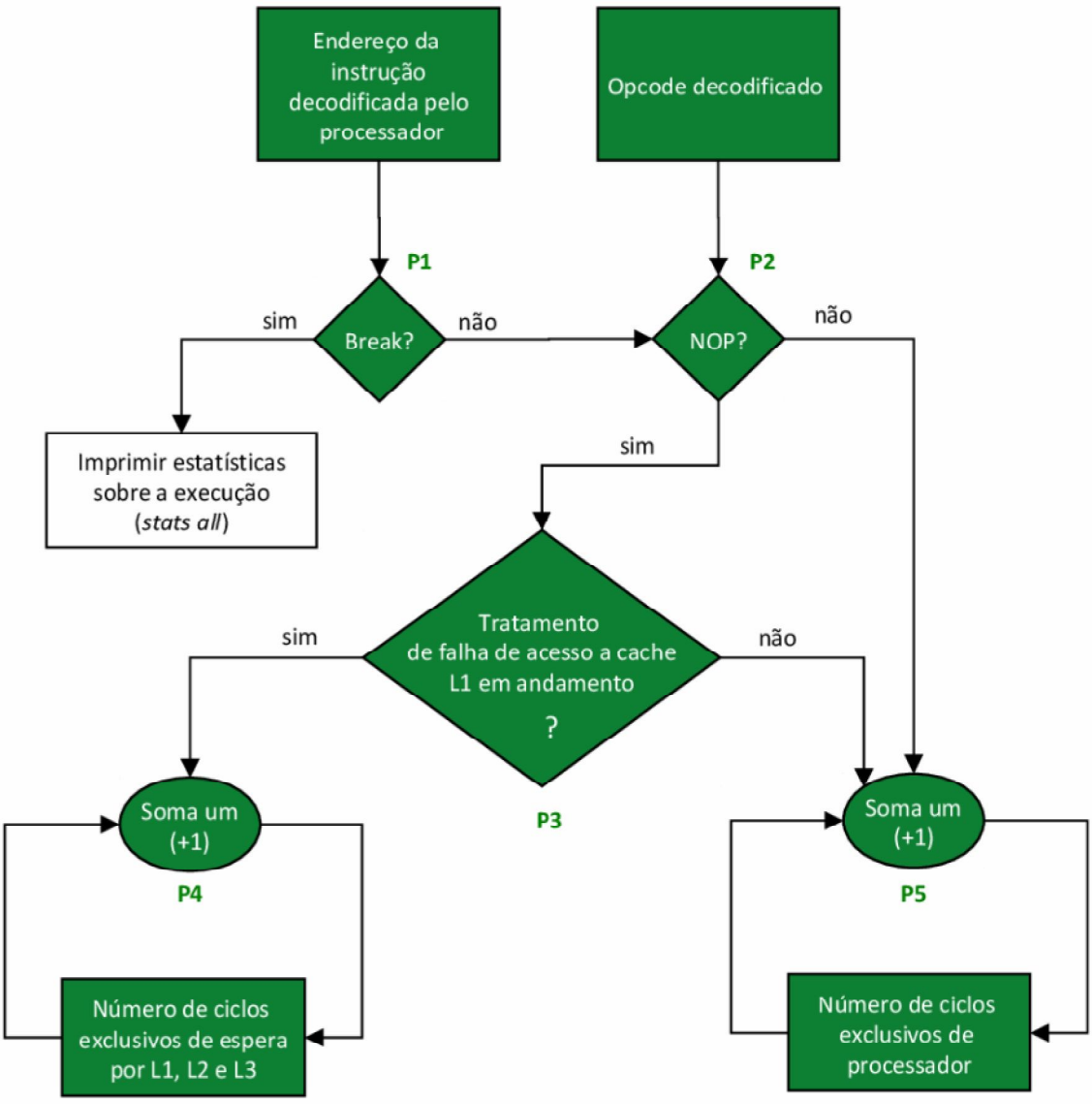
Uma vez realizada a configuração do Simplescalar no passo 3 da figura 4.13, no passo 4 são executadas as rodadas de serviço do *proxy*. A figura 4.20 descreve como é feita a contagem dos ciclos de processador e de espera pela hierarquia de memória no âmbito do passo 4 da figura 4.13. Para implementar as operações descritas na figura 4.20, foi necessário realizar adaptações sobre o código fonte do processador simulado no âmbito do Simplescalar.

O primeiro passo do fluxo apresentado na figura 4.20 consiste em verificar se o endereço decodificado pelo processador corresponde ao endereço de pausa para exportação de dados de desempenho (configurado no terceiro passo do fluxo descrito pela figura 4.13. Se a correspondência for positiva, o Simplescalar interrompe a simulação para exportação dos dados de desempenho. Se a correspondência for negativa, é verificada a equivalência entre o opcode da instrução decodificada e o código numérico da instrução NOP (*No Operation*).

Se a instrução for um NOP, o passo 3 do fluxo apresentado na figura 4.20 consiste em verificar se, no ciclo corrente, o subsistema de memória está tratando uma falha de acesso à cache de nível um. Caso sim, o Simplescalar executa o passo 4 para incrementar o número de

ciclos de espera pela memória. Do contrário, o simulador executa o passo 5 incrementando o número de ciclos executados (execução útil) pelo processador. Conforme também pode ser percebido na figura 4.20, o passo 5 também é executado imediatamente após o passo 2, caso a instrução decodificada pelo processador não seja um NOP.

Figura 4.20 - Fluxo interno de contagem dos ciclos durante uma rodada de serviço no modo de tempo discreto



Fonte: do autor (2015).

Retornando ao fluxo descrito na figura 4.13, no passo 5 as diferentes contagens de ciclos exportadas pelo Simplescalar para uma rodada de serviço são multiplicada pelo período de relógio da máquina simulada. É importante salientar que o período de relógio da máquina, assim como o total de ciclos e o perfil de instruções já são exportados por default pelo

comando *stats all* do Simplescalar. Os ciclos exclusivos de processador e de memória, calculados de acordo com o fluxo descrito pela figura 4.20, precisaram ser explicitamente registrados na lista de variáveis a serem exportadas através do comando *stats all*. Esta lista é mantida pelo pacote *Stats* do Simplescalar.

Por último, o passo 6 finaliza o fluxo descrito pela figura 4.11 com armazenamento dos dados de desempenho produzidos pela rodada de serviço.

4.6 Validação dos Dados de Desempenho

Para uma melhor organização desta secção, optou-se por descrever a validação do modo funcional na subseção 4.6.1. Assim, a apresentação da metodologia para a validação dos modos de tempo real e discreto foi feita em conjunto numa subseção à parte (4.6.2), uma vez que ambos os modos visam a avaliação do *proxy* VoD a partir de métricas físicas de desempenho.

4.6.1 Validação do Modo Funcional

Para validar o modo funcional do SIMPRO, um aspecto preponderante deveria ser a validação do comportamento de cada algoritmo de cacheamento por ele suportado. Para isso, levando-se em consideração que um dos objetivos principais deste estudo é avaliar o desempenho de um grupo de algoritmos de cacheamento, comparando-os em desempenho com o novo algoritmo proposto, o CARTE, foram selecionados dois algoritmos da literatura, que também seguem o paradigma “olhar à frente", para implementação e avaliação: CCVC e CC, ambos descritos resumidamente no capítulo 2.

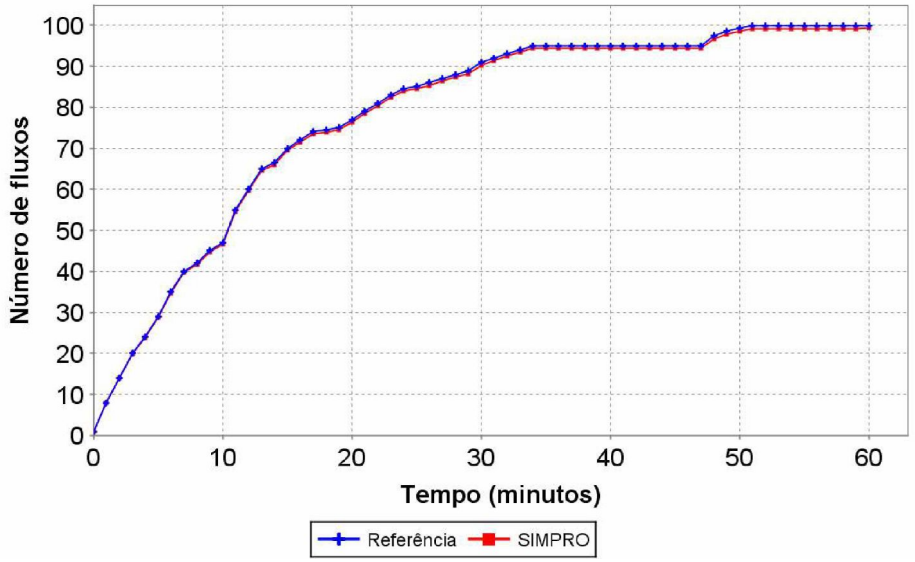
A razão principal para o uso desses algoritmos decorre do fato deles serem os pioneiros em relação à proposição do paradigma de cacheamento seguido pelo CARTE. Por causa disso, sob o ponto de vista do presente estudo, eles se tornaram as principais referências para a avaliação do desempenho de novas propostas feitas sob este contexto.

Todavia, no que diz respeito ao algoritmo CC, não foi possível encontrar na literatura disponível todo o conjunto de informações necessário para reproduzir o ambiente de simulação usado na geração dos dados de desempenho publicados pelos autores deste algoritmo. A exemplo, nenhum documento disponível, para o qual se obteve acesso, descreve

a relação de tamanho entre a unidade básica de tempo de simulação, usada no simulador desenvolvido pelos autores do CC, e o tamanho base dos segmentos de vídeo. Frente à ausência destes dados, não foi possível configurar e validar a réplica deste algoritmo executada pelo SIMPRO.

Por outro lado, no que diz respeito à validação do CCVC, os autores deste algoritmo disponibilizaram em (GRANADO, 2010) um ambiente de simulação reproduzível, juntamente com um conjunto de dados de desempenho que possibilitaram a realização de um experimento de validação. Face à disponibilidade destes recursos, o CCVC foi reproduzido e integrado ao SIMPRO para geração de dados comparativos visando a sua validação. Os parâmetros de simulação usados originalmente pelos autores do CCVC são descritos na tabela 4.4. Os resultados de desempenho originais (sobre o consumo de rede, em termos de fluxos de vídeo, no enlace servidor-*proxy*) são apresentados na figura 4.21 sob o nome curva "Referência".

Figura 4.21 - Validação dos dados produzidos pelo SIMPRO no modo funcional



Fonte: do autor (2015).

Como é possível notar no mesmo gráfico, onde foram plotados os resultados produzidos pela simulação[18] funcional do SIMPRO para as mesmas condições de carga[19], ambos os resultados são muito próximos com uma diferença máxima de 2% (quando tempo =

[18] Foram executadas 30 simulações obtendo-se uma confiança de no mínimo 95% para um intervalo de confidência menor que 1% do valor da medida.

[19] usando uma rodada de serviço como tempo para retorno do servidor principal.

58 min.[20]). Isso permite concluir que os resultados produzidos pelo SIMPRO no modo funcional são válidos, apesar de não ter sido possível validar o conjunto inteiro de métricas suportadas pelo simulador devido à indisponibilidade de resultados produzidos com as mesmas métricas pelos autores do CCVC.

Nesse ponto, cabe salientar que, no contexto deste estudo, o modo funcional foi utilizado para mensurar as taxas de acertos produzidas pelos algoritmos usados na avaliação comparativa de desempenho. A principal razão para essa escolha reside no fato de que essa métrica é menos restritiva do que o consumo de banda de rede, haja vista que a largura de banda associada ao enlace servidor-*proxy* pode ser muito limitada em alguns cenários de operação.

Tabela 4.4 - Parâmetros de carga para validação funcional

Parâmetro	**Valor**
Coeficiente Zipf - θ (inclinação)	0,271
Coeficiente de Poisson - λ (seg.)	3
Número de vídeos	100
Tamanho da cache (percentual sobre o tamanho total do acervo)	10
Duração da simulação (min.)	60
Taxa de transmissão do vídeo (Mbps)	1
Duração dos vídeos (min.)	60 (= 450MB)
Largura de Banda no enlace servidor-*proxy* (fluxos)	1000

Fonte: Granado (2010, p. 46).

4.6.2 Validação dos Modos de Tempo Real e Discreto

Para a validação dos modos de tempo real e discreto do simulador, primeiro, criou-se um conjunto de resultados de referência através da coleta de dados de desempenho durante sua execução do *proxy* VoD em um contexto mais realista, o qual denominou-se no escopo deste estudo de "implementação em rede".

Para essa implementação, usou-se dois computadores, ambos equipados com dois processadores Intel Xeon E5530, 16 GB de DDR3 1066 MHz RAM e interfaces de rede Broadcom 57810S-k de dupla porta e vazão de 10 Gbps, capazes de executar operações TCP/IP diretamente em hardware com o uso da tecnologia TOE. Ambos computadores rodam o sistema operacional SUSE Linux Enterprise Server (3.0 kernel) e eles foram conectados um ao outro usando fibra ótica. Todo o tráfego de rede entre esses computadores foi

[20] Equivalente a execução de 3.480 rodadas de serviço ou, à transmissão de aproximadamente 3.480 Mbits de vídeo a 1Mbps (considerando a transmissão para o primeiro cliente a se tornar ativo no sistema).

implementado usando a biblioteca de programação de sockets para ambientes Unix (POSIX, 2008).

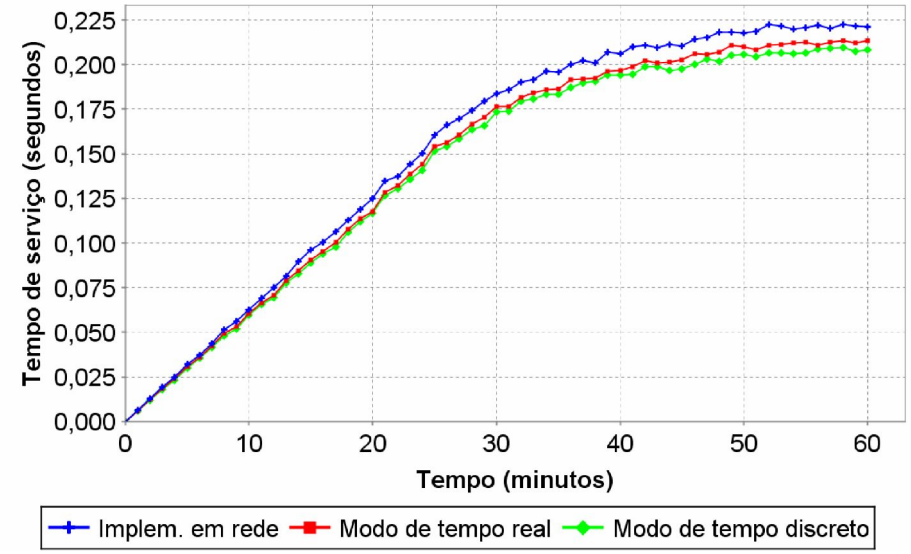
Além disso, para reproduzir um cenário de operação típico de VoD, um dos computadores foi configurado para executar o *proxy* VoD usando o algoritmo CCVC. O outro computador foi configurado para desempenhar simultaneamente o comportamento dos clientes, produzindo uma carga de trabalho para o *proxy* nos mesmos moldes da tabela 4.4, e do servidor principal do sistema VoD. Nesse experimento, os blocos de vídeo não foram decodificados pelos clientes após o recebimento.

Sob essas condições, a validação foi feita através da medição e comparação do tempo de serviço (tempo para execução de uma rodada de serviço) produzido pela implementação em rede e do tempo de serviço resultante da execução usando o SIMPRO nos modos de tempo real e discreto. O tempo de processamento produzido pela implementação em rede foi mensurado através do software IPCM, levando em consideração todos os atrasos causados pela RAM. Para obter o tempo de serviço através do modo discreto do SIMPRO, foi usada uma arquitetura PISA (disponível na biblioteca de núcleos do Simplescalar), a qual, segundo (DECKER et al., 2006), apresenta várias semelhanças com o processador Xeon usado para coleta de dados no modo de tempo real. A tabela 4.5, descreve as configurações da arquitetura para a obtenção do tempo de serviço através do modo de tempo discreto. Estas configurações seguiram as especificações descritas em (BERRENDORF, 2014) para a arquitetura Xeon.

Como mostra a figura 4.22, ambos os modos de tempo real e discreto produzem resultados muito próximos daqueles produzidos pela implementação em rede. O erro produzido pela simulação no modo de tempo real não foi maior que 5,8% (quando o tempo é igual a 26 min.), enquanto que o erro produzido pelo modo discreto não foi maior que 6,2% (para tempo = 59 min.). Esses erros são aceitáveis de acordo com os resultados apresentados em (JIANG et al., 2013; SMIT; STROULIA, 2013; ABAD et al., 2012) para experimentos de natureza semelhante.

Levando-se em consideração que o simulador foi configurado para não levar em conta o custo de processamento do protocolo TCP/IP, baseado na capacidade das interfaces de rede TOE, disponíveis atualmente, atribuíu-se os erros obtidos à execução de algumas operações do protocolo TPC/IP sobre o processador, uma vez que a tecnologia TOE presente nos computadores usados nesse experimento não suporta o conjunto completo de camadas deste protocolo.

Figura 4.22 - Validação dos dados produzidos pelo SIMPRO nos modos de tempo real e discreto



Fonte: do autor (2015).

Tabela 4.5 - Configuração da arquitetura para simulações no modo discreto

ID	Nome do parâmetro	Config. adotada
14	Conj. de instruções (ARM, x86 SPARC,PISA, Alpha e MIPS)	PISA
15	Tam. das caches de nível 1 - separadas para dados e instr. (KB)	32
16	Tam. da cache nível 2 - unificada para dados e instr. (KB)	256
17	Tam. da cache nível 3 - unificada para dados e instr. (MB)	8
18	Tam. de bloco para as caches de nível 1 (bytes)	64
19	Tam. de bloco da cache nível 2 (bytes)	64
20	Tam. de bloco da cache nível 3 (bytes)	64
21	Associatividade da cache nível 1 (número de vias)	4
22	Associatividade da cache nível 2 (número de vias)	8
23	Associatividade da cache nível 3 (número de vias)	16
24	Pol. de subst. para as caches nível 1 (LRU, LFU ou Random)	LRU
25	Política de subst. para a cache nível 2 (LRU, LFU ou Random)	LRU
26	Política de subst. para a cache nível 3 (LRU, LFU ou Random)	LRU
27	Tempo para acesso ao nível 1 (ciclos)	4
28	Tempo para acesso ao nível 2 (ciclos)	10
29	Tempo para acesso ao nível 3 (ciclos)	44
30	Unidade de predição de desvio	bimodal c/ 2048 entr.
31	Latências para acesso a memória RAM (x-y-y-y, onde x é tempo para o primeiro acesso e y o tempo para realizar o acesso aos bytes subsequentes)	120-1-1-1
32	Tam. da fila de requisições load e store da RAM (entradas)	8

Fonte: do autor (2015).

5 RESULTADOS E DISCUSSÕES

Este capítulo apresenta e discute os resultados da comparação de desempenho dos três algoritmos de cacheamento de vídeo previamente apresentados:

- CCVC e CC, ambos introduzidos na seção 2.
- O algoritmo CARTE, descrito na seção 3.

Como estratégia para obtenção de cada resultado, foram executadas 30 simulações obtendo-se uma confiança de no mínimo 95% para um intervalo de confidência inferior a 1% do valor da medida.

As seções a seguir estão organizadas da seguinte forma: a seção 5.1 descreve os parâmetros de carga e de configuração dos algoritmos para execução das simulações. A seção 5.2 analisa comparativamente as taxas de acertos e os tempos de serviço obtidos e a seção 5.3 apresenta uma avaliação dos recursos computacionais consumidos por cada algoritmo analisado. A seção 5.4 mostra uma análise do desempenho dos algoritmos quando executados em cenários com saída prematura de clientes. A seção 5.5 apresenta um estudo dos impactos produzidos sobre a taxa de acertos em função da variação do principal parâmetro de projeto do CARTE: o tamanho da janela de tempo usada para contagem do número de clientes que irão realizar acesso a cada sequência de vídeo. Por fim, a seção 5.6 apresenta a estratégia atualmente em uso para configurar o algoritmo CARTE adequadamente para cada cenário.

5.1 Parâmetros de Carga e de Funcionamento para a Avaliação do *Proxy*

Os resultados para taxa de acertos, tempo de serviço e uso de recursos (tempo exclusivo de processador e de memória) apresentados nesta seção foram todos obtidos, respectivamente, através de simulações com os modos funcional e de tempo real do SIMPRO. Estas simulações foram realizadas com uso de um servidor de propósito geral organizado com um processador Intel Xeon E5530 (2.4 GHz), 16 GB de memória DDR3 (1066MHz) e sistema operacional SUSE Linux Enterprise Server (3.0 kernel).

Através do modo de tempo discreto do SIMPRO, produziu-se um conjunto complementar de resultados, sobre o perfil das instruções executadas pelo *proxy*, o qual também é apresentado nessa seção. Para obter esses dados suplementares, as simulações no modo discreto foram executadas usando como base para execução do *proxy* uma arquitetura PISA, seguindo a mesma estratégia usada para realizar a validação do SIMPRO (seção 4.6).

A topologia assumida para os experimentos considera cenários onde um conjunto de requisições para acesso ao conteúdo fornecido pelo sistema VoD são roteadas através de um *proxy*. Dessa forma, usando a posição instantânea dos clientes ativos, cada algoritmo de cacheamento determina (baseado na sua lógica de priorização) quais trechos de vídeo devem ser preservados na memória (se os trechos já estão alocados na memória) ou solicitados para o servidor principal. Adicionalmente, as requisições direcionadas pelo *proxy* ao servidor principal são feitas assumindo uma existência de uma largura de banda limitada entre estes dois componentes, como ocorre em cenários reais.

Tabela 5.1 - Parâmetros utilizados para configuração do SIMPRO

ID	Acrônimo	Nome do parâmetro	Valor padrão	Faixa de variação	Passo de incremento
01	LB	Máxima Largura de banda no enlace servidor-*proxy* (fluxos/seg.)	200;700	200;700	50
02	TM	Tamanho da memória (GB)	14	4-14	1
03	RS	Tempo de retorno do servidor (rodadas de serviço)	1	-	-
04	AC	Algoritmo de cacheamento (CCVC, CC ou CARTE)	-	-	-
05	JT	Tamanho da janela de tempo do CARTE (seg.)	-	1-300	1
06	TC	Tamanho dos segmentos de vídeo usado pelo CC(seg.)	1 (≈0.63MB)	-	-
07	TT	Taxa de transmissão dos vídeos (Mbps)	5	-	-
08	DV	Duração dos vídeos (min)	90 (=3,4GB)	90-240 (=3,4-9,15GB)	30
09	NV	Número de vídeos	100	50-150	10
10	CZ	Coeficiente Zipf- θ (inclinação)	0,271	0-1	0,25
11	IR	Intervalo médio entre chegada das requisições - λ (seg.)	3	1-10	1
12	DS	Duração da simulação (min.)	DV + 1500 rodadas	-	-
13	IA	Início do intervalo de análise (min.)	DV	-	-

Fonte: do autor (2015).

A tabela 5.1 mostra as configurações usadas para construção da carga de trabalho, bem como os parâmetros para funcionamento dos algoritmos nas simulações. O conjunto de

acrônimos presentes, na tabela, foi usado para substituir os nomes completos dos parâmetros ao longo deste capítulo.

A metodologia experimental consistiu na seleção individual de cada parâmetro da carga de trabalho para variação e avaliação, fixando todos os outros parâmetros usando um valor padrão. Esta estratégia foi seguida com o objetivo de observar claramente as influências causadas por cada parâmetro sobre a eficiência do *proxy*. Para isso, a segunda coluna da tabela 5.1 mostra o valor padrão usado quando o parâmetro de interesse não esteve sob variação. De outra forma, a terceira e a quarta coluna apresentam, respectivamente, o intervalo e o passo de incremento para fazer a variação do parâmetro.

A motivação para o uso das configurações padrões apresentadas na tabela 5.1 é descrita a seguir em cada item. O intervalo para variação é obtido por proximidade em relação ao valor padrão utilizado.

- *Taxa de transmissão dos vídeos*: A configuração adotada corresponde à largura de banda mínima necessária para transmissão de vídeos com qualidade HD (NETFLIX, 2014a).
- *Máxima largura de banda no enlace servidor-proxy*: Em [8], os autores demonstram que o requisito médio para LB, para condições de carga de trabalho similares, não é superior a 700 fluxos. Assim, usou-se este valor como configuração padrão, exceto durante a variação do parâmetro IR onde o valor padrão do LB foi reduzido a 200 fluxos para prevenir que todos os algoritmos obtivessem uma taxa de acertos alta (em resposta ao aumento no IR) a ponto de dificultar a observação dos impactos causados pela variação desse parâmetro.
- *Tamanho da memória*: de acordo com as explicações feitas na seção 4.2, o valor padrão para o parâmetro TM foi configurado para a quantidade máxima de RAM disponível no computador usado para realizar os experimentos, permitindo dessa forma executar simulações com a carga de trabalho mais alta possível.
- *Duração dos vídeos*: as configurações usadas seguem as estimativas apresentadas em (YU et al., 2006; ADHIKARI et al., 2012; CHAN; XU; LIU, 2013; JI, 2013; CAMPANOTTI; HURT, 2013; DEWANGAN; JALIHAL, 2013), com relação à duração média dos filmes disponibilizados pelos provedores VoD.
- *Número de vídeos*: considerando a informação fornecida na seção 4.2, a configuração padrão foi inspirada na quantidade de novos (e altamente populares) vídeos introduzidos diariamente/semanalmente pelo provedores de VoD, tais como (NETFLIX, 2014b; YU et al., 2006; AVRAMOVA et al., 2009).

- *Coeficiente Zipf*: o mesmo valor padrão para esse parâmetro mostrou-se ser amplamente usado para modelar a popularidade de vídeos em (DAN; SITARAM; SHAHABUDDIN, 1994; QUDAH; SARHAN, 2010; YU et al., 2006; RYU; KIM; RAMACHANDRAN, 2011; BATAA et al., 2012; LI; YANG; XI, 2009).
- *Intervalo médio entre chegada das requisições*: segundo (ISHIKAWA; AMORIM, 2003) e (WU et al., 2012), a configuração usada corresponde a de um cenário típico de alta carga de trabalho para *proxies* VoD, levando em consideração tamanhos de vídeo similares ao usado neste estudo.
- *Duração da simulação*: em todas as simulações, o simulador foi configurado para executar uma quantidade de rodadas de serviço equivalente ao tamanho do vídeo, dados em segundos, e mais 1500 rodadas adicionais (cerca de 30 minutos de vídeo ou um terço do valor padrão para DV). Assim, espera-se que durante o intervalo de rodadas extras ocorra tanto a entrada como saída de clientes no *proxy*. Então, para produzir cada resultado básico, se mediu o desempenho médio durante esse intervalo adicional de tempo, uma vez que, dessa forma, cada resultado representa o desempenho obtido para um funcionamento completo do *proxy* VoD.
- *Tamanho da janela de tempo*: para cada cenário, foi avaliada a melhor configuração para o CARTE através da variação do tamanho da janela de tempo sobre um intervalo de 1 a 300 segundos. Assim, nos gráficos apresentados ao longo deste capítulo, o tamanho da janela que gerou uma maior eficiência para o CARTE, considerando esta variação, foi apresentado juntamente com o desempenho obtido pelo algoritmo como o uso desta configuração.
- *Tamanho dos segmentos*: em todos os experimentos, o tamanho dos segmentos de vídeo usado pelo algoritmo CC é equivalente a um bloco de vídeo com um segundo de vídeo. Essa escolha foi feita para prevenir que o algoritmo CC sofresse uma potencial queda na sua taxa de acertos devido ao overhead em memória causado pelo uso de segmentos de tamanhos maiores, como explicado na seção 3.1.
- *Tempo de retorno do servidor*: levando-se em consideração os objetivos e a metodologia utilizada para coleta dos resultados, previamente descritos durante a apresentação da estratégia para configuração do parâmetro DS, optou-se pelo uso deste valor padrão para RS como forma restringir a influência causada por esse parâmetro sobre os valores médios produzidos em cada simulação.

Em adição, seguindo a mesma estratégia empregada pelos autores dos algoritmos usados nas comparações com o CARTE, em todos os cenários experimentais (exceto aqueles apresentados na seção 5.4), os clientes requisitam o vídeo do início ao fim.

5.2 Avaliação da Taxa de Acertos e Tempo de Serviço para a Execução dos Algoritmos de Cacheamento

Com base nas condições de funcionamento descritas na seção anterior (5.1), os gráficos das figuras 5.1(*a*-*f*) e 5.2(*a*-*f*) descrevem os impactos causados pela variação dos parâmetros de carga, respectivamente, sobre a taxa de acertos média e o tempo médio de serviço produzidos pelos algoritmos analisados. Os números que aparecem sobre cada ponto das curvas de desempenho do CARTE correspondem à configuração do tamanho da janela de tempo usado para produzir o desempenho representado por cada ponto.

Uma primeira e geral conclusão sobre os dados apresentados é que, ao custo de um tempo de serviço um pouco mais alto, devido a um mecanismo de substituição mais complexo, a taxa de acertos produzida pelo CARTE é maior do que aquela fornecida pelo CCVC e CC.

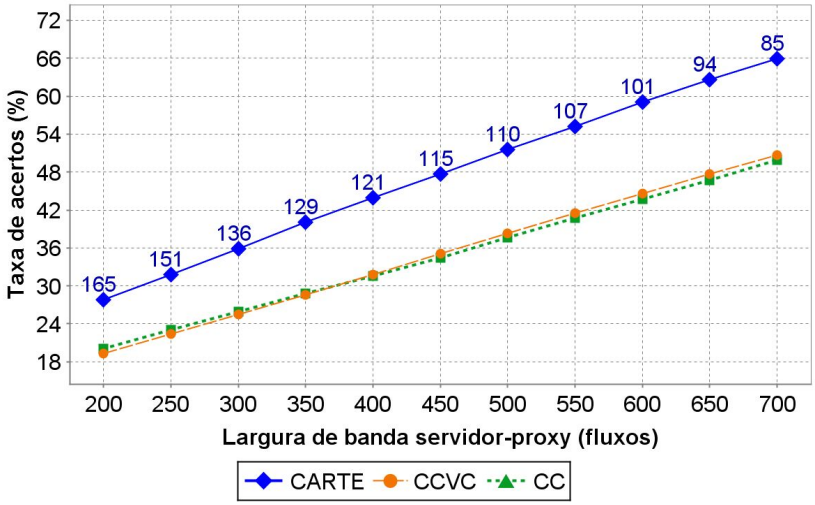
Como demonstrado pelas figuras 5.1(*a*, *b*, *e* e *f*), o aumento aplicado sobre os parâmetros LB, TM, CZ e IR produziu também o aumento da taxa de acertos para todos os algoritmos. Isso ocorre, respectivamente, porque:

- respeitando as limitações impostas pela capacidade de armazenamento da memória, o aumento no LB possibilita que o *proxy* realize um volume maior de substituições na sua memória, permitindo também desta forma que um número maior de clientes possa ter suas demandas atendidas;
- quanto maior o valor para TM, maior é a probabilidade dos conteúdos já armazenados permanecerem na memória, podendo assim serem novamente acessados por clientes mais recentes do *proxy*;
- o aumento de CZ faz com que a demanda do *proxy* se concentre em alguns poucos vídeos mais populares do acervo. Como consequência, os requisitos de memória para sustentar os clientes ativos são reduzidos e um número maior de blocos pertencentes aos vídeos solicitados pode permanecer na memória para serem acessados por clientes mais recentes;

- um valor maior para IR, para um tamanho fixo de vídeo, faz com que o número de clientes ativos do *proxy* se torne menor. Consequentemente, tanto os requisitos de memória como os de rede para atender a demanda se tornam menores em relação ao volume de recursos disponíveis, causando o aumento do desempenho do sistema nesse caso.

Figura 5.1 - Desempenho dos algoritmos de cacheamento: resultados para taxa de acertos

(a) Taxa de acertos para a variação do número de fluxos no canal *proxy*-servidor principal



(b) Taxa de acertos para a variação do tamanho da memória de vídeo do *proxy*

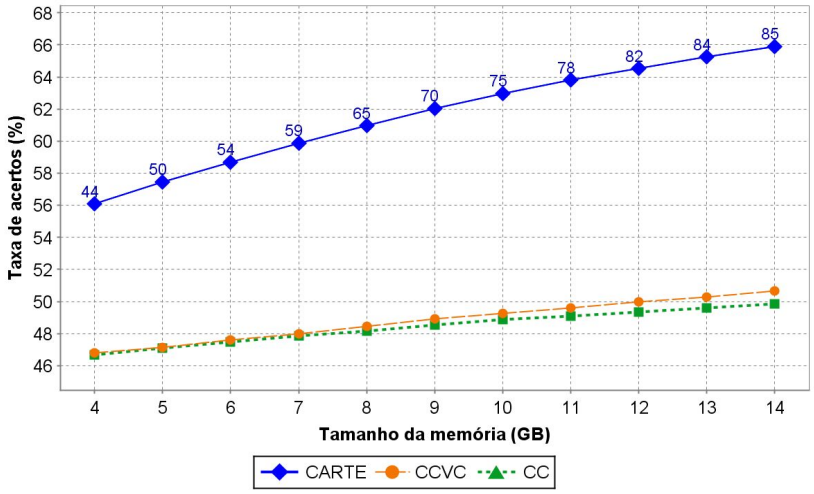
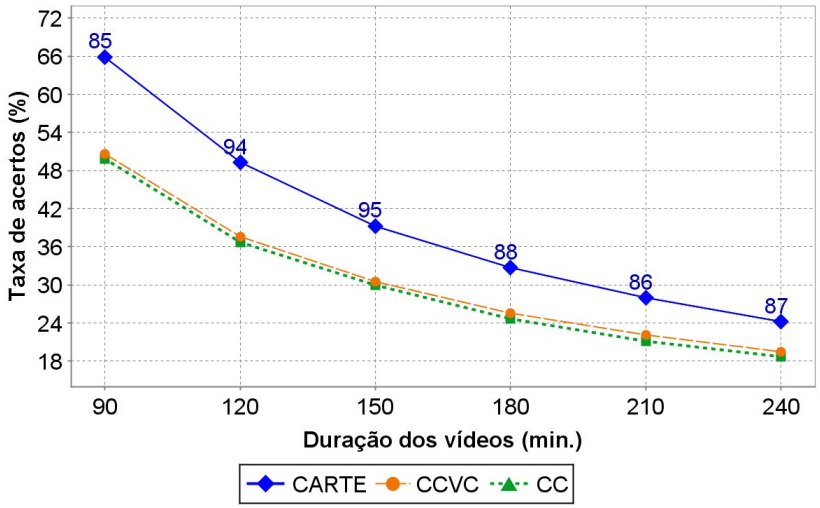
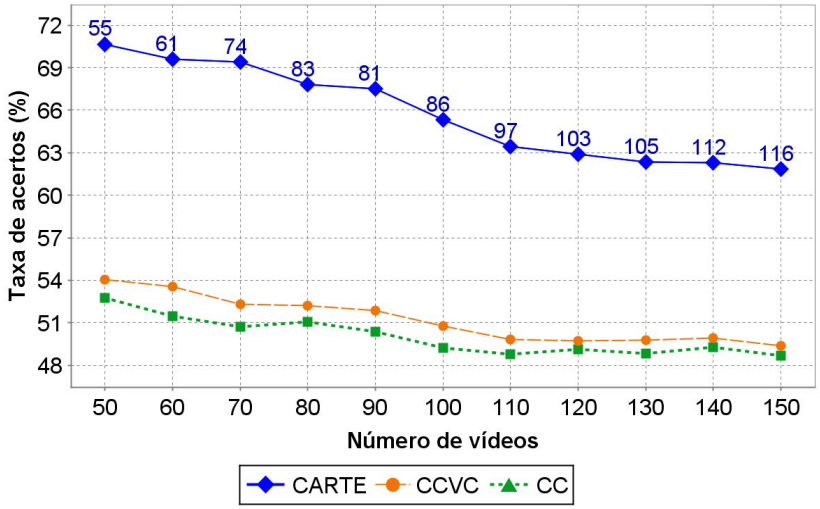


Figura 5.1 - Desempenho dos algoritmos de cacheamento: resultados para taxa de acertos (cont.)

(c) Taxa de acertos para a variação da duração dos vídeos



(d) Taxa de acertos para a variação do número de vídeos disponíveis no acervo



(e) Taxa de acertos para a variação do coeficiente Zipf

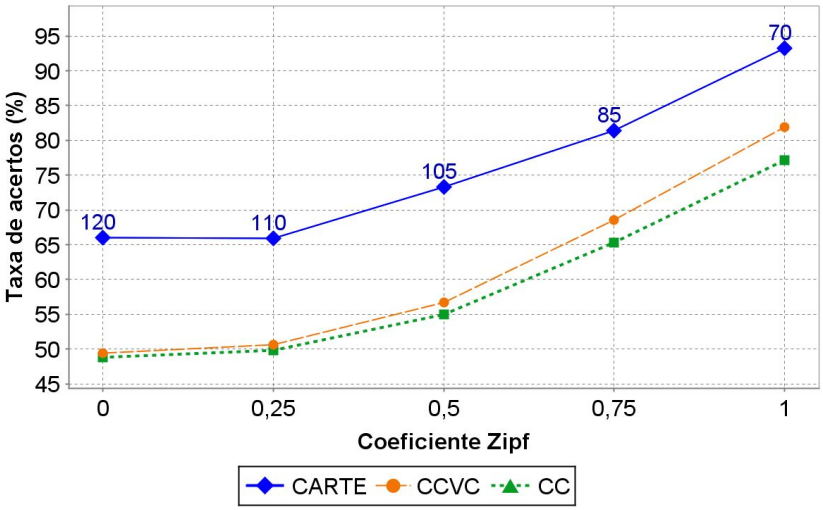
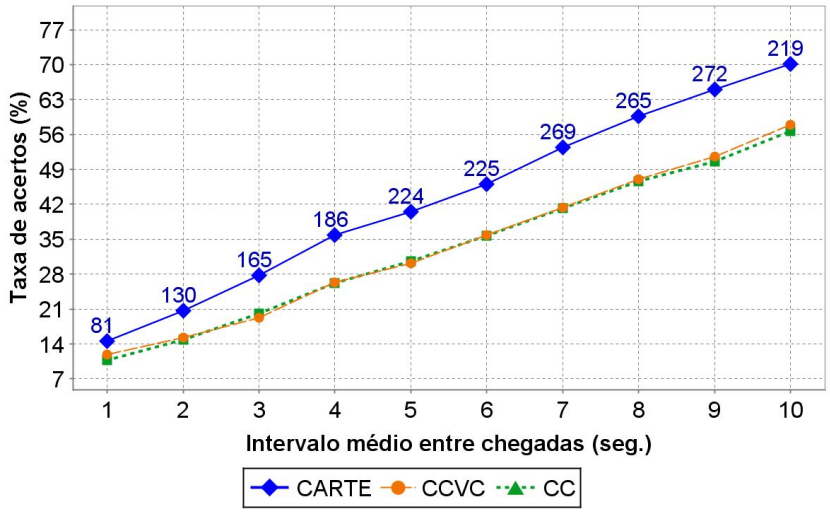


Figura 5.1 - Desempenho dos algoritmos de cacheamento: resultados para taxa de acertos (cont.)

(f) Taxa de acertos para a variação do intervalo médio entre as chegadas das requisições ao *proxy*



Fonte: do autor (2015).

Também considerando as figuras 5.1(*a*, *b* e *e*), é importante destacar dois aspectos levando em conta especificamente o desempenho do CARTE: primeiro, quanto maior a disponibilidade de recursos (capacidade de memória e largura de banda de rede) em relação à demanda, maior a taxa de acertos fornecida pelo CARTE em comparação com aquela fornecida pelos outros dois algoritmos. Segundo, o CARTE alcança uma taxa de acertos comparativamente maior até mesmo em cenários onde a popularidade de vídeos tende em direção à igualdade (ex.: quando CZ assume um valor igual a zero). Com isso, conclui-se que a nova abordagem proposta é não apenas mais escalável, mas também capaz de prover um melhor desempenho ao *proxy* em cenários de pior caso, quando a probabilidade de acesso ao vídeo não pode ser facilmente prevista, como ocorre, por exemplo, em certos períodos para os itens mais populares do acervo (JUNG; KRISHNA-MURTHY; RABINOVICH, 2002).

Adicionalmente, o CARTE também apresenta um desempenho superior quando CZ assume um valor ainda mais alto do que os apresentados na figura 5.1(e). Para exemplificar, foi simulado o cenário onde as configurações padrão são usadas, exceto pelo valor de CZ, configurado para ser igual a 5, e de TM, configurado para ser igual a 2GB para prevenir que todos os algoritmos analisados obtivessem uma alta taxa de acertos devido à significativa redução dos requisitos de memória provocada pelo aumento de ZC[21]. Sob estas condições de

[21] Considerando o número de vídeos utilizados para compor o acervo neste cenário, o uso de CZ=5 cria a tendência para que todos as requisições sejam direcionadas para um mesmo vídeo.

carga, a diferença entre as taxas de acertos do CARTE e do CCVC de 8,1%, enquanto que a diferença para o CC é de 11,8%.

O maior desempenho fornecido pelo CARTE nestes cenários resulta tanto da alocação de sequências menores na memória, fazendo um melhor uso de recursos armazenados, assim como da consideração em nível local das diferentes densidades de clientes existentes em regiões distintas dos vídeos. Esse último fator contrasta, em especial, com o comportamento do algoritmo CC que considera como um todo os clientes que acessam um determinado vídeo para decidir que partes dos conteúdos devem ser mantidas na memória.

Em relação aos gráficos *c* e *d* da figura 5.1, ao contrário do que ocorre nos gráficos *a*, *b*, *e* e *f*, é possível notar que o aumento aplicado aos parâmetros de carga produziram a queda de taxa de acertos para todos os algoritmos. Esse efeito oposto ocorre, respectivamente, porque:

- Para cada aumento aplicado sobre DV, os clientes ativos do *proxy* permanecem recebendo seus respectivos vídeos por um período maior. Como os clientes continuam aderindo ao sistema na mesma frequência de chegada, isso gera o aumento do número de clientes ativos, os quais ficam dispersos sobre um espaço de vídeo maior, resultando na demanda pelo aumento da capacidade de armazenamento da memória, assim como pelo aumento da banda de rede, causando assim a queda da taxa de acertos.
- O aumento do número de vídeos causa a dispersão dos clientes sobre um acervo maior. Como consequência, uma vez que CZ permanece constante ao longo do experimento, os clientes são direcionados para porções do acervo que ainda não foram acessadas por outros clientes ou foram acessadas por um período de tempo suficientemente grande para permitir a sua remoção da memória.

Complementarmente, ao analisar as informações disponíveis na figura 5.2, é possível ver que nos gráficos *a*, *d* e *e* o tempo de serviço para cada algoritmo, em geral, segue o crescimento produzido na taxa de acertos. Isso ocorre porque cada algoritmo precisa empregar um esforço para transferir os blocos de vídeo da memória local para a interface de rede em proporção com o número de clientes a serem servidos. Porém, nos cenários representados pelos gráficos *b*, *c* e *f* na figura 5.2, o tempo de serviço não segue o crescimento da taxa de acertos. Isso ocorre, respectivamente, porque:

- O maior tempo de serviço obtido para tamanhos de memória menores é resultante de um processamento maior para mudar, continuamente, o conteúdo disponível na

memória do *proxy*, objetivando, minimamente, armazenar os blocos de vídeo que estão um passo à frente na linha de execução dos clientes.

- Quanto menor for o valor de DV, assumindo uma configuração fixa para IR, menor tende a ser também o número de clientes ativos servidos pelo *proxy*. Porém, quando DV assume valores pequenos, a disponibilidade de recursos em relação ao número de clientes ativos é maior e, por consequência, a taxa de acertos também se torna maior. Seguindo um raciocínio inverso, à medida que o valor de DV aumenta, a taxa de acertos diminui, mas o número de clientes ativos aumenta. Logo, o número de clientes efetivamente servidos pelo *proxy* tende a permanecer constante durante a variação de DV e, como resultado, o tempo de serviço também permanece quase estável durante esse período.
- Conforme IR aumenta e DV permanece constante no experimento, o número de clientes ativos é reduzido, causando a queda do tempo de serviço.

Figura 5.2 - Desempenho dos algoritmos de cacheamento: resultados para tempo de serviço

(a) Tempo de serviço para a variação do número de fluxos no canal *proxy*-servidor principal

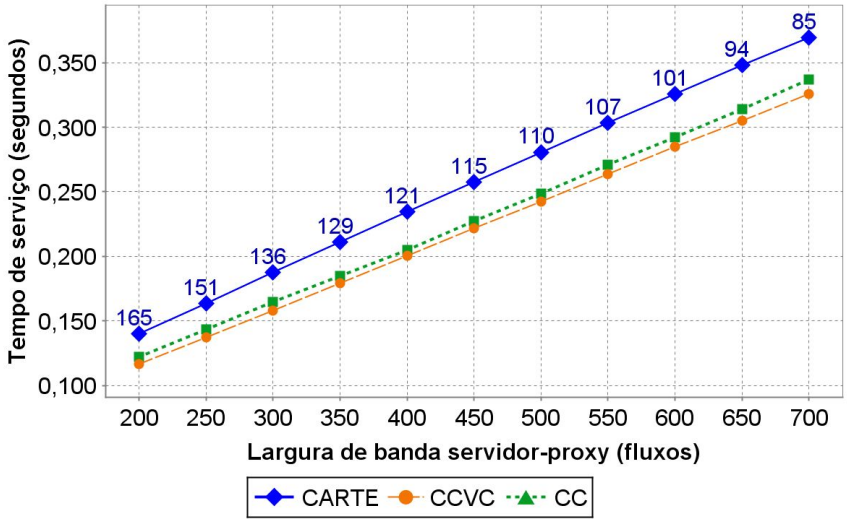
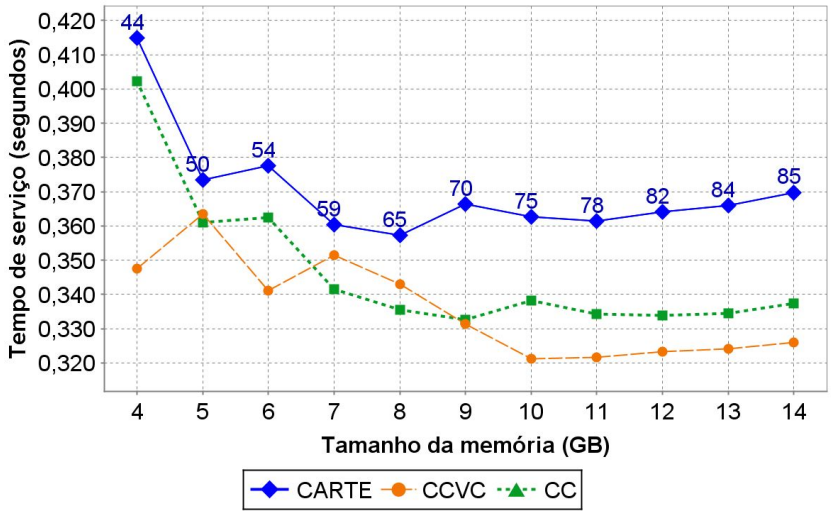
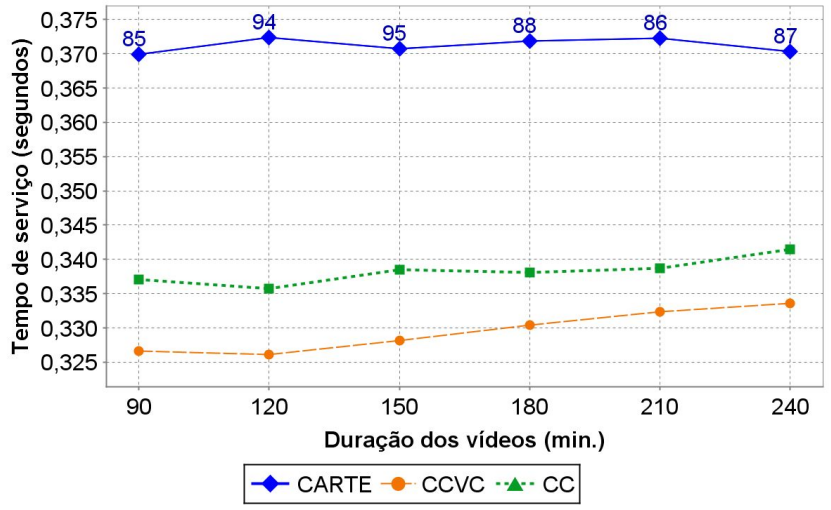


Figura 5.2 - Desempenho dos algoritmos de cacheamento: resultados para tempo de serviço (cont.)

(b) Tempo de serviço para a variação do tamanho da memória de vídeo do *proxy*



(c) Tempo de serviço para a variação da duração dos vídeos



(d) Tempo de serviço para a variação do número de vídeos disponíveis no acervo

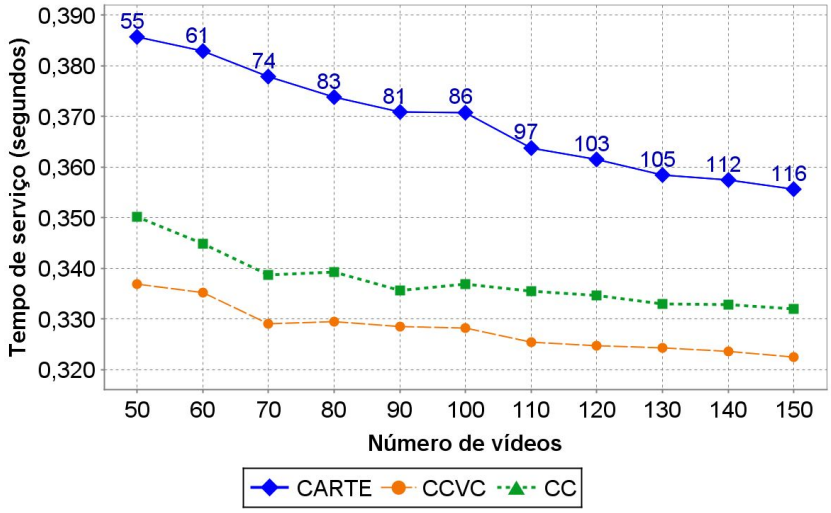
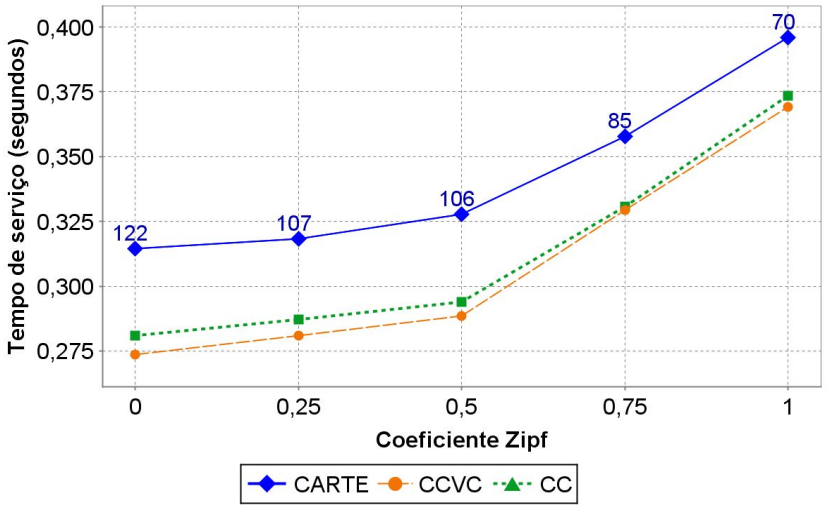
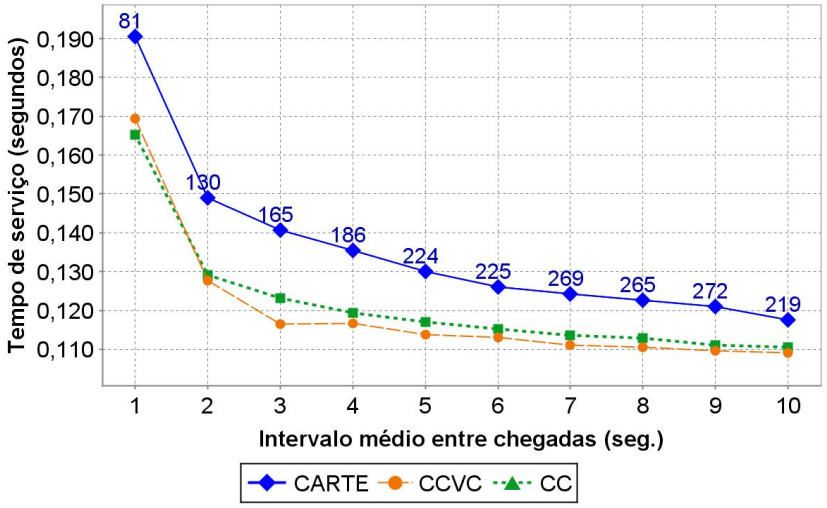


Figura 5.2 - Desempenho dos algoritmos de cacheamento: resultados para tempo de serviço (cont.)

(e) Tempo de serviço para a variação do coeficiente Zipf



(f) Tempo de serviço para a variação do intervalo médio entre as chegadas das requisições ao *proxy*



Fonte: do autor (2015).

A tabela 5.2 descreve os resultados gerais em termos de taxa de acertos e tempo de serviço produzidos pelos algoritmos avaliados. As diferenças médias obtidas entre o CARTE e o CCVC para a taxa de acertos e o tempo de serviço, considerando todas as simulações realizadas, foram, respectivamente, de 12,9% e 11,3%. Para as mesmas métricas, as diferenças médias obtidas quando os parâmetros de funcionamento assumiram seus valores padrão foram, respectivamente, de 14,6% e 11,6%. Complementarmente, a diferença máxima obtida entre as taxas de acertos produzidas por esses algoritmos foi de 17,3%, diferença que foi atingida durante a variação do parâmetro NV - figura 5.1(d) – quando NV assumiu o valor

igual a setenta. Para este mesmo cenário, a diferença entre os tempos de serviço produzidos por estes algoritmos foi de 12,8%.

Nesse contexto, há duas razões para a melhor taxa de acertos obtida, em geral, pelo CARTE. Primeiro, como antes mencionado, o CCVC emprega um algoritmo LFU (*Least Frequently Used*) no estágio inicial do processo de substituição para definir a partir de quais vídeos serão descartas sequências de vídeo. Deste modo, diferente da política executada pelo CARTE, onde as sequências a serem removidas podem provir de qualquer vídeo armazenado na memória, a estratégia usada pelo CCVC reduz o espaço de alternativas para a ação do algoritmo de substituição, haja vista que a seleção de uma sequência vítima deve respeitar a escala de popularidade (frequências de acesso) dos vídeos.

Segundo, uma vez que o vídeo de menor popularidade foi escolhido, o CCVC seleciona as sequências mais próximas do final do vídeo para desalocação, visto que, dessa forma, segundo a visão dos autores, o *proxy* precisa consumir recursos da rede por um tempo mais curto até que o conteúdo removido possa ser recuperado novamente junto ao servidor principal, caso exista necessidade. Porém, ao fazer isso, uma vez que as sequências se deslocam para o final do vídeo solidariamente à movimentação dos clientes, até mesmo as sequências que tem uma quantidade maior de clientes prévios no curto prazo tendem a ser desalocadas à medida que elas se aproximam do final do vídeo. Dessa forma, a eficiência do algoritmo CCVC tende a ser diminuída devido a essa inversão de prioridade durante o estágio final da transmissão do vídeo para os clientes.

Por último, conforme mostra em geral a figura 5.1, a comparação de desempenho com o *proxy* CC mostrou ganhos maiores. A diferença média total obtida para a taxa de acertos foi de 13,8% e de 9% para o tempo de serviço. As diferenças obtidas, respectivamente, para as mesmas métricas, quando os parâmetros de funcionamento assumiram seus valores padrão foram de 16,1% e 11,4%. A diferença máxima obtida entre as taxas de acertos produzidas por esses algoritmos foi de 18,9%, diferença esta que também foi atingida durante a variação do parâmetro NV - figura 5.1(d) - quando NV assumiu o valor igual a setenta. Para este mesmo cenário, a diferença entre os tempos de serviço produzidos por estes algoritmos foi de 10,4%.

Como mencionado na seção 2, a origem da taxa de acertos mais baixa relacionada ao algoritmo CC reside na tendência de priorização excessiva de segmentos posicionados no final do vídeo. Isso ocorre porque esse algoritmo considera todos os clientes ativos que não acessaram ainda um determinado segmento de vídeo como forma de estimar o número de acessos a serem realizados no futuro para esse segmento. Ao fazer isso, o algoritmo CC prioriza o armazenamento de trechos de vídeos relacionados a demandas de mais longo prazo,

ao invés de favorecer a alocação de trechos com demandas mais urgentes. Com isso, o algoritmo experimenta uma queda no seu desempenho quando as condições de carga momentâneas apontam para o armazenamento das partes iniciais e intermediárias dos vídeos.

Tabela 5.2 - Resultados gerais sobre taxa de acertos e tempo de serviço

Tipo de análise	Comparação com	Diferença na taxa de acerto	Diferença no tempo de serviço
Valor médio	CCVC	+12.9%	+11.3%
Valor médio	CC	+13.8%	+9.0%
Config. padrão	CCVC	+14.6%	+11.6%
Config. padrão	CC	+16.1%	+11.4%
Diferença máxima	CCVC	+17.3%	+12.8%
Diferença máxima	CC	+18.9%	+10.4%

Fonte: do autor (2015).

5.3 Avaliação dos Recursos Computacionais Usados por cada Algoritmo

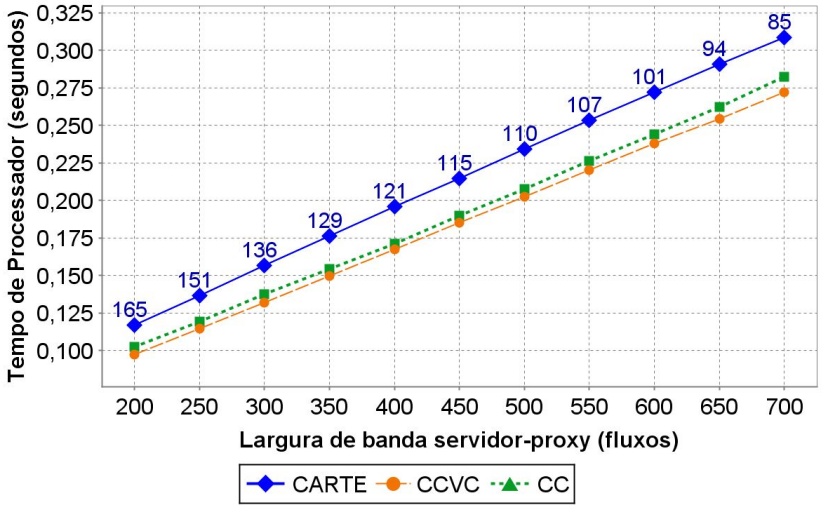
As figuras 5.3 e 5.4 demonstram, para as mesmas configurações de sistema e de carga usadas na construção da figura 5.1, como são usados os recursos físicos da arquitetura alvo.

O gráficos da figura 5.3 descrevem o tempo consumido exclusivamente para a execução de instruções sobre o processador sem os atrasos produzidos pelo sistema de memória. Os gráficos da figura 5.4 descrevem o tempo produzido para realizar a transferência de dados para dentro e fora da memória RAM. Este tempo de memória considera, além do fluxo de entrada e saída dos blocos de vídeo, o tráfego resultante do acesso às variáveis de programa e demais estruturas de dados usadas para sustentar as operações de *proxy*.

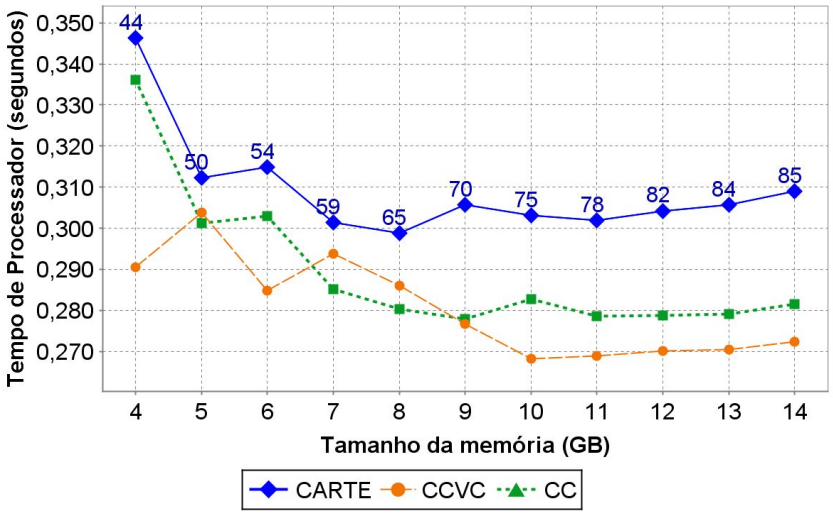
Como pode ser percebido na maior parte dos gráficos disponíveis nas figuras 5.3 e 5.4, os tempos de processador e de memória seguem (embora não na mesma proporção) tendências comportamentais similares. Isso ocorre porque, na maioria dos cenários de funcionamento, o trabalho realizado pelos componentes físicos do *proxy* é determinado pelo número de clientes servidos em cada instante pelo sistema. Porém, sob certas circunstâncias, como aquelas representadas pelas figuras 5.3(b) e 5.4(b), os comportamentos dos tempos do processador e da memória podem diferir.

Figura 5.3 - Uso de recursos: resultados para tempo de processador

(a) Tempo de processador para a variação do número de fluxos no canal *proxy*-servidor principal



(b) Tempo de processador para a variação do tamanho da memória de vídeo do *proxy*



(c) Tempo de processador para a variação da duração dos vídeos

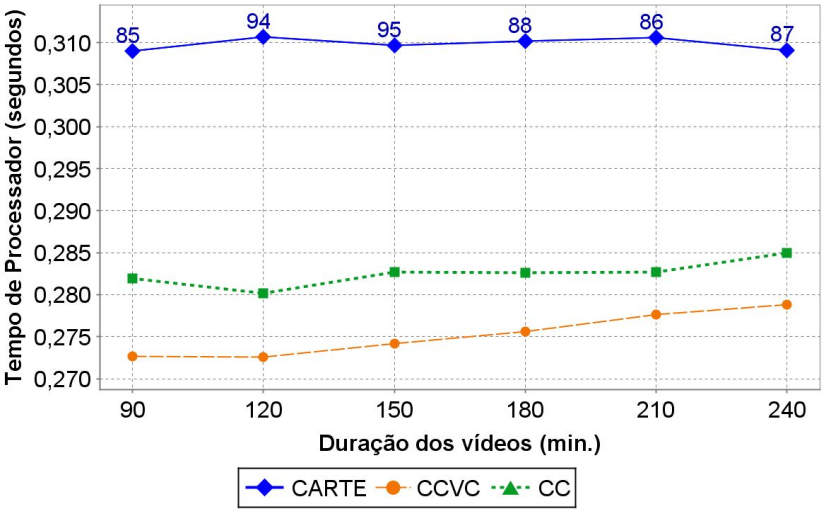
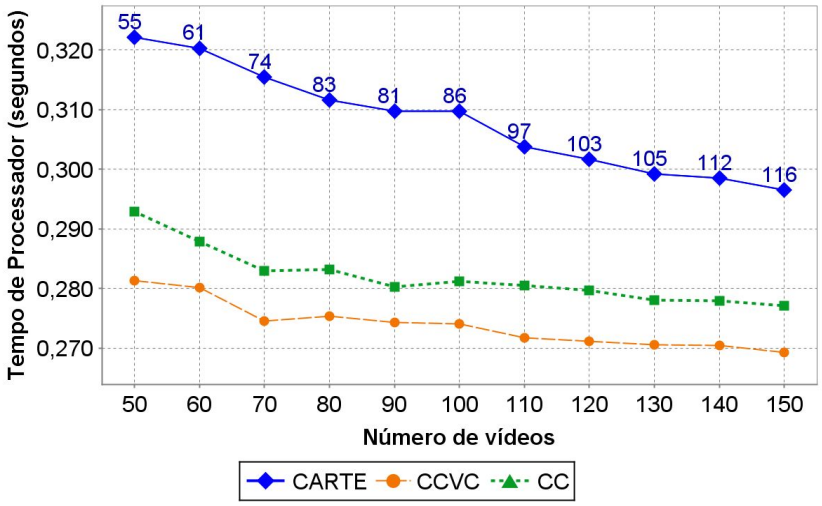
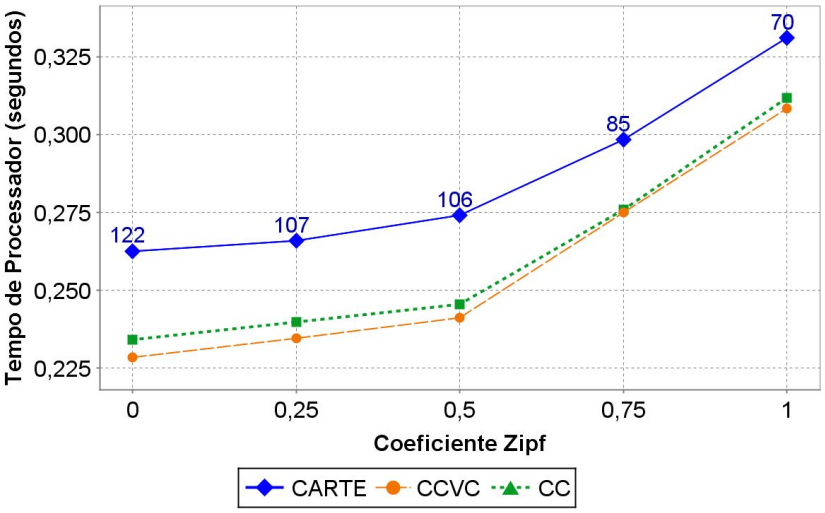


Figura 5.3 - Uso de recursos: resultados para tempo de processador (cont.)

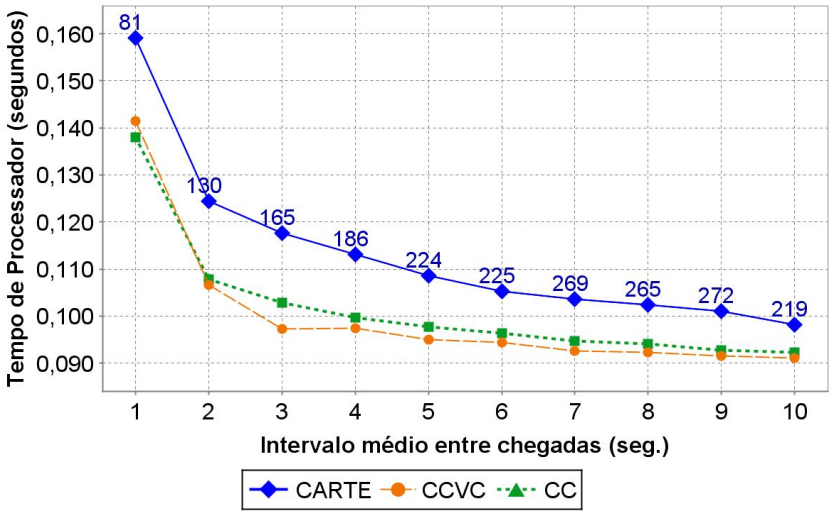
(d) Tempo de processador para a variação do número de vídeos disponíveis no acervo



(e) Tempo de processador para a variação do coeficiente Zipf



(f) Tempo de processador para a variação do intervalo médio entre as chegadas das requisições ao *proxy*

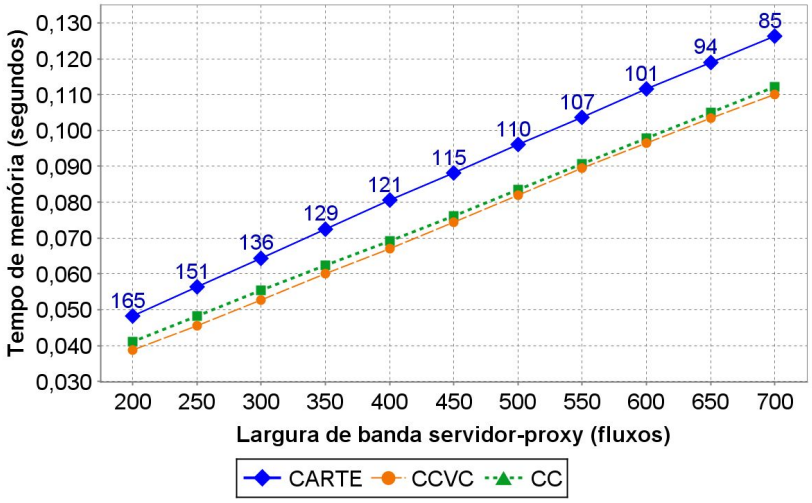


Fonte: do autor (2015).

Essa diferença ocorre porque, à medida que o tamanho da memória diminui, os algoritmos tendem a demandar um tempo de processamento (ou de processador) maior para promover continuamente a substituição dos blocos de vídeo alocados na memória. Por outro lado, conforme o tamanho da memória é reduzido, a taxa de acertos também diminui, resultando em um volume menor de clientes servidos pelo *proxy* e, por consequência, também em um tempo de memória mais baixo para transferir a quantidade necessária de blocos de vídeo da memória para a rede.

Figura 5.4 - Uso de recursos: resultados para tempo de memória

(a) Tempo de memória para a variação do tamanho da memória de vídeo do *proxy*



(b) Tempo de memória para a variação do tamanho da memória de vídeo do *proxy*

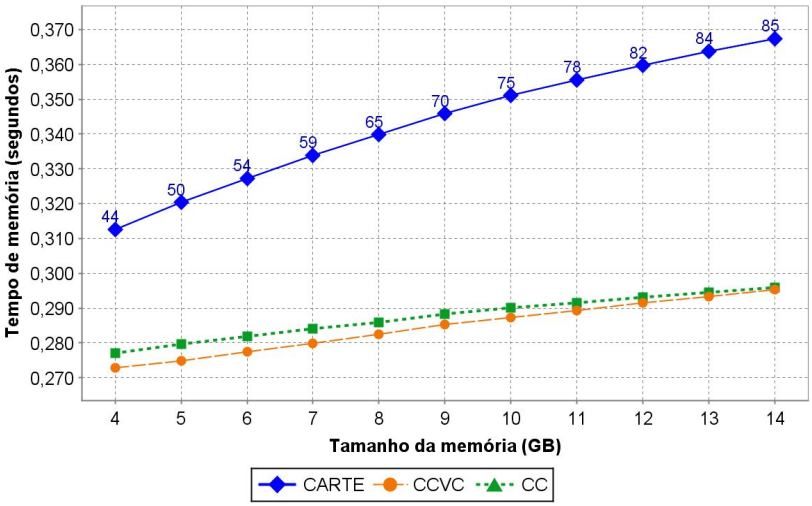
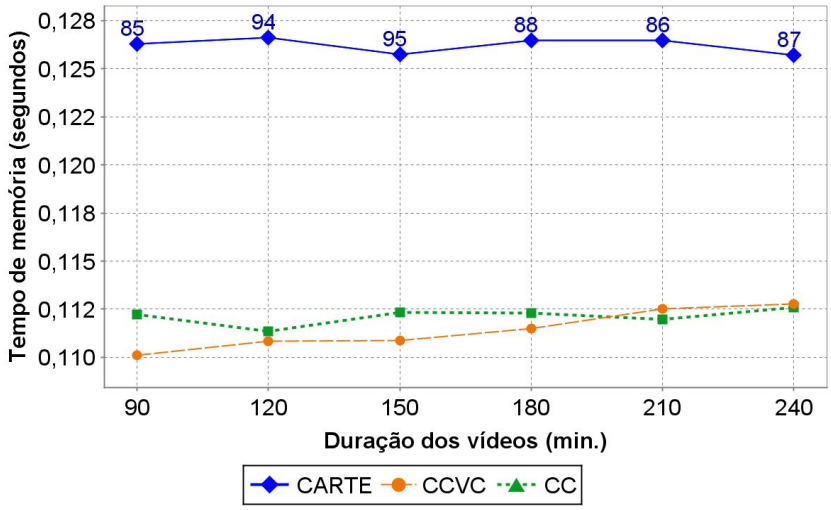
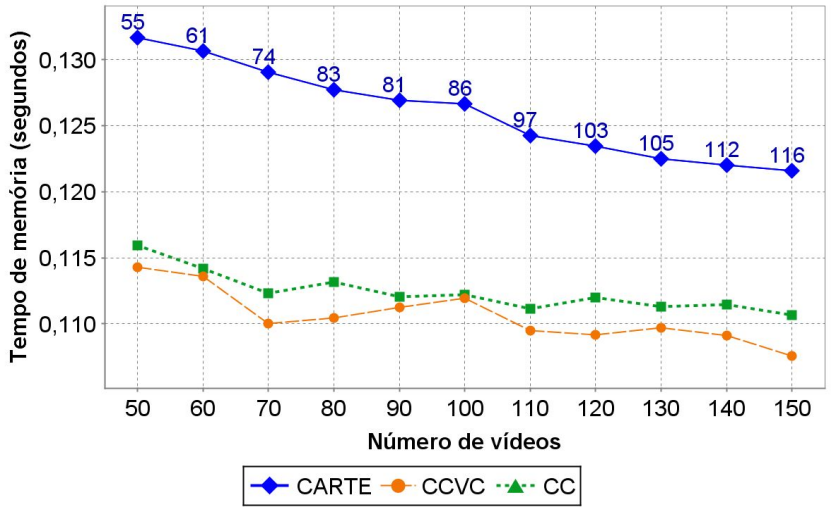


Figura 5.4 - Uso de recursos: resultados para tempo de memória (cont.)

(c) Tempo de memória para a variação da duração dos vídeos



(d) Tempo de memória para a variação do número de vídeos disponíveis no acervo



(e) Tempo de memória para a variação do coeficiente Zipf

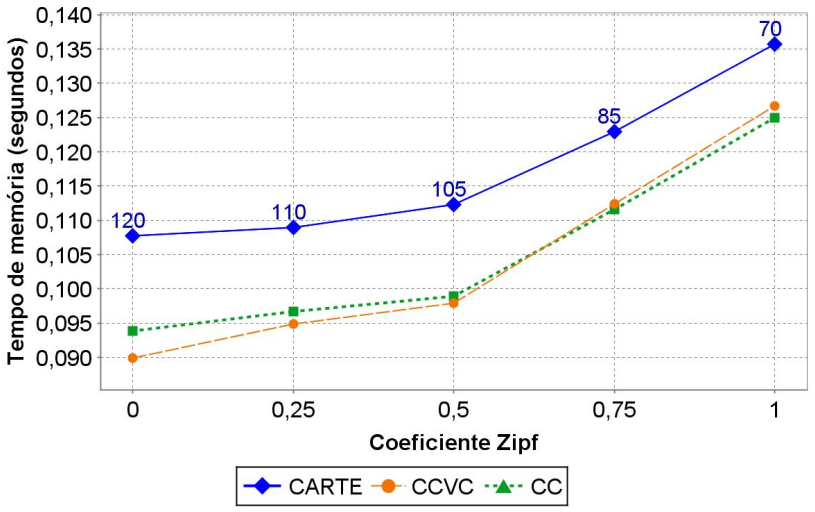
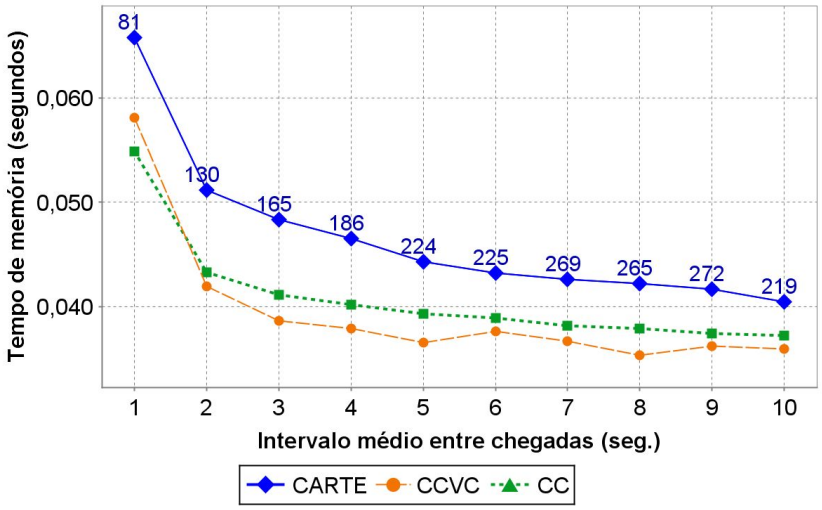


Figura 5.4 - Uso de recursos: resultados para tempo de memória (cont.)

(f) Tempo de memória para a variação do intervalo médio entre as chegadas das requisições ao *proxy*



Fonte: do autor (2015).

À parte disso, como é possível notar com base no conjunto completo de resultados sobre o consumo de recursos, disponível nas duas figuras, o processador consome em média cerca de duas a três vezes o tempo despendido pela memória para executar as tarefas da aplicação. A diferença entre o tempo consumido por esses componentes demonstra que o núcleo de processamento tende a ser o principal gargalo para o fornecimento do serviço quando a demanda aumenta em relação à quantidade de recursos disponíveis.

Complementarmente, as análises dos dados produzidos pela simulação no modo discreto revelou que, em média, a maior parte das instruções executadas, 63,2%, são do tipo aritmético ou lógico, enquanto que as instruções de acesso à memória e de controle de fluxo representam, respectivamente, 28,32% e 8,38% do total de instruções executadas. Esses resultados, em conjunto com aqueles produzidos pela simulação no modo tempo real, sugerem que o desenvolvimento de um hardware dedicado para suporte às operações básicas de *proxy* tende a contribuir para o aumento significativo da eficiência dessa aplicação e, consequentemente, também para a redução dos custos para implementação do sistema VoD.

A tabela 5.3 descreve os resultados gerais com relação à análise comparativa dos recursos consumidos por cada algoritmo, os resultados demonstram que o CARTE demandou em média 11,4% mais tempo do processador que o CCVC e 8,7% mais tempo que o CC, produzindo também um tempo médio de acesso à memória superior em, respectivamente, 12% e 9,7%. Quando os parâmetros de carga assumiram seus valores padrão, o tempo de processador consumido pelo CARTE foi 11,8% maior do que aquele consumido pelo CCVC e

8,9% maior do que aquele produzido pelo CC, também consumindo, respectivamente, 12,8% e 11,1% mais tempo para realizar os acessos necessários à memória.

Tabela 5.3 - Resultados gerais sobre uso de recursos

Tipo de análise	Comparação com	Diferença no tempo de processador	Diferença no tempo de memória
Valor médio	CCVC	+11,4%	+12,0%
Valor médio	CC	+8,7%	+9,7%
Config. padrão	CCVC	+11,8%	+12,8%
Config. padrão	CC	+8,9%	+11,1%
Cenário de maior dif. na taxa de acertos	CCVC	+12,6%	+14,7%
Cenário de maior dif. na taxa de acertosa	CC	+10,1%	+12,4%

Fonte: do autor (2015).

Além disso, considerando os mesmos cenários onde o CARTE obteve as maiores diferenças em termos de taxa de acertos para o CCVC e o CC, a diferença encontrada sobre os tempos de processador foi de, respectivamente, 12,6% e 10,1%. Também considerando esses cenários, a diferença encontrada em relação ao tempo de memória para comparações entre o CARTE e o CC foi de 12,4%, enquanto que para as comparações entre CARTE e CCVC, a diferença encontrada para essa mesma métrica foi de 14,7%. Nesta conjuntura, a origem dessas diferenças se baseia em três aspectos principais.

Primeiro, como já explicado, o tempo de processamento tende a manter uma proporção aproximada com o número de clientes servidos pelo sistema. Como consequência, as distâncias existentes entre as taxas de acertos produzidas pelos algoritmos contribuem predominantemente para a geração das diferenças observadas entre os consumos de recursos praticados por eles.

Segundo, o tempo de processador produzido pelo algoritmo CC também é influenciado pelo maior custo de gerenciamento vinculado à organização do vídeo em segmentos, cujo tamanho, para estes experimentos, equivale a um bloco (ou segundo) de vídeo. Enquanto esta estratégia possibilita que o algoritmo CC alcance sua máxima eficiência em termos de taxa de acertos (de acordo com a descrição feita na seção 3.1), ela também contribui para o aumento do tempo do processamento desse algoritmo. Isso porque, diferentemente do CARTE, que vincula uma única prioridade de cacheamento a múltiplos blocos de vídeo englobados em cada sequência, o CC atribui uma prioridade para cada bloco de vídeo alocado na memória, necessitando, portanto, mais tempo do processador para esse trabalho. Assim, apesar do mecanismo de vinculação de prioridades do CC não produzir um uso de recursos a ponto de exceder o consumo produzido pelo CARTE (que resulta da sua taxa de acertos mais alta), ele

leva o CC a utilizar mais recursos do que o CCVC, apesar da taxa de acertos produzida pelo CC ser menor do que aquela produzida pelo CCVC na maioria dos cenários.

Terceiro, o critério usado, tanto pelo CCVC como pelo CC, para determinar a prioridade de cacheamento para cada trecho de vídeo, resulta em um número menor de substituições de conteúdo na memória do *proxy*, produzindo um consumo menor de recursos para executar essa tarefa. No que diz respeito ao CCVC, isso ocorre porque o algoritmo concentra as substituições do conteúdo sobre os vídeos com poucos clientes ativos e prioriza a remoção das partes finais desses vídeos. Dessa maneira, somente uma mudança significativa na popularidade dos vídeos pode modificar as prioridades de cacheamento das sequências a ponto de causar um número maior de substituições de conteúdo. Todavia, tal mudança na popularidade dos vídeos raramente ocorre em intervalos menores do que um dia de serviço (HONG; DE VLEESCHAUWER; BACCELLI, 2010).

No que diz respeito ao comportamento do algoritmo CC, apesar de haver uma tendência para os segmentos mais próximos ao final de cada vídeo recebam uma prioridade maior de cacheamento em relação aos segmentos mais próximos ao começo, o algoritmo tende a possibilitar uma competição maior pelos espaços disponíveis na memória do *proxy*, se comparado ao CCVC. Isso ocorre porque, em certos momentos, as condições de carga podem sugerir que alguns segmentos perto do começo de um vídeo de popularidade maior possam ter precedência na alocação em relação a segmentos próximos ao final de um vídeo menos popular. Dessa forma, o algoritmo CC cria condições para a execução de um volume maior de substituições de conteúdo, consumindo uma quantidade correspondente de recursos para este fim.

Em contraste a estas políticas, o CARTE foi criado para permitir que qualquer trecho de vídeo pertencente ao acervo (indiferente da posição ocupada dentro de cada vídeo) possa receber uma maior prioridade de cacheamento, baseando, para isso, suas decisões exclusivamente nas condições atuais de carga existentes dentro de cada janela de tempo.

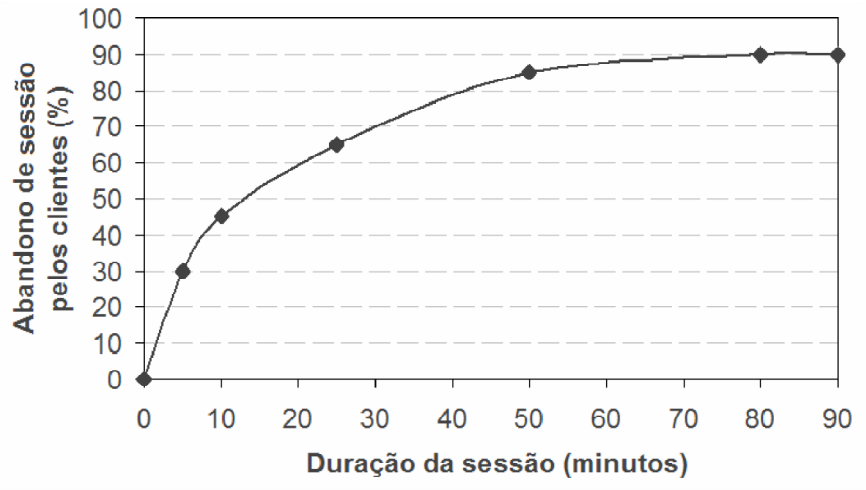
Como resultado, esse algoritmo tende a reagir com maior frequência às variações momentâneas que ocorrem na distribuição dos clientes durante a execução. Com isso, ele tende a produzir um volume maior de atualizações de conteúdo na memória do *proxy*, com impactos proporcionais sobre o consumo dos recursos.

5.4 Impactos Produzidos pela Saída Prematura dos Clientes

Para analisar os impactos produzidos pelo fechamento proativo de sessão pelos clientes VoD, reavaliou-se um subconjunto dos cenários apresentados na seção 5.2, considerando desta vez, contudo, a ocorrência do abandono de sessão durante as simulações. Para isto, foram reexecutadas simulações para os cenários correspondentes aos gráficos das figuras 5.1(b) e 5.1(c), uma vez que o tamanho da memória e o comprimento dos vídeos no acervo (parâmetros MS e DV) estão mais diretamente relacionados com o fenômeno do abandono.

Para modelar o comportamento da saída prematura dos clientes, foi utilizada a curva de abandono apresentada na figura 5.5, a qual se baseia nos dados de acesso a um *proxy* VoD comercial apresentados em (YU et al., 2006). A figura 5.6 apresenta o desempenho obtido, neste novo contexto, para os algoritmos analisados e a tabela 5.4 apresenta um resumo dos dados apresentados na figura 5.6. A terceira coluna desta tabela apresenta o crescimento (+) ou decrescimento da diferença relativa entre os algoritmos usando como referência os resultados obtidos para os cenários equivalentes sem saída prematura de clientes.

Figura 5.5 - Curva de abandono dos clientes VoD: percentual de fechamento cumulativo das sessões para cada instante de exibição do vídeo (baseada em (YU et al., 2006))



Fonte: do autor (2015).

No novo cenário com abandono de sessão, as diferenças média e máxima entre as taxas de acertos do CARTE e do CCVC diminuíram, respectivamente, em 2,3 e 3,4 pontos percentuais, em comparação com os cenários sem abandono apresentados nas figuras 5.1(b) e 5.1(c), ficando em 8,6% e 11,8%. A diferença obtida entre estes dois sistemas quando as configurações default foram empregadas foi reduzida em 3,4 pontos percentuais, ficando em

11,8%. Isto mostra que o CCVC consegue melhorar comparativamente seu desempenho, através do descarte de sequências próximas ao final do vídeo[22] .

Todavia, o CARTE consegue manter uma taxa de acertos maior por que, mesmo sob a ocorrência do abandono, o CCVC não descarta sequências pertencentes aos vídeos de maior popularidade, onde a desistência de sessão também ocorre. Consequentemente, quando os clientes abortam o vídeo, as sequências de vídeo mantidas pelo CCVC na memória do *proxy*, para o acesso dos clientes que desistiram, precisam esperar por um tempo maior até que os próximos clientes (que não abandonaram o vídeo) realizem acesso a elas. Neste contexto, o efeito da saída prematura dos clientes sobre desempenho do CCVC é ainda maior à medida que o comprimento do vídeo aumenta em relação ao comprimento de sessão dos pontos de abando, como mostra a figura 5.6(a). Em contrapartida, o CARTE permite manter as sequências na memória, enquanto houverem clientes suficientes para justificar a alocação delas. Deste modo, o algoritmo tende a favorecer a alocação das sequências mais próximas do início de cada vídeo, que possuem maior concentração de clientes prévios em razão da redução gradativa do número de clientes ativos à medida que eles avançam para o final do vídeo.

Tabela 5.4 - Resultados gerais para cenários com saída prematura de clientes

Tipo de análise	**Comparação com**	**Aumento(+) / redução(-) da dif. relativa**	**Valor absoluto**
Valor médio	CCVC	-2,3%	8,6%
Valor médio	CC	+3,5%	15,0%
Config. padrão	CCVC	-3,4%	11,8%
Config. padrão	CC	+3,9%	19,9%
Diferença máxima	CCVC	-3,4%	11,8%
Diferença máxima	CC	+4,7%	20,7%

Fonte: do autor (2015).

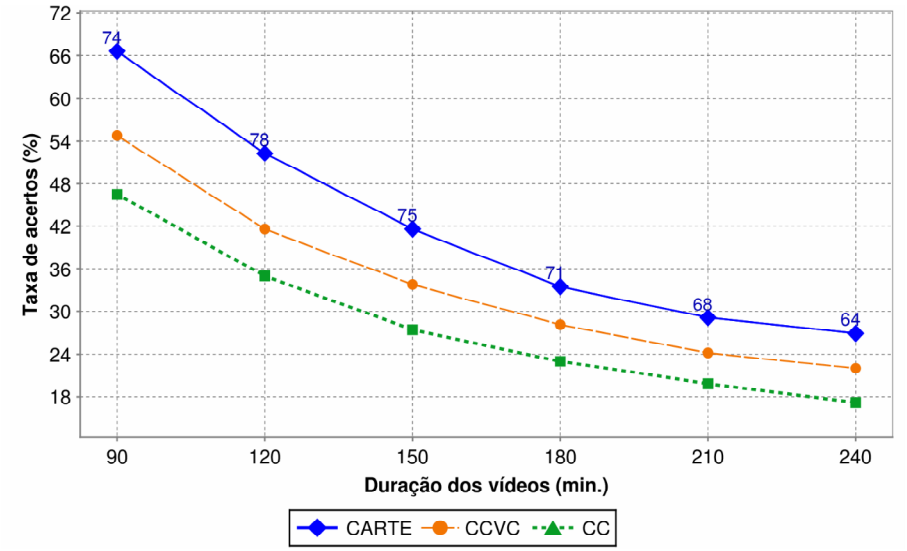
No que diz respeito à comparação com o CC, as diferenças média e máxima entre as taxas de acertos aumentaram, respectivamente, em 3,5 e 4,7 pontos percentuais, nos cenários com abandono de sessão, ficando em 15% e 20,7%. A diferença obtida entre estes dois sistemas quando as configurações default foram empregadas foi aumentada em 3,9 ponto percentual, ficando em 19,9%. Isto ocorre por que o CC tende a priorizar o cacheamento das porções finais de cada vídeo, a qual é menos acessada devido à saída prematura dos clientes. Assim, conforme o comprimento de vídeo aumenta em relação aos pontos de saída dos clientes, ocorre o aumento do número de blocos existentes com maior prioridade de

[22] Como critério parcial, juntamente com o tamanho das sequências.

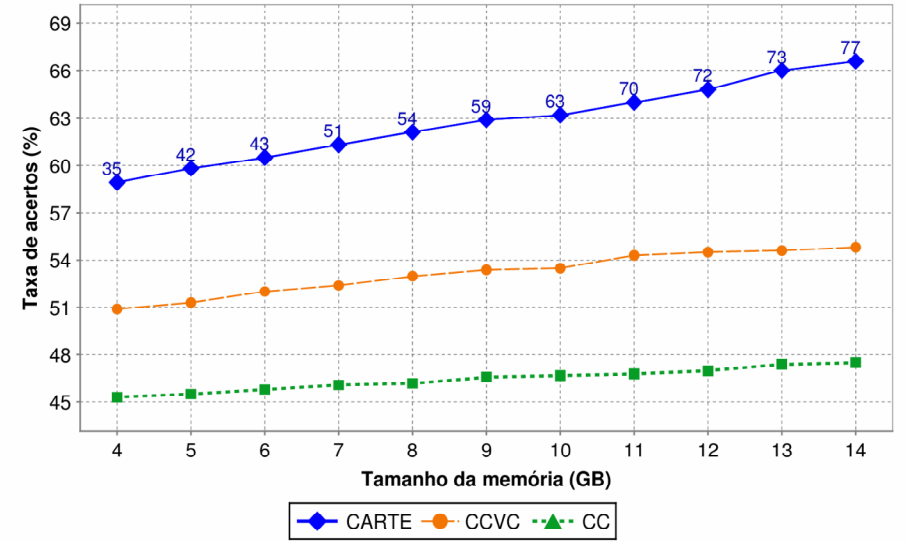
cacheamento no final do vídeo. Como consequência, estes blocos dominam os espaços existentes na memória de vídeo do *proxy*, resultando na queda de eficiência do CC.

Figura 5.6 - Desempenho sob abandono de sessão: taxa de acertos para variação do tamanho dos vídeos e da memória

(a) Taxa de acertos para a variação da duração dos vídeos



(b) Taxa de acertos para a variação do tamanho da memória de vídeo do *proxy*



Fonte: do autor (2015).

Em contrapartida, o mecanismo de janelamento empregado pelo CARTE proporciona que a maior densidade de clientes existente no início de cada vídeo impactue apenas sobre as sequências que ficam no escopo das janelas de tempo também localizadas nos inícios destes

vídeos. Assim, as sequências de vídeo localizadas próximas ao sufixo[23] não são mantidas na memória sem que existam clientes prévios presentes em suas próprias janelas. Por esta razão, o uso de janelas de tempo menores nos cenários com abandono, se comparados aos cenários análogos onde o aborto de sessão não ocorre, fez com que o CARTE produzisse, em geral, melhores resultados.

Com relação à figura 5.6(b), os resultados mostram que, embora sob a ocorrência da desistência dos clientes, o CARTE mantém sua tendência de escalar melhor seu desempenho, em comparação com os dois algoritmos de referência, quando o tamanho da memória do *proxy* aumenta. Isto demonstra que a lógica de cacheamento do CARTE é capaz de explorar a memória adicional com maior eficiência, através da distinção das várias concentrações de clientes existentes ao longo dos vídeos.

Por último, o tempo de serviço produzido pelo CARTE nos cenários com abandono de sessão apresentou uma redução média de 7.8% em relação aos cenários análogos onde a saída prematura dos clientes não ocorre. Esta redução decorre do menor número de clientes simultâneos sustentados pelo *proxy* em razão da desistência dos clientes. Analogamente, apesar da possível queda de eficiência esperada para os cenários com abandono, a redução do número de clientes concorrentes se mostrou ser um fator preponderante para causar a elevação das taxas de acertos média produzidas pelo CARTE e pelo CCVC, respectivamente, em 1,5% e 3,7%, se comparada com os resultados obtidos para os mesmos cenários quando o abandono não ocorre. Pelos motivos já descritos, o CC não acompanhou este crescimento, apresentando uma queda de sua eficiência média em 2,1%.

5.5 Análise da Relação entre o Tamanho da Janela de Tempo e a Carga de Trabalho

Conforme sugerido na seção 5.3, a configuração adotada para o parâmetro JT do CARTE é capaz de criar impactos diretos não apenas sobre a taxa de acertos produzida pelo CARTE, mas também sobre o número de substituições realizadas por este algoritmo. Conforme também já demonstrado, estes dois fatores criam repercussões sobre o consumo de recursos da arquitetura subjacente do *proxy* VoD. Como consequência, o aspecto chave para alcançar a eficiência máxima do algoritmo consiste em identificar o melhor ajuste para o JT.

Por esse motivo, essa seção apresenta um estudo da correlação entre o parâmetro JT e a carga de trabalho para o *proxy*, objetivando demonstrar de que forma as condições de

[23] Parte final de cada vídeo.

funcionamento podem influenciar a escolha feita pelo projetista VoD para alcançar a maior eficiência possível para o sistema. Para isso, foram analisados os efeitos produzidos pela variação de JT, sob diferentes cenários de carga, usando, como métrica para avaliação de eficiência, a taxa de acertos. Com isso, espera-se construir diretrizes sólidas para fazer o ajuste do algoritmo, extraindo, dessa forma, o máximo desempenho para qualquer ocasião.

Dito isso, a figura 5.7(a) apresenta a correlação existente entre IR e JT. Notoriamente, a variação na IR causa uma mudança na concentração (densidade) de clientes por vídeo. Quando IR assume um valor baixo, a concentração de clientes se torna alta e JT precisa ser configurado com um valor pequeno para definir com eficiência a prioridade de caching para cada sequência. Analogamente, em cenários onde os clientes estão mais dispersos (IR alto), é necessário usar valores maiores para JT.

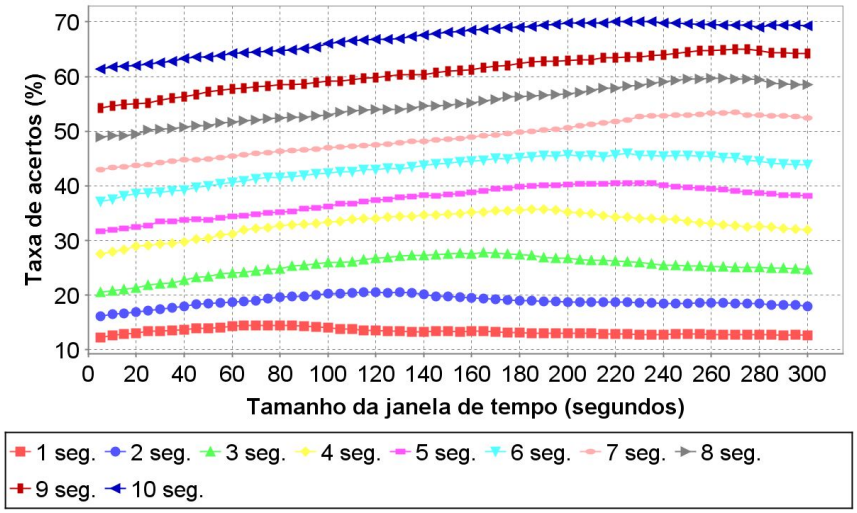
Isso ocorre porque quando os clientes estão mais concentrados, o uso de uma janela de tempo maior tende a abranger não somente um maior número de clientes mas também uma quantidade maior de sequências dentro de cada janela, fazendo com que os mesmos clientes participem do cálculo da densidade para diferentes sequências simultaneamente. Assim, à medida que o valor de JT é aumentado gradativamente, ocorre a tendência para que as sequências mais próximas do final de cada vídeo tenham um maior número de clientes prévios englobados por suas janelas de tempo e, como resultado, elas recebem prioridades de cacheamento mais altas. Em contrapartida, as sequências próximas ao começo de cada vídeo recebem prioridades de cacheamento menores, apesar de possivelmente terem uma maior demanda associada a elas a curto prazo. Quando isso ocorre, o comportamento do algoritmo CARTE tende a se assemelhar àquele desempenhado pelo algoritmo CC.

Adicionalmente, quando IR possui um valor igual ou próximo a um segundo, produzindo a maior quantidade de clientes simultâneos usada nas simulações, mesmo o uso das melhores configurações para JT leva a observação de ganhos significativamente menores na taxa de acertos, se comparado aos cenários onde IR assume valores maiores.

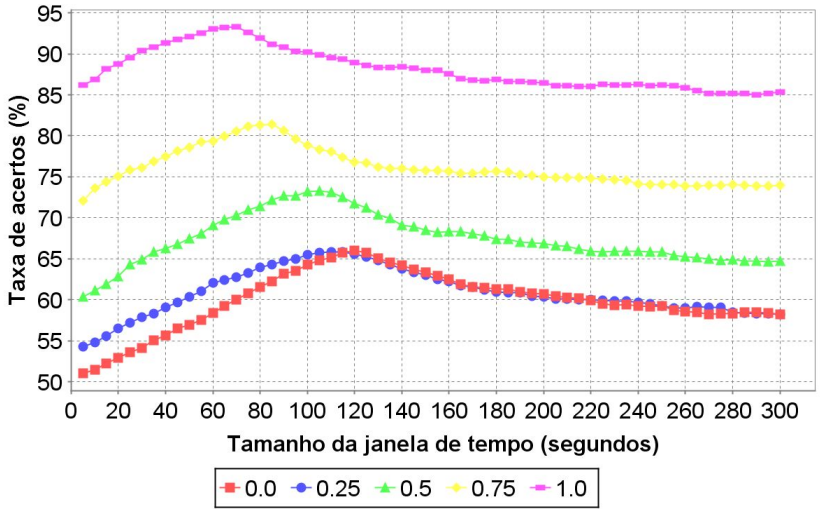
Isso acontece porque, sob essas circunstâncias, tanto o tamanho das sequências como a densidade de clientes dentro das janelas de tempo dos vídeos vitimados se tornam muito semelhantes, fazendo com que o benefício de manter uma ou outra sequência na memória seja aproximadamente o mesmo.

Figura 5.7 - Taxa de acertos para variação do tamanho da janela de tempo

(a) Variação de IR



(b) Variação de ZC



(c) Variação de NV

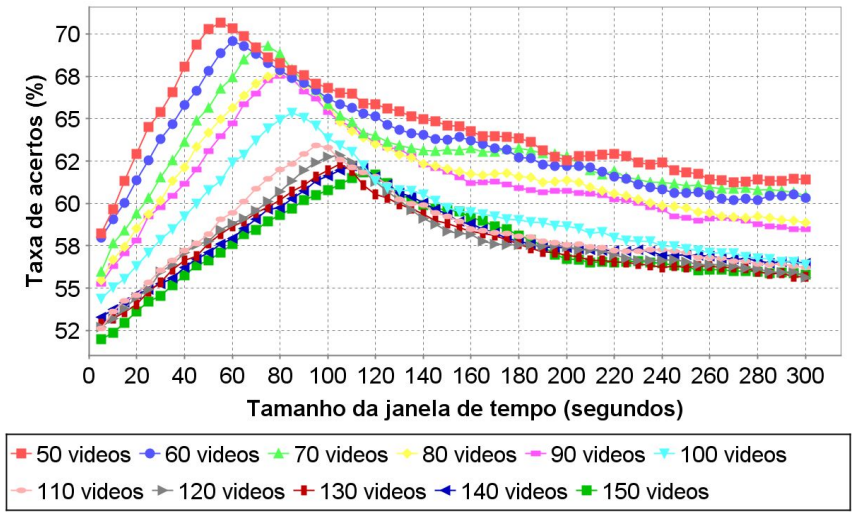
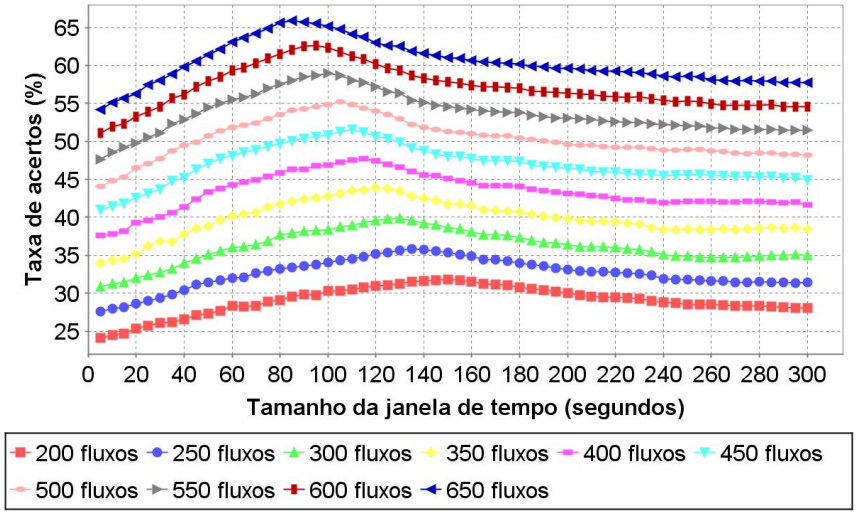
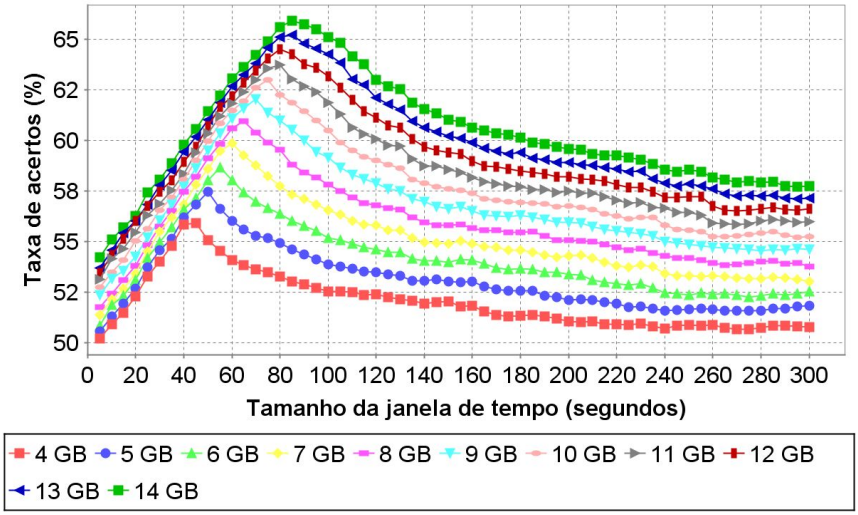


Figura 5.7 - Taxa de acertos para variação do tamanho da janela de tempo (cont.)

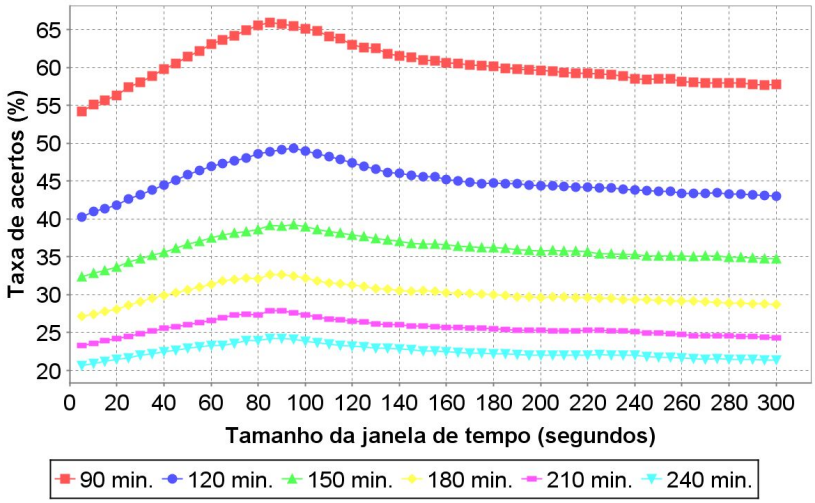
(d) Variação de LB



(e) Variação de TM



(f) Variação de DV



Fonte: do autor (2015).

A figura 5.7(b) mostra a influência causada pela variação do parâmetro CZ sobre a configuração de JT. Considerando que NV e IR foram mantidos constantes nesse experimento, os aumentos produzidos sobre CZ causaram a transferência de parte dos clientes dos vídeos menos populares para os mais populares. Com isso, uma quantidade menor de clientes permaneceu requisitando os vídeos menos populares, produzindo igualmente um menor número de sequências para descartes nesses vídeos.

Desta forma, parte das sequências elegidas para substituição precisou ser obtida a partir dos vídeos mais populares onde a concentração de clientes cresceu devido ao aumento causado sobre CZ. Assim, em razão deste aumento da concentração nos vídeos mais populares, os clientes passaram a ficar mais próximos uns dos outros, criando a demanda para que JT assumisse valores menores para possibilitar a distinção das densidades de clientes observadas nas janelas de tempo existentes.

Em contrapartida, à medida que CZ foi reduzido, parte dos clientes deixou de solicitar acesso aos vídeos de grande popularidade e, ao invés disso, passou a solicitar o acesso aos vídeos menos populares. Com isso, a concentração de clientes sobre os vídeos menos populares passou a ser suficiente para permitir a formação de sequências que alimentaram o processo de substituição. Neste contexto, uma vez que os poucos clientes existentes sobre os vídeos menos populares ficaram separados por uma quantidade grande de blocos, foi necessário aumentar o valor de JT para acomodar uma quantidade maior de clientes em cada janela, de tal forma a produzir um desequilíbrio no número de clientes presentes em cada janela.

À parte disso, uma outra forma de modificar a concentração dos clientes é mostrada na figura 5.7(c), que apresenta a correlação entre o tamanho da janela de tempo e o número de vídeos oferecidos para acesso. Considerando que IR e CZ foram mantidos constantes nessas simulações (uma vez que, conforme demonstrado, esses parâmetros podem afetar diretamente a concentração de clientes), o aumento sobre NV causou uma maior dispersão dos clientes entre os vídeos disponíveis, de tal modo que o número de clientes presentes em cada vídeo se tornou menor. Assim, conforme o valor de NV foi aumentado, foi necessário aumentar também o valor para JT para criar uma distinção entre as quantidades de clientes presentes em cada janela.

Deixando à parte a necessidade de ajustar o tamanho da janela temporal para acompanhar a mudança na densidade de clientes em cada vídeo (como ocorre quando os parâmetros IR, CZ ou NV são variados), a figura 5.7(d) mostra que o ajuste do parâmetro JT também pode ser influenciado pela largura de banda associada ao enlace que conecta o *proxy*

ao servidor principal. Conforme a largura de banda desse canal é reduzida, o algoritmo precisa trabalhar com uma janela de tempo maior para identificar sequências que, uma vez cacheadas no *proxy*, possibilitarão o fornecimento de serviço para um número maior de clientes. Isso ocorre porque, em cenários onde o valor de LB é baixo, o CARTE precisa escolher as sequências que levam ao alcance de um custo benefício melhor a mais longo prazo, visto que, de outro modo, o algoritmo não disporá de banda suficiente na rede para, se necessário, realizar a troca do conteúdo armazenado, caso os dados alocados devido a maior demanda a curto prazo tenham sua procura reduzida depois deste curto período.

Por outro lado, à medida que a largura de banda do enlace servidor-*proxy* se torna maior, o algoritmo trabalha mais eficientemente com tamanhos menores de janela, dado que, sob essas condições, é possível aplicar a largura de banda adicional para priorizar a alocação simultânea e frequente de um número maior de sequências que tem grande procura a curto prazo, mas não necessariamente a longo prazo. Assim, o CARTE alcança uma alta escalabilidade, maximizando não somente o uso da rede, mas também do processador e da memória para promover as trocas necessárias de conteúdo e, assim, aumentar a produtividade do *proxy*.

Analogamente, a figura 5.7(e) mostra que, em resposta ao aumento produzido sobre o tamanho da memória, o valor de JT precisa ser expandido para prover um melhor desempenho para o *proxy*. Isso ocorre porque, com o uso de uma memória maior, o CARTE pode investir em sequências que a um prazo mais longo apresentam uma demanda maior. Nesse contexto, a memória maior possibilita a preservação dessas sequências por um tempo grande o suficiente para extrair os benefícios resultantes do cacheamento desse conteúdo.

Por fim, a figura 5.7(f) mostra que JT foi pouco influenciado pela maior parte dos tamanhos de vídeos usados nas simulações. Sob este cenário, a variação mais significativa sobre JT ocorreu quando DV foi aumentado de 90 para 120 minutos. Neste caso, foi necessário aumentar o JT para selecionar com maior eficiência as sequências disponíveis, uma vez que, face à preservação do valor constante de IR, o aumento de DV criou uma quantidade maior de sequências em cada vídeo, as quais contribuíram para o aumento da disputa pelos espaços disponíveis na memória.

Porém, os aumentos subsequentes no DV não produzem o mesmo impacto sobre JT. Isso ocorre porque, apesar de ser esperado que as adições subsequentes sobre DV pudessem gerar um aumento proporcional sobre JT, a capacidade da memória disponível no *proxy* não seria grande o suficiente para armazenar essas sequências por um período tão longo. Desta maneira, o valor de JT permaneceu aproximadamente estável ao longo dos acréscimos

subsequentes sobre DV, visto que essa configuração mantém uma proporção melhor com o tamanho da memória usado nesse experimento.

5.6 Procedimentos para configurar o CARTE

A estratégia correntemente em uso para ajustar o parâmetro JT baseia-se no uso de um algoritmo explorador, nomeado JSS (*JT Space Scanner*), que executa um pequeno número de simulações do CARTE, para cada uma das quais JT é variado, até que a melhor configuração para um dado cenário alvo possa ser encontrada.

O JSS funciona em três etapas. Na primeira, ele realiza um escaneamento de granularidade grossa sobre um conjunto de amostras de JT obtidas a partir de um intervalo inicial (por exemplo, de 1 a 300). Neste escaneamento, cada simulação usa uma amostra diferente, a qual é selecionada através da soma do passo de incremento (*coarse_step*) sobre o valor da amostra anterior, sendo que a primeira amostra equivale ao primeiro valor de JT presente no intervalo inicial.

Com base nos experimentos realizados, identificou-se que quando *coarse_step* = 20 o TTS consegue detectar falsos pontos máximos sobre a curva de eficiência (resultante das taxas de acertos produzidas pelos valores de JT existentes no intervalo inicial). Assim, quando esta configuração é usada, o topo da curva de eficiência em geral é claramente destacado em relação aos demais pontos pertencentes ao intervalo inicial.

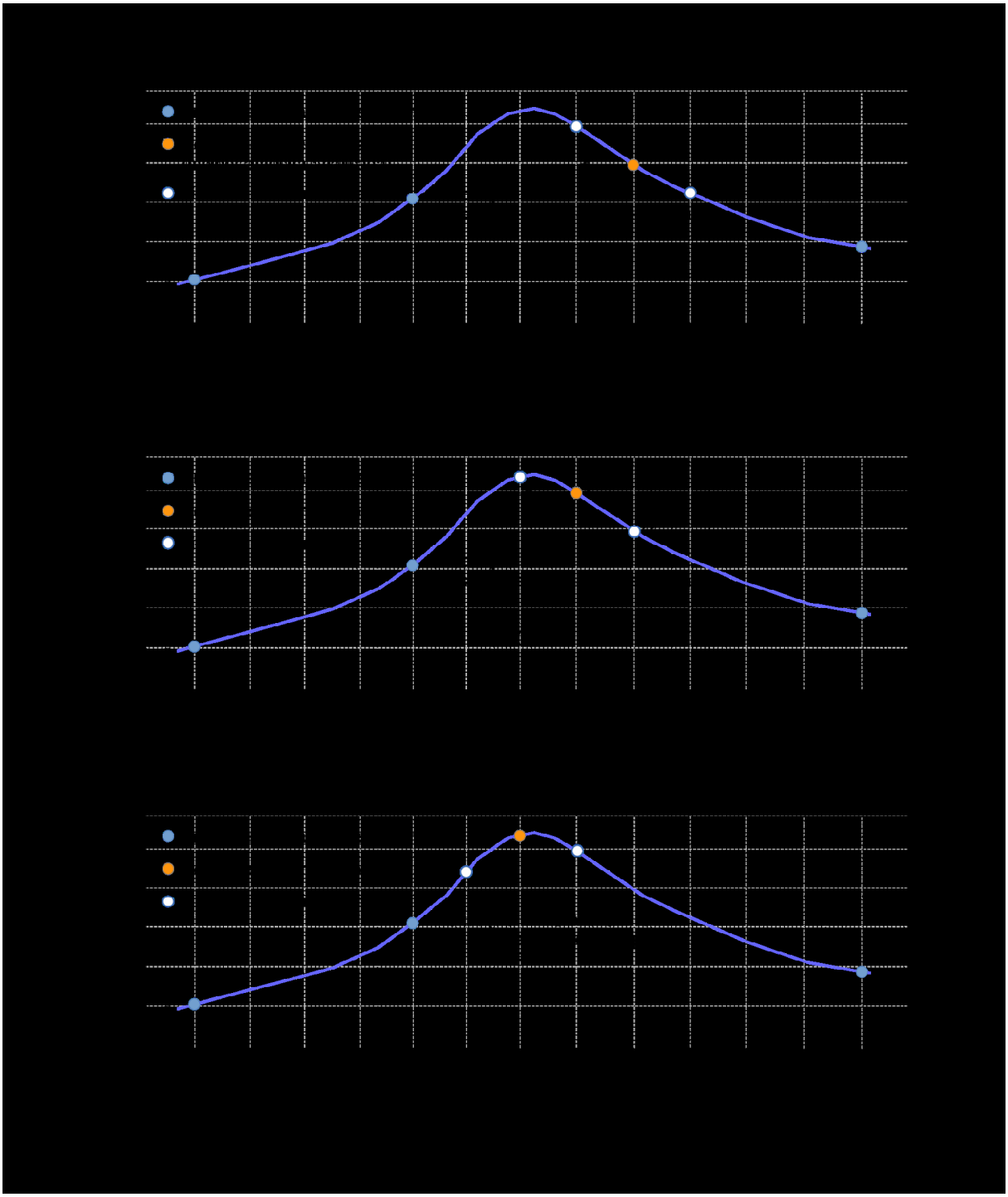
Após o scan grosso, a amostra de JT que resulta na maior taxa de acertos é usada como referência inicial para execução da segunda etapa que consiste em um escaneamento de granularidade fina realizado conforme ilustra a figura 5.8.

A figura 5.8(a) mostra que o JSS usa um passo de variação fino (*fine_step* = 5 nas simulações realizadas) para calcular, a partir do ponto de referência produzido pelo scan grosso, os valores de L e R que limitam, respectivamente, pela esquerda e pela direita um subintervalo que tem C como centro. Depois disso, como mostram as Figuras 5.8(b) e 5.8(c), o algoritmo itera deslocando L, C e R para um dos lados usando o passo de variação até que $L \leq C$ e $C \geq R$. O deslocamento ocorre em direção ao limite lateral que apresenta uma maior taxa de acertos em comparação ao centro do intervalo.

Quando a condição de parada é alcançada, como mostra a figura 5.8(c), o algoritmo inicia a sua última etapa realizando um scan sequencial (usando passo de incremento igual a 1) dentro do subintervalo compreendido entre os limites L e R. Como resultado do scan

sequencial, o JSS identifica o valor de JT que conduz a melhor eficiência para o CARTE dentro do cenário alvo.

Figura 5.8 - Funcionamento do scan fino: (a) estado inicial com a definição dos primeiros valores para L, C e R. (b) e (c) iterações do scan fino com deslocamento de L, C e R. (c) estado final com L ≤ C and C ≥ R



Fonte: do autor (2015).

As equações 5.1, 5.2 e 5.3 descrevem como é feito o cálculo do número máximo de simulações, respectivamente, para a primeira, segunda e terceira etapa do algoritmo JSS. A variável *max_TW*, presente na equação 5.1, informa o comprimento máximo de JT para o intervalo inicial, que se estende de 1 até *max_TW*. Para cenários existentes dentro da faixa de funcionamento especificada pela tabela 5.1, *max_TW* tende a não assumir valores maiores do que 300. Neste contexto, considerando a configuração dos parâmetros *fine_step* e *coarse_step*, respectivamente, com os valores 5 e 20, a execução do JSS produz 30 simulações até encontrar o valor ideal de JT para o cenário alvo.

$$Coarse\_runs = (max\_TW / coarse\_step) + 1 \quad (5.1)$$

$$fine\_runs = (coarse\_step / fine\_step) \quad (5.2)$$

$$sequential\_runs = (fine\_step * 2) - 2 \quad (5.3)$$

Espera-se para o futuro realizar um estudo mais profundo dos impactos produzidos sobre JT pela variação conjunta dos parâmetros de carga. A partir disso, planeja-se criar um modelo que permita estimar o melhor valor aproximado de JT para um cenário de interesse. Uma vez feito isso, pretende-se mensurar os benefícios proporcionados pela substituição do scan grosso, atualmente executado pelo JSS, por uma representação em software deste modelo. Por fim, intenciona-se derivar a partir desta versão modificada do JSS uma solução para ajuste dinâmico do JT que possibilite ao CARTE automaticamente se reconfigurar perante às possíveis flutuações na carga de trabalho.

6 CONCLUSÃO

Ao longo deste estudo foi apresentado o algoritmo para cacheamento de vídeo chamado CARTE (*Current demAnd Rather Than futurE*), criado para uso em *proxies* de vídeo sob demanda. Esse algoritmo foi projetado no escopo de um novo paradigma, no qual as decisões de cacheamento se baseiam exclusivamente no posicionamento atual dos clientes ativos do sistema para calcular a demanda específica que esses clientes irão criar para cada trecho de vídeo no futuro.

Como um diferencial da nova estratégia criada, em relação a outras similares existentes, as decisões tomadas pelo algoritmo levam em consideração a quantidade de clientes presentes dentro de uma janela de tempo localizada à frente de cada trecho de vídeo, prevenindo, assim, que os clientes distantes de um determinado trecho possam interferir no cálculo da prioridade de cacheamento deste trecho. Adicionalmente, o tamanho dessa janela de tempo é o parâmetro chave para configurar o CARTE para alcançar seu desempenho máximo em cada cenário.

Uma vez que o número de clientes ativos dentro de cada janela de tempo pode variar durante a operação do *proxy*, devido ao fluxo de entrada e saída constante dos clientes, o algoritmo de cacheamento deve continuamente (na mesma frequência de entrada e saída dos clientes) avaliar quais trechos de vídeo têm maior demanda, definindo assim o conteúdo que deve permanecer armazenado na memória do *proxy* de forma a maximizar a sua eficiência.

Como consequência, diferentemente das abordagens anteriores, que tendem a subutilizar a largura de banda sustentada pela memória e pelo processador do *proxy*, visando a qualificação do serviço a longo prazo, o CARTE explora esses recursos mais ativamente de modo a alcançar a escalabilidade necessária para sustentar a alta demanda que, em geral, recai sobre os vídeos mais populares de um acervo. Com isso, o algoritmo provê uma maior eficiência para o *proxy*, a qual, por sua vez, cria impactos subsequentes nos custos de implementação do sistema VoD.

Sob esta conjuntura, para conhecer o consumo de recursos praticado por um algoritmo de cacheamento como o CARTE e, ao mesmo tempo, avaliar seu desempenho em termos de métricas mais comuns como as taxas de acertos, criou-se um novo simulador chamado SIMPRO (*proxy SIMulator*). Como um diferencial deste simulador, em relação a outros existentes, ele é capaz de avaliar o desempenho da arquitetura de base usada na implementação do *proxy* VoD, permitindo a identificação dos principais gargalos físicos criados face ao aumento da demanda. Até onde se conhece, essa característica torna o

SIMPRO o primeiro simulador existente dedicado à análise de desempenho de *proxies* VoD sob um ponto de vista arquitetural.

Os resultados comparativos obtidos através desse ambiente de simulação mostram que o novo algoritmo de cacheamento de vídeo desenvolvido no escopo deste estudo é capaz de alcançar uma taxa de acertos significativamente maior do que a produzida por outros de natureza similar (17,3 a 20,7% de diferença no pico), consumindo, para isso, um quantidade também maior de recursos computacionais (10,1 a 12,6% a mais que os outros algoritmos também analisados).

Assumindo o uso de uma arquitetura típica para implementação de um *proxy* VoD, tal como aquela usada nos experimentos realizados neste trabalho, os recursos extras consumidos pelo novo algoritmo tendem a ficar ociosos quando os demais algoritmos também avaliados neste estudo são usados. Com base nisso, é possível considerar que o custo extra, no que diz respeito ao hardware necessário para implementação do *proxy*, é nulo quando o algoritmo proposto neste estudo é utilizado. Por outro lado, considerando o cenário em que cada um dos clientes não servidos pelo *proxy* precisa ser abastecido diretamente pelo servidor principal, a maior taxa de acertos produzida pelo novo algoritmo tende a contribuir para a redução do consumo de recursos (e subsequentes custos) no backbone da rede, uma vez que, neste caso, um menor número de clientes tende a consumir recursos diretos deste canal.

No diz respeito ainda à análise dos recursos consumidos pelos algoritmos analisados, os dados produzidos sugerem que o processador tende a ser o principal gargalo para a aplicação à medida que a demanda aumenta, consumindo, em média, de duas a três vezes o tempo produzido pela memória para desempenhar as tarefas necessárias. Isso permite concluir que o desenvolvimento de um novo particionamento de hardware e software, especificamente dedicado ao *proxy* VoD, tende a contribuir para o aumento significativo da sua eficiência.

Portanto, como um trabalho futuro, pretende-se conduzir uma revisão da arquitetura do *proxy* VoD, como forma de planejar mapeamentos mais eficientes para os seus componentes em termos de hardware/software, sob um ponto de vista da relação custo-benefício. Assim, espera-se contribuir para o aumento da escalabilidade da infra-estrutura, atualmente disponível para VoD e, ao mesmo tempo, propiciar a redução dos investimentos para a implantação desse serviço.

Além disso, considerando que um *proxy* VoD é projetado para operar no modo contínuo e as condições de carga poderem variar durante a provisão do serviço, planeja-se implementar as funcionalidades necessárias para fornecer ao CARTE a habilidade de realizar o ajuste dinâmico e incremental do seu parâmetro de trabalho, o tamanho da janela de tempo.

Assim, o algoritmo será capaz de se adaptar às diferentes circunstâncias, preservando sua eficiência máxima mesmo frente a flutuações na carga de trabalho.

Para incorporar essa nova capacidade ao algoritmo, far-se-á o uso das diretrizes apresentadas nesse estudo, muitas das quais se baseiam na observação da densidade de clientes (número de clientes por tempo de intervalo) produzida pelas condições de carga de trabalho do *proxy*. Uma vez que a densidade produz impactos diretos sobre a escolha do melhor tamanho para a janela de tempo, o aspecto chave para o ajuste dinâmico do algoritmo consiste em monitorar o fluxo de entrada e saída dos clientes para, de maneira incremental, identificar e produzir as mudanças necessárias sobre o tamanho da janela de tempo de modo a encontrar as configurações ótimas para o sistema.

Por fim, a partir da capacidade de ajuste dinâmico da janela de tempo, pretende-se implementar e avaliar os benefícios produzidos pelo uso de um esquema de janelas segregadas por vídeo. Assim, cada vídeo usará um tamanho de janela mais adequado à suas condições específicas de carga. Este esquema tende a ser especialmente importante em cenários onde a quantidade de memória disponível é pequena em proporção ao número de clientes servidos. Isto porque, nestes casos, o algoritmo de cacheamento precisa promover a reciclagem de sequências disponíveis em diferentes vídeos para liberar o volume de memória necessário para abastecer todos os fluxos de vídeo recebidos pelo *proxy*.

REFERÊNCIAS

ABAD, P. et al. TOPAZ: an open-source interconnection network simulator for chip multiprocessors and supercomputers. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON NETWORKS ON CHIP, 6., 2012, Copenhagen, Denmark. **Proceedings. . .** Los Alamitos: IEEE, 2012. p. 99-106.

ABEYWICKRAMA, S.; WONG, E. Dynamic bandwidth allocation algorithms for local storage based VoD delivery: comparison between single and dual receiver configurations. **Optics Communications**, [S.l.], v. 336, n. 0, 2015. p. 40-46.

ADHIKARI, V. et al. Unreeling netflix: understanding and improving multi-cdn movie delivery. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATIONS, 26., 2012, Orlando, Flórida, United States of America. **Proceedings. . .** Los Alamitos: IEEE, 2012. p.1620-1628.

ALMASRI, I. et al. Universal ISA simulator with soft processor FPGA implementation. In: IEEE JORDAN CONFERENCE ON APPLIED ELECTRICAL ENGINEERING AND COMPUTING TECHNOLOGIES, 2011, Amman, Jordania. **Proceedings. . .** Los Alamitos: IEEE, 2011. p.1-6.

ALMEIDA, J. M. et al. Analysis of Educational Media Server Workloads. In: INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, 11., 2011, Port Jefferson, New York, United States of America. **Proceedings. . .** New York: ACM, 2001. p. 21-30.

APPLEGATE, D. et al. Optimal Content Placement for a Large-scale VoD System. In: CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES, 6., 2010, Philadelphia, Pennsylvania, United States of America. **Proceedings. . .** New York: ACM, 2010. p. 4:1-4:12.

AUSTIN, T.; LARSON, E.; ERNST, D. SimpleScalar: an infrastructure for computer system modeling. **Computer**, Los Alamitos, CA, United States of America, v. 35, n. 2, feb. 2002. p. 59-67.

AVRAMOVA, Z. et al. Analysis and Modeling of Video Popularity Evolution in Various Online Video Content Systems: power-law versus exponential decay. In: INTERNATIONAL CONFERENCE ON EVOLVING INTERNET, 1., 2009, Cannes/La Bocca. **Proceedings. . .** Los Alamitos: IEEE, 2009. p. 95-100.

BATAA, O. et al. Service control algorithm of providing efficient video-on-demand service using hybrid mechanism. In: INTERNATIONAL CONFERENCE ON CONSUMER ELECTRONICS, COMMUNICATIONS AND NETWORKS, 2., 2012, Yichang, China. **Proceedings. . .** New York: IEEE, 2012. p. 3507-3512.

BEN ABDESSLEM, F.; LINDGREN, A. Cacheability of YouTube Videos in Cellular Networks. In: ACM WORKSHOP ON ALL THINGS CELLULAR: OPERATIONS, APPLICATIONS AND CHALLENGES, 4., 2014, Chicago, Illinois, United States of America. **Proceedings. . .** New York: ACM, 2014. p. 53-58.

BERRENDORF, R. **WR Cluster Hardware**. Sankt Augustin: Bonn-Rhein-Sieg University, 2014. Disponível em: <http://wr0.wr.inf.hbrs.de/wr/hardware/hardware.html>. Acesso em: 05 jun. 2014.

BORST, S.; GUPTA, V.; WALID, A. Distributed Caching Algorithms for Content Distribution Networks. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATIONS, 24., 2010, San Diego, California, United States of America. **Proceedings. . .** Los Alamitos: IEEE, 2010. p. 1-9.

BROADCOM. **1-Gigabit TCP Offload Engine**. Irvine: Broadcom, 2009. Disponível em: <https://www.broadcom.com/collateral/wp/5709-WP101.pdf >. Acesso em: 05 jan. 2012.

BRUNEAU-QUEYREIX, J. et al. Home-Boxes: context-aware distributed middleware assisting content delivery solutions. In: ACM MIDDLEWARE POSTERS & DEMOS SESSION, 3., 2014, Bordeaux, France. **Proceedings. . .** New York: ACM, 2014. p. 37-38.

BURGER, D.; AUSTIN, T. M. The SimpleScalar Tool Set, Version 2.0. **ACM SIGARCH Computer Architecture News**, New York, United States of America, v. 25, n. 3, p.13-25, jun. 1997.

CAMPANOTTI, B.; HURT, A. Building Real World Media in the Cloud. **SMPTE Conferences**, Hollywood, California, v. 2013, n. 10, p. 1-7, 2013.

CARBUNAR, B. et al. Predictive Caching for Video on Demand CDNs. In: IEEE GLOBAL COMMUNICATIONS CONFERENCE, 34., 2011, Houston, Texas, United States of America. **Proceedings. . .** Los Alamitos: IEEE, 2011. p. 1-5.

CHAN, S.-H.; XU, Z.; LIU, N. Optimizing video-on-demand with source coding. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO, 2013, San Jose, California, United States of America. **Proceedings. . .** Los Alamitos: IEEE, 2013. p. 1-6.

CHEN, H. et al. A New *Proxy* Caching Scheme for Parallel Video Servers. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER NETWORKS AND MOBILE COMPUTING, 2. 2003, Washington, United States of America. **Proceedings. . .** Los Alamitos: IEEE, 2003. p. 438-441.

CHIU, H. et al. Window-based popularity caching for IPTV on-demand services. **ISRN Communications and Networking**, New York, United States of America, v. 2011, n. 1, p. 32-43, jan. 2011.

CISCO. **Visual networking index: forecast and methodology, 2011-2016**. Jose, California, United States of America: CISCO, 2012. Disponível em: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html>. Acesso em: 19 jun. 2012.

CODE::BLOCKS. **The open source, cross platform, free C, C++ and Fortran IDE**. Boston: Free Software Foundation, 2014. Disponível em: <http://www.codeblocks.org/>. Acesso em: 20 jun. 2013.

CORBET, J.; RUBINI, A.; KROAH-HARTMAN, G. **Linux Device Drivers**, 3rd ed. Sebastopol, California, United States of America: O'Reilly Media, 2005.

DAN, A.; SITARAM, D.; SHAHABUDDIN, P. Scheduling policies for an on-demand video server with batching. In: ACM INTERNATIONAL CONFERENCE ON MULTIMEDIA, 2., 1994, San Francisco, California, United States of America. **Proceedings. . .** New York: ACM, 1994. p.15-23.

DAN, A.; SITARAM, D.; SHAHABUDDIN, P. Dynamic batching policies for an on-demand video server. **Multimedia Systems**, Berlim, v. 4, n. 3, p.112-121, 1996.

DECKER, C. et al. A file system for system programming in ubiquitous computing. **Personal Ubiquitous Computing**, London, v. 11, n. 1, p. 21-31, oct. 2006.

DEWANGAN, A.; JALIHAL, D. Statistics based energy efficient caching decisions for IPTV services. In: NATIONAL CONFERENCE ON COMMUNICATIONS, 13., 2013, New Delhi, India. **Proceedings. . .** Los Alamitos: IEEE, 2013. p. 1-5.

DHAGE, S. N.; MESHRAM, B. Design and implementation of video servers for VoD system. **International Journal on Cloud Computing**, Olney, v. 2, n. 1, p. 61-88, jan. 2013.

ERMAN, J. et al. Over the Top Video: the gorilla in cellular networks. In: ACM SIGCOMM CONFERENCE ON INTERNET MEASUREMENT CONFERENCE, 11., 2011, Berlin, Germany. **Proceedings. . .** New York: ACM, 2011. p. 127-136.

FAMAEY, J. et al. Towards a predictive cache replacement strategy for multimedia content. **Journal of Network Computing. Applications.**, London, UK, v. 36, n. 1, p. 219-227, jan. 2013.

FAN, Q.; PANKANTI, S. Robust Foreground and Abandonment Analysis for Large-Scale Abandoned Object Detection in Complex Surveillance Videos. In: IEEE INTERNATIONAL CONFERENCE ON ADVANCED VIDEO AND SIGNAL-BASED SURVEILLANCE, 9., 2012, Beijing, Peking. **Proceedings. . .** Los Alamitos: IEEE, 2012. p. 58-63.

GODSE, A. P. **Microprocessors and Microcontrollers**, 1st ed. Oxford: Technical Publications, 2011.

GOLREZAEI, N. et al. FemtoCaching: wireless video content delivery through distributed caching helpers. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATIONS, 26., 2012, Orlando, Flórida. **Proceedings. . .** Los Alamitos: IEEE, 2012. p. 1107-1115.

GRANADO, A. C. **Experimental Evaluation of the Collapsed Cooperative Video Cache for Video on Demand Systems**. 2010. Dissertação (Mestrado em Ciência da Computação), Federal University of Rio de Janeiro, COPPE, Rio de Janeiro, 2010.

HO, C.-P.; YU, J.-Y.; LEE, S.-Y. Efficient data replication for the delivery of high-quality video content over P2P VoD advertising networks. **EURASIP Journal on Advances in Signal Processing**, Gewerbestrasse, v. 2011, p. 105-115, 2011.

HONG, D.; DE VLEESCHAUWER, D.; BACCELLI, F. A chunk-based caching algorithm for streaming video. In: WORKSHOP ON NETWORK CONTROL AND OPTIMIZATION, 4., 2010, Gent, Belgium. **Proceedings. . .** [S.l.: s.n.], 2010.

HWANG, K.-W. et al. Abandonment and its impact on P2P VoD streaming. In: IEEE INTERNATIONAL CONFERENCE ON PEER-TO-PEER COMPUTING, 13., 2013, Trento, Italy. **Proceedings. . .** Los Alamitos: IEEE, 2013. p. 1-10.

INTEL. **PCI Express Ethernet Networking**. Santa Clara: Intel, 2005. Disponível em: <http://www.intel.com/content/www/us/en/pci-express/pci-express-ethernet-networking-paper.html>. Acesso em: 23 ago. 2011.

INTEL. **Accelerating High-Speed Networking with Intel® I/O Acceleration Technology**. Santa Clara: Intel, 2006. Disponível em: <http://download.intel.com/support/network/sb/98856.pdf >. Acesso em: 13 set. 2011.

ISHIKAWA, E.; AMORIM, C. L. Collapsed Cooperative Video Cache for Content Distribution Networks. In: BRAZILIAN SIMPOSIUM ON COMPUTER NETWORKS, 21., Florianópolis, Brazil. **Proceedings. . .** Porto Alegre: Sociedade Brasileira de Computação, 2003. p. 249-264.

ISHIKAWA, E.; AMORIM, C. L. **Collapsed Distributed Cooperative Memory for Interactive and Scalable Media-on-demand Systems**. U.S. Patent 7,596,664, to COOPE/UFRJ. 2009.

JI, W. The Design of Passive Optical Networking+Ethernet over Coaxial Cable Access Networking and Video-on-Demand Services Carrying. **Fiber and Integrated Optics**, Abingdon, v. 32, n. 4, p. 268-279, 2013.

JIANG, C. et al. PCantorSim: accelerating parallel architecture simulation through fractal-based sampling. **ACM Transactions on Architecture Code Optimization.**, New York, NY, United States of America, v. 10, n. 4, p. 49:1-49:24, dec. 2013.

JUNG, J.; KRISHNAMURTHY, B.; RABINOVICH, M. Flash crowds and denial of service attacks: characterization and implications for cdns and web sites. In: INTERNATIONAL CONFERENCE WORLD WIDE WEB, 11., New York, NY, United States of America. **Proceedings. . .** New York: ACM, 2002. p. 293-304.

KAI-CHUN, L.; YU, H.-F. Adjustable Two-Tier Cache for IPTV Based on Segmented Streaming. **International Journal on Digital Multimedia Broadcasting**, Nasr, v. 2012, n. 1, 2012.

KHEMMARAT, S. et al. Watching user generated videos with prefetching. In: ACM CONFERENCE ON MULTIMEDIA SYSTEMS, 2., 2011, San Jose, California, United States of America. **Proceedings. . .** New York: ACM, 2011. p. 187-198.

KRISHNAN, S.; SITARAMAN, R. Video Stream Quality Impacts Viewer Behavior: inferring causality using quasi-experimental designs. **IEEE Transactions on Networking**, Los Alamitos, v. 21, n. 6, p. 2001-2014, dec. 2013.

LAI, J.-H. et al. Tennis Video 2.0: a new presentation of sports videos with content separation and rendering. **Journal of Visual Communication and Image Representation**, Orlando, Florida, United States of America, v. 22, n. 3, p. 271-283, apr. 2011.

LEE, M.-C.; LEU, F.-Y.; CHEN, Y.-P. Cache Replacement Algorithms for YouTube. In: IEEE INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS, 28., 2014, Victoria, British Columbia, Canada. **Proceedings. . .** Los Alamitos: IEEE, 2014. p. 743-750.

LI, C.; DING, C.; SHEN, K. Quantifying the Cost of Context Switch. In: WORKSHOP ON EXPERIMENTAL COMPUTER SCIENCE, 1., 2007, San Diego, California. **Proceedings. . .** New York: ACM, 2007.

LI, J.; CHEN, Z. Sliding-window caching algorithm for streaming media server. In: INTERNATIONAL CONFERENCE ON INTERACTION SCIENCES: INFORMATION TECHNOLOGY, CULTURE AND HUMAN, 2., 2009, Seoul, Korea. **Proceedings. . .** New York: ACM, 2009. p. 1152-1159.

LI, J.; LI, F.; JIANG, X. Flexible-segmentation-jumping strategy to reduce user-perceived latency for video on demand. **Applied Computational Intelligence and Soft Computing**, New York, v. 2011, p. 1:1-1:8, jan. 2011.

LI, J.; YANG, J.; XI, H. A Scalable and Cooperative Caching Scheme in a Distributed VOD System. In: INTERNATIONAL CONFERENCE ON COMMUNICATION SOFTWARE AND NETWORKS, 2., 2009, Macau, China, Asia. **Proceedings. . .** Los Alamitos: IEEE, 2009. p. 247-250.

LI, P.; ZHENG, W.; ZHANG, K. The design of streaming media *proxy* server based on patching first algorithm. In: INTERNATIONAL FORUM ON STRATEGIC TECHNOLOGY, 6., 2011, Harbin, Heilongjiang. **Proceedings. . .** Los Alamitos: IEEE, 2011. p. 643-647.

LING, Q. et al. An adaptive caching algorithm suitable for time-varying user accesses in VOD systems. **Multimedia Tools and Applications**, New York, p. 1-21, 2014.

LO, C.-W.; SU, Y.-Y. P2P video streaming replication scheme for P2P VoD services. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGY CONVERGENCE, 5., 2014, Busan, Korea. **Proceedings. . .** Los Alamitos: IEEE, 2014. p. 391-393.

MA, K. J.; BARTOS, R.; BHATIA, S. Review: a survey of schemes for internet-based video delivery. **Journal of Network and Computer Applications**, London, United Kingdon, England, v. 34, n. 5, p. 1572-1586, sept. 2011.

MOLLER, Steffen. **Cbp2make: Makefile Generation Tool for Code::Blocks IDE**. New York: Mirai Computing, 2015. Disponível em: <http://sourceforge.net/p/cbp2make/wiki/Home/>. Acesso em: 20 jun. 2013.

NETFLIX. **Internet Connection Speed Recommendations**. Los Gatos, California: Netflix, 2014. Disponível em: <https://help.netflix.com/en/node/306>. Acesso em: 14 jan. 2014.

NETFLIX. **All DVDs Releasing This Week**. Los Gatos, California: Netflix, 2013. Disponível em: <http://dvd.netflix.com/AllNewReleases?lnkctr=NavAllNewReleases>. Acesso em: 5 dec. 2013.

PARK, J. G.; CHOI, H.; LEE, B. C. Content caching with bi-level control for efficient IPTV content steaming service. In: INTERNATIONAL CONFERENCE ON INFORMATION NETWORKING, 29. 2014, Phuket, Thailand. **Proceedings. . .** Los Alamitos: IEEE, 2014. p. 439-443.

PASSARELLA, A. Review: a survey on content-centric technologies for the current internet: cdn and p2p solutions. **Computer Communication**, Amsterdam, Netherlands, v. 35, n. 1, p. 1-32, jan. 2012.

PATHAN, M.; BUYYA, R.; VAKALI, A. Content Delivery Networks: state of the art, insights, and imperatives. In: BUYYA, R.; PATHAN, M.; VAKALI, A. **Content Delivery Networks**, Springer: Berlin, 2008. p. 3-32. (Lecture Notes Electrical Engineering, v.9).

POSIX. **The Open Group Base Specifications Issue 7**. Los Alamitos: IEEE, 2013. Disponível em: <http://pubs.opengroup.org/onlinepubs/9699919799/>. Acesso em: 6 jan. 2014.

POURMIR, A.; RAMANATHAN, P. Distributed caching and coding in VoD. In: IEEE CONFERENCE ON COMPUTER COMMUNICATIONS WORKSHOPS, 28., 2014, Toronto, Ontario, Canada. **Proceedings. . .** Los Alamitos: IEEE, 2014. p. 233-238.

QUDAH, B.; SARHAN, N. J. Efficient Delivery of On-demand Video Streams to Heterogeneous Receivers. **ACM Transactions on Multimedia Computing, Communications and Applications**, New York, NY, United States of America, v. 6, n. 3, p. 20:1-20:25, aug. 2010.

REN, D. et al. Distributed joint optimization for large-scale video-on-demand. **Computer Networks**, Melbourne, Australia, v. 75, Part A, n. 0, p. 86-98, 2014.

RYU, M.; KIM, H.; RAMACHANDRAN, U. Impact of Flash Memory on Video-on-demand Storage: analysis of tradeoffs. In: ACM CONFERENCE ON MULTIMEDIA SYSTEMS, 2., 2011, San Jose, California, United States of America. **Proceedings. . .** New York: ACM, 2011. p. 175-186.

SAVI, M. et al. Energy-efficient VoD content delivery and replication in integrated metro/access networks. In: IEEE LATIN-AMERICA CONFERENCE ON COMMUNICATIONS, 6., 2014, Cartagena de Indias, Colômbia. **Proceedings. . .** Los Alamitos: IEEE, 2014. p. 1-6.

SHEN, L.; TU, W.; STEINBACH, E. A Flexible Starting Point Based Partial Caching Algorithm for Video on Demand. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO, 2007, Beijing, Peking. **Proceedings. . .** Los Alamitos: IEEE, 2007. p. 76-79.

SIGOURE, B. **How long does it take to make a context switch?** New York: Google, 2014. Disponível em: <http://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make-context.html>. Acesso em: 27 nov. 2013.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating System Concepts**. 9th ed. Chichester: John Wiley, 2013.

SMIT, M.; STROULIA, E. Simulating Service-Oriented Systems: a survey and the services-aware simulation framework. **IEEE Transactions on Services Computing**, Los Alamitos: IEEE, v. 6, n. 4, p. 443-456, dec. 2013.

SUMMERS, J. et al. Methodologies for Generating HTTP Streaming Video Workloads to Evaluate Web Server Performance. In: INTERNATIONAL SYSTEMS AND STORAGE CONFERENCE, 5., 2012, Haifa, Israel. **Proceedings. . .** New York: ACM, 2012. p. 2:1-2:12.

SUN, Y. et al. Trace-Driven Analysis of ICN Caching Algorithms on Video-on-Demand Workloads. In: ACM INTERNATIONAL CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES, 10., Sydney, Australia. **Proceedings. . .** New York: ACM, 2014. p. 363-376.

TU, W. et al. *Proxy* Caching for Video-on-Demand Using Flexible Starting Point Selection. **IEEE Transactions on Multimedia**, Los Alamitos, v. 11, n. 4, p. 716-729, jun. 2009.

USTELECOM. **Broadband Investment**. New York, Washington, United States of America: United States Telecom Association, 2015. Disponível em: <http://www.ustelecom.org/broadband-industry/broadband-industry-stats/investment>. Acesso em: 15 jan. 2015.

VIJENDRAN, A. S.; THAVAMANI, S. Survey of Caching and Replica Placement Algorithm for Content Distribution in Peer to Peer Overlay Networks. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE, ENGINEERING AND INFORMATION TECHNOLOGY, 2., 2012, Coimbatore, India. **Proceedings. . .** New York: ACM, 2012. p. 248-252.

VINAY, A. et al. A multithreaded based load balancing framework for video-on-demand systems. In: INTERNATIONAL CONFERENCE ON EMERGING TRENDS IN TECHNOLOGY, 1. 2011, Mumbai, Maharashtra, India. **Proceedings. . .** New York: ACM, 2011. p. 363-369.

VO, N.-S. et al. An optimization problem for replication in VoD service over broad-band wireless Internet. In: INTERNATIONAL CONFERENCE ON COMPUTING, MANAGEMENT AND TELECOMMUNICATIONS, 2., 2014, Da Nang, Vietnam. **Proceedings. . .** Los Alamitos: IEEE, 2014. p. 80-83.

WILLHALM, T.; DEMENTIEV, R.; FAY, P. **Intel Performance Counter Monitor - A better way to measure CPU utilization**. Santa Clara: Intel, 2012. Disponível em: <https://software.intel.com/en-us/articles/intel-performance-counter-monitor>. Acesso em: 5 jan. 2012.

WU, K.-L.; YU, P. S.; WOLF, J. L. Segment-based *Proxy* Caching of Multimedia Streams. In: INTERNATIONAL CONFERENCE ON WORLD WIDE WEB, 10., 2001, Hong Kong, Hong Kong. **Proceedings. . .** New York: ACM, 2001. p. 36-44.

WU, T. et al. Reuse time based caching policy for video streaming. In: IEEE CONSUMER COMMUNICATIONS AND NETWORKING CONFERENCE, 9., 2012, Las Vegas, Nevada. **Proceedings. . .** Los Alamitos: IEEE, 2012. p. 89-93.

YANG, J.; HAJEK, B. A hybrid algorithm for content placement in distributed video on demand systems. In: IEEE INTERNATIONAL SYMPOSIUM ON INFORMATION THEORY, 22. 2014, Honolulu, Havaí. **Proceedings. . .** Los Alamitos: IEEE, 2014. p. 816-820.

YU, H. et al. Understanding User Behavior in Large-scale Video-on-demand Systems. **ACM SIGOPS Operating Systems Review**, New York, v. 40, n. 4, p. 333-344, apr. 2006.

YU, J. et al. A dynamic caching algorithm based on internal popularity distribution of streaming media. **Multimedia Systems**, Berlim, v. 12, n. 2, p. 135-149, jul. 2006.

YU, J. et al. Internal popularity of streaming video and its implication on caching. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS, 20., 2006, Viena, Austria. **Proceedings. . .** Los Alamitos: IEEE, 2006. p. 35-40.

ZENG, Z.; VEERAVALLI, B.; LI, K. A novel server-side *proxy* caching strategy for large-scale multimedia applications. **Journal of Parallel and Distributed Computing**, Orlando, Florida, United States of America, v. 71, n. 4, p. 525-536, apr. 2011.

APÊNDICE A <FORMATO DO ARQUIVO SIV>

ID: Identificador do vídeo

AS: Instante (segundo) para abertura da sessão do cliente

FS: Instante (segundo) para fechamento da sessão do cliente

-ID-	-AS-	-FS-
095	0.003	1.800
078	0.006	0.240
054	0.009	0.000
019	0.012	2.340
007	0.015	0.960
035	0.016	0.660
024	0.020	1.080
033	0.023	1.380
094	0.024	2.340
056	0.027	0.120
086	0.030	5.400
016	0.034	0.180
079	0.036	0.420
043	0.038	5.400
014	0.041	2.520
054	0.046	0.600
036	0.047	0.420
000	0.049	0.180
073	0.054	4.320
065	0.059	1.680
046	0.062	1.440
011	0.066	0.420
074	0.066	0.420
000	0.069	0.600
025	0.070	1.440
053	0.073	0.060
068	0.074	0.000

023	0.078	1.800
048	0.081	0.840
015	0.082	0.060
039	0.086	2.580
061	0.090	1.740
063	0.093	0.120
076	0.097	5.400
075	0.101	0.660
034	0.102	1.140
015	0.102	4.320
062	0.105	1.320
000	0.110	0.540
002	0.112	0.480
083	0.113	0.000
099	0.114	0.120
001	0.116	2.220
079	0.119	0.060
017	0.123	1.380
014	0.126	0.120
038	0.130	0.180
002	0.134	0.060
001	0.136	0.600
037	0.138	0.420
090	0.142	0.240
043	0.147	0.240
018	0.148	1.920
078	0.154	0.600
051	0.156	5.400
089	0.157	5.400
036	0.160	0.000
062	0.161	5.400
098	0.162	5.400
074	0.170	4.680
036	0.173	0.120

014	0.179	5.400
022	0.180	0.000
012	0.182	5.400
041	0.186	2.400
077	0.188	0.240
027	0.190	0.300
007	0.191	5.400
036	0.196	0.300
057	0.200	3.360
086	0.203	0.300
080	0.207	1.320
052	0.210	0.360
082	0.212	2.580
043	0.214	1.020
056	0.215	0.060
070	0.217	0.360
063	0.221	0.300
040	0.224	0.780
001	0.227	0.120
064	0.233	1.380
068	0.234	3.000
012	0.236	1.320
023	0.237	5.400
005	0.243	0.540
052	0.251	0.000
026	0.253	1.800
052	0.256	0.900
088	0.260	1.080
067	0.264	1.080
083	0.267	5.400
082	0.270	1.800
094	0.273	0.780
060	0.277	0.180
069	0.280	0.240

074	0.283	0.360
063	0.285	5.400
043	0.288	0.000
064	0.289	0.240
071	0.291	5.400
070	0.292	0.600
084	0.296	4.740
042	0.296	2.520
026	0.298	1.260
005	0.302	0.780
063	0.307	1.260
009	0.309	0.180
078	0.316	2.220
028	0.322	0.480
099	0.327	0.120
034	0.330	5.400
041	0.337	0.000
067	0.339	0.420
044	0.344	0.540
017	0.345	0.060
002	0.346	0.900
098	0.349	0.000
036	0.353	0.120
085	0.354	0.960
057	0.357	0.120
020	0.366	2.040
085	0.369	2.640
093	0.377	0.420
075	0.380	5.400
070	0.383	1.500
034	0.389	0.840
044	0.391	0.720
051	0.394	0.000
035	0.398	0.360

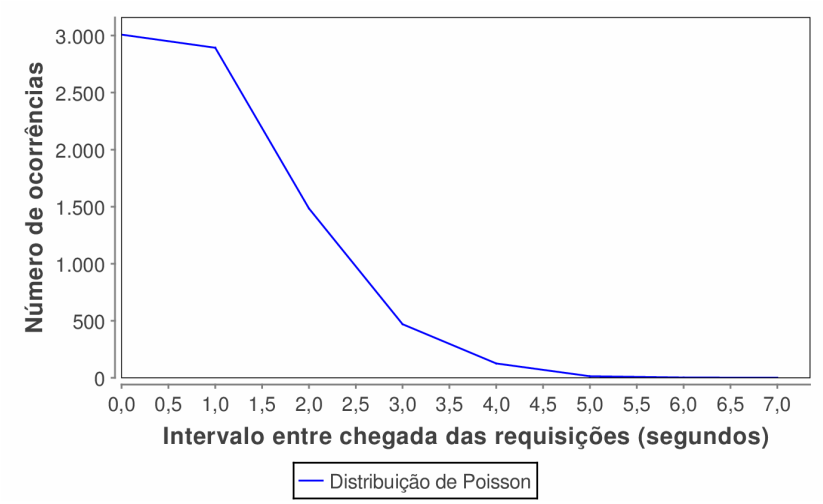
058	0.400	5.400
030	0.401	0.960
021	0.402	0.000
066	0.402	0.420
026	0.406	5.400
077	0.409	4.620
039	0.411	0.000
051	0.413	0.840
021	0.420	0.000
074	0.424	3.360
066	0.425	0.540
096	0.429	5.400
068	0.432	2.460
068	0.433	0.720
060	0.437	5.400
055	0.441	0.720
002	0.443	1.740
013	0.444	0.420
096	0.450	1.320
069	0.458	1.380
034	0.459	2.820
080	0.460	1.320
099	0.464	0.060
069	0.466	5.400
062	0.468	2.160
023	0.471	0.360
047	0.472	3.360
094	0.477	2.340
038	0.480	4.020
037	0.484	0.060
010	0.486	0.900
065	0.487	2.640
012	0.488	1.020
063	0.494	1.440

070	0.497	0.300
043	0.498	0.240
091	0.502	0.420
009	0.505	5.400
043	0.506	5.400
042	0.508	2.280
097	0.510	2.760
087	0.514	0.180
046	0.516	0.240
002	0.516	1.620
083	0.523	0.300
003	0.526	0.480
081	0.529	0.900
053	0.530	1.560
086	0.533	0.600
027	0.538	0.000
009	0.539	0.600
068	0.234	3.000
012	0.236	1.320
023	0.237	5.400
005	0.243	0.540
052	0.251	0.000
026	0.253	1.800
052	0.256	0.900
088	0.260	1.080
067	0.264	1.080
071	0.291	5.400
070	0.292	0.600
084	0.296	4.740
042	0.296	2.520
026	0.298	1.260
005	0.302	0.780
063	0.307	1.260

APÊNDICE B < CURVAS PARA AS DISTRIBUIÇÕES DE POISSON E ZIPF: DIFERENTES RESULTADOS PARA A VARIAÇÃO DOS PARÂMETROS DE CONTROLE DAS DIS-TRIBUIÇÕES>

Figura B.1 - Distribuição de Poisson para os diferentes valores de λ usados neste trabalho. Nos Gráficos da figura, o número total de requisições recebidas pelo *proxy* foi 8000

(a) λ = 1s



(b) λ = 2s

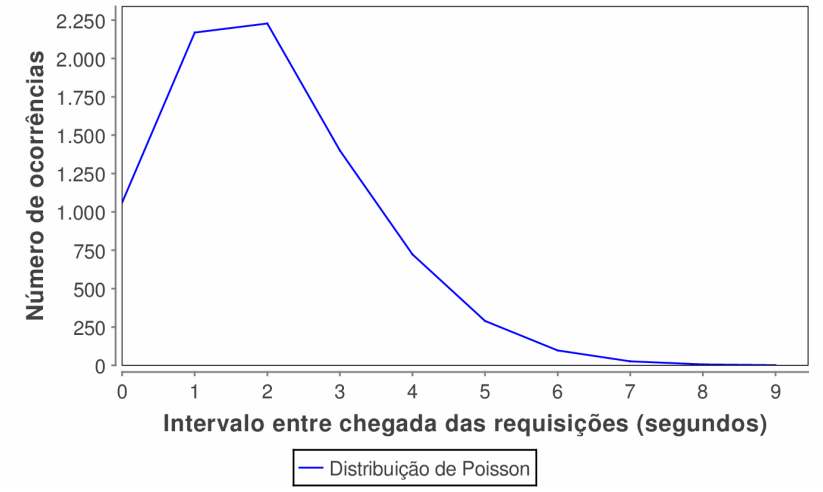
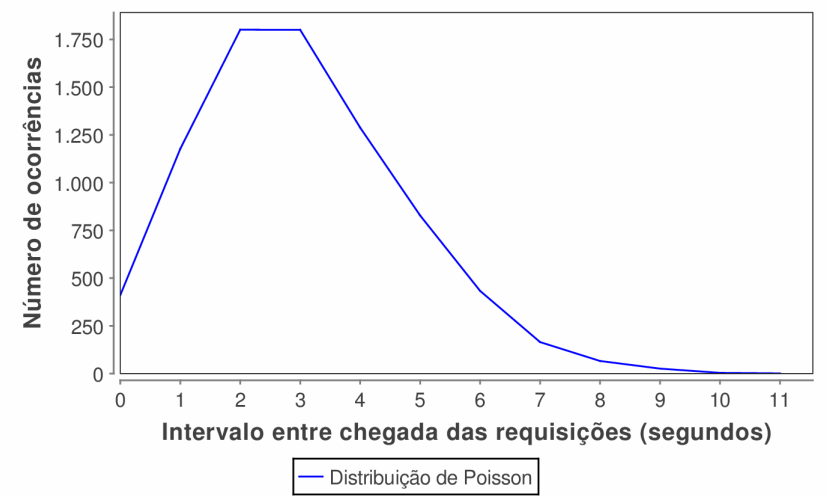
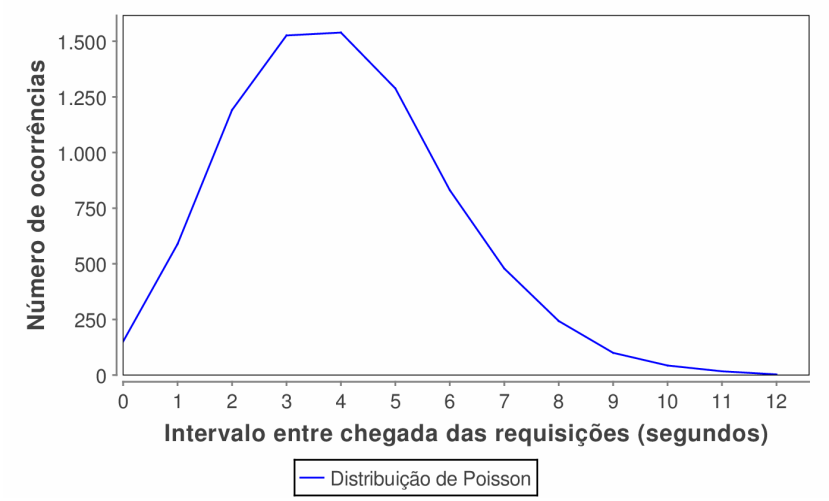


Figura B.1 - Distribuição de Poisson para os diferentes valores de λ usados neste trabalho. Nos Gráficos da figura, o número total de requisições recebidas pelo *proxy* foi 8000 (cont.)

(c) λ = 3s



(d) λ = 4s



(e) λ = 5s

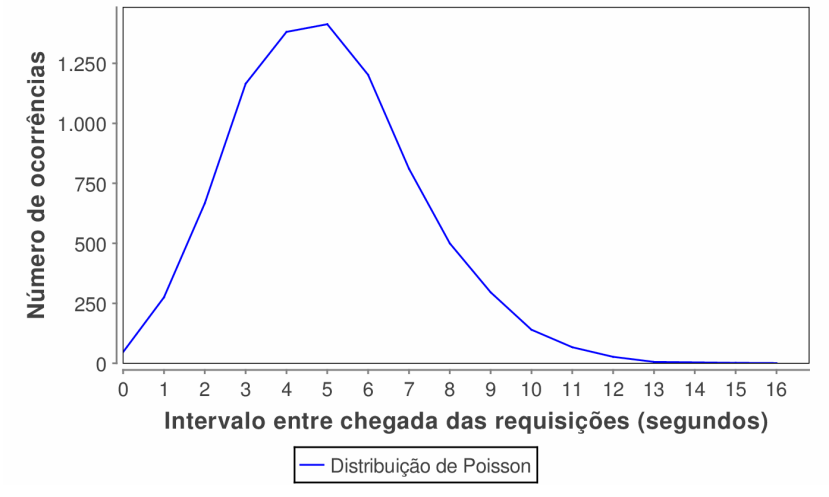
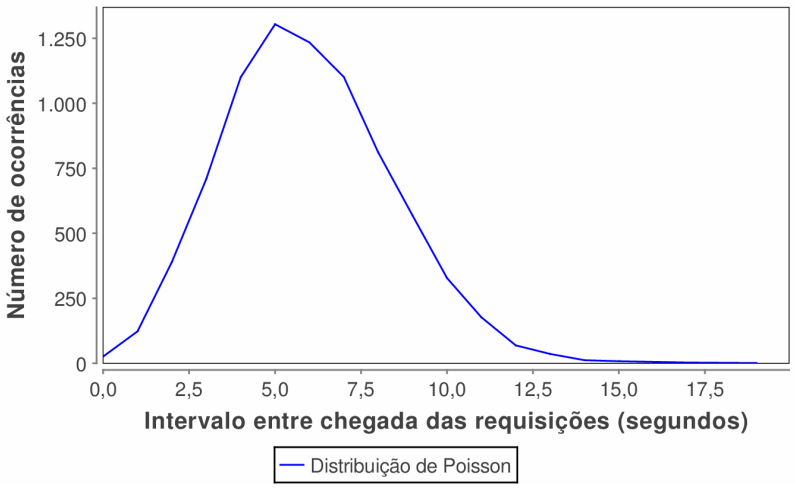
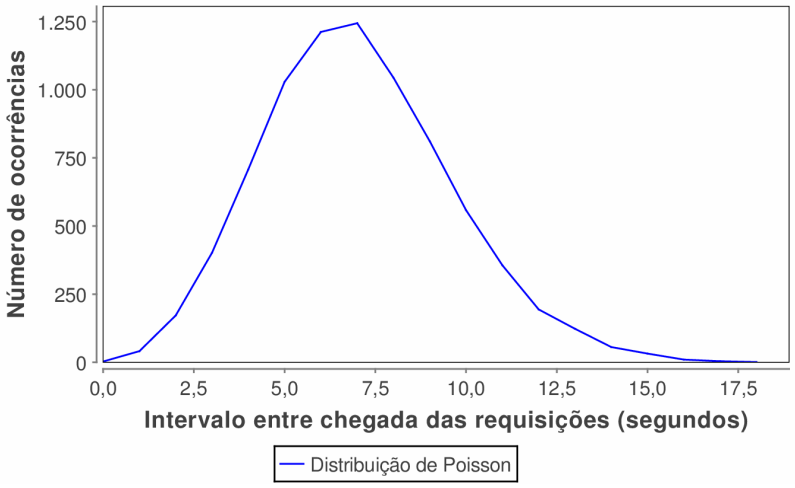


Figura B.1 - Distribuição de Poisson para os diferentes valores de λ usados neste trabalho. Nos Gráficos da figura, o número total de requisições recebidas pelo *proxy* foi 8000 (cont.)

(f) λ = 6s



(g) λ = 7s



(h) λ = 8s

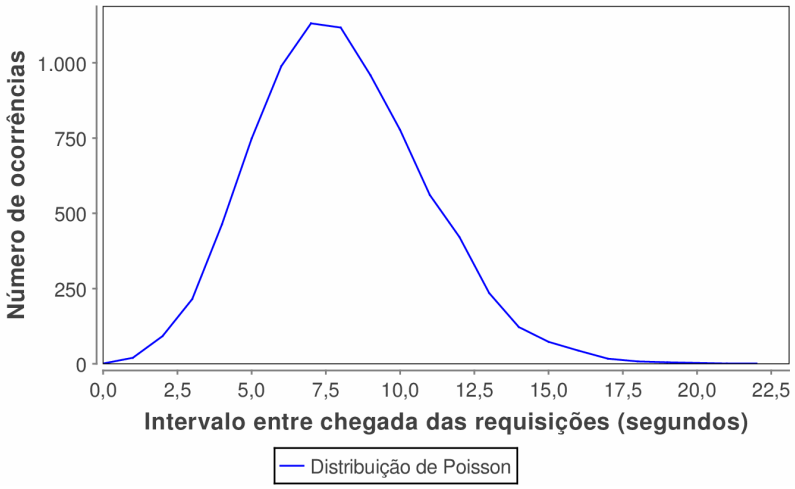
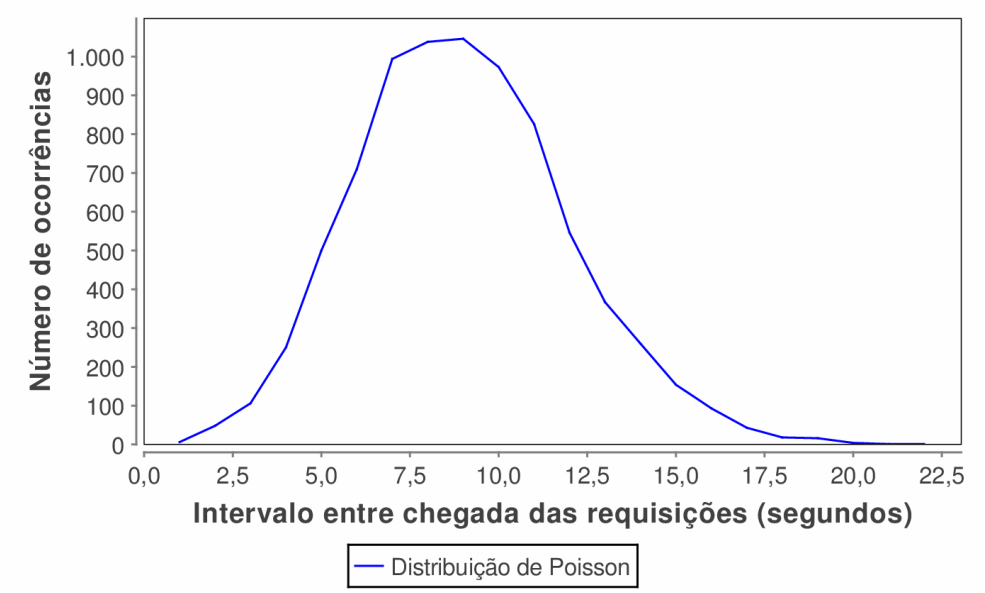


Figura B.1 - Distribuição de Poisson para os diferentes valores de λ usados neste trabalho. Nos Gráficos da figura, o número total de requisições recebidas pelo *proxy* foi 8000 (cont.)

(i) λ = 9s



Fonte: do autor (2015).

Figura B.2 - Distribuição de Zipf para os diferentes valores de θ usados neste trabalho. Nos Gráficos da figura, NV = 100 e o número total de requisições recebidas pelo *proxy* foi 8000

(a) $\theta = 0$

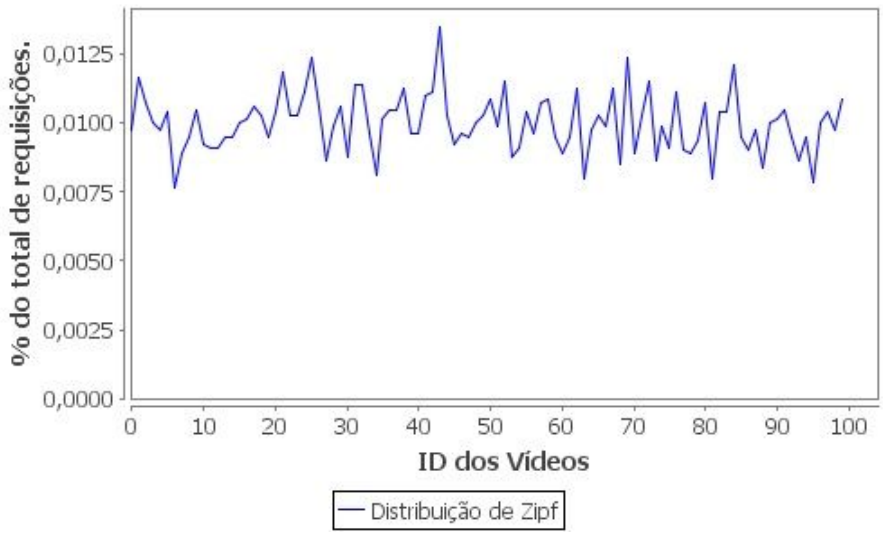
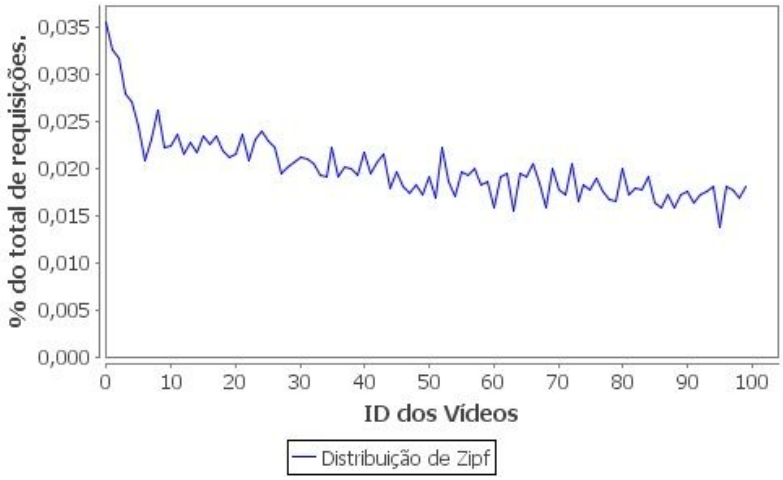
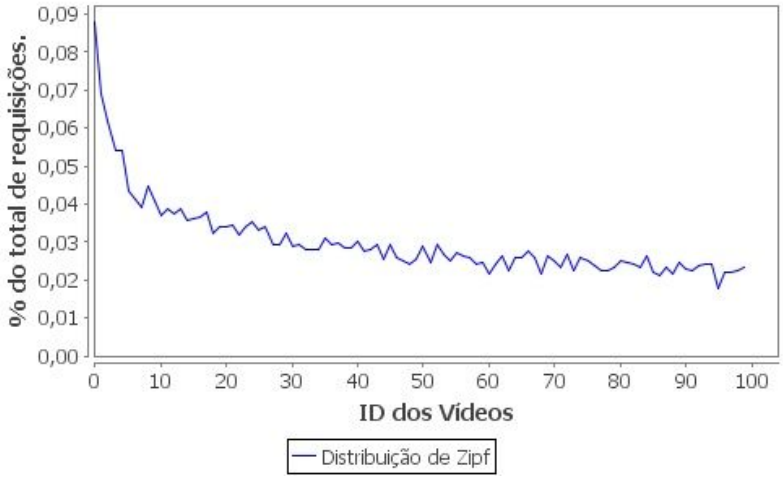


Figura B.2 - Distribuição de Zipf para os diferentes valores de θ usados neste trabalho. Nos Gráficos da figura, NV = 100 e o número total de requisições recebidas pelo *proxy* foi 8000 (cont.)

(b) θ = 0,25



(c) θ = 0,5



(d) θ = 0,75

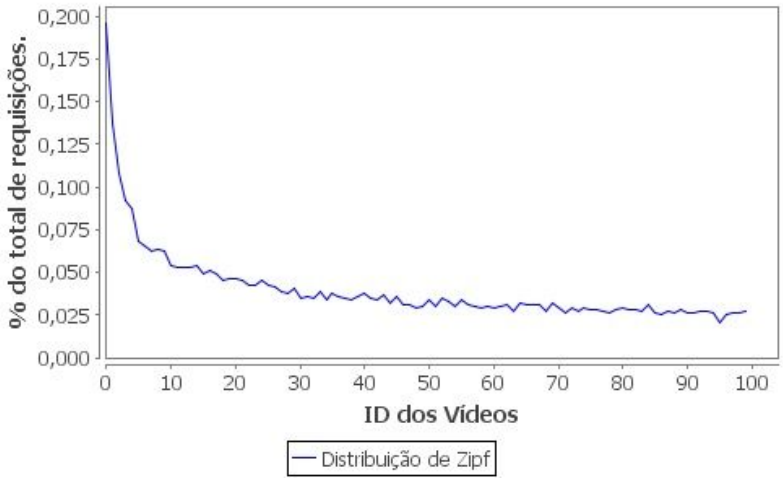
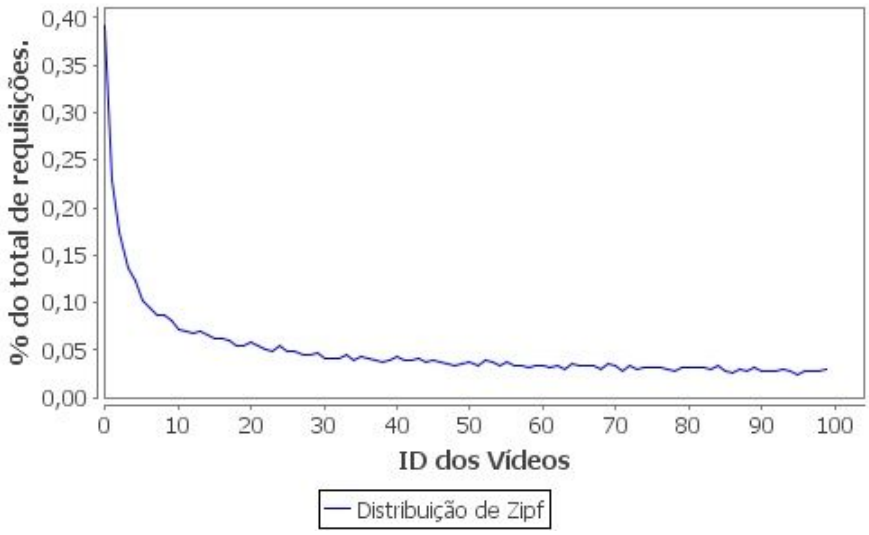


Figura B.2 - Distribuição de Zipf para os diferentes valores de θ usados neste trabalho. Nos Gráficos da figura, NV = 100 e o número total de requisições recebidas pelo *proxy* foi 8000 (cont.)

(e) $\theta = 0{,}1$



Fonte: do autor (2015).

APÊNDICE C < EXTRATO DE CÓDIGO FONTE: USO DO IPCM PARA COLETA DE DADOS DE DESEMPENHO>

```
//########## Início da inicialização dos registradores dedicados ##########

SystemCounterState before_sstate, after_sstate;
PCM * m = PCM::getInstance();
PCM::ErrorCode status = m->program();

//########## Fim da inicialização dos registradores dedicados ##########
    ...
while (round ≥ minRound && round ≤ maxRound){
    before_sstate = getSystemCounterState();
    ...
    //execução da rodada de serviço
    ...
    after_sstate = getSystemCounterState();
    total_cycles[round] = getCycles(before_sstate,after_sstate);
    L2_hits[round] = getL2CacheHits(before_sstate,after_sstate);
    L2_misses[round] = getL2CacheMisses(before_sstate,after_sstate);
    L2_lost[round] = getCyclesLostDueL2CacheMisses(before_sstate,after_sstate);
    L3_hits[round] = getL3CacheHits(before_sstate,after_sstate);
    L3_misses[round] = getL3CacheMisses(before_sstate,after_sstate);
    L3_lost[round] = getCyclesLostDueL3CacheMisses(before_sstate,after_sstate);
    m->resetPMU();
}
...
m->cleanup();
```

APÊNDICE D < SCRIPT PARA INSTALAÇÃO DO SIMPLESCALAR COM *TARGET* PISA>

```
#!/bin/bash

sudo apt-get update
sudo apt-get install flex-old bison binutils

mkdir /Documents/simplescalar;
cp simplescalar.tgz /Documents/simplescalar;
cd /Documents/simplescalar;
sudo tar xvfz simplescalar.tgz;

export HOST=i686-unknown-linux;
export TARGET=sslittle-na-sstrix;
export IDIR=/opt/simplescalar;
export CC=gcc-4.3;

#Simpletools

tar xvfz simpletools-2v0.tgz;
rm -rf gcc-2.6.3;
sudo mkdir -p - -mode=+xw /opt/simplescalar;
mv f2c-1994.09.27/ glibc-1.09/ ssbig-na-sstrix/ sslittle-na-sstrix/ /opt/simplescalar/;

#SimpleUtils

cd /Documents/simplescalar;
tar xvfz simpleutils-990811.tar.gz;
cd simpleutils-990811;
echo “ ";

echo “Edit /Documents/simplescalar/simpleutils-990811/ld/ldlex.l and replace
all instances of yy_current_buffer with YY_CURRENT_BUFFER";
```

```
echo “ ";
sudo gedit /Documents/simplescalar/simpleutils-990811/ld/ldlex.l;

echo "./configure - -host=$HOST - -target=$TARGET - -with-gnu-as -with-gnu-ld -
-prefix=$IDIR";
./configure - -host=$HOST - -target=$TARGET - -with-gnu-as - -with-gnu-ld - -
prefix=$IDIR;
make CC=gcc-4.3;
make install CC=gcc-4.3;

echo "./configure - -host=$HOST - -target=$TARGET - -with-gnu-as -with-gnu-ld -
-prefix=$IDIR";
./configure - -host=$HOST - -target=$TARGET - -with-gnu-as - -with-gnu-ld - -
prefix=$IDIR;
make CC=gcc-4.3;
make install CC=gcc-4.3;

#SimpleSim
cd /Documents/simplescalar;
tar xvfz simplesim-3v0d.tgz;
cp dlite.c simplesim-3.0;
cd simplesim-3.0;
make config-pisa CC=gcc-4.3;
make CC=gcc-4.3;
cd /Documents/simplescalar;
mv simplesim-3.0 /opt/simplescalar;

#CrossCompiler
cd /Documents/simplescalar;
tar xvfz gcc-2.7.2.3.ss.tar.gz;
cd gcc-2.7.2.3;
export PATH=$PATH:$IDIR/sslittle-na-sstrix/bin;
echo "./configure - -host=$HOST - -target=$TARGET - -with-gnu-as - -with-gnu-ld
- -prefix=$IDIR";
```

```
./configure - -host=$HOST - -target=$TARGET - -with-gnu-as - -with-gnu-ld -
- prefix=$IDIR;
chmod -R +w .;
make LANGUAGES="c c++"CFLAGS=-O CC=gcc-4.3;
echo “";
echo "Edit /Documents/simplescalar/gcc-2.7.2.3/insn-output.c : add a \ (backslash) at
the end of lines 675, 750, and 823";
echo “";
sudo gedit /Documents/simplescalar/gcc-2.7.2.3/insn-output.c;

make LANGUAGES="c c++"CFLAGS=-O CC=gcc-4.3;

echo “";
echo "Edit /Documents/simplescalar/gcc-2.7.2.3/Makefile : Add -I/usr/include to the
end of line 130";
echo “";
sudo gedit /Documents/simplescalar/gcc-2.7.2.3/Makefile;

echo “";
echo "Edit /Documents/simplescalar/gcc-2.7.2.3/obstack.h : at line 341 and change";
echo "*((void **)__o>next_free)++=((void *)datum);";
echo "with";
echo "*((void **)__o>next_free++)=((void *)datum);";
echo “";
sudo gedit /Documents/simplescalar/gcc-2.7.2.3/obstack.h;

echo “";
echo "Edit /Documents/simplescalar/gcc-2.7.2.3/protoize.c : at line 60 replace
#include <varargs.h> with #include <stdarg.h>";
sudo gedit /Documents/simplescalar/gcc-2.7.2.3/protoize.c;
echo “";

sudo cp /Documents/simplescalar/gcc-2.7.2.3/patched/sys/cdefs.h /opt/simplescalar
/sslittle-na-sstrix/include/sys/cdefs.h;
```

```
sudo cp /opt/simplescalar/sslittle-na-sstrix/lib/libc.a /opt/simplescalar/lib/;
sudo cp /opt/simplescalar/sslittle-na-sstrix/lib/crt0.o /opt/simplescalar/lib/;

make LANGUAGES="c c++"CFLAGS=-O CC=gcc-4.3;

echo “";
echo "Edit /Documents/simplescalar/gcc-2.7.2.3/cxxmain.c : remove lines 2978 and
2979";
sudo gedit /Documents/simplescalar/gcc-2.7.2.3/cxxmain.c;
echo “";

make LANGUAGES="c c++"CFLAGS=-O CC=gcc-4.3;
make install LANGUAGES="c c++"CFLAGS=-O CC=gcc-4.3;
```

APÊNDICE E < ARQUIVO .CFG PARA CONFIGURAÇÃO DO *TARGET* PISA>

```
#———————————————— cache configurations ————————————————

-cache:dl1 dl1:128:64:4:l
-cache:il1 il1:128:64:4:l
-cache:dl2 dl2:512:64:8:l
-cache:il2 dl2
-cache:dl3 dl3:8192:64:16:l
-cache:il3 dl3
-cache:dl1lat 4
-cache:il1lat 4
-cache:d12lat 10
-cache:il2lat 10
-cache:d13lat 44
-cache:il3lat 44

#———————————————— RAM configurations ————————————————-

-mem:lat 120 1
-mem:width 64

#————————————————- branch predictor configurations ————————————————-

-bpred:bimod 2048
```

APÊNDICE F < SCRIPT PARA CONFIGURAÇÃO E EXECUÇÃO DO SOFTWARE DO *PROXY*>

```
#!/bin/bash
echo "Script for variation of the Zipf coeficient";

curdir=$(pwd)

TR=5; # Transmission Rate
FPS=30; # Frames Per Second
NC=8000; # Number of Clients
NV=100; # Number of Videos
SM=3000; # Size of Memory
SV=4500; # Size of Video
SB=1; # Server Bandwidth
FI=0; # Frame Interval
TLPC=7700; # Time Limit fo aleatory Placement of Clients
DSCC=7200; # Demand Space of CC "equal to the number of blocks in the video"
SE=NUMBLOCKS; # Substitution Strategy

dadospath=$curdir/DadosFive
mkdir $dadospath

declare -a tlpcArray=(’4100’ ’5940’ ’7780’ ’9620’ ’11460’ ’13300’ ’15140’)
declare -a svArray=(’2250’ ’3400’ ’4550’ ’5700’ ’6850’ ’8000’ ’9150’)
declare -a svInblocksArray=(’3600’ ’5440’ ’7280’ ’9120’ ’10960’ ’12800’ ’14640’)

i=0;
for((k=0;k<=10;k++))
do
    if [ "$k"== "0"]; then
        inputfile="Inputfiles/config_IBA_3s_Zipf_0.0.txt"
    fi
    if [ "$k"== "1"]; then
```

```
    inputfile="Inputfiles/config_IBA_3s_Zipf_1.0.txt"
fi
if [ "$k"== "2"]; then
    inputfile="Inputfiles/config_IBA_3s_Zipf_2.0.txt"
fi
if [ "$k"== "3"]; then
    inputfile="Inputfiles/config_IBA_3s_Zipf_3.0.txt"
fi
if [ "$k"== "4"]; then
    inputfile="Inputfiles/config_IBA_3s_Zipf_4.0.txt"
fi
if [ "$k"== "5"]; then
    inputfile="Inputfiles/config_IBA_3s_Zipf_5.0.txt"
fi
if [ "$k"== "6"]; then
    inputfile="Inputfiles/config_IBA_3s_Zipf_6.0.txt"
fi
if [ "$k"== "7"]; then
    inputfile="Inputfiles/config_IBA_3s_Zipf_7.0.txt"
fi
if [ "$k"== "8"]; then
    inputfile="Inputfiles/config_IBA_3s_Zipf_8.0.txt"
fi
if [ "$k"== "9"]; then
    inputfile="Inputfiles/config_IBA_3s_Zipf_9.0.txt"
fi
if [ "$k"== "10"]; then
    inputfile="Inputfiles/config_IBA_3s_Zipf_10.0.txt"
fi

TLPC=${tlpcArray[1]};
SV=${svArray[1]};
DSCC=${svInblocksArray[1]};
```

```
######## BEGIN OF THE LOCAL CONFIGURATIONS FOR CARTE ########
echo "CARTE "$k"started!mail -s "CARTE "$k"started!"bsnpampa@gmail.com
echo ; echo "######## CARTE "$k"started! ########";
for((DS=10; DS<=200; DS=DS+10))
do
    echo ; echo "Running CARTE "$k"with DS = "$DS;
    cd \$curdir
    make clean » /dev/null;
    make functional_PREVUSERS demandSpace=$DS mode=-DGLOBAL»> /dev/null;
    echo ./main \$inputfile \$TR \$FPS \$NC \$NV \$SM \$SV \$SB \ $TLPC \$FI
    \$TLPC;
    ./main \$inputfile \$TR \$FPS \$NC \$NV \$SM \$SV \$SB \$TLPC \$FI \$TLPC »
    /dev/null;
    for filename in *.txt;
    do
      iformated=$i
      len=$(expr length \$iformated)
      while [ $len -lt 5 ]
      do
          iformated="0"$iformated
          len=$(expr length \$iformated)
      done
      newname=$iformated"_CARTE_"$filename
      mv \$filename \$newname;
      echo \$newname;
      i=$(expr \$i + 1)
    done;
    mv *.txt \$dadospath
done # exit

######## END OF THE LOCAL CONFIGURATIONS FOR CARTE ########

echo "CCVC "$k"started!mail -s "CCVC "$k"started!"bsnpampa@gmail.com
echo ; echo "############ CCVC "$k"started! ############"; echo ;
```

```
cd \$curdir
make clean
make functional_NUMBLOCKS mode=-DLOCAL»> /dev/null;
echo ./main \$inputfile \$TR \$FPS \$NC \$NV \$SM \$SV \$SB \$TLPC \$FI \$TLPC;
./main $inputfile $TR $FPS $NC $NV $SM $SV $SB $TLPC $FI $TLPC » /dev/null;
for filename in *.txt;
do
    iformated=$i
    len=$(expr length $iformated)
    while [ $len -lt 5 ]
    do
        iformated="0"$iformated
        len=$(expr length $iformated)
    done
    newname=$iformated"_CCVC_"$filename
    mv $filename $newname;
    echo $newname;
    i=$(expr $i + 1)
done;
mv *.txt $dadospath

echo "CC "$k"started!mail -s "CC "$k"started!"bsnpampa@gmail.com
echo ; echo "############ CC "$k"started! ############"; echo ;
cd $curdir
make clean
make functional_PREVUSERS demandSpace=$DSCC mode=-DGLOBAL»> /dev/null;
echo ./main $inputfile $TR $FPS $NC $NV $SM $SV $SB $TLPC $FI $TLPC;
./main $inputfile $TR $FPS $NC $NV $SM $SV $SB $TLPC $FI $TLPC » /dev/null;
for filename in *.txt;
do
    iformated=$i
    len=$(expr length $iformated)
    while [ $len -lt 5 ]
    do
```

```
            iformated="0"$iformated
            len=$(expr length $iformated)
        done
        newname=$iformated"_CC_"$filename
        mv $filename $newname;
        echo $newname;
        i=$(expr $i + 1)
        done;
        mv *.txt $dadospath
    done
echo "Simulation finalizedmail -s "Simulation finalized" bsnpampa@gmail.com
```

APÊNDICE G < DESCRIÇÃO DO PACOTE DE SOFTWARE E TUTORIAL PARA CONFIGURAÇÃO EM MODO GRÁFICO COM O CODE::BLOCKS>

Para este trabalho, foram escritas 16.386 linhas de código distribuídas em dez módulos de software, conforme descrito na tabela G.1. Os experimentos realizados no escopo do presente trabalho utilizaram a versão v12.11 IDE de projeto Code::Blocks, juntamente com o *plug-in* cbp2make v1.47 (gerador de makefile), utilizado para realizar a síntese do Makefile, necessário para compilação do código fonte do *proxy*. A seção G.1 deste tutorial apresenta o procedimento utilizado para instalação Code::Blocks e do *plug-in* cpb2make. A seção G.2 descreve como o cp2make é utilizado para configurar os principais parâmetros de funcionamento do SIMPRO.

Tabela G.1 - Descrição do código desenvolvido neste trabalho

Software	**Descrição**	**Núm. de linhas de código**
SIV Gen	Gerador de arquivo SIV para cenários com e sem abandono de seção.	448
SIMPRO (Cabeçalhos .h)	Arquivos de cabeçalho (*headers*) contendo definições gerais para o programa.	1544
SIMPRO (Programa .c)	Rotinas do simulador SIMPRO.	3114
CARTE (Programa .c)	Algoritmo de cacheamento proposto neste trabalho.	1435
CCVC (Programa .c)	Algoritmo de referência para uso em análises comparativas de desempenho.	1222
CC (Programa .c)	Algoritmo de referência para uso em análises comparativas de desempenho.	1058
Analyzer (java)	Analisador dos traços (*traces* de execução produzidos pelo SIMPRO.).	2434
Scripts (.sh)	Scripts para configuração dos makefiles para compilação do SIMPRO e dos algoritmos de cacheamento.	1830
Impl. em rede (cabeçalhos .h)	Arquivos de cabeçalho (*headers*) contendo definições gerais para a implementação em rede.	460
Impl. em rede (Programa .c)	Programa da implementação em rede.	2841

Fonte: do autor (2015).

G.1 Code::Blocks e cbp2make - Instalação e Configuração

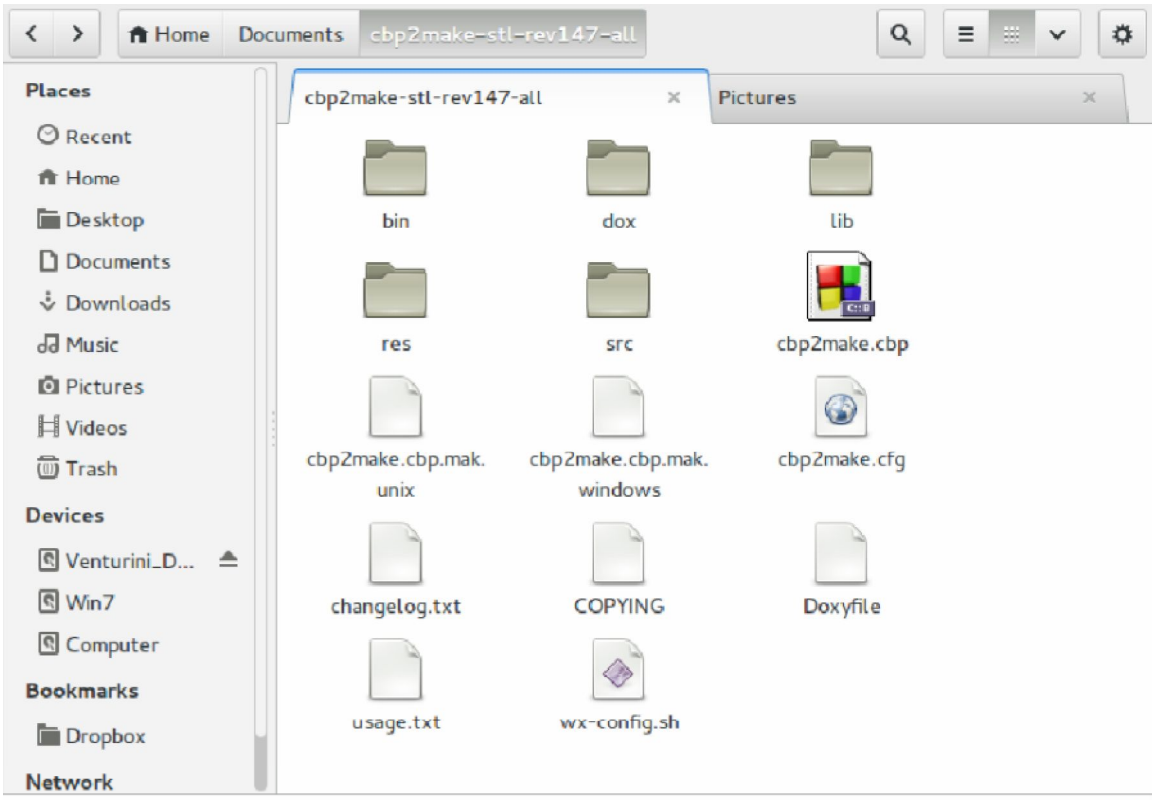
O download do Code::Blocks para plataformas Windows, Mac OS e Linux pode ser feito livremente em (CODE::BLOCKS, 2015). Sendo uma ferramenta *open source*, sua instalação pode ser feita em diferentes modalidades:

- *Código binários*: o usuário baixa um instalador (windows) ou script de shell (linux) que configura o IDE para a plataforma alvo desejada.
- *Compilação do código fonte*: o usuário baixa os códigos fontes e um makefile para compilação do software para a plataforma alvo.
- *Apache Subversion (SVN):* o usuário usa o sistema de subversão para escolher entre as diferentes versões do IDE disponibilizadas através de um repositório.

Uma vez que o Code::Blocks tenha sido instalado na plataforma alvo, a instalação do cbp2make é feita através dos seguintes passos:

1. Executar o download do aplicativo, em formato compactado (.tar.7z), em (MOLLER, 2015).
2. Descompactar o arquivo compactado em diretório desejado, alocado no espaço de armazenamento do usuário. A pasta selecionada conterá os arquivos apresentados na figura G.1.

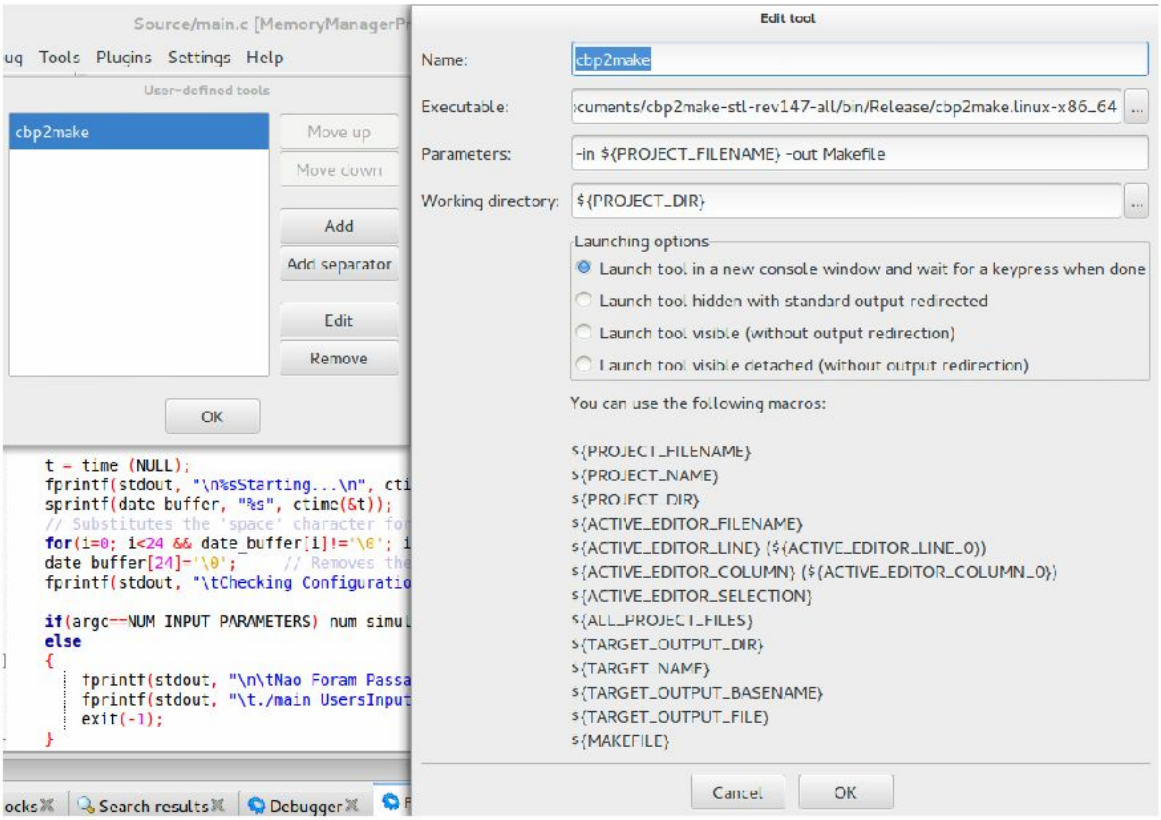
Figura G.1 - Pasta com arquivos de instalação do Code::Blocks



Fonte: do autor (2015).

3. Executar o IDE do Code::Blocks e selecionar o menu *Tools -> Configure Tools*. A janela sob o label *User-defined* tools irá aparecer. Clique em *Add* para adicionar um novo *plug-in*. A janela Edit tool será exibida, conforme mostra a figura G.2.

Figura G.2 - Interface do Code:Blocks para instalação e configuração de *plug-ins*: configuração do *plug-in* cbp2make

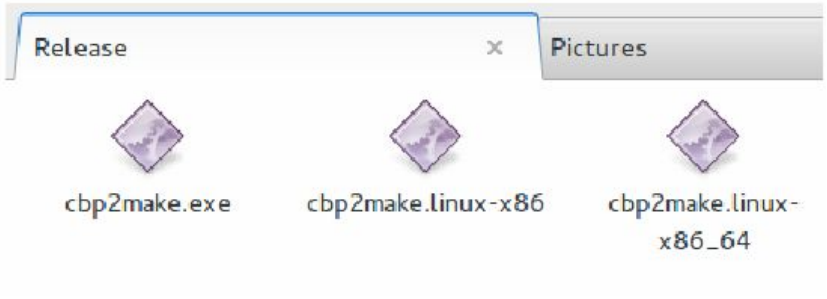


Fonte: do autor (2015).

4. Inserir um nome para a ferramenta (cbp2make, por exemplo). Na caixa *Executable*, selecione o código binário da ferramenta, usando, para isso, o localizador dentro do sistema de arquivos. Os binários do cbp2make estão alocados dentro da pasta alvo da descompactação executada no passo 2, mais precisamente dentro das subpastas *bin -> Release*. Conforme mostra a figura G.3 , a pasta *Release* contém os arquivos executáveis da ferramenta compilados para três diferentes plataformas (windows e linux, 32 e 64 bits). O executável deve ser selecionado de acordo com a plataforma alvo. Na caixa *Parameters* digitar o seguinte comando: *-in ${PROJECT_FILENAME} -out Makefile*. Ele fará com que todas as modificações realizadas sobre o projeto ativo sejam transportadas para um Makefile descrevendo a sequência de passos para compilação dos *targets* configurados. Por fim, na caixa *Working Directory*, digitar o

comando *${PROJECT_DIR}* para informar ao cbp2make o endereço do projeto correntemente aberto.

Figura G.3 - Pasta release com códigos binários do cbp2make para diferentes sistemas operacionais



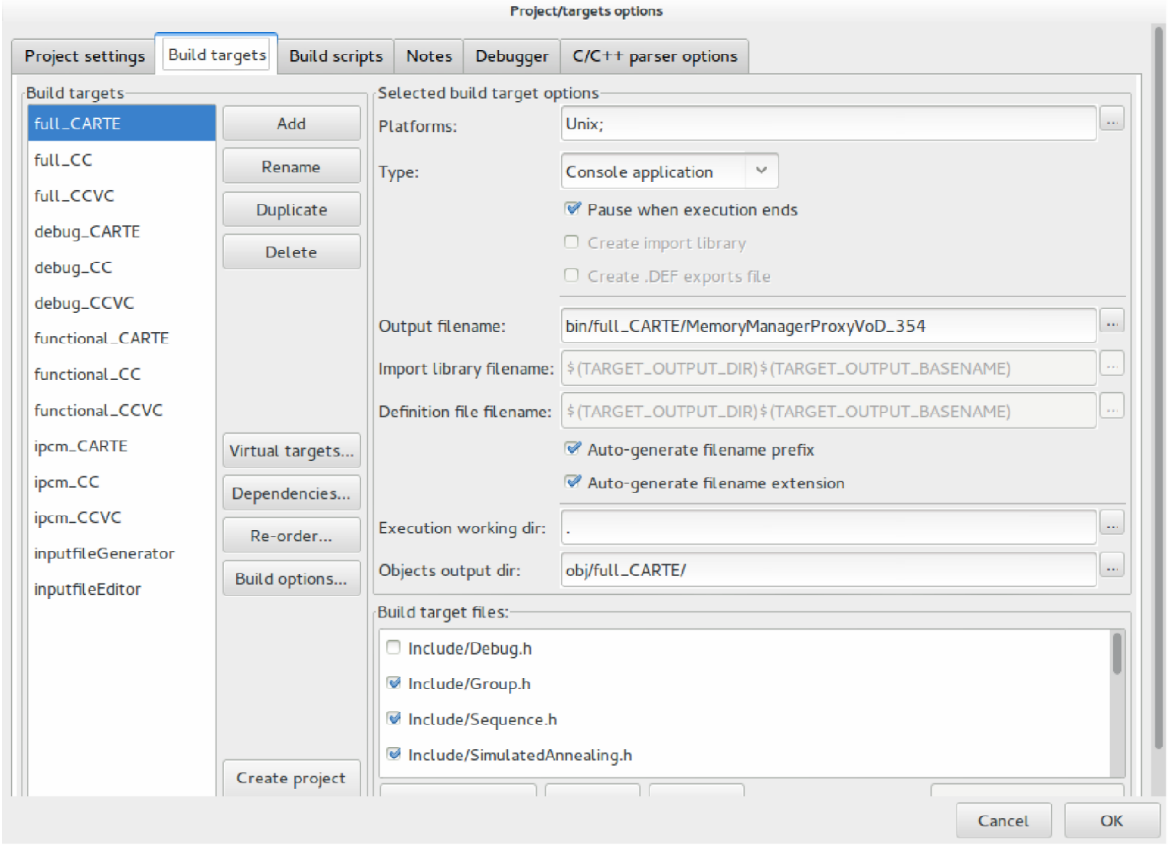
Fonte: do autor (2015).

G.2 Configuração do SIMPRO usando o Code::Blocks

A configuração do SIMPRO através do Code::Blocks ocorre através dos seguintes passos:

1. Conforme mostra a figura G.4, através do menu *Project -> Properties -> Build targets* são apresentadas as características gerais do projeto como os *targets* implementados e os arquivos-fonte utilizados. Para adicionar novos *targets* pode-se usar o botão *Add* ou usar o botão *Duplicate* para duplicar um *target* existente e, em seguida, modificar as especificações. Estas opções são úteis quando o projetista deseja expandir as funcionalidades do SIMPRO (como, por exemplo, incorporando um novo algoritmo de cacheamento ou novas ferramentas para coleta de dados).
2. Através do menu *Project -> Build Options* (ou usando diretamente o botão *Build Options* a partir da janela apresentada na figura G.4), é possível conhecer e modificar as configurações gerais de compilação dos *targets* disponíveis. Através da guia *#defines* (G.5) é possível acessar todas as definições de programas, flags de compilação e etapas de pré/pós compilação (*building*) gerais para cada um dos principais *targets* implementados, que são:
   - *FULL*: gera o Makefile que sintetiza uma versão do SIMPRO para cada um dos três modos de simulação suportados atualmente para coleta de dados de desempenho (modo funcional, modo de tempo real e modo de tempo discreto).
   - *FUNCTIONAL*: gera o Makefile que sintetiza uma versão do SIMPRO especificamente para o modo funcional.

Figura G.4 - Visão geral sobre os *targets* do SIMPRO implementados sobre o Code::Blocks



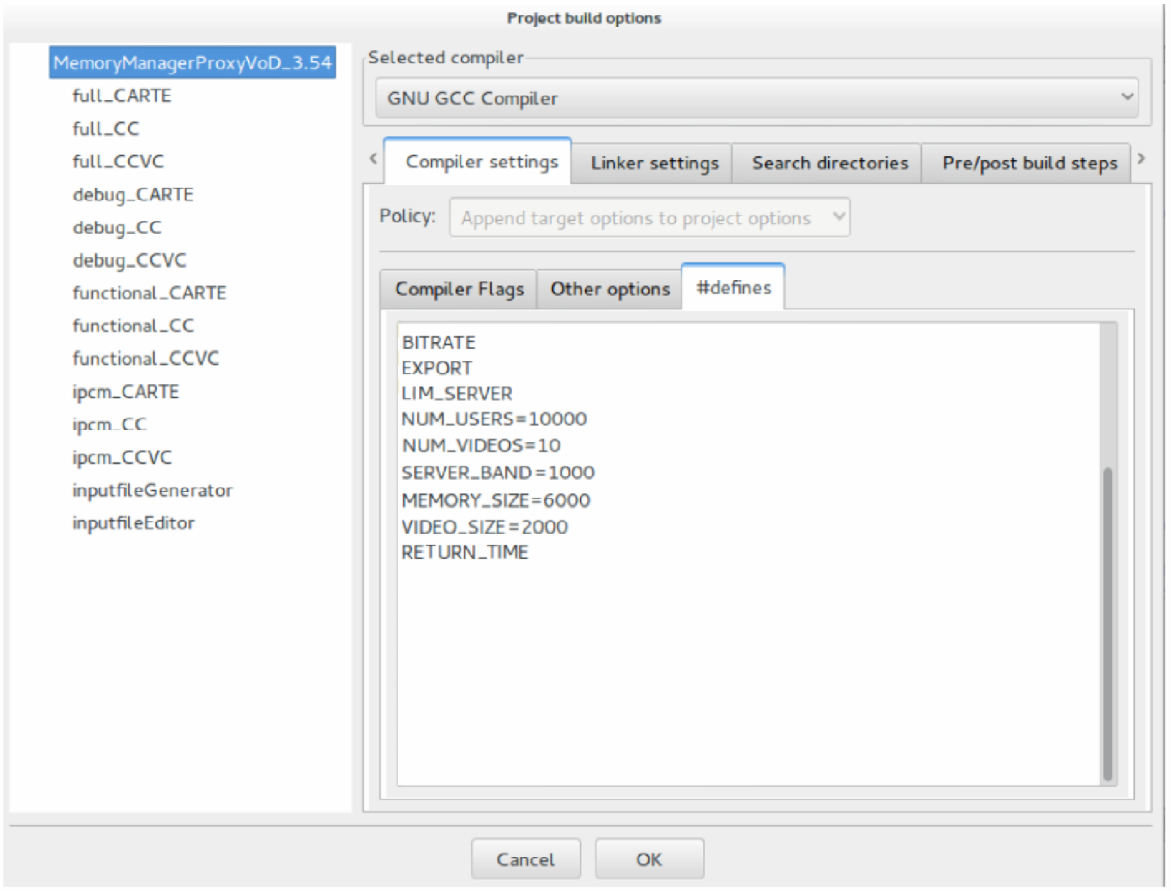
Fonte: do autor (2015).

- *IPCM*: gera o Makefile que sintetiza uma versão do SIMPRO especificamente para o modo de tempo real.
- *DEBUG*: gera o Makefile que sintetiza uma versão do SIMPRO especificamente para o modo de depuração. O modo de depuração é destinado exclusivamente para esta finalidade e permite que o usuário exporte os dados contidos nas estruturas do *proxy* para análise. Os dados exportados podem ser utilizados como ponto de partida para futuras simulações.

As definições de programa (guia *#defines* da G.5) são usadas para configuração das estruturas internas do *proxy*, em tempo de compilação, e estão descritas na tabela G.2.

Para que não seja preciso inserir e remover as definições cada vez que se deseja ativar/desativar uma funcionalidade, é possível adicionar um "_"(underscore) em frente à variável definida (por exemplo: _LIM_SERVER desativa o limite de banda para o canal de comunicação do *proxy* com o servidor principal).

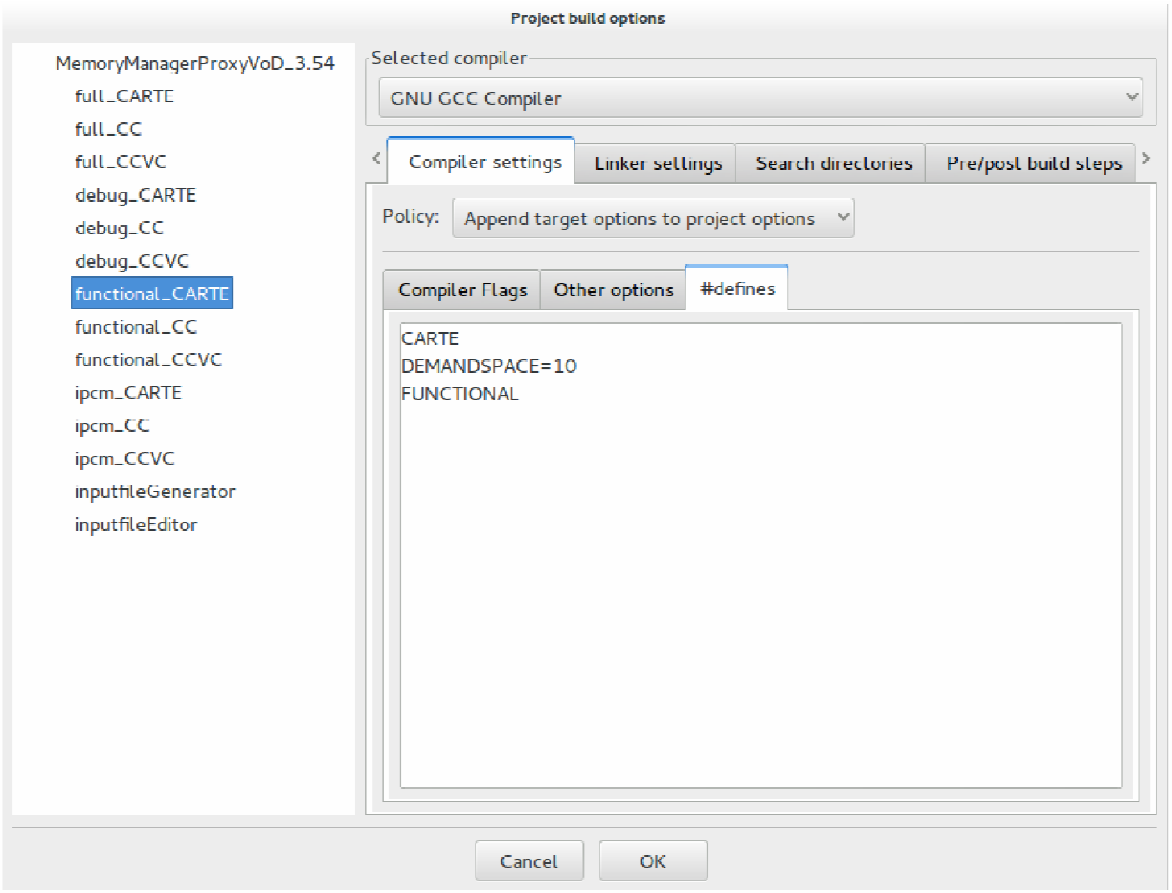
Figura G.5 - Interface do Code::Blocks para configuração das definições gerais de programa para os *targets* do SIMPRO



Fonte: do autor (2015).

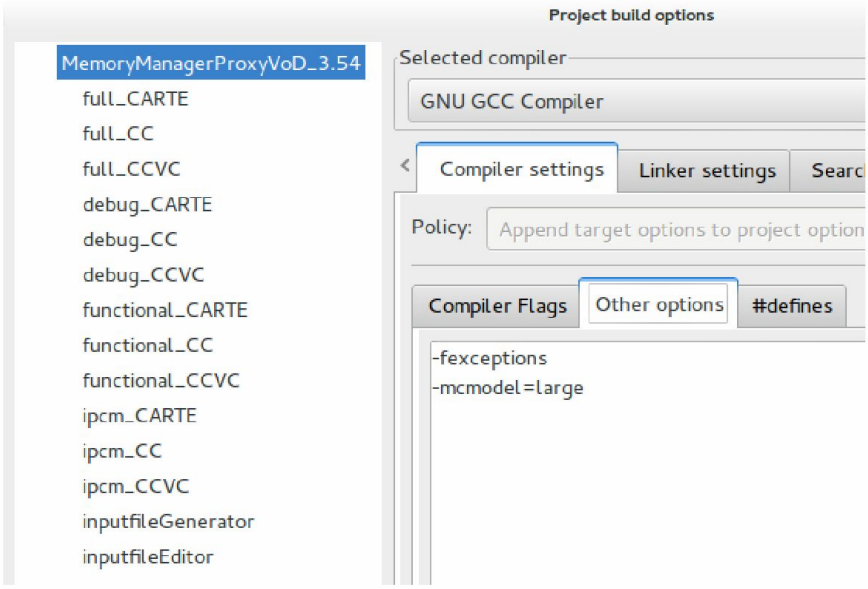
3. Ao selecionar um *target* específico (à esquerda na figura G.5), é possível acessar as definições de programa exclusivas do *target* escolhido. A figura G.6 ilustra as definições da guia *#defines* para o *target funcional_CARTE*, onde a variável DEMANDSPACE informa o tamanho da janela de trabalho do algoritmo CARTE.
4. Conforme mostra a figura G.7, seguindo o menu *Project -> Build Options -> Compiler Settings -> Other Options*, são acessadas algumas definições gerais para que o compilador gcc possa sintetizar o SIMPRO adequadamente para cada plataforma de execução. A variável *-mcmodel*, presente na guia *Other options* da figura G.7, estabelece a forma como os dados de um aplicativo irão ocupar a memória da máquina hospedeira. Os valores possíveis para esta variável são:

Figura G.6 - Exemplificação da interface para entrada das definições de programa específicas para o *target funcional*_CARTE do SIMPRO



Fonte: do autor (2015).

Figura G.7 - Opções gerais de configuração do compilador para síntese do SIMPRO



Fonte: do autor (2015).

Tabela G.2 - Descrição do código desenvolvido neste trabalho

Software	Descrição
BITRATE	Taxa de codificação do vídeo (também usada para configurar o tamanho do bloco de vídeo) em Mbps.
EXPORT	Ativa a captação e exportação dos dados de desempenho para cada rodada de serviço e em vez de para toda a simulação.
LIM_SERVER	Ativa o limite de banda para o canal de comunicação do *proxy* com o servidor principal.
NUM_USERS	Define o número máximo de requisições a serem recebi das pelo *proxy*. Se o número de simulações exceder à MAX_NUM_USERS, o SIMPRO suspende a entrada de clientes.
NUM_VIDEOS	Define o número máximo de vídeos disponíveis para acesso.
SERVER_BAND	Define a largura de banda máxima (em Mbps) para o canal de comunicação do *proxy* com o servidor principal. Para converter a largura de banda de fluxos por segundo para Mbps (Mega bits por segundo), é necessário multiplicar o número de fluxos pela taxa de transmissão dos vídeos.
MEMORY_SIZE	Define o tamanho (em MB) da memória (RAM) de vídeo do *proxy*. Para converter o tamanho da memória de vídeo de GB (giga bytes) para MB (Mega Bytes), é necessário multiplicar o tamanho dado em GB por 1024.
VIDEO_SIZE	Define o tamanho para cada um dos vídeos do acervo (em MB). Para converter a duração do vídeo de minutos para MB, é necessário primeiramente multiplicar o tamanho do vídeo (em minutos) por 60, obtendo assim a duração equivalente em segundos. Em seguida, a duração (em segundos) deve ser multiplicada pela taxa de transmissão dos vídeos para obtenção do tamanho do vídeo em MB.
RETURN_TIME	Define o tempo (em segundos ou rodadas de serviço) de retorno do servidor principal para o *proxy*.

Fonte: do autor (2015).

- *small*: compila o *target* para memórias RAM com capacidade máxima para alocação de 2 GB de dados e código. Todos os ponteiros são de 64 bits e os dados podem ser alocados tanto de forma estática como dinâmica. Esta opção é usada quando o *proxy* executa sobre arquiteturas com baixa capacidade de armazenamento.
- *kernel*: usado para compilar *target* para módulos do kernel do sistema operacional hospedeiro. Esta opção é usada em circunstâncias onde existe necessidade de instalação de novas rotinas no kernel ou para realizar a adaptação de serviços já existentes.

- *medium*: separa os espaços para alocação de pequenas e grandes variáveis de programa. Não há uso previsto desta opção para compilação de *targets* para o SIMPRO.
- *large*: o compilador não impõe qualquer restrição com relação aos endereços e capacidade dos segmentos de memória para alocação do programa. Esta opção é utilizada para viabilizar a compilação e simulação do *proxy* quando o tamanho da memória de vídeo excede o limite de 4 GB.

REFERÊNCIAS

CODE::BLOCKS. **The open source, cross platform, free C, C++ and Fortran IDE**. Boston: Free Software Foundation, 2014. Disponível em: <http://www.codeblocks.org/>. Acesso em: 20 jun. 2013.

MOLLER, Steffen. **Cbp2make: Makefile Generation Tool for Code::Blocks IDE**. New York: Mirai Computing, 2015. Disponível em: <http://sourceforge.net/p/cbp2make/wiki/Home/>. Acesso em: 20 jun. 2013.