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

RODRIGO BRANDÃO MANSILHA

Paralelizando Unidades de Cache Hierárquicas para Roteadores ICN

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação

Prof. Ph.D. Marinho Pilla Barcellos
Orientador

Porto Alegre, 2017

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Mansilha, Rodrigo Brandão

Paralelizando Unidades de Cache Hierárquicas para Roteadores ICN / Rodrigo Brandão Mansilha. – Porto Alegre: PPGC da UFRGS, 2017.

88 f.: il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2017. Orientador: Marinho Pilla Barcellos.

1. Redes orientadas a conteúdo. 2. Caching. 3. Memória hierárquica. 4. Avaliação. 5. Emulação. I. Barcellos, Marinho Pilla. II. Título.

“Do not pray for tasks equal to your powers;
pray for powers equal to your tasks.
Then the doing of your work shall be no miracle,
but you shall be the miracle.”
— PHILLIPS BROOKS

AGRADECIMENTOS / ACKNOWLEDGMENTS

Eu agradeço a Deus por me proporcionar o desafio do doutorado e por apresentar pessoas que me ajudaram a superar esse desafio.

Eu agradeço a toda minha família. Em especial, agradeço aos meus pais, Edison Mansilha e Venina Mansilha, pelo amor e exemplo de superação. Eu também agradeço aos meus irmãos, Raquel Mansilha e Ricardo Mansilha e aos seus respectivos pares, Juliano Oliveira e Marília Rodrigues, pela força, compreensão e apoio durante esta caminhada. Eu dedico essa conquista a todos vocês!

Eu agradeço a todos os meus professores. Em especial, agradeço ao meu orientador, Prof. Marinho Barcellos. Sou muito grato por ele ter me aceito como aluno na graduação, no mestrado e no doutorado. Ao longo de dez anos, nós passamos por muitas coisas, de forma que seria inviável explicar toda a minha gratidão neste espaço. Eu faço questão de resgistrar, porém, o meu reconhecimento aos seus ensinamentos, que me formaram um professor pesquisador; e aos seus conselhos e exemplos, que iluminaram e sempre iluminarão o meu crescimento como ser humano. Também agradeço ao Prof. Luciano Gaspary, que contribuiu bastante para a minha formação como pesquisador, desde a época de mestrado.

I am also very grateful to Prof. Dario Rossi for hosting me as a visiting student at Télécom ParisTech. He was always readily available to discuss my research and provide me with important guidance to solve the issues I had, even after my departure from Paris. I am also thankful to Prof. Emilio Leonardi (Politecnico di Torino, Italy) for supporting my research.

Não poderia deixar de agradecer aos meus colegas e amigos. Sou grato aos "colegas de ICN", Rodolfo Antunes e Matheus Lehmann, por todas as discussões técnico-filosóficas (e, porque não, humorísticas), em particular sobre a nossa área de pesquisa, que desbravamos conjuntamente. Outros agradecimentos especiais são para os amigos Bruno Dalmazo, Daniel Marcon, Lucas Müller, Miguel Neves, Ricardo dos Santos, Rodrigo Oliveira, Oscar Caicedo, Pedro Marcos, Weverton Cordeiro e, enfim, todos os colegas do Grupo de Redes da UFRGS pelas discussões científicas e pelos momentos de descontração.

I also thank my friends from Paris for all the technical discussions and amazing parties we had together. I am particularly thankful to Andrea Araldo, Danilo Cicalese, Diana Joumblatt, Diego Pacheco, Gong Yixi, Leonardo Linguaglossa, Lorenzo Saino, Martino Trevisan and Michele Tortelli. They made this journey much more enriching and pleasant.

Por último, mas não menos importante, agradeço à minha namorada, Camila Martini, pelo carinho e compreensão dedicados a mim na reta final desta jornada.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CCN	*Content Centric Network*
CCNx	*Content Centric Network Router* (protótipo de roteador CCN)
CDN	*Content Delivery Network*
CS	*Content Store*
CPU	*Central Processing Unit*
DRAM	*Dynamic Random Access Memory*
E/S	Entrada/Saída
FIB	*Forwarding Information Base*
HCS	*Hierarchical Content Store*
HDD	*Hard Disk Drive*
ICN	*Information-centric Network*
IRM	*Independent Reference Model*
L1	*Layer 1* (memória menor e mais rápida)
L2	*Layer 2* (memória maior e mais lenta)
LCE	*Leave-a-Copy-Everywhere*
LCP	*Leave-a-Copy-Probabilistically*
LCD	*Leave-a-Copy-Down*
LRU	*Least Recently Used*
FIFO	*First In First Out*
NDN	*Named Data Networking*
NFD	*Network Forwarder Daemon* (protótipo de roteador NDN)
NIC	*Network Interface Card*
NUMA	*Non-Uniform Memory Access*
PIT	*Pending Interest Table*
P2P	*Peer-to-Peer*
PASTA	*Poisson Arrivals See Time Averages*

REQM	Raiz do Erro Quadrático Médio
SSD	*Solid-state Drive*

LISTA DE SÍMBOLOS

α	Parâmetro da distribuição Zipf de popularidade de conteúdos
Λ	Taxa de chegada global de requisições seguindo um processo de Poisson
λ_n	Taxa de chegada de requisições para um determinado chunk n
τ_{L2}	Taxa de leitura da *Layer 2*
B	Tamanho[1] do *batch* de transferência entre as camadas de memória
C	Tamanho[1] de uma unidade de cache
$\|c\|$	Tamanho[1] de um *chunk*
D	Parâmetro de emulação de espera da L2
d	Tempo de acesso ($delay$)
$\mathbb{E}(x)$	Estimativa de uma variável x
F	Tamanho[1] da fatia de carga de trabalho
H	Parâmetro de emulação da entrada para função hash
HT	Parâmetro de sistema para *Hyper-Threading*
$\|L1\|$	Tamanho[1] da *Layer 1*
$\|L2\|$	Tamanho[1] da *Layer 2*
M	Parâmetro de emulação da alocação de memória para L2
N	Tamanho[1] do catálogo de conteúdos
O	Tamanho[1] da carga de trabalho
n	Índice de chunks do catálogo, $1 \leq n \leq \|N\|$
p_n	Probabilidade de um chunk n ser solicitado
P	Probabilidade de ocorrer acerto em cache
R	Grau de paralelismo
t	Unidade discreta de tempo
T_C	Tempo estimado para um conteúdo ser despejado da cache
W	Parâmetro de tipo da carga de trabalho

[1] Tamanhos são alternadamente expressos em termos de bits, bytes ou chunks, dependendo do contexto.

LISTA DE ALGORITMOS

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Um desafio fundamental em ICN (do inglês *Information-Centric Networking*) é desenvolver *Content Stores* (ou seja, unidades de cache) que satisfaçam três requisitos: espaço de armazenamento grande, velocidade de operação rápida e custo acessível. A chamada *Hierarchical Content Store* (HCS) é uma abordagem promissora para atender a esses requisitos. Ela explora a correlação temporal entre requisições para prever futuras solicitações. Por exemplo, assume-se que um usuário que solicita o primeiro minuto de um filme também solicitará o segundo minuto. Teoricamente, essa premissa permitiria transferir proativamente conteúdos de uma área de cache relativamente grande, mas lenta (*Layer 2* - L2), para uma área de cache mais rápida, porém menor (*Layer 1* - L1). A estrutura hierárquica tem potencial para incrementar o desempenho da CS em uma ordem de grandeza tanto em termos de vazão como de tamanho, mantendo o custo.

Contudo, o desenvolvimento de HCS apresenta diversos desafios práticos. É necessário acoplar as hierarquias de memória L2 e L1 considerando as suas taxas de transferência e tamanhos, que dependem tanto de aspectos de hardware (por exemplo, taxa de leitura da L2, uso de múltiplos SSD físicos em paralelo, velocidade de barramento, etc.), como de software (por exemplo, controlador do SSD, gerenciamento de memória, etc.).

Nesse contexto, esta tese apresenta duas contribuições principais. Primeiramente, é proposta uma arquitetura para superar os gargalos inerentes ao sistema através da paralelização de múltiplas HCS. Em resumo, o esquema proposto supera desafios inerentes à concorrência (especificamente, sincronismo) através do particionamento determinístico das requisições de conteúdos entre múltiplas *threads*. Em segundo lugar, é proposta uma metodologia para investigar o desenvolvimento de HCS explorando técnicas de emulação e modelagem analítica conjuntamente. A metodologia proposta apresenta vantagens em relação a metodologias baseadas em prototipação e simulação. A L2 é emulada para viabilizar a investigação de uma variedade de cenários de contorno (tanto em termos de hardware como de software) maior do que seria possível através de prototipação (considerando as tecnologias atuais). Além disso, a emulação emprega código real de um protótipo para os outros componentes do HCS (por exemplo L1, gerência das camadas e API) para fornecer resultados mais realistas do que seriam obtidos através de simulação.

Palavras-chave: Redes orientadas a conteúdo, caching, memória hierárquica, avaliação, emulação.

Parallelizing Hierarchical Content Stores for ICN Routers

ABSTRACT

A key challenge in *Information Centric Networking* (ICN) is to develop cache units (also called *Content Store* - CS) that meet three requirements: large storage space, fast operation, and affordable cost. The so-called HCS (*Hierarchical Content Store*) is a promising approach to satisfy these requirements jointly. It explores the correlation between content requests to predict future demands. Theoretically, this idea would enable proactively content transfers from a relatively large but slow cache area (Layer 2 - L2) to a faster but smaller cache area (Layer 1 - L1). Thereby, it would be possible to increase the throughput and size of CS in one order of magnitude, while keeping the cost.

However, the development of HCS introduces several practical challenges. HCS requires a careful coupling of L2 and L1 memory levels considering their transfer rates and sizes. This requirement depends on both hardware specifications (e.g., read rate L2, use of multiple physical SSD in parallel, bus speed, etc.), and software aspects (e.g., the SSD controller, memory management, etc.).

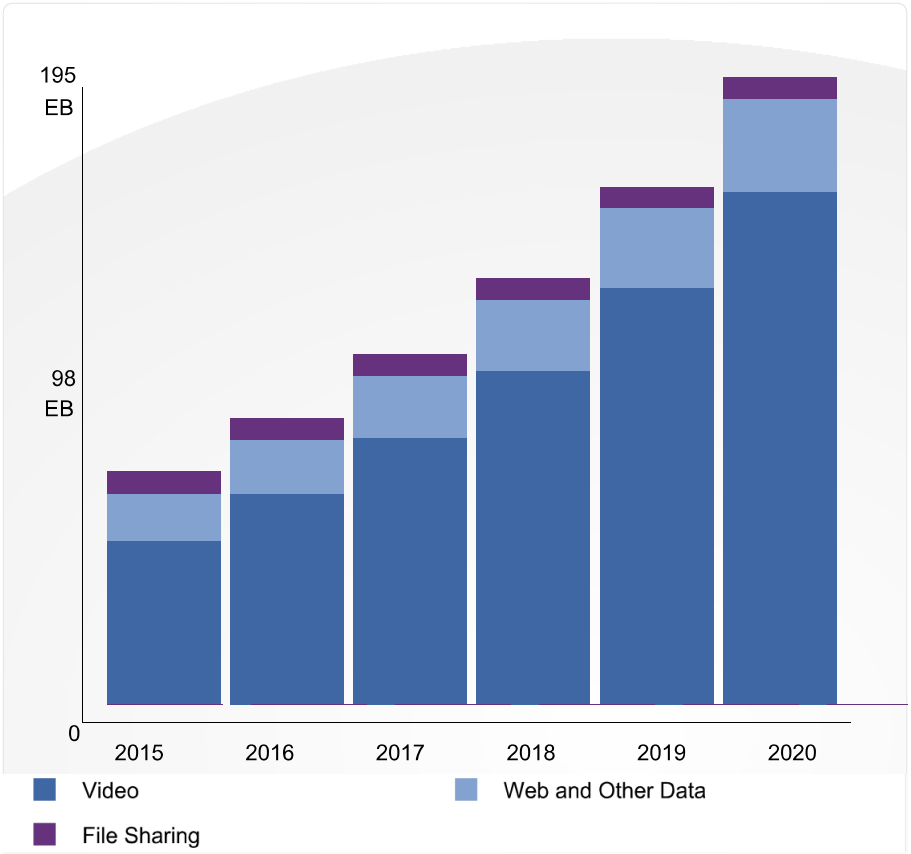
In this context, this thesis presents two main contributions. First, we propose an architecture for overcoming the HCS bottlenecks by parallelizing multiple HCS. In summary, the proposed scheme overcomes racing condition related challenges through deterministic partitioning of content requests among multiple threads. Second, we propose a methodology to investigate the development of HCS exploiting emulation techniques and analytical modeling jointly. The proposed methodology offers advantages over prototyping and simulation-based methods. We emulate the L2 to enable the investigation of a variety of boundary scenarios that are richer (regarding both hardware and software aspects) than would be possible through prototyping (considering current technologies). Moreover, the emulation employs real code from a prototype for the other components of the HCS (e.g., L1, layers management and API) to provide more realistic results than would be obtained through simulation.

Keywords: information centric networking, caching, hierarquical memory, evaluation, emulation.

1 INTRODUÇÃO

De acordo com o Cisco Visual Networking Index 2016 (CISCO, 2016a), o tráfego IP global crescerá quase três vezes até 2020, devendo alcançar 194 EB por mês. A maior parte do crescimento do tráfego é atribuída à distribuição de vídeo (por exemplo, o serviço Netflix), como ilustrado na Figura 1.1. A Cisco prevê que o tráfego de vídeo irá representar 82 % de todo o tráfego IP em 2020, o que constitui um crescimento significativo em comparação com os 70 % registrados em 2015 (CISCO, 2016a).

Figura 1.1: Projeção do tráfego na Internet



Fonte: CISCO (2016b)

Embora essas previsões devam ser interpretadas com cautela, elas são intuitivas considerando três observações. Primeiro, a disseminação de conteúdo (por exemplo, notícias, músicas, filmes), a exemplo dos meios de comunicação (por exemplo, telefonia), está convergindo para a Internet. Segundo, observa-se uma expansão do número de geradores

de conteúdo (vide o número de publicadores independentes no YouTube), que é potencializada pelos dispositivos móveis e a chamada *Internet das Coisas*. Terceiro, espera-se que a resolução dos conteúdos multimídia seja ampliada na medida em que telas de alta definição[1] tornem-se economicamente acessíveis, e a rede seja capaz de atender essa demanda.

Ainda que a arquitetura IP até agora tenha suportado bem a distribuição de conteúdo, uma mudança gradual para uma nova arquitetura projetada especialmente para esse fim poderia atender de uma forma muito melhor as expectativas da sociedade no futuro. Redes sobrepostas, tais como redes *peer-to-peer* (P2P) e *content delivery networks* (CDNs) foram instanciadas para acomodar a demanda por distribuição de conteúdo na arquitetura IP (PASSARELLA, 2012). No entanto, essas soluções dependem de um modelo fim-a-fim de comunicação na camada de aplicação da pilha TCP/IP, o que resulta em ineficiência (CAROFIGLIO et al., 2013).

Para atender eficientemente o tráfego IP em vigor e futuro, várias arquiteturas de rede foram propostas (KOPONEN et al., 2007; JACOBSON et al., 2009; KATSAROS; XYLOMENOS; POLYZOS, 2011; FOTIOU et al., 2012; DANNEWITZ et al., 2013), estabelecendo o paradigma chamado Redes Centradas em Informações (do inglês *Information-Centric Networking* - ICN). Em comparação geral com o paradigma da Internet atual, o ICN pretende dissociar os conteúdos de sua localização (ou da localização dos servidores que os armazenam) em nível de rede – e não no nível de aplicação. ICN localiza e roteia dados através de uma *nomenclatura de conteúdos padronizada*, em contraste com o endereçamento IP. A principal vantagem do ICN é fornecer suporte nativo para recuperação de conteúdo de forma escalável e eficiente, ao mesmo tempo que, entre outros benefícios, promete maiores níveis de segurança (ABDALLAH; HASSANEIN; ZULKERNINE, 2015), melhor suporte à mobilidade (TYSON et al., 2013), e novas oportunidades de negócio (AGYAPONG; SIRBU, 2012; TROSSEN; KOSTOPOULOS, 2012).

As arquiteturas ICN adotam caching em nível de rede para alcançar a eficiência prometida. O caching em nível de rede difere dos sistemas de caching atuais (por exemplo *Web proxies*, CDN), que rodam em nível de aplicação, em termos de *transparência* e *ubiquidade* (ZHANG; LI; LIN, 2013). Para cada uma dessas propriedades, existe uma relação direta entre os ganhos potenciais e os desafios envolvidos, que variam de acordo com o grau de adoção, conforme será explicado nos próximos parágrafos.

Em ICN, a rede de caching é dita transparente porque pode beneficiar todo e qualquer conteúdo. Caching transparente é possível em ICN graças ao seu esquema de nomenclatura padronizado. Em contraste, a nomenclatura de conteúdos da arquitetura atual é livremente determinada pelos protocolos executados na camada de aplicação. Por exemplo, sistemas de caching de Web proxies utilizam URL dos objetos como seus identificadores. Quando duas cópias exatas de um mesmo objeto são armazenadas em servidores distintos, elas são identificadas por URLs diferentes. Como resultado, o Web proxy irá tratar as duas cópias como objetos distintos. Uma ICN com um sistema de caching idealmente transparente (ou seja, no qual todos os conteúdos são nomeados seguindo uma mesma padronização, definida pela rede) trataria as duas cópias como um mesmo objeto, resultando em maior eficiência do sistema de caching. No entanto, caching transparente tem um preço: ele exige mais espaço ou esforço de filtragem, porque qualquer conteúdo da

[1] Por exemplo, a Sony lançou recentemente um serviço de *streaming* em 4 K para seus televisores (Sony Oficial News, 2016) e o Playstation 4 Pro suporta aplicativos de vídeo da Netflix e da Amazon com resolução em 4 K (The Guardian, 2016).

Internet pode ser armazenado em cache.

Em ICN, o sistema de caching é dito ubíquo porque cada nodo da rede é um potencial elemento da rede de cache. Quanto mais ubíqua a rede de cache for, mais largura de banda da rede ela tende a economizar, na medida em que mais pedidos tendem a ser servidos por unidades de cache mais próximas do que a fonte original. Porém, para oferecer um sistema de caching ubíquo é necessário desenvolver unidades de cache capazes de operarem em taxa de linha para que elas possam ser acopladas aos roteadores.

1.1 Definição do Problema

Um desafio fundamental para o sucesso do paradigma ICN é o desenvolvimento de unidades de armazenamento em cache (ou *Content Stores* - CS, empregando uma nomenclatura ICN) que satisfaçam três requisitos: espaço de armazenamento grande (GHODSI et al., 2011), velocidade de operação rápida (ARIANFAR; NIKANDER, 2010) e custo acessível (PERINO; VARVELLO, 2011). Conforme a Tabela 1.1, por um lado, as tecnologias atraentes devido ao tamanho grande e preço baixo não atingem as velocidades necessárias para a operação em taxa de linha. Esse é o caso das tecnologias SSD, com alta capacidade (tamanho máximo na ordem dos TBs) e baixo custo (preço na ordem de 1 USD/GB), mas com alta latência de acesso (na ordem de 10 μs). Por outro lado, as tecnologias de memória que atendem os requisitos de velocidade são relativamente caras e possuem tamanho limitado. Por exemplo, as tecnologias DRAM atingem latências de acesso da ordem de 10 ns, porém possuem tamanho máximo na ordem de 10 GB por banco de memória e preço na ordem de 10 USD/GB. Como resultado, o tamanho máximo de uma CS que pode sustentar uma taxa de transferência na ordem de 10 Gbps é estimado em cerca de 10 GB (PERINO; VARVELLO, 2011). Essa ordem de magnitude de tamanho é pequena, especialmente considerando o grau de transparência almejado pelo sistema de caching das ICNs.

Tabela 1.1: Exemplos de tecnologias (Corsair, 2017) adequadas para HCS e algumas de suas especificações

Métrica	**DRAM** (a)	**SSD** (b)	**SSD** (c)
Interface de hardware	288 Pin	PCIe-3 x4	SATA 3
Taxa de leitura sequencial máxima	-	3.000 MB/seg	550 MB/seg
Taxa de leitura aleatória máxima	-	250.000 IOPS	80.000 IOPS
Taxa de transferência máxima	25.600 MB/seg	-	-
Tamanho	2×4 GB	480 GB	1.920 GB
Preço (USD/MB)	0,0232	0,0007	0,0005

Fonte: o autor

Em (ROSSINI et al., 2014), os autores introduziram o conceito denominado *Hierarchical Content Stores* (HCS) para superar o desafio descrito acima. HCS propõe o emprego de memória hierárquica no projeto de CS. Embora memória hierárquica não seja uma ideia nova, ela é particularmente atraente nesse contexto devido à peculiaridade da nomenclatura padronizada e a correlação entre as chegadas de requisições intrínsecas à ICN. A correlação entre requisições de conteúdos com nomenclatura padronizada permite o uso de uma chegada de uma requisição para um determinado pedaço do conteúdo (*chunk*) como indicativo para futuros pedidos de chunks subsequentes do mesmo conteúdo. Tal correlação pode ser explorada para transferir antecipadamente (ou seja, realizar *prefetching*) de chunks (a serem solicitados) de uma área de cache relativamente grande,

mas lenta, como SSD (Layer 2 - L2) para uma área de cache mais rápida, porém menor, como DRAM (Layer 1 - L1). Assim, cada transferência pode transportar um batch de chunks, ou simplesmente *batch*, que contém o chunk solicitado e um determinado número de chunks subsequentes ao solicitado. A transferência em batch proporcionaria dois benefícios principais. Primeiro, a quantidade de operações de transferência entre L1 e L2 seria reduzida proporcionalmente ao tamanho do batch. Segundo, a transferência de dados em batch permitiria que um SSD operasse em sua taxa de *acesso sequencial* em vez de sua taxa de *acesso aleatório*, como seria o acesso a chunks individuais[2]. Desse modo, seria possível melhorar o desempenho do sistema em mais de uma ordem de grandeza, tanto em termos de taxa de acesso, como em termos de tamanho da unidade de cache, mantendo-se a mesma grandeza de custo (ROSSINI et al., 2014).

Até onde se sabe, existem apenas dois trabalhos anteriores relacionados sobre HCS. O primeiro trabalho (ROSSINI et al., 2014), citado acima, introduz a ideia seminal e mostra uma avaliação analítica sobre os benefícios teóricos da HCS para a rede. O segundo trabalho (SO et al., 2014) apresenta uma arquitetura de software alternativa para acoplar SSD em unidades de cache de roteadores ICN.

1.2 Hipótese

Em (MANSILHA et al., 2015), nós investigamos os desafios de se implementar uma HCS. A principal conclusão daquele estudo foi que uma HCS composta por tecnologias de memória DRAM e SSD pode sustentar operações de cache de até 10 Gbps executando em hardware comum. Para superar o gargalo inerentes à tecnologia SSD, foi introduzida uma nova arquitetura de processamento paralelo. Baseada nesse estudo, definimos a seguinte hipótese.

> *"Uma arquitetura de paralelização que acople as camadas de memória de maneira adequada pode viabilizar unidades de cache hierárquicas suficientemente grandes e rápidas para roteadores ICN."*

Em (MANSILHA et al., 2017), nós investigamos a hipótese acima. Para tanto, nós apresentamos um conjunto de técnicas para emulação de HCS. O conjunto de técnicas permitiu a avaliação de uma gama de parâmetros de sistema mais ampla do que aquelas consideradas em trabalhos anteriores. Um conjunto de avaliações baseado na metodologia emulacional nos permitiu confirmar a hipótese acima e compreender o impacto de cada parâmetro no desempenho de HCS paralelizadas.

[2]Por exemplo, considerando o SSD de 1 TB de melhor desempenho segundo a avaliação apresentada em (Tom's Hardware, 2016), a diferença entre a taxa de leitura de blocos aleatórios e a taxa de leitura de blocos sequenciais é de ≈150 MB/s (a leitura aleatória permite uma taxa de 100.000 4 KB IOP/s, ou ≈ 400 MB/s, enquanto que a leitura sequencial permite uma taxa de 550 MB/s).

1.3 Contribuições da Tese

Esta tese apresenta duas contribuições principais para o campo de pesquisa em redes de computadores, descritas a seguir em ordem de relevância.

Arquitetura para paralelização de HCS. Uma arquitetura para paralelizar múltiplas HCS é proposta para superar os gargalos inerentes à L2 tirando proveito das plataformas de hardware *multi-core* comumente existentes no mercado. Em resumo, o esquema proposto supera desafios relacionados à concorrência (especificamente, sincronismo) através do particionamento determinístico das requisições de conteúdos entre múltiplas threads. A ideia é assegurar que todas as requisições de chunks de um determinado batch sejam sempre tratadas pela mesma thread de processamento. Como resultado, múltiplas HCS podem ser executadas paralelamente de maneira isolada, evitando, assim, a necessidade de acesso sincronizado ao primeiro nível, L1.

Metodologia para avaliação de HCS. Uma metodologia é proposta para avaliar o desempenho de HCS, incluindo a arquitetura paralela proposta e esquemas alternativos. A metodologia explora técnicas de emulação e modelagem analítica conjuntamente e oferece vantagens em relação às metodologias prototipal e simulacional. Em comparação com a prototipação, a emulação, por um lado, oferece resultados menos realísticos. Por outro lado, a emulação viabiliza a investigação de uma variedade de cenários de contorno (tanto em termos de hardware como de software) maior do que seria possível caso fosse empregada prototipação, com hardware real (restringidos por aspectos de custo e tecnológicos). Em comparação com simulação, a emulação fornece resultados mais realistas já que estende código real de um protótipo para os outros componentes da HCS. O modelo analítico, por sua vez, é útil para validar o sistema emulado e comparar a proposta com esquemas paralelos hipotéticos alternativos.

1.4 Organização

O restante desta tese está organizado da seguinte forma.

- O **Capítulo 2** revisa os conceitos relevantes de ICN, e posiciona a presente investigação no contexto de trabalhos relacionados, incluindo o projeto seminal de HCS e o estado da arte.
- O **Capítulo 3** apresenta as soluções propostas para paralelizar e avaliar HCS.
- O **Capítulo 4** discute o conjunto de avaliações que sustentam esta tese.
- O **Capítulo 5** resume as conclusões apresentadas nesta tese e perspectivas de trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico desta tese e está organizado da seguinte forma. A Seção 2.1 resume os principais conceitos do paradigma ICN, enquanto a Seção 2.2 descreve em maiores detalhes a arquitetura ICN adotada como referência (*Named Data Networking* – NDN), e a Seção 2.3, os conceitos relacionados à *Content Store* (CS) dos roteadores ICN. Por fim, a Seção 2.4 discute os esforços de pesquisa relacionados à unidade de cache hierárquica (*Hierarchical Content Store* - HCS), desde a sua concepção seminal até o estado-da-arte.

2.1 Information-Centric Networking - ICN

As redes ICN representam uma mudança de paradigma porque alteram o foco da comunicação da localização dos dados para o seu conteúdo. Nesse paradigma, as aplicações precisam determinar apenas qual conteúdo desejam. Encontrar a localização de um nodo qualquer que armazena os dados passa a ser uma tarefa da rede (e não mais da aplicação).

O paradigma ICN pode ser sintetizado através de um conjunto de abstrações que interagem entre si da seguinte forma. *Publicadores* podem disponibilizar conteúdos publicando objetos de informação, ou simplesmente objetos. Um *objeto* pode representar, por exemplo, documentos, livros e vídeos. A comunicação é dirigida por *requisitantes*, que podem solicitar objetos. A rede deve atender uma requisição entregando dados provenientes de qualquer *fonte* (seja ela o próprio publicador ou uma unidade de cache) que possua uma cópia válida do objeto. Para viabilizar esse desacoplamento em termos de tempo e espaço entre publicadores e requisitantes, em cada objeto são adicionados metadados que permitem verificar sua integridade e autenticidade.

O paradigma ICN é concretizado por meio de três mecanismos fundamentais (AHLGREN et al., 2012): nomeação de objetos, roteamento e caching. A **nomeação de objetos** é o mecanismo responsável por associar um identificador à informação que se deseja publicar ou obter. O nome é dado à própria informação, de forma independente de sua localização, forma de transmissão e armazenamento. Existem dois esquemas principais de nomeação adotados por arquiteturas ICN (Figura 2.1): plano e hierárquico. A nomeação plana utiliza um conjunto de bytes de tamanho pré-definido para identificar o conteúdo. Tal sequência pode ser gerada, por exemplo, por meio de uma função *hash* aplicada sobre o conteúdo. O esquema hierárquico, por sua vez, utiliza uma estrutura similar àquela das URLs. Alguns esquemas de nomeação planos apresentam a vantagem de serem autocertificáveis, ou seja, o próprio nome serve de garantia de que o conteúdo representado é aquele que se está procurando. A nomeação hierárquica, por sua vez, possui como vantagens a inteligibilidade e agregabilidade.

Objetos são divididos em pedaços menores (*chunks*) de tamanho fixo. Este esquema,

Figura 2.1: Esquemas básicos de nomenclatura do paradigma ICN

(a) Plano (auto-certificável, menos agregável) (b) Hierárquico (inteligível, mais agregável)

Fonte: o autor

conhecido como "*chunking*", representa uma mudança na granularidade de conteúdo: de um nível de arquivo (total) para um nível de chunk (parcial). Chunking oferece dois benefícios principais. Primeiro, permite um balanceamento de carga de grão fino, independentemente do tamanho do conteúdo, entre diversos nós e de forma paralelizável. Ou seja, um par pode ajudar a disseminar um conteúdo antes de concluir o seu download e um par pode receber um conteúdo de múltiplas fontes ao mesmo tempo. Em segundo lugar, chunking permite políticas de cache melhor informadas, que consideram valores de popularidade diversificados para diferentes partes de um mesmo conteúdo. Por exemplo, o início de um filme tende a ser mais popular que o final já que muitos expectadores desistem de assistir o conteúdo ao longo da exibição.

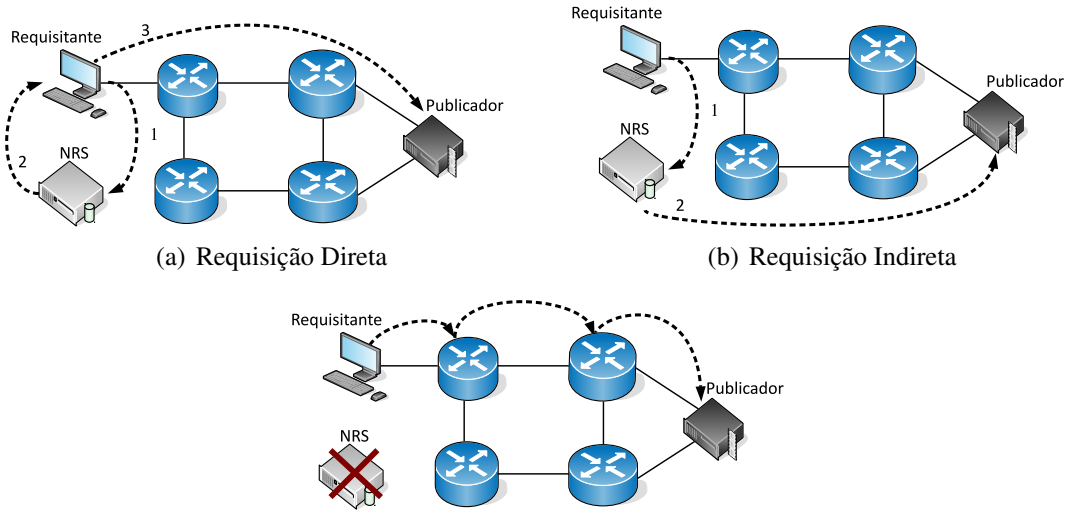
O segundo mecanismo fundamental de ICN, **roteamento**, divide-se em duas etapas. A primeira, chamada roteamento de requisições, refere-se ao caminho percorrido entre um requisitante e uma fonte que contenha uma cópia do objeto requisitado. A segunda etapa, roteamento de objetos, refere-se ao caminho de volta, ou seja, da fonte de uma cópia do objeto até o requisitante. As opções existentes para cada etapa são discutidas em maior nível de detalhes nos próximos parágrafos.

As soluções de **roteamento de requisições** podem ser classificadas em três modelos, os quais são ilustrados na Figura 2.2. A solução de roteamento de requisição inclui um serviço de resolução de nomes (do inglês *Name Resolution Service* - NRS) quando a nomeação é plana. O primeiro modelo utiliza um NRS para transformar o interesse do requisitante (como um filme) em localizador (por exemplo, endereço IP de uma fonte) e um nome plano. O localizador e o nome são retornados para o requisitante, que os utiliza para encaminhar a requisição diretamente para a fonte indicada do objeto. No segundo modelo também é necessário um NRS, porém o próprio serviço encaminha a requisição para a fonte do objeto. No terceiro modelo, por fim, o nome do objeto é suficiente para o processo de roteamento, similarmente ao protocolo IP. A forma como as tabelas de roteamento são construídas é dependente da arquitetura ICN.

A segunda etapa de roteamento refere-se à entrega dos objetos de informação. Basicamente, as soluções de **roteamento de objetos** podem ser classificadas em três modelos, ilustrados na Figura 2.3: conexão direta, caminho reverso e utilizando filtro de Bloom. Algumas arquiteturas suportam conexão direta para viabilizar entrega eficiente um-para-um, que pode ser útil em determinadas aplicações, em particular as que transmitem informações sensíveis (por exemplo, dados bancários). A segunda opção consiste em manter um rastro durante a etapa de roteamento das requisições e utilizá-lo para entregar o conteúdo através do caminho reverso. A terceira opção prevê o uso de filtros de Bloom[1] para enca-

[1]Filtro de Bloom é uma estrutura de dados que serve para testar probabilisticamente se um elemento é membro de um conjunto. Sua estrutura ocupa menos espaço que um vetor determinístico correspondente. Seu formalismo matemático garante que (i) falsos negativos não ocorram, e (ii) falsos positivos ocorram com determinada probabilidade.

Figura 2.2: Esquemas básicos de roteamento de requisição do paradigma ICN



(a) Requisição Direta

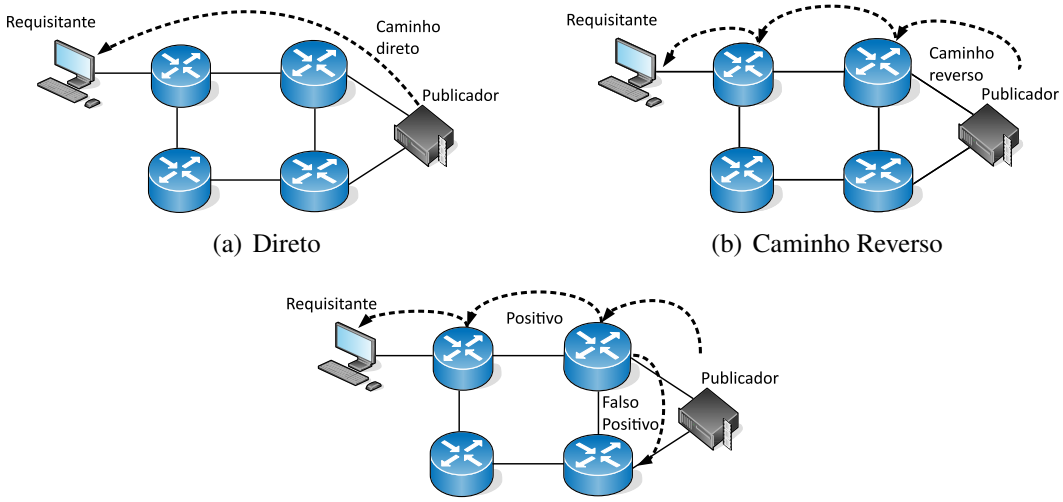
(b) Requisição Indireta

(c) Requisição por Nome

Fonte: o autor

minhar o conteúdo. Esse tipo de filtro diminui a sobrecarga com estruturas auxiliares nos roteadores, porém gera uma sobrecarga na rede, na forma de mensagens transmitidas em decorrência de falsos positivos.

Figura 2.3: Esquemas básicos de roteamento de conteúdo do paradigma ICN



(a) Direto

(b) Caminho Reverso

(c) Filtro de Bloom

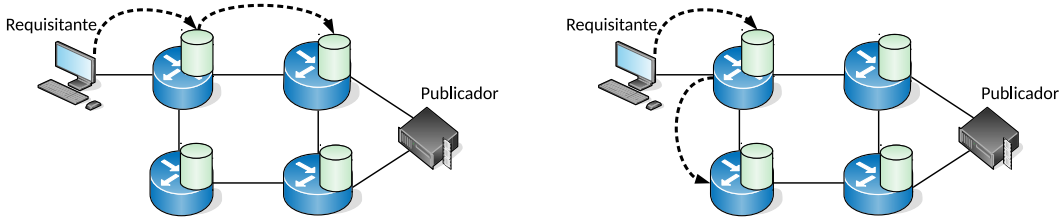
Fonte: o autor

O terceiro e último mecanismo fundamental em arquiteturas ICN é o uso de **caching** no núcleo da rede. Isto é, enquanto são encaminhados das fontes para os requisitantes, os dados podem ser armazenados por qualquer elemento intermediário. Caso uma nova requisição para o mesmo nome seja realizada, pode-se respondê-la com uma cópia válida do objeto armazenada em um elemento de caching mais próximo.

Em geral, uma rede de cache pode ser classificada considerando a posição das unidades de cache consultadas em relação ao caminho de roteamento de requisições (ZHANG; LI; LIN, 2013): *on-path* (Figura 2.4(a)) e *off-path* (Figura 2.4(b)). No primeiro caso,

apenas as unidades de cache presentes no caminho do roteamento de requisição são verificadas (tipicamente, antes da requisição ser propagada para o próximo nodo da rede). O caching *on-path* é simples, porém não explora completamente a infraestrutura já que cópias armazenadas fora da rota padrão são ignoradas, mesmo estando mais próximas. Para preencher esta lacuna é possível adotar caching *off-path*. A desvantagem nesse caso é a sobrecarga de gerenciamento dos conteúdos disponíveis nas proximidades (por exemplo, manutenção de índices). As especificidades das unidades de cache em termos locais serão discutidas na Seção 2.3.

Figura 2.4: Esquemas básicos de caching do paradigma ICN



(a) Cache On-Path (b) Cache Off-Path

Fonte: o autor

A capacidade total de redes de cache é um recurso frequentemente limitado por causa do seu custo. Assim, alguns trabalhos buscam otimizar a alocação de armazenamento com o objetivo de minimizar o tráfego na rede considerando como restrição um determinado orçamento (LAOUTARIS; ZISSIMOPOULOS; STAVRAKAKIS, 2005; WANG et al., 2016). Especificamente, LAOUTARIS; ZISSIMOPOULOS; STAVRAKAKIS (2005) investigaram o referido problema de otimização considerando topologias em árvore em redes de cache da Internet atual. Por sua vez, WANG et al. (2016) investigaram o problema de otimização considerando redes ICN e topologias genéricas.

2.2 Named Data Networking - NDN

Entre as arquiteturas ICN existentes, NDN é a aquela que tem recebido maior atenção da comunidade. Por isso, este estudo adota NDN como sua arquitetura ICN de referência. Argumenta-se, no entanto, que a contribuição principal deste estudo não se restringe apenas à arquitetura NDN, uma vez que o mecanismo de cache existe em todas as arquiteturas ICN.

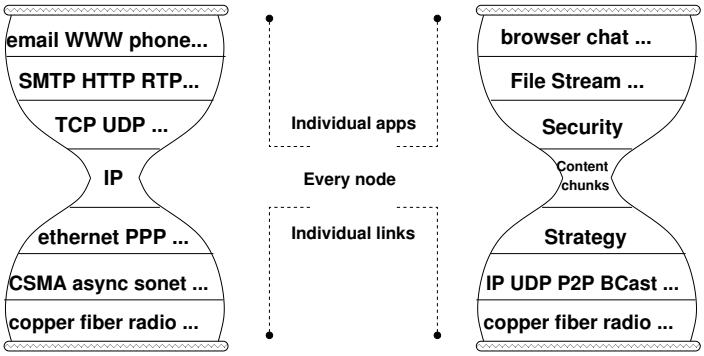
Existem dois protótipos principais da arquitetura NDN disponíveis atualmente. O primeiro, denominado CCNx (PARC, 2016), é um projeto de código fechado mantido pela PARC. O segundo protótipo, chamado NFD (NDN Project Team, 2016), é um *fork* de código aberto do CCNx 1.0 e é mantido pelo projeto Named-Data Networking (NDN) (ZHANG et al., 2010). Ambos os protótipos seguem os princípios básicos de arquitetura NDN, apesar de divergirem em termos de projeto de protocolos e licenciamento. De fato, recentemente, os dois grupos têm unido esforços visando padronização dos protocolos (ICNRG, 2016). O trabalho experimental apresentado nesta proposta é baseado na implementação NFD devido ao acesso livre ao código fonte.

A arquitetura NDN emprega as seguintes opções para cada componente: nomeação hierárquica, roteamento de requisição por nome, requisição de conteúdo por caminho reverso e caching *on-path*. Para implementar esses mecanismos, a arquitetura NDN estabelece uma pilha de protocolos e seus protótipos estabelecem tipos de pacotes e seguem um

fluxograma de processamento conforme explicado em maiores detalhes nos parágrafos seguintes.

A **pilha de protocolos** NDN pode ser comparada com a pilha TCP/IP, como ilustrado pela Figura 2.5, destacando-se três aspectos principais. Primeiramente, é possível estabelecer relações bilaterais para todas as camadas (por exemplo, entre a camada de transporte do IP e a camada de segurança do NDN). Em segundo lugar, note-se que a pilha NDN, a exemplo da pilha TCP/IP, também estabelece uma camada chamada "*cintura fina*", no sentido de ser a única a exigir acordo universal na rede (o qual seja a nomeação dos conteúdos). Por fim, a arquitetura NDN pode ser instanciada sobre outros protocolos de rede, incluindo o IP.

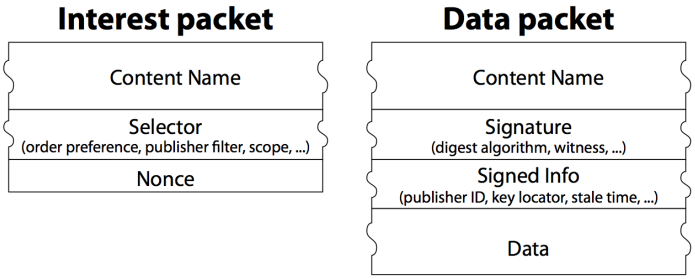
Figura 2.5: Comparação entre as pilhas IP e NDN



Fonte: JACOBSON et al. (2009)

A arquitetura NDN especifica dois **tipos de pacotes**, como ilustrado na Figura 2.6. Os requisitantes enviam pacotes de interesse (*interest*) e recebem pacotes de dados (*data*) como resposta. O primeiro tipo de pacote carrega o nome do conteúdo desejado. O segundo tipo, de dados, leva o nome de conteúdo, o respectivo objeto/*chunk*[2], e a assinatura do objeto realizada com a chave privada do publicador.

Figura 2.6: Pacotes NDN



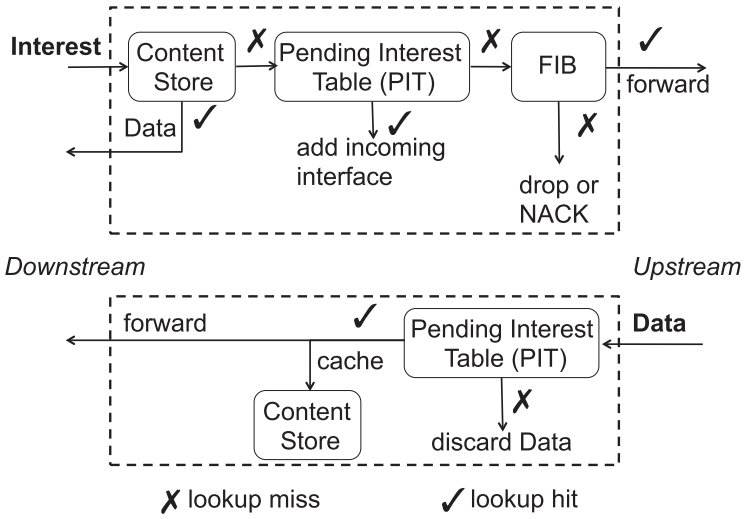
Fonte: ZHANG et al. (2010)

Para processar os pacotes acima, cada roteador NDN mantém três estruturas de dados principais indexadas pelo nome do conteúdo: a unidade de cache (*Content Store* - CS),

[2]Os termos "objeto" e "chunk" são utilizados de maneira intercambiável dependendo do contexto para melhorar a clareza do texto, mas representam a mesma granularidade: uma parte de tamanho fixo do conteúdo.

uma tabela de interesses pendentes (*Pending Interest Table* - PIT), e uma tabela de informações de encaminhamento (*Forwarder Information Base* - FIB). A Figura 2.7 mostra o **fluxograma de um roteador NDN** para tratar um pacote de interesse (na parte superior) e para tratar um pacote de dados (na parte inferior). O roteador realiza operações de pesquisa em cada estrutura de dados seguindo uma determinada sequência, conforme explicado detalhadamente abaixo.

Figura 2.7: Fluxograma de atravessamento de pacotes nos componentes de um roteador NDN



Fonte: YI et al. (2013)

Ao receber um pacote de interesse, o roteador primeiro verifica se o objeto correspondente se encontra armazenado na CS. Em caso positivo, um pacote contendo o objeto é enviado para a interface de chegada do pacote de interesse. Em caso negativo, o roteador verifica se o nome do objeto requisitado está registrado na PIT. Um caso positivo pode ser consequência de duas possibilidades:

- Interesse duplicado, que é descartado; ou,
- Interesse novo, advindo de uma interface diferente, que é anotada na entrada PIT correspondente ao nome do objeto solicitado.

Em caso negativo, um novo registro de interesse é adicionado na PIT e a requisição é verificada na FIB. Normalmente, o roteador deverá encontrar uma entrada seguindo o algoritmo de prefixo mais longo na FIB e encaminhar a requisição para a interface retornada. Casos onde nenhuma entrada satisfatória é encontrada significam que o roteador desconhece a interface para a qual a requisição deve ser enviada. Nesses casos de exceção, o roteador tipicamente descartaria o pacote.

Quando um roteador NDN recebe um pacote de dados, ele primeiro verifica se o nome consta na PIT. Em caso negativo, o pacote de dados é considerado não solicitado e, portanto, descartado. Caso contrário, três operações são executadas sequencialmente. Primeiramente, o roteador envia o pacote de dados para a(s) interface(s) a partir do qual o interesse foi recebido (conforme registrado na PIT). Em segundo lugar, os dados são opcionalmente armazenados na CS (dependendo da política de inserção, conforme será explicado na próxima seção). Por fim, a entrada é removida da PIT.

2.3 Content Store - CS

Esta seção revisa os conceitos relacionados ao estudo de caching e está organizada em duas partes. Primeiramente, são elencadas as principais políticas de substituição, considerando suas duas etapas: inserção (isto é, como uma unidade de caching decide se um determinado objeto deve ser armazenado) e remoção (isto é, como uma unidade de cache escolhe qual objeto deve ser desalojado quando precisa liberar espaço para outro objeto). Em segundo lugar, são revisados os modelos matemáticos básicos utilizados para avaliação analítica de unidades de cache.

Uma unidade de cache utiliza uma **política de inserção** para decidir se um determinado objeto, que chega através de um pacote de dados, deve ou não ser armazenado localmente. Assumindo que um pacote de dados percorre um caminho desde a fonte até o requisitante, existem quatro políticas de inserção principais:

- **LCE** (*Leave-a-Copy-Everywhere*). Todos os nodos do caminho armazenam o objeto.
- **LCP** (*Leave-a-Copy-Probabilistically*). Cada nodo do caminho armazena o objeto de acordo com uma determinada probabilidade.
- **LCD** (*Leave-a-Copy-Down*). O objeto é armazenado apenas no nodo seguinte àquele onde foi encontrado. Assim, o objeto tende a se aproximar cada vez mais do requisitante a cada nova requisição.
- **2-LRU** (*Least Recently Used* de segunda ordem). Essa política foi proposta mais recentemente (MARTINA; GARETTO; LEONARDI, 2014) e visa evitar o armazenamento de objetos que são requisitados poucas vezes ao custo de uma estrutura de dados auxiliar. Tal estrutura, denominada cache virtual, armazena apenas o nome do objeto e por isso pode ser mantida na memória cache da CPU, oferecendo taxas operacionais maiores do que uma cache real, que armazena conteúdo também e, portanto, ocupa mais espaço. A ideia é que todo objeto seja inserido na cache virtual, mas apenas aqueles que já estejam na cache virtual sejam inseridos na cache real. A política é dita de segunda ordem porque é composta por apenas uma unidade de cache virtual, além da cache real. Essa política pode ser generalizada para n-ésima ordem, considerando uma quantidade $n - 1$ caches virtuais em série. É importante observar que o nome advém da política de remoção de ambas as caches: a LRU, que será explicada a seguir.

Uma unidade de cache emprega uma **política de remoção** para escolher o objeto a ser removido quando está cheia e decide armazenar um outro objeto. As políticas mais comuns para tomar essa decisão são as seguintes:

- **Aleatório**. O objeto é escolhido aleatoriamente (tipicamente de maneira uniforme).
- **FIFO** (*First-In-First-Out*). O objeto escolhido é aquele que está há mais tempo armazenado.
- **LRU** (*Least-Recently-Used*). O objeto escolhido é aquele que está há mais tempo sem ser requisitado.

Para analisar o comportamento de sistemas de cache através de **modelagem analítica**, três conceitos fundamentais são tipicamente empregados na literatura: a distribuição Zipf para modelar a popularidade dos conteúdos, o chamado *Independent Reference Model* (IRM) como premissa das requisições, e a aproximação de Che (CHE; TUNG; WANG, 2002) para avalição de desempenho. Esses conceitos são discutidos nos parágrafos seguintes.

A distribuição Zipf é frequentemente observada em medições de tráfego e amplamente adotada em estudos de avaliação de desempenho. Na sua forma mais simples, a distribuição Zipf diz que a probabilidade do n-ésimo conteúdo mais popular ser solicitado é proporcional a $1/n^{\alpha}$. O parâmetro α depende do sistema considerado (por exemplo, Web, VoD, P2P) e afeta o desempenho da cache; tipicamente, quanto maior α, melhor é o desempenho da cache (FRICKER et al., 2012). Os valores de α relatados na literatura relacionada a ICN variam entre $0{,}64$ e $1{,}00$ para a maioria dos tipos de conteúdos, e $\alpha \geq 2{,}00$ para conteúdos gerados por usuários (PENTIKOUSIS et al., 2016).

O IRM é a abordagem padrão de fato adotada na literatura para caracterizar a chegada de requisições a uma unidade de cache. Ele assume que a popularidade dos conteúdos solicitados segue uma distribuição Zipf e se caracteriza por duas premissas principais. Primeiro, o conjunto de conteúdos disponíveis não se altera ao longo do tempo e possui tamanho definido (N). Segundo, a probabilidade p_n de um determinado conteúdo n, $1 \leq n \leq N$, ser requisitado é constante (isto é, a popularidade do conteúdo não varia ao longo do tempo) e *independe* dos pedidos anteriores (ou seja, ignora a correlação temporal).

A aproximação de Che (CHE; TUNG; WANG, 2002) simplifica a estimação de desempenho de uma cache LRU alimentada com requisições que seguem o IRM. A simplificação ocorre em relação à dinamicidade do desempenho do sistema em relação aos múltiplos objetos. Em suma, o desempenho do sistema é resumido por um componente (T_C) que atua como único valor representativo sobre o desempenho da cache, independentemente do objeto (n) solicitado. Uma justificativa teórica para a aproximação de Che pode ser encontrada em (FRICKER; ROBERT; ROBERTS, 2012).

Em maiores detalhes, considere-se uma cache capaz de armazenar C objetos distintos. Seja $T_C(n)$ o tempo médio necessário para que C objetos (excluindo-se n) sejam solicitados por usuários. Em uma cache LRU, $T_C(n)$ representa, também, o tempo de despejo para o objeto n. Um objeto n se encontrará na cache no instante de tempo t se, e somente se, um tempo menor do que $T_C(n)$ tenha decorrido desde a última solicitação para o objeto n (ou seja, ocorrer uma requisição para n no intervalo $[t - T_C(n), t]$).

Seja λ_n a taxa de solicitações para um objeto n, e $\Lambda = \sum_{n=1}^{N} \lambda_n$, a taxa global de solicitações que chegam conforme um processo de Poisson. Considerando uma lei de popularidade de objetos análoga à considerada pelo IRM, temos $\lambda_n = \Lambda p_n$ (MARTINA; GARETTO; LEONARDI, 2014). Segundo (CHE; TUNG; WANG, 2002), a probabilidade média $P_{in}(n)$ para um objeto n ser encontrado em cache pode ser calculado por:

$$P_{in}(n) = 1 - e^{-\lambda_n T_C} \tag{2.1}$$

De acordo com (MARTINA; GARETTO; LEONARDI, 2014), como consequência imediata da propriedade PASTA (*Poisson Arrivals See Time Averages* (WOLFF, 1982)) do processo de Poisson, $P_{in}(n)$ permite inferir, por construção, a probabilidade de ocorrer *acerto em cache* $P_{hit}(n)$ (isto é, a probabilidade de uma solicitação para um objeto n encontrar o objeto n em cache). Assim, considerando uma unidade de cache com tamanho C, por construção, temos que:

$$C = \sum_{n=1}^{N} \mathbb{E}[\mathbb{I}_{\{\text{n em cache}\}}] = \sum_{n=1}^{N} P_{in}(n) \tag{2.2}$$

Note-se que a única variável da Equação 2.2 com valor desconhecido é T_C. Portanto, a Equação 2.2 permite calcular o valor de T_C, com precisão arbitrária, através de um método de ponto fixo. Após calcular T_C, podemos calcular a probabilidade média de uma requisição qualquer resultar em acerto na cache:

$$P_{hit} = \sum_{n=1}^{N} \frac{\lambda_n}{\Lambda} P_{hit}(n) \tag{2.3}$$

De maneira mais concisa (anulando-se Λ), a probabilidade média de uma requisição qualquer resultar em acerto em uma cache que segue política de remoção LRU pode ser calculada por:

$$P_{hit} = \sum_{n=1}^{N} p_n P_{hit}(n) \tag{2.4}$$

Mais recentemente, MARTINA; GARETTO; LEONARDI (2014) propuseram diversas extensões para a aproximação de Che visando contemplar outras políticas de remoção (além da LRU) e o impacto de correlações temporais (em contraste com o IRM). Entre elas, a extensão para a política de remoção FIFO é de particular importância para esta tese. O sistema de cache hierárquico implementado nesta tese (conforme será descrito na Seção 3.4) adota a política de substituição FIFO e é avaliado analiticamente (Seção 4.3) com base na respectiva aproximação, que é apresentada a seguir.

Assumindo que requisições para um objeto n, $1 \leq n \leq N$, chegam de acordo com um processo de Poisson com taxa λ_n, tem-se:

$$P_{FIFO}(n) = \frac{\lambda_n T_C}{1 + \lambda_n T_C} \tag{2.5}$$

sendo T_C a única solução de:

$$C = \sum_{n=1}^{N} P_{FIFO}(n) \tag{2.6}$$

Na Equação 2.6, a única variável com valor desconhecido é T_C (de maneira análoga à Equação 2.2), que pode ser calculado com precisão arbitrária através de um método de ponto fixo. Como resultado, a probabilidade média de uma requisição resultar em acerto em uma cache que segue política de substituição FIFO pode ser estimada, de maneira análoga à Equação 2.4, por:

$$P_{FIFO} = \sum_{n=1}^{N} p_n P_{FIFO}(n) \tag{2.7}$$

2.4 Estado da Arte: Hierarchical Content Store - HCS

Antes de discutir os trabalhos relacionados, é importante salientar as diferenças do foco desta tese com três tópicos de pesquisa que, embora invoquem um conjunto de pala-

vras chave semelhantes, se referem a desafios distintos. Em primeiro lugar, trabalhos sobre "caches hierárquicas" (CHE; WANG; TUNG, 2001; LAOUTARIS; SYNTILA; STAVRAKAKIS, 2004) tratam da estrutura topológica de unidades de cache em uma rede. Esta tese, em contraste, se refere à memória hierárquica dentro de uma única unidade de cache. Em segundo lugar, "armazenamento híbrido" (KIM; KIM, 2015) refere-se à combinação de tecnologias SSD e HDD para diminuir o custo de servidores de cache para CDN, que apresentam requisitos em termos de ubiquidade e transparência diferentes das redes ICN. Em terceiro lugar, trabalhos sobre "memória hierárquica" no contexto das arquiteturas de computadores (AGGARWAL et al., 1987) consideram outras cargas de trabalho diferentes das sequências de interesses gerados em redes ICN, que justamente motivam o desenvolvimento de HCS, investigada nesta tese.

No domínio de ICN, a grande maioria dos trabalhos se concentra em algoritmos, protocolos e aspectos de desempenho da rede. Em quantidade menor, existem estudos que investigam o projeto de roteadores ICN e que são, portanto, mais próximos a esta tese. Artigos seminais (ARIANFAR; NIKANDER, 2010; PERINO; VARVELLO, 2011) abordam o projeto de roteadores ICN. Trabalhos mais recentes refinam aspectos específicos de roteadores ICN, como gerenciamento da PIT e FIB (YOU et al., 2012a,b; VARVELLO; PERINO; LINGUAGLOSSA, 2013; CAROFIGLIO et al., 2015; DAI; LIU, 2016), encaminhamento de pacotes e filtragem (SONG et al., 2015; PAPALINI et al., 2016; SHI et al., 2016) e eficiência energética (HASEGAWA et al., 2014). Essas referências são relevantes a partir da perspectiva de um roteador ICN, mas a avaliação de CS é limitada (ou ausente) e, portanto, apresentam metas distintas da presente tese.

O particionamento determinístico de solicitações entre múltiplas threads de processamento, como o explorado nesta tese, é cada vez mais comum em software de redes (JAMSHED et al., 2012; KIM; CHOI; CHOI, 2014), mas não havia sido explorado no contexto de CS até então. Esse esquema de paralelismo foi originalmente trazido à área de ICN por (SO; NARAYANAN; ORAN, 2013) e (SO et al., 2013). Mais recentemente, o mesmo princípio foi explorado por (KIRCHNER et al., 2016) para propor um roteador NDN para redes programáveis e por (TANIGUCHI et al., 2016) para desenvolver uma metodologia de projeto de roteadores ICN de alta vazão. Esses trabalhos consideram o paralelismo como um conceito-chave para permitir operações de alta velocidade, que estão de acordo com as conclusões apresentadas nesta tese. A conclusão é reforçada ainda mais por esforços complementares tais como (MARCHAL; CHOLEZ; FESTOR, 2016), que avaliam os limites (severos) das implementações de roteadores ICN de núcleo único. No entanto, a investigação de roteadores ICN de múltiplas threads (SO et al., 2013; KIRCHNER et al., 2016; TANIGUCHI et al., 2016) concentram-se sobre outros componentes além da cache, de maneira que a CS ou é minimamente avaliada (KIRCHNER et al., 2016; TANIGUCHI et al., 2016), ou é sequer considerada explicitamente (SO et al., 2013).

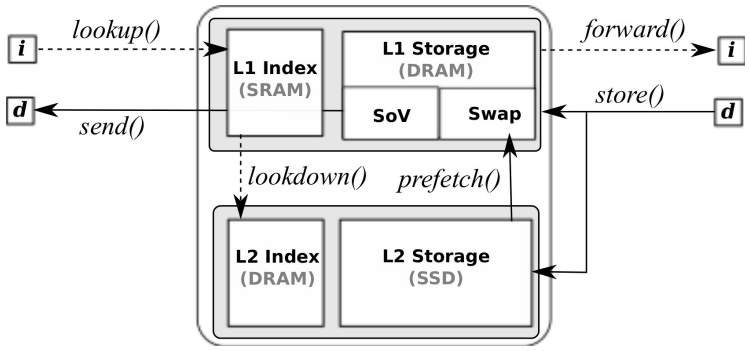
Até onde se sabe, existem apenas quatro trabalhos anteriores relacionados sobre o tema HCS (ROSSINI et al., 2014; SO et al., 2014; MANSILHA et al., 2015, 2017). Esses trabalhos são discutidos nas próximas subseções, em ordem cronológica.

2.4.1 Projeto Seminal

A ideia de explorar "memórias hierárquicas" dentro do contexto do ICN foi proposta pela primeira vez em (ROSSINI et al., 2014). Mais precisamente, os autores propuseram o projeto seminal de HCS, modelaram analiticamente o sistema proposto, e realizaram uma avaliação analítica do desempenho do sistema. Dada sua relevância para esta tese, o projeto original da HCS é discutido a seguir, com ajuda da Figura 2.8. O sistema processa

pacotes *Interest* i e *Data* d, e contém uma hierarquia de memórias: um nível relativamente rápido e pequeno (L1, baseado em DRAM), e um nível relativamente grande e lento (L2, baseado em SSD). A HCS visa mascarar a taxa de leitura menor da L2 através da transferência antecipada (*prefetching*) de batches de chunks da L2 para L1. Além do esquema básico, duas otimizações foram propostas. A primeira otimização é realizar prefetching do batch seguinte quando o último chunk do batch atual for lido da L1. A segunda otimização é dividir a L1 em duas áreas: uma área de permuta (*swap*), que armazena chunks de transferências em curso, e uma segunda área denominada *Start of Video* (SoV), que mantém o primeiro chunk de conteúdos populares. A área SoV permite que o sistema leia o primeiro chunk de conteúdos populares diretamente da L1 (diminuindo o atraso para o usuário) enquanto que realiza em paralelo o prefetching do primeiro batch.

Figura 2.8: Arquitetura seminal da HCS



Fonte: ROSSINI et al. (2014)

2.4.2 Protótipo da Cisco

Em (SO et al., 2014), a Cisco apresenta uma extensão para um roteador apresentado anteriormente pelo grupo (SO; NARAYANAN; ORAN, 2013). O objetivo da extensão é utilizar SSDs para aumentar o tamanho da CS enquanto mantém a vazão de pacotes em taxa de linha. A ideia principal é permitir que as operações de entrada e saída (E/S) sejam processadas em rotas alternativas no roteador para não limitar a taxa de encaminhamento. Além disso, um gerenciador de cache realiza operações de leitura e gravação via acesso assíncrono para maximizar a utilização dos SSDs. O esquema proposto foi avaliado experimentalmente utilizando os parâmetros resumidos na Tabela 2.1.

Os resultados da única avaliação realizada são reproduzidos pela Figura 2.9. Ela mostra a largura de banda de E/S utilizada como função do número de pacotes processados por vetor de E/S (v). As linhas contínuas resultam da utilização uma carga de trabalho sintética onde as leituras e escritas ocorrem em ordem aleatória. As linhas pontilhadas resultam de uma carga de trabalho sintética onde as leituras e as escritas acontecem em ordem sequencial. Com base nesses resultados, os autores apresentam duas observações. Primeiro, o desempenho de escrita é menos da metade do desempenho de leitura em todos os casos (confirmando uma propriedade assimétrica bem conhecida da tecnologia SSD). Segundo, tanto para operações de leitura como de escrita, a largura de banda de E/S utilizada é afetada positivamente à medida que v aumenta, alcançando, finalmente, patamares que são próximos às especificações do SSD utilizado.

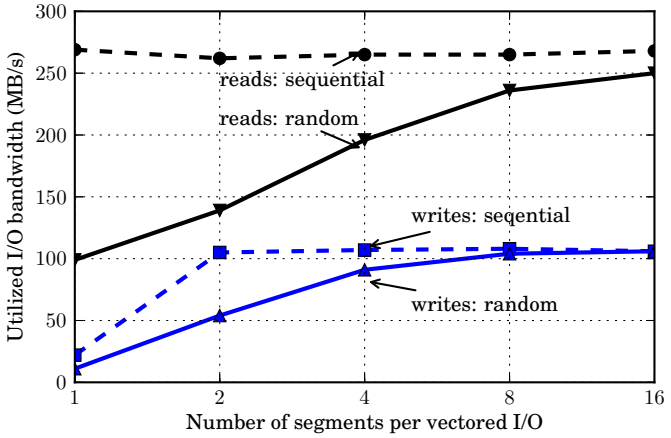
Note-se que esse trabalho se concentra em como maximizar a leitura de SSDs para fins de memória hierárquica. Portanto, como será discutido na Seção 3.1, esse trabalho

Tabela 2.1: Especificações da avalição do protótipo desenvolvido pela Cisco

Classe	**Parâmetro**	**Valor**
Hardware	CPU	Intel Xeon 2,6 GHz
	NUMA	1 nodo, 8 núcleos
	DRAM	32 GB - 1.333 MHz (0,8 ns)
	Memória secundária	Intel SATA2 SSD 160 GB
Software	Sistema operacional	Não disponível
Carga de trabalho	Traço base	IRCache traces (IRCache, 2007)
	Tamanho	100.000 pacotes (interest + data)
	Sintético 1	Sequencial
	Sintético 2	Aleatório
	Parâmetro α da distribuição Zipf	Não disponível
Content Store	Tamanho	128 GB

Fonte: adaptado de (SO et al., 2014)

Figura 2.9: Desempenho do protótipo desenvolvido pela Cisco



Fonte: (SO et al., 2014)

pode ser visto como um esforço de pesquisa complementar.

2.4.3 Protótipo da Alcatel-Lucent

Mais recentemente, em (MANSILHA et al., 2015), nós introduzimos uma arquitetura de processamento paralelo para HCS. A arquitetura proposta particiona as requisições entre múltiplas threads deterministicamente para prescindir de mecanismo de sincronização. A arquitetura foi investigada por meio de duas metodologias complementares, emulação e prototipagem, que conduziram a resultados coerentes. O autor desta tese realizou um estudo baseado em emulação enquanto que um grupo de pesquisa da Alcatel-Lucent desenvolveu um estudo baseado em prototipação. Os autores do protótipo o avaliaram experimentalmente utilizando as configurações especificadas na Tabela 2.2.

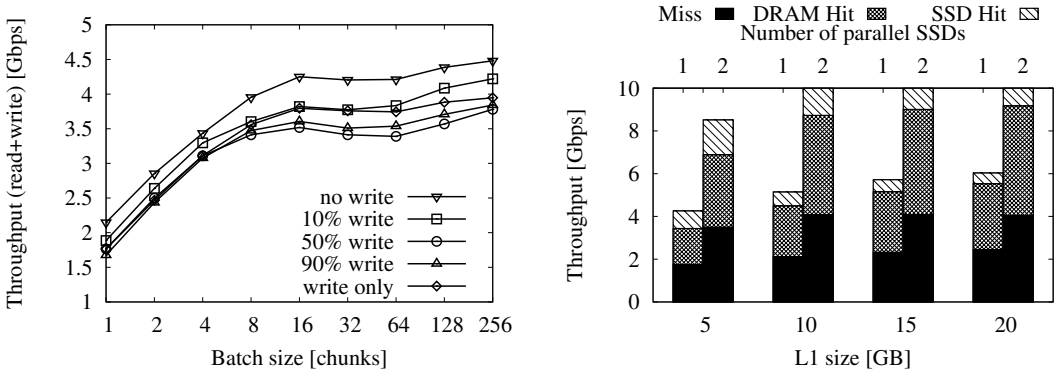
Tabela 2.2: Especificações da avaliação do protótipo desenvolvido pela Alcatel-Lucent

Classe	**Parâmetro**	**Valor**
Hardware	CPU	(2 ×) Intel Xeon E5540 2,53 GHz
	NUMA	2 nodos, 4 cores/nodo
	DRAM	32 GB - 1,3 GHz (0,8 ns)
	SSD	(2 ×) 200 GB HP enterprise SAS
	NIC	Dual-port 10 GbE Intel 82599EB
Software	Sistema operacional	Ubuntu 12.04 LTS
Carga de trabalho	Tamanho do catálogo	1,3 Milhão
	Tamanho do conteúdo	10 MB
	Tamanho do chunk	8 KB
	Parâmetro α da distribuição Zipf	1
Contet Store	Tamanho	(até) 120 GB

Fonte: adaptado de (MANSILHA et al., 2015)

Os autores submeteram o protótipo a duas avaliações: desempenho do SSD e desempenho do roteador completo. Na primeira avaliação, a vazão do SSD foi observada na medida em que foram variados, entre outros fatores, o tamanho do batch e a razão de leitura/escrita da carga de trabalho sintética (isto é, o percentual de operações de leitura/escrita em SSD para um determinado lote aleatório). A Figura 2.10 apresenta os resultados de duas avaliações[3]. A Figura 2.11(a) mostra a vazão do SSD cumulativa (leitura + escrita) como função do tamanho do batch para diferentes razões de leitura/escrita. Os autores observam que um batch com 16 chunks (128 KB) é suficiente para alcançar uma vazão do SSD próxima ao máximo do hardware para todas as combinações de leitura/escrita. Aumentar o tamanho do batch além desse patamar penaliza proporcionalmente a latência, mas não gera ganhos perceptíveis em termos de vazão.

Figura 2.10: Desempenho do protótipo desenvolvido pela Alcatel-Lucent



(a) Desempenho do SSD (b) Desempenho do Protótipo

Fonte: (MANSILHA et al., 2015)

A segunda avaliação mediu o desempenho do protótipo configurado com tamanho de batch igual a 16 e distribuindo a L2 em 1 ou 2 SSDs. Os resultados dessa avaliação são apresentados na Figura 2.11(b). Segundo os autores, as lições mais importantes são que o sistema é capaz de sustentar uma vazão de 10 Gbps quando a L2 é distribuída entre duas unidades SSD, e que o desempenho cai significativamente ao se utilizar apenas uma unidade SSD.

[3]O referido artigo apresenta três avaliações do protótipo, dos quais uma é omitida por questões de simplicidade.

Em suma, concluiu-se que o tamanho de batch que oferece melhor desempenho varia entre 8 e 16 chunks. Demonstrou-se, também, que é possível atingir a taxa de transferência sequencial de um hardware SSD através de transferências de batches (por volta ≈3,8 Gbps no protótipo implementado).

2.4.4 Discussão

Como metodologia de avaliação, a prototipagem demonstrou a viabilidade técnica de HCS. Por outro lado, a prototipagem não esclarece os compromissos que surgem ao se acoplar as duas camadas de memória. Mais precisamente, argumenta-se que a prototipagem oferece duas limitações principais. Primeiro, ela depende das especificidades de hardware, que estão intrinsecamente ligadas às tecnologias atuais. Em segundo lugar, os protótipos são, tipicamente, implementações de código fonte fechado, o que torna seus resultados mais difíceis de serem reproduzidos, comparados e estendidos pela comunidade.

Conforme mencionado anteriormente, a arquitetura de processamento paralelo para HCS, apresentada em (MANSILHA et al., 2015), foi investigada pelo autor desta tese por uma metodologia baseada em emulação. A emulação estende um protótipo de código aberto (NFD) e permitiu o estudo de uma gama de parâmetros do sistema mais ampla do que o estudo baseado em prototipação. Em (MANSILHA et al., 2017), nós aprofundamos o estudo da arquitetura paralela através da metodologia emulacional e assumimos como premissas os resultados obtidos via prototipação (como, por exemplo, o tamanho do batch de transferência). Esta tese reúne e detalha o estudo baseado em emulação apresentado inicialmente em (MANSILHA et al., 2015) e aprofundado em (MANSILHA et al., 2017).

A Tabela 2.3 apresenta uma comparação dos fatores considerados nos trabalhos experimentais e nesta tese. Observe-se que o protótipo da Cisco explora apenas a L2, de maneira isolada, enquanto o protótipo da Alcatel explora ou apenas a L2, ou o roteador completo. Esta tese, portanto, preenche uma lacuna ao focar em HCS. Além disso, esta tese apresenta algoritmos que são simples de serem implementados, estendem uma implementação de roteador NDN de código fonte aberto e foram executados em uma máquina virtual alocada em uma nuvem pública, que pode ser acessada por qualquer pesquisador interessado.

Tabela 2.3: Comparação entre os fatores considerados nesta Tese e no estado-da-arte

Trabalho	**Escopo**	**Algoritmo**	**Acesso ao Hardware**	**Metodologia**	**Grau de Paralelismo**	**Tamanho da L1**	**Tamanho da L2**	**Vazão da L2**
Cisco	L2	Fechado	Não	Prototipação	1	–	128 GB	–
Alcatel	L2	Fechado	Não	Prototipação	1	–	100 GB	–
	Roteador	Fechado	Não	Prototipação	[1, 2]	{5, 10} GB	100 GB	≈ 4 Gbps
Esta Tese	Cache	Aberto	Sim (nuvem)	Emulação Analítica	[1, 16]	[1, 10] GB	[10 GB,1 TB]	[1, 64] Gbps

Fonte: o autor

Graças à adoção da metodologia emulacional, é possível explorar uma gama de parâmetros maior que aquelas consideradas nos trabalhos anteriores, para todos os fatores relevantes (Grau de Paralelismo, Tamanho da L1, Tamanho da L2 e Vazão da L2). Por outro lado, para viabilizar a avaliação, a metodologia emulacional proposta emprega premissas simplificatórias. Dado o foco da tese, os componentes que não a cache hierárquica, de menor interesse (como tabelas de encaminhamento), não são considerados. Assim,

assume-se que todos os conteúdos a serem acessados estão disponíveis no roteador, enfatizando o papel da cache hierárquica e apresentando um limite superior de desempenho dos sistemas avaliados.

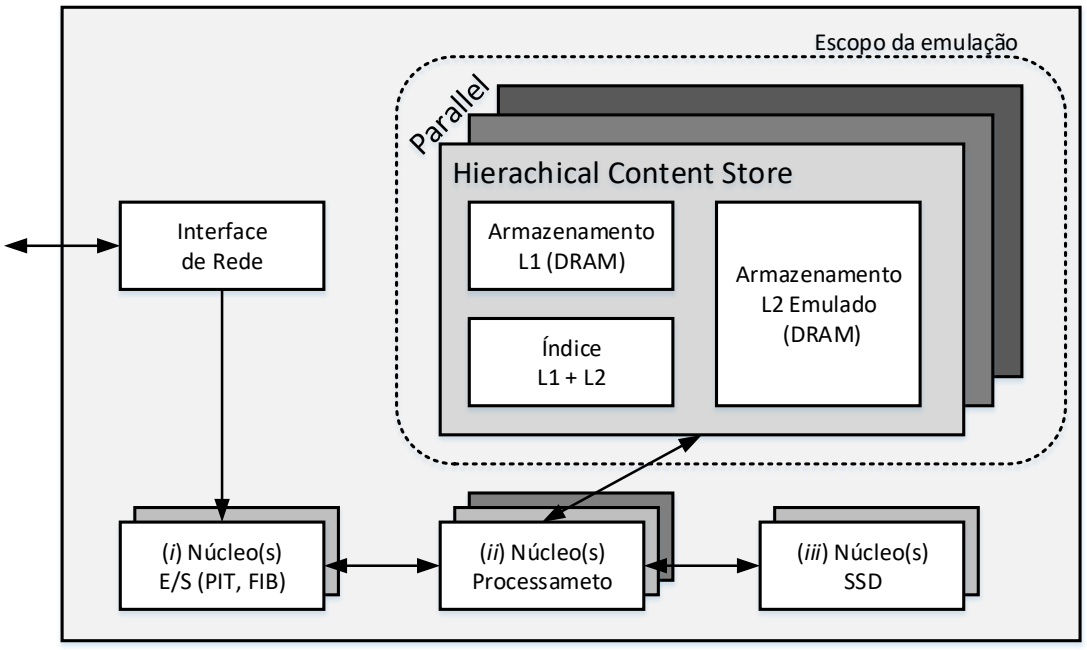
3 SOLUÇÕES PROPOSTAS

Este capítulo apresenta as soluções propostas para os desafios apresentados no Capítulo 1 e está organizado como segue. Inicialmente, a Seção 3.1 restringe o escopo do estudo realizado, enquanto a Seção 3.2 explica os objetivos do (sub)sistema investigado. Em seguida, as seções 3.3 e 3.4 apresentam as soluções propostas para paralelização e avaliação de HCS, respectivamente.

3.1 Escopo do Estudo

Esta seção apresenta a visão completa do roteador para, em seguida, restringir o âmbito de investigação especificando os componentes de interesse. A Figura 3.1 oferece uma visão de alto nível do sistema considerado neste trabalho. O roteador lida com chegada de pacotes através de um encadeamento de núcleos (ou *threads*), que realizam: (i) entrada e saída (E/S) de pacotes, (ii) processamento de pacotes e (iii) leitura e escrita no SSD.

Figura 3.1: Esboço do roteador NDN e escopo de investigação considerados



Fonte: o autor

Os *núcleos de E/S* são responsáveis pelos outros componentes de um roteador NDN, como a PIT e a FIB, e encaminham, conforme a demanda, pacotes para os *núcleos de*

processamento (do componente de cache). Um *núcleo de processamento*, por sua vez, gerencia uma HCS e executa suas operações, incluindo aquelas de leitura/escrita nos *núcleos de gerenciamento de SSDs*.

Por uma questão de simplificação da avaliação, o trabalho se concentra na (H)CS e abstrai os outros componentes do plano de dados do NDN (PIT, FIB, etc.). Como consequência, assume-se que todos os conteúdos do catálogo se encontram armazenados localmente. Como os conteúdos armazenados em cache não são substituídos ao longo das avaliações, apenas operações de leitura são efetuadas e nenhuma operação de escrita é executada. Portanto, este trabalho se concentra em operações de leitura, enquanto operações de escrita são deixadas para trabalhos futuros.

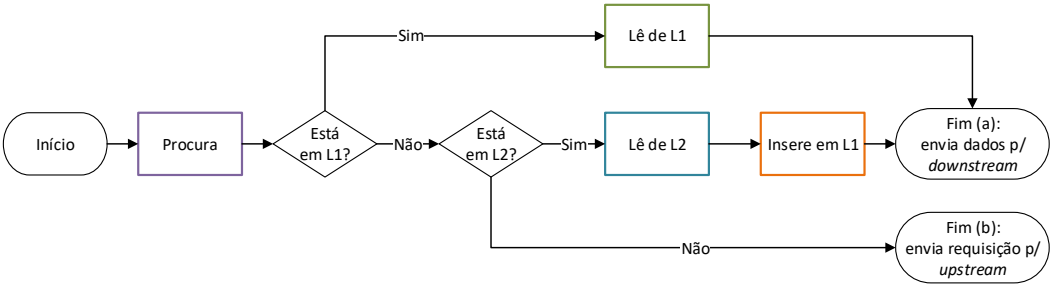
Para evitar medições que representem apenas tecnologias de memória SSD específicas e atuais, *emula-se* o hardware e os drivers da L2, abstraindo o SSD como um dispositivo de armazenamento com determinados tamanho e vazão. Isso significa também que neste trabalho, os recursos necessários para gerenciar a camada 2 de uma HCS, como operações de baixo nível em núcleos SSD, não são utilizados diretamente. Portanto, os resultados da avaliação de desempenho representam um limiar superior, na medida em que núcleos extras seriam necessários para completar o sistema HCS sob teste, por exemplo, para demultiplexar pacotes, pesquisar na PIT/FIB e gerenciar SSDs.

3.2 Objetivos do Sistema

Em suma, o objetivo de projeto do sistema investigado é *acoplar de maneira otimizada as hierarquias de memória* L1 e L2 considerando as suas taxas de transferência e tamanhos. Essa é uma tarefa desafiadora porque envolve aspectos tanto de hardware (por exemplo, taxa de leitura da L2, uso de múltiplos SSDs físicos em paralelo, velocidade de barramento) como de software (por exemplo, *driver* do SSD, gerenciamento de memória).

A Figura 3.2 apresenta um modelo conceitual de processamento de leitura da HCS. Quando um pacote de interesse é recebido pela cache, o respectivo conteúdo é procurado no índice. Em caso de acerto na L1, os dados são lidos da L1 e retornados para o roteador enviar para a rede (*downstream*). Em caso de falha na L1 e acerto na L2, o batch de dados que contém o dado solicitado é lido da L2 e os seus chunks são, então, armazenados na L1. Posteriormente, o dado solicitado é enviado para o roteador enviar para a rede (*downstream*). Em caso de falha na L2 (e na L1), uma requisição é enviada para rede (*upstream*).

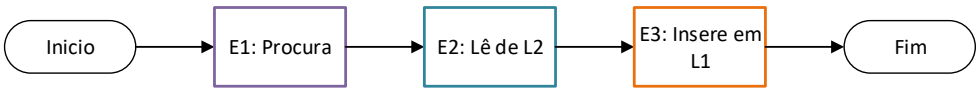
Figura 3.2: Fluxograma das operações de leitura de uma HCS



Fonte: o autor

O caminho mais longo do fluxograma apresentado na Figura 3.2 resulta dos casos onde ocorrem falha na L1 e acerto na L2. Portanto, essa sequência de operações representa o *caminho crítico* a ser otimizado para acoplar as memórias hierárquicas eficientemente e assim maximizar o rendimento do sistema. A Figura 3.3 apresenta o fluxograma do caminho crítico, que é composto por três estágios.

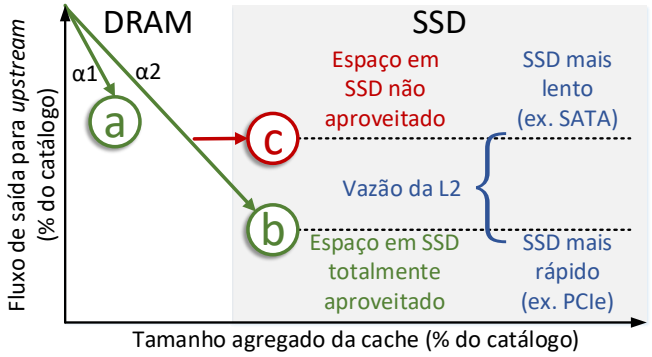
Figura 3.3: Fluxograma do caminho crítico da operação de leitura de uma HCS



Fonte: o autor

As implicações da otimização do caminho crítico são explicadas com ajuda da Figura 3.4. Ela mostra no eixo vertical o fluxo de requisições que deixam o roteador para *upstream* como função do tamanho da memória agregada (eixo horizontal) de um roteador recebendo pedidos na sua taxa de linha máxima. Note-se que valores próximos à origem (0,0) são preferíveis: conforme ilustrado pela Figura 2.7, o roteador evita encaminhamento de requisições para a rede (*forward*) em caso de acerto na *Content Store*, que tem custo proporcional ao seu tamanho.

Figura 3.4: Desempenho esperado de uma unidade de cache



Fonte: o autor

Duas regiões são destacadas e correspondem a diferentes tecnologias de memória (isto é, DRAM e SSD). Intuitivamente, quanto maior é a taxa de acerto (ou seja, o percentual do catálogo encontrado) na memória do roteador, menor o fluxo que sai dele para a rede (*upstream*). O gráfico apresenta, para fins de exemplo, duas linhas, com diferentes inclinações. A inclinação é afetada pela carga de trabalho (isto é, o tamanho percentual do catálogo e o parâmetro α da distribuição Zipf) e pela composição da memória do roteador (DRAM apenas, ou DRAM e SSD conjuntamente, lembrando a Tabela 1.1). Quando se utiliza um único nível de memória, baseado em DRAM (ou seja, que normalmente fornece uma taxa de acesso maior do que a taxa de linha dos roteadores), o fluxo de saída segue o declive até a capacidade da memória (por exemplo, o ponto ⓐ). Conforme será visto na Seção 4.2, aumentar o tamanho da cache de nível único para além do espaço disponível em memória DRAM resultaria em perda de desempenho significativa, que não compensaria a diminuição do fluxo de saída do roteador para a rede.

Por sua vez, quando um esquema de memória hierárquica (HCS) é utilizado, o fluxo de saída pode ou acompanhar a inclinação da curva (até um certo ponto (b)), ou ser limitado pela taxa de transferência da L2 (ponto (c)), resultando em "desperdício" de espaço da L2. O segundo caso ocorre quando os conteúdos não encontrados em L1 são localizados em L2 a uma taxa maior que a capacidade de leitura dela. A demanda de leitura de L2 depende linearmente da probabilidade de acerto em L2 que, por sua vez, depende da capacidade de armazenamento de L2. Consequentemente, o sistema opera no ponto operacional máximo (isto é, segue o declive) até que a taxa de operações resultantes em falha na L1 (que geram buscas na L2) exceda a taxa de leitura de L2. Após este ponto, o tamanho excedente de L2 não gera benefício porque não é possível ler o conteúdo na velocidade necessária. Por essa razão, é desejado operar em pontos tais como o (b) e evitar situações tais como a (c).

3.3 Arquiteturas para Paralelização de HCS

Para acoplar de maneira otimizada as hierarquias de memória, propõe-se a investigação de esquemas de paralelismo como componente arquitetural complementar de HCS. As plataformas de hardware atuais oferecem recursos de paralelismo que podem ser usadas para superar o principal gargalo do sistema – a taxa de transferência da L2.

O desempenho do sistema pode ser melhorado aumentando-se a quantidade média de execuções dos caminhos (em particular o crítico) por unidade de tempo. Esse efeito pode ser alcançado através das chamadas *escalabilidade vertical* e *escalabilidade horizontal*. A escalabilidade vertical diz respeito à diminuição da latência de cada estágio de processamento. Por exemplo, caso a latência média seja diminuída pela metade, o rendimento do sistema será duplicado. A latência do caminho crítico pode ser diminuída, por exemplo, melhorando-se o desempenho da L2 através do uso de tecnologias mais rápidas relacionadas ao SSD, como drivers, hardware e software. A escalabilidade horizontal, por sua vez, diz respeito ao emprego de paralelismo na execução dos estágios. Por exemplo, caso o número de caminhos executados por unidade de tempo seja multiplicado, o desempenho do sistema será aumentado na mesma proporção.

Para explorar a escalabilidade horizontal, ou, simplesmente, paralelismo, nós nos inspiramos em uma analogia aos conceitos de paralelismo em arquitetura de processadores. Nós assumimos que o caminho crítico é composto por três estágios de processamento e cada thread representa uma "unidade de processamento". Como resultado, nós propomos duas arquiteturas de processamento paralelo para HCS, denominadas Pipeline e Escalar, que são detalhadas nas próximas subseções.

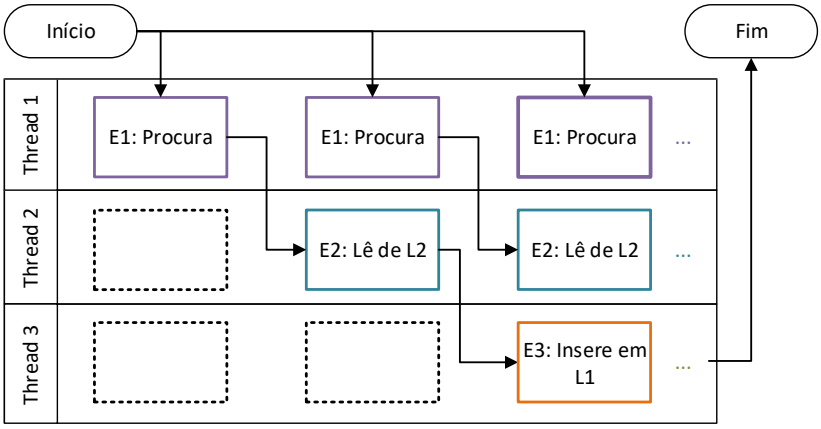
3.3.1 Arquitetura Pipeline

A arquitetura Pipeline aloca uma thread para cada estágio de processamento, como mostrado na Figura 3.5. Em teoria, esse modelo permitiria entregar, em média[1], um pacote a cada "ciclo" de processamento. Na prática, porém, alcançar esse desempenho pode ser desafiador devido ao gerenciamento de memória. Na arquitetura Pipeline, o espaço completo de memória é compartilhado entre as threads, o que gera a necessidade de um mecanismo de sincronização para negociar o acesso a uma única HCS.

A arquitetura Pipeline requer um conjunto de peças de software complementares. Primeiramente, podem ser necessários a implementação e o dimensionamento de *buffers*

[1] Ou seja, considerando que o tempo de preenchimento do pipeline tende a tornar-se neglível assintoticamente.

Figura 3.5: Fluxograma de arquitetura Pipeline com três threads



Fonte: o autor

entre cada estágio de processamento. Também é preciso tratar adequadamente casos de estouro de buffer e sincronismo entre as threads de processamento para garantir entrega na mesma ordem recebida. Finalmente, é imprescindível sincronizar as operações de leitura e escrita na L1 para evitar condições de disputa entre as várias threads.

3.3.2 Arquitetura Escalar

Na arquitetura Escalar, cada thread executa sequencialmente todas os estágios de processamento, como mostrado na Figura 3.6. Esse modelo permite entregar um pacote a cada três "ciclos" de processamento por thread de processamento.

Na arquitetura Escalar, o espaço de memória é dividido entre as threads, de maneira que cada thread gerencie uma HCS (parcial) independente. Assume-se que a placa de rede é capaz de realizar operações de *hash* em campos de cabeçalho não-IP. Essa premissa tem se tornado comum em projeto de roteadores baseados em múltiplos núcleos como, por exemplo, em (JAMSHED et al., 2012; KIM; CHOI; CHOI, 2014). A função hash garante que todos os chunks de um batch específico serão sempre tratados pela mesma thread de processamento. Assim, várias HCS podem executar operações de busca e inserção de uma forma paralelizada e livre de bloqueios sincronizantes.

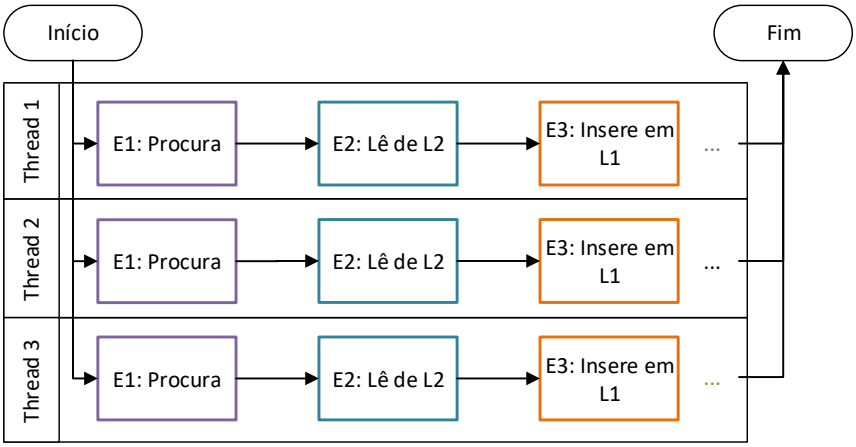
A arquitetura Escalar é mais simples de ser implementada que a arquitetura Pipeline. A divisão de carga de trabalho permite acesso concorrente livre de bloqueio sincronizante às duas camadas de memória. Além disso, como os estágios são executados sequencialmente, não há a necessidade de gerenciamento de buffers intermediários entre eles.

Como principal desvantagem, o esquema escalar requer um processo de divisão de carga entre as threads de processamento. Embora esse componente seja tipicamente encontrado em arquiteturas de roteadores atuais e tenha sido amplamente adotado em soluções de processamento de pacotes de alto desempenho, é importante observar que ele é crucial para o desempenho do esquema e pode se tornar o gargalo do sistema. Uma segunda desvantagem notável da arquitetura Escalar é que suas threads de processamento permanecem ociosas enquanto ocorrem operações de E/S da L2.

3.3.3 Discussão

Como será visto no Capítulo 4, resultados de uma série de avaliações indicam que a arquitetura Escalar é mais eficiente que a arquitetura Pipeline em termos de ganho de de-

Figura 3.6: Fluxograma de uma arquitetura Escalar com três threads



Fonte: o autor

sempenho por thread de processamento. Em suma, o fator crítico para o desempenho está na duração do segundo estágio (leitura de L2), que limita drasticamente o desempenho da arquitetura Pipeline.

3.4 Metodologia para Avaliação de HCS

Esta seção propõe uma metodologia para avaliação de desempenho de HCS e está organizada em três subseções, como segue. A subseção 3.4.1 apresenta o sistema implementado. A Subseção 3.4.2 discute as técnicas de emulação implementadas. Por fim, a Subseção 3.4.3 explica o procedimento de micro-avaliação

3.4.1 NFD-HCS

Uma versão da HCS, denominada NFD-HCS, foi implementada estendendo-se o NDN Forwarding Daemon (NFD). O NFD-HCS implementa as duas camadas de memória ($L1$ e $L2$) e suas operações ($L1.lookup$, $L1.insert$, $L2.read$ e $L2.insert$), como segue. A camada L1 instancia a classe de armazenamento de conteúdo do NFD, denominada Content Store (denotada neste trabalho por NFD-CS). Internamente, o NFD-CS emprega uma estrutura de dados *Skiplist* e a política de substituição FIFO[2].

A camada L2, por sua vez, também armazena os dados em DRAM, mas emula uma tecnologia de memória mais lenta (por exemplo, SSD). A memória mais lenta é emulada aguardando-se um determinado tempo antes de retornar os dados da DRAM, como será explicado em detalhes na próxima subseção, sobre técnicas de emulação.

O Algoritmo 3.1 mostra a operação de leitura do NFD-HCS. Ele primeiro procura ler um chunk da L1 (Linha 2). Em caso de acerto, o chunk correspondente é retornado (Linha 4); caso contrário, um batch contendo o chunk desejado é lido da L2 (Linha 8), e cada chunk do batch é inserido na L1 (Linha 10). Após a cópia da L2 para L1, o chunk solicitado é retornado (Linha 12).

[2]Mais precisamente, o NFD-CS emprega um conjunto de filas FIFO priorizadas, composto por três filas (dados solicitados, dados obsoletos e dados não solicitados). No entanto, neste estudo, todos os chunks passam necessariamente pela fila de dados requisitados, uma vez que não são considerados cenários que contemplam dados obsoletos e nem cenários que contemplam dados não solicitados.

Algoritmo 3.1: Leitura do NFD-HCS

```
// co:  índice do conteúdo
// ch:  índice do chunk
Entrada: name_{co,ch} ;                    // nome da peça solicitada
Saída: chunk_{co,ch} ;                              // peça solicitada
1  início
2      chunk_{co,ch} ← nulo; ;        // inicializa variável de retorno
3      chunk_{co,ch} ← leia chunk de nome name_{co,ch} da L1;
4      se chunk_{co,ch} = nulo então
           // variável ba:  índice do batch
           // constante B: tamanho do batch
5          ba ← trunca(ch/B);
6          batch_{co,ba} ← leia batch contendo name_{co,ch} da L2;
7          para cada chunk do batch_{co,ba} faça
8              insere chunk na L1;
9              se nome do chunk = name_{co,ch} então
10                 chunk_{co,ch} ← chunk
11             fim
12         fim
13     fim
14     retorna chunk_{co,ch};
15 fim
```

Fonte: o autor

O NFD-HCS possui seis parâmetros, como mostrado na Tabela 3.1: tamanho do chunk, tamanho do batch, tamanho da L1, tamanho da L2, taxa de transferência da L2 e grau de paralelismo. O primeiro parâmetro, tamanho do chunk, denotado por $|c|$[3], define a quantidade de dados que podem ser armazenados dentro de um chunk. Para esse parâmetro, adota-se o valor padrão atual do NFD ($|c| = 8$ KB).

O segundo parâmetro, tamanho do batch, denotado por B, define a quantidade de dados copiados da L2 para a L1, em cada operação $L2.read$. É importante observar que o tamanho do batch deve ser suficientemente grande para permitir que o SSD seja efetivamente acessado usando sua taxa de dados sequencial. Como esse aspecto não pode ser investigado através da metodologia emulacional, é necessário referenciar a literatura para escolher um valor adequado para o tamanho do batch. O valor $B = 10$ chunks foi escolhido com base nos seguintes dois argumentos. Primeiro, o estudo analítico apresentado em (ROSSINI et al., 2014) mostra que um tamanho de batch $B = 10$ chunks é operacionalmente efetivo. Segundo, os resultados experimentais do protótipo completo envolvendo um SSD real apresentado em (MANSILHA et al., 2015) mostram que os valores no intervalo $B \in [8, 16]$ chunks apresentam resultados satisfatórios.

O parâmetro tamanho da L1, denotado por $|L1|$, define o espaço de memória disponível na camada 1. O parâmetro tamanho da L2, denotado por $|L2|$, define o espaço de memória disponível na camada 2. Os parâmetros $|L1|$ e $|L2|$ têm importância fundamental para o presente estudo e, por isso, são amplamente investigados neste trabalho. Os intervalos de valores $L1 \in [10\,\text{MB}, 10\,\text{GB}]$ e $L2 \in [10\,\text{GB}, 1\,\text{TB}]$ são considerados para

[3] O tamanho é alternadamente expresso em termos de bits, bytes, ou pedaços, dependendo do contexto.

Tabela 3.1: Configurações do NFD-HCS

Significado	Parâmetro	Valores
Tamanho do Chunk	$\|c\|$	8 KB
Tamanho do Batch	B	10 chunks
Tamanho da L1	$\|L1\|$	[100 MB, 10 GB]
Tamanho da L2	$\|L2\|$	[10 GB, 1 TB]
Taxa de transferência da L2	τ_{L2}	[4, 64] Gbps
Grau de Paralalelismo	R	[1, 16] threads

Fonte: o autor

os tamanhos das camadas L1 e L2, respectivamente.

A taxa de transferência da L2, denotada por τ_{L2}, contempla conjuntamente tanto aspectos de hardware (como velocidades dos discos individuais), como de software (por exemplo, drivers de dispositivo). O valor de τ_{L2} resulta da duração da espera da L2, denotada por $d_{\text{L2.read}}$: para um batch de tamanho B chunks, cada um de tamanho $|c|$, tem-se:

$$d_{\text{L2.read}} = \frac{B|c|}{\tau_{L2}} \tag{3.1}$$

O valor da taxa de transferência da L2 foi variado no intervalo $\tau_{L2} \in [4, 64]$ Gbps. O limite inferior $\tau_{L2} = 4$ Gbps corresponde à taxa de leitura sequencial de uma unidade SSD atual, enquanto o limite superior $\tau_{L2} = 64$ Gbps corresponde à taxa de leitura máxima de oito unidades SSD atuais acessadas em paralelo. O limite superior foi definido com base na taxa máxima possível de ser emulada confiavelmente no ambiente de testes utilizado, conforme será mostrado na Subseção 4.3.1.

Por fim, o grau de paralelismo, denotado por R, define o número de threads lógicas instanciadas. No caso da arquitetura paralela Escalar, por exemplo, cada thread executa um NFD-HCS. O valor do grau de paralelismo é variado no intervalo $R \in [1, 16]$.

3.4.2 Emulação

O hardware da plataforma de testes restringe o intervalo de valores de parâmetros avaliados. Por exemplo, o tamanho agregado do NFD-HCS é limitado pela memória DRAM disponível. Para estender a gama de configurações do NFD-HCS possíveis de serem emuladas (por exemplo, avaliar um NFD-HCS com capacidade de armazenamento de 1 TB utilizando um sistema de testes com apenas 32 GB de DRAM), são propostas três técnicas de emulação, e duas *estratégias* alternativas para cada uma delas, totalizando seis estratégias de emulação. As técnicas, estratégias e respectivos compromissos são resumidos na Tabela 3.2 e detalhados em seguida.

Tabela 3.2: Técnicas de emulação, estratégias implementadas e seus respectivos compromissos

Técnica	Parâmetro	Estratégia	Prós	Contras
1. Alocação de Memória para L2	M	$Static$	Vazão de L2 mais alta	Tamanho de L2 menor
		$Dynamic$	Tamanho de L2 maior	Vazão de L2 mais baixa
2. Implementação do atraso da L2 emulada	D	$Sleep$	Menos precisa	Nenhuma sobrecarga
		$Busy$	Mais precisa	Sobrecarga
3. Entrada para função Hash	H	$Content$	Implementação mais simples	Desbalanceamento de carga
		$Batch$	Balanceamento	Implementação mais complexa

Fonte: o autor

Alocação de Memória para L2

A primeira técnica de emulação, cujo parâmetro é denotado por M, escolhe o momento em que o espaço de memória necessário para armazenar os dados lidos da L2 é alocado. Duas estratégias, $Static$ e $Dynamic$, são propostas a seguir.

Static. Esta estratégia aloca toda a memória necessária para armazenar os dados de L2, além de L1, durante a fase de instanciação do experimento (isto é, antes de iniciar a execução). Assim, os tamanhos de L1 e L2 são limitados pelo espaço disponível na DRAM, pois $|L1| + |L2| \leq |DRAM|$. Durante o experimento, os batches são encontrados usando índices de conteúdo e chunks, resultando em uma complexidade para operação $L2.read$ de ordem O(1).

Dynamic. Para estender o tamanho de L2 (para além da capacidade da memória DRAM), é possível alocar dinamicamente a memória necessária ao ler dados da L2. Seguindo essa intuição, a estratégia de alocação $Dynamic$ aloca um batch de dados dinamicamente a cada operação de leitura da L2, sob demanda. O batch de dados lido substitui algum outro batch armazenado em L1, que é então desalojado[4]. Seguindo essa estratégia, o uso de memória é limitado pelos tamanhos da L1 e do batch, resultando na restrição $|L1|$ + $R|c|B \leq |DRAM|$ (onde R é o grau de paralelismo, $|c|$ o tamanho do chunk e B o tamanho do batch). Portanto, a estratégia $Dynamic$ desassocia o tamanho L2 da quantidade de memória disponível na DRAM.

Comparativamente, a estratégia de alocação de memória dinâmica ($M = Dynamic$) permite avaliar cenários onde o tamanho do NFD-HCS é maior do que sua alternativa ($M = Static$). Por outro lado, a opção $M = Static$ permite avaliar cenários onde a vazão de L2 é maior que aquelas permitidas pela opção $M = Dynamic$. O motivo é que as operações de alocação/desalocação de memória da opção $M = Dynamic$ representam uma sobrecarga significativa de processamento, inexistente em $M = Static$. Além disso, essa sobrecarga possivelmente seria agravada em sistemas instanciados com alto grau de paralelismo. O par de estratégias $M \in \{Static, Dynamic\}$ é avaliado na Subseção 4.3.1.

Implementação do atraso da L2 emulada

A segunda técnica de emulação, cujo parâmetro é denotado por D, define o algoritmo usado para implementar o atraso da memória L2 emulada. Duas estratégias de implementação para essa técnica, $Sleep$ e $Busy$, são explicadas a seguir.

Sleep. A primeira estratégia realiza uma chamada de sistema *usleep()*. Essa estratégia é limitada pela granularidade de tempo da chamada do sistema (normalmente nano segundos em sistemas operacionais atuais), pela precisão de relógio (geralmente milissegundos em sistemas atuais) e pela sobrecarga imposta pela mudança de contexto no processador.

Busy. A segunda estratégia baseia-se em um laço de espera ocupada, conforme descrito no Algoritmo 3.2. O laço armazena o tempo do relógio (Linha 2) antes de ler o conteúdo da L2 (Linha 3). Após a leitura, inicia-se o laço de espera ocupada, no

[4]O NFD (e, consequentemente, a implementação da HCS proposta nesta tese) usa um tipo especial de ponteiro de memória C++ que desaloja espaço de memória quando o seu contador de referências atinge zero.

qual o relógio é verificado a cada iteração (Linha 5) até que o atraso desejado seja atingido (Linha 6).

Algoritmo 3.2: Atraso da L2 via espera ocupada ($D = Busy$)

```
// co:  índice do conteúdo
// ch:  índice do chunk
Entrada: name_{co,ch} ;            // nome da peça solicitada
Saída: batch_{co,ch} ;  // batch que contém a peça solicitada
1 início
2     define time_start;
3     batch_{co,ch} ← leia batch contendo name_{co,ch} da L2;
      // constante D_{L2.read}:  latência emulada
4     faça
5         atualiza time_current;
6     enquanto (time_current − time_start) > D_{L2.read} ;
7     retorna batch_{co,ch};
8 fim
```

Fonte: o autor

Em comparação com a estratégia $D = Sleep$, a estratégia $D = Busy$ apresenta como desvantagem o consumo de ciclos de CPU que podem ser úteis para outras operações executadas em paralelo (por exemplo, em cenários onde é considerado paralelismo, $R > 1$). Como vantagem, a estratégia $D = Busy$ fornece resultados mais conservadores e precisos do que caso fosse usada uma chamada de sistema. O par de estratégias $D \in \{Sleep, Busy\}$ é avaliado na Subseção 4.3.1.

Entrada para a função Hash

A terceira e última técnica de emulação, cujo parâmetro é denotado por H, define a entrada para a função hash empregada para realizar a divisão de solicitações entre as múltiplas threads de execução na arquitetura paralela Escalar. Duas estratégias, $Content$ e $Batch$, são explicadas a seguir.

Content. A primeira estratégia usa os *nomes dos conteúdos* como entrada para a função hash. Essa estratégia tende a gerar desbalanceamento de carga entre as threads. Como a popularidade dos conteúdos normalmente segue uma distribuição Zipf, algumas threads podem ficar responsáveis pelos conteúdos mais populares, enquanto as outras ficam responsáveis por conteúdos menos populares. A função hash baseada em conteúdo é implementada conforme o Algoritmo 3.4.

Algoritmo 3.3: Função Hash por Conteúdo ($H = Content$)

```
// co:  índice do conteúdo
// ch:  índice do chunk
Entrada: name_{co,ch} ;            // nome da peça solicitada
Saída: thread ;      // índice da thread de processamento
1 início
    // constante R: grau de paralelismo
2   retorna co % R;
3 fim
```

Fonte: o autor

Batch. A segunda estratégia utiliza um *identificador de batch* como entrada para a função hash. A função hash baseada em batch é apresentada no Algoritmo 3.4. Nesse caso, o balanceamento de carga não é afetado pela distribuição Zipf porque os batches de um mesmo conteúdo popular são distribuídos entre threads distintas.

Algoritmo 3.4: Função Hash por Batch ($H = Batch$)

```
// co:  índice do conteúdo
// ch:  índice do chunk
Entrada: name_{co,ch} ;            // nome da peça solicitada
Saída: thread ;      // índice da thread de processamento
1 início
    // constante B: tamanho do batch
    // constante R: grau de paralelismo
2   retorna ( co + trunca(ch/B) ) % R;
3 fim
```

Fonte: o autor

A estratégia $H = Content$ é mais simples do que a estratégia $H = Batch$ porque requer duas operações matemáticas a menos, um detalhe que pode ser não negligível em taxa de linha. A opção $H = Batch$, embora mais "complexa", tende a mitigar o problema de balanceamento de carga. O impacto das estratégias $H \in \{Content, Batch\}$ no balanceamento de carga e o impacto do balanceamento de carga no desempenho do sistema são avaliados na Subseção 4.4.3.

3.4.3 Sistema de Micro-avaliação

Para realizar as medições, foi desenvolvido um sistema de micro-avaliação para o NFD composto por dois módulos de propósito específico. Os módulos, um para a arquitetura de única thread (Algoritmo 3.5) e outro para a arquitetura de múltiplas threads (Algoritmo 3.6), incluem apenas as funcionalidades necessárias para avaliação.

Os módulos recebem como entrada um componente de interesse (por exemplo NFD-CS ou NFD-HCS) e uma carga de trabalho. A carga de trabalho é instanciada antes da medição para evitar interferências nos resultados.

Algoritmo 3.5: Micro-avaliação de arquiteturas de única thread

Entrada: $component$, $workload$
Saída: performance $iops$

```
1 início
2     atualiza time_start;
3     para cada content_name do workload faça
4         procura content_name no component;
5     fim
6     atualiza time_end;
7     retorna tamanho(workload) / (time_end − time_start);
8 fim
```

Fonte: o autor

Quando uma arquitetura paralela Escalar é avaliada, a carga de trabalho é distribuída entre as threads antes da medição. Conforme explicado anteriormente, assume-se que a distribuição da carga de trabalho entre as threads é realizada pela NIC (Seção 3.3.2), e que esse componente é externo ao escopo do estudo (Seção 3.1).

Ambos os módulos de micro-avaliação medem o tempo que um sistema de cache leva para processar uma determinada carga de trabalho como métrica de desempenho principal. Além disso, os módulos medem três recursos da plataforma subjacente[5] para revelar possíveis efeitos colaterais. Quando avaliam um NFD-HCS, os módulos também registram a taxa de acertos na L1 para fins de validação do sistema.

Uma carga de trabalho pode ser *fatiada* em partes menores para economizar memória DRAM e, assim, maximizar o tamanho de NFD-HCS possíveis de serem emuladas. Isso ocorre quando a carga de trabalho é maior do que o tamanho definido da fatia (ou seja, $O > F$, onde F representa o tamanho da fatia). Nesse caso, cada fatia da carga de trabalho é processada sequencialmente, em um laço (que foi omitido do Algoritmo 3.5 e do Algoritmo 3.6 para fins de simplificação). A cada iteração do laço, os dados referentes a uma fatia em particular são coletados e somados em totalizadores. Entre cada iteração, o contador do relógio é pausado. Ao final do laço, as métricas são calculadas como nos outros casos (onde $O \leq F$).

[5]Os módulos utilizam a ferramenta GNU Time (Linux User's Manual, 2016) (uso de CPU, uso de memória e número de páginas faltantes).

Algoritmo 3.6: Micro-avaliação de arquiteturas de múltiplas threads

```
// component:  componente que será avaliado
// workload:  carga de trabalho que será utilizada
Entrada: component, workload
// iops:  taxa de operações
Saída: iops
1  início
       // constante R: grau de paralelismo
       // variável i:  índice
       // variável workloads:  lista de cargas de trabalho
2      workloads ← carga de trabalho [0, 1, ..., R − 1];
       // variável threads:  lista de threads
3      threads ← thread [0, 1, ..., R − 1];
4      para cada thread em threads faça
5          para cada content_name em workload faça
6              i ← função_hash(content_name);
7              adiciona content_name na workloads[i];
8          fim
9      fim
10     atualiza time_start;
11     i ← 0;
12     para cada thread em threads faça
13         thread ← inicia thread(micro-avaliação component, workloads[i);
14         i ← i + 1;
15     fim
16     para cada thread em threads faça
17         aguarda término da thread;
18     fim
19     atualiza time_end;
20     retorna tamanho(workload) / (time_end − time_start);
21 fim
```

Fonte: o autor

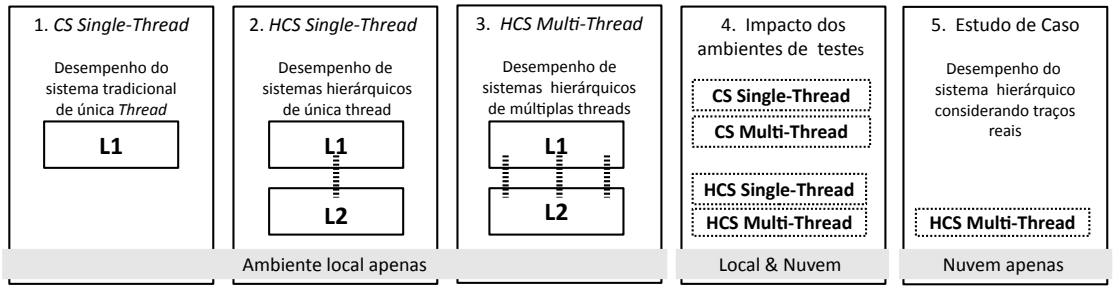
O tamanho da fatia deve obedecer dois limiares. No limite superior, uma fatia com tamanho igual ao da carga de trabalho ($F = O$) oferece nenhum benefício. No limite inferior, uma fatia deve gerar um ciclo de operações que resultem em dados de desempenho estatisticamente válidos.

O tamanho da fatia de carga de trabalho foi definido como $F = 10\,\text{GB}$. Com base em uma série de experimentos de propósito específico, concluiu-se que o tamanho estipulado permitiria executar os experimentos planejados e não geraria interferências nos resultados.

4 AVALIAÇÃO

Este capítulo apresenta as avaliações que são o cerne desta tese e está organizado em sete seções, como segue. A Seção 4.1 discute os parâmetros das avaliações. As avaliações estão organizadas em cinco grupos, conforme ilustrado pela Figura 4.1. Os dois primeiros grupos consideram sistemas com uma única thread de processamento. Especificamente, o primeiro grupo (Seção 4.2) mede o desempenho de dois componentes de interesse do NFD, considerando sua implementação original. O segundo grupo de avaliações (Seção 4.3) mede o desempenho do sistema hierárquico e o impacto dos parâmetros tamanho e vazão da L2 no sistema, após validar as técnicas de emulação e o sistema implementado. O terceiro grupo de avaliações (Seção 4.4) investiga as arquiteturas paralelizadas, Pipeline e Escalar. O quarto grupo de avaliações (Seção 4.5) mede o impacto do ambiente de testes nos resultados obtidos pela emulação. O quinto e último grupo (Seção 4.6) apresenta um estudo de caso considerando um traço real, onde foco é medir o desempenho da arquitetura Escalar em um cenário particular – e não compreender os fenômenos subjacentes. Por fim, a Seção 4.7 apresenta uma análise crítica dos resultados e suas limitações.

Figura 4.1: Visão geral das avaliações



Fonte: o autor

4.1 Parâmetros das Avaliações

Duas plataformas de hardware, cujas especificações são apresentadas na Tabela 4.1, foram usadas no estudo: uma máquina local e uma máquina instanciada na nuvem. A plataforma local oferece um ambiente completamente controlado e tem como vantagem permitir o ajuste de opções como *hyper-threading*. Por outro lado, o seu poder de computação e a capacidade de memória são relativamente limitados. A segunda plataforma é uma máquina virtual hospedada em um servidor de nuvem pública. A máquina instanciada na nuvem, além de possuir maior capacidade de processamento e armazenamento, facilita a reprodução da avaliação por outros investigadores com acesso à mesma nuvem.

É importante observar que, por se tratar de uma nuvem pública, os experimentos executados nela compartilham recursos e, portanto, estão sujeitos a interferências. No entanto, resultados estatisticamente válidos podem ser extraídos a partir de múltiplas repetições de cada experimento.

Tabela 4.1: Especificações de Hardware

Plataforma	**Parâmetro**	**Valor**
Local	CPU	1,90 GHz Intel E52420
	NUMA	1 nodo, 6 núcleos
	DRAM	32 GB - 1.333 MHz (0,8 ns)
	Opções	CPU Gov., Hyper-threading
Nuvem (Microsoft Azure G3)	CPU	2,00 GHz Intel E52698B
	NUMA	1 nodo, 8 núcleos
	DRAM	112 GB (frequência desconhecida)
	Opções	Nenhuma

Fonte: o autor

As duas máquinas rodam um conjunto igual de softwares. Esse conjunto é listado na Tabela 4.2.

Tabela 4.2: Especificações de Software

Grupo	**Nome**	**Versão**
Sistema Operacional	Distribuição	Ubuntu 12.04 LTS
	Kernel	Linux 3.14.21
Compilador	gcc	4.7.3
Bibliotecas de Apoio	Boost	1.54
	Crypto++	5.6.2
Bibliotecas Específicas	NDN	0.3.1
	NFD	0.3.1

Fonte: o autor

O estudo considera três tipos de carga de trabalho sintéticas e uma carga de trabalho baseada em traço real. As cargas sintéticas facilitam a generalização, reprodução e comparação de resultados. A carga de trabalho baseada em traço real, por sua vez, permite testar o sistema em um caso relativamente mais fidedigno, embora mais específico. As cargas de trabalho sintéticas e a carga de trabalho real são detalhadas a seguir.

A Tabela 4.3 resume os parâmetros das cargas de trabalho sintéticas. Todos os conteúdos apresentam um mesmo tamanho, igual a 10,25 MB (ou 1.280 chunks, cada um contendo 8 KB), à exemplo de (ROSSINI et al., 2014). Observe-se que foram definidas as seguintes cargas de trabalho: sequencial (Seq), uniformemente aleatória ($Unif$) e realística ($Real$). As cargas de trabalho Seq e $Unif$ foram incluídas como melhor e pior casos, respectivamente, para fins de referência. Além disso, essas cargas de trabalho são particularmente importantes para o processo de validação do sistema: como será visto na próxima seção, a modelagem do desempenho do sistema alimentado com essas cargas de trabalhos é trivial. Na primeira carga de trabalho (Seq), cada chunk de cada conteúdo é solicitado sequencialmente. Na segunda carga de trabalho ($Unif$), os chunks são escolhidos aleatoriamente seguindo uma probabilidade uniforme. A terceira carga de trabalho ($Real$) pretende reproduzir cenários de uso típico. Nessa carga de trabalho, os conteúdos são sorteados seguindo uma distribuição de popularidade Zipf com parâmetro α. As requisições do primeiro chunk chegam de acordo com um processo de Poisson com taxa λ. Os chunks subsequentes seguem a taxa de *streaming* e são periodicamente espaçados (ou seja, interferências da rede, como congestionamento, não são consideradas).

Tabela 4.3: Parâmetros das cargas de trabalho sintéticas (tamanho de chunk $|c| = 8$ KB)

Carga de trabalho (W)	**Significado**	**Parâmetro**	**Valor**
Todas sintéticas (Seq, $Unif$ e $Real$)	Tamanho do conteúdo		10,25 MB (1.280 chunks)
	Tamanho da carga de trabalho	O	(até) 1,1 TB
	Tamanho do catálogo	N	(até) 1 TB
Apenas sintética $Real$	Taxa do conteúdo		512 Kbps (8 chunks/seg)
	Taxa de chegada de requisições	λ	1 requisição/seg
	Parâmetro da distribuição Zipf	α	1,0

Fonte: o autor

O tamanho do catálogo, denotado por N, varia conforme o cenário investigado. Propositalmente, o tamanho do catálogo é atrelado ao tamanho do nível de cache mais baixo do sistema considerado. Ou seja, $N = |L2|$ se a unidade de cache avaliada possuir duas camadas de memória e $N = |L1|$, caso contrário. Assim, é possível focar o estudo no desempenho das operações principais do NFD envolvendo o componente de cache (por exemplo, taxas de leitura e escrita), evitando outras operações que não encontrar objeto em cache obrigatoriamente implicaria (tais como gerenciamento da PIT, pesquisa na FIB). É importante ressaltar que, conforme explicado anteriormente (ver discussão relacionada à Figura 3.3), o cenário mais estressante para uma HCS é aquele no qual a taxa de acertos na L2 é maxima (já que cada acerto na L2 resulta em operações de transferência da L2 para L1). Portanto, o cenário onde $N = |L2|$ embora não seja realístico no que se refere à taxa de acertos de cache (que não é o foco deste trabalho), permite uma avaliação que representa o *pior caso* para a capacidade de leitura da HCS (foco desta tese).

O tamanho da carga de trabalho, denotado por O, também depende do componente a ser avaliado. Ao avaliar uma unidade de cache de camada única, nós estipulamos ao tamanho da carga de trabalho um valor igual ao tamanho do catálogo, o que resulta em $O = N$. Ao avaliar uma unidade de cache hierárquica, nós estendemos a carga de trabalho para preencher a L1 antes de iniciar a avaliação ("arranque a quente"). O tamanho da carga de aquecimento é definido empiricamente como sendo igual a dez vezes o tamanho da L1. Portanto, em caso de avaliação de HCS, o tamanho da carga de trabalho é determinado por $O = N + 10|L1|$.

A carga de trabalho baseada em traço real ($Trace$), por sua vez, consiste em requisições de conteúdo da Web coletados pelo IBM T.J. Watson Research Center (ZERFOS et al., 2013). O traço foi analisado analiticamente e os parâmetros da carga de trabalho baseada nele são resumidos na Tabela 4.4.

A carga de trabalho $Trace$ apresenta duas diferenças básicas em relação às cargas de trabalho sintéticas. Primeiro, os tamanhos dos conteúdos variam entre si (em contraste, o tamanho dos conteúdos das cargas sintéticas é fixado em 10 MB). As requisições foram filtradas adotando-se como critério um tamanho mínimo de conteúdo igual ao tamanho de um batch (80 KB). Dessa forma, nós consideramos apenas os casos onde uma L2 poderia ser acessada usando a taxa de leitura sequencial máxima do SSD. A segunda diferença é que as requisições dos chunks de um mesmo conteúdo não são periodicamente espaçadas, já que essa informação não está disponível no traço. A ausência de espaçamento reflete aplicações de download (em contraste, as cargas sintéticas representam aplicações de streaming com taxa de 512 Kbps).

Todos os resultados experimentais foram coletados após cinco rodadas executadas com sementes aleatórias distintas. Os resultados são mostrados com intervalo de confiança de 95 % (distribuição t-student com 4 graus de liberdade).

Tabela 4.4: Análise do traço (assumindo tamanho de chunk $|c| = 8\,\text{KB}$)

Carga de trabalho (W)	**Significado**	**Parâmetro**	**Valor**	
	Tamanho da carga de trabalho	O	44,61 GB	(5.577.176 chunks)
	Tamanho do catálogo	N	2,46 GB	(307.863 chunks)
	Tamanho mínimo dos conteúdos		80 KB	(10 chunks)
	Tamanho máximo dos conteúdos		134,61 MB	(16.827 chunks)
$Trace$	Tamanho médio dos conteúdos		64,81 MB	(8.101 chunks)
	Desvio padrão do tamanho dos conteúdos		25,62 MB	(3.203 chunks)
	Tamanho médio das requisições		0,41 MB	(51 chunks)
	Desvio padrão do tamanho das requisições		1,78 MB	(222 chunks)
	Distribuição Zipf	α	0,65	

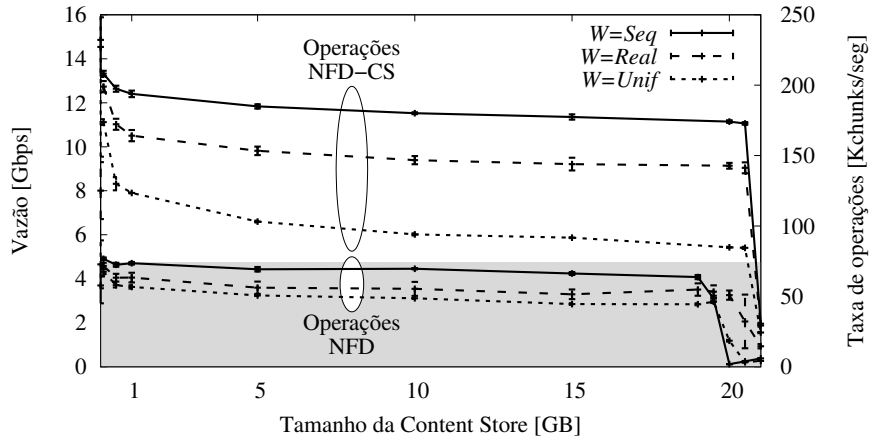
Fonte: o autor

4.2 Desempenho do Sistema Tradicional

A primeira avaliação mede o desempenho do roteador (NFD) e do componente *Content Store* (NFD-CS), executados em uma thread única. O objetivo é medir a capacidade do sistema original, antes de avaliar os aperfeiçoamentos considerados (ou seja, a hierarquia de memórias e o paralelismo).

A Figura 4.2 apresenta o desempenho dos dois sistemas considerados (NFD e NFD-CS) como função do tamanho da cache (que foi variado entre 100 MB e 23 GB). O desempenho é expresso em termos de vazão (à esquerda) e taxa de leitura de chunks (à direita). Duas famílias de três curvas (uma para cada carga de trabalho) são mostradas: uma família para o NFD-CS (superior) e outra para o NFD (inferior, sombreada).

Figura 4.2: Desempenho de dois componentes do NFD em função do tamanho da Content Store



Fonte: o autor

Comparando-se as duas famílias, observa-se que o NFD (roteador completo) impõe sobrecarga significativa em relação ao NFD-CS isolado (apenas cache). O ganho permitido por uma reengenharia do NFD-CS pode ser perdido pelas outras operações do NFD. Observa-se também que as propriedades estatísticas dos processos de chegada das cargas de trabalho impactam significativamente no rendimento dos componentes. O comportamento consistente nas duas famílias de curvas confirma a escolha da carga de trabalho $Real$ como um caso intermediário entre as cargas de trabalho Seq (melhor caso) e $Unif$ (pior caso). Além disso, note-se que, para as todas as curvas, o desempenho é bi-modal. Há um grande planalto quando o tamanho da Content Store é menor que $\approx 20\,\text{GB}$. Após

esse limiar, o desempenho cai drasticamente devido ao gerenciamento de memória do sistema operacional, conforme explicado a seguir.

A Figura 4.3 apresenta o impacto do tamanho do NFD-CS em três recursos do sistema: memória, número de páginas faltantes e uso da CPU. As três cargas de trabalho sintéticas foram avaliadas e retornaram resultados semelhantes porque o consumo de recursos depende essencialmente do tamanho do sistema – e não da maneira como ele é utilizado.

Observe-se na Figura 4.3(a) que o uso de memória cresce linearmente como função do tamanho da Content Store até atingir a capacidade disponível no sistema. Além disso, note-se que o crescimento do uso de memória é mais agudo do que o crescimento do tamanho da Content Store. Esse fenômeno se deve ao uso de memória para armazenar a carga de trabalho. É importante lembrar que o tamanho da carga de trabalho é proporcional ao tamanho da cache e que a carga de trabalho é "fatiada" em partes com tamanho máximo de 10 GB. Por essas razões, o espaço de 32 GB de memória disponível na plataforma de testes é esgotado com uma Content Store de apenas 22 GB.

A Figura 4.3(b) mostra que a partir do limiar onde a memória utilizada alcança a memória disponível, o número de páginas faltantes explode. A Figura 4.3(c) mostra que, a partir do mesmo limiar de saturação de memória, o uso da CPU decresce para cerca de $\approx 70\,\%$. Em suma, quando o tamanho da cache ultrapassa o espaço disponível na memória principal, a parte excedente é armazenada em disco, gerando falta de páginas em memória, o que reduz o uso da CPU e, consequentemente, o desempenho do NFD-CS.

A Tabela 4.5 apresenta os resultados da correlação de Pearson entre o fator avaliado (tamanho da cache) e as métricas monitoradas (taxa de operações, utilização de memória, número de páginas faltantes e uso de CPU). Por razões de simplicidade, é apresentada apenas a análise da carga de trabalho $W = Real$. Contudo, é importante mencionar que as outras cargas de trabalho sintéticas geraram resultados similares. Como esperado, observa-se uma correlação positiva entre tamanho da cache, uso de memória e número de páginas faltantes. Note-se, também, uma correlação negativa entre tamanho da cache, taxa de leitura e uso de CPU.

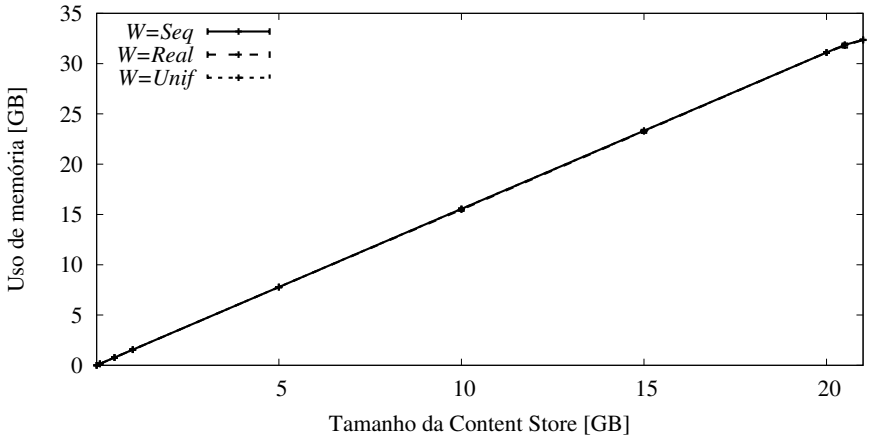
Tabela 4.5: Correlação entre o tamanho do NFD-CS e uso de recursos ($W = Real$)

	Tamanho do NFD-CS	**Taxa de operações**	**Uso de memória**	**Páginas faltantes**	**Uso de CPU**
Tamanho do NFD-CS	1,00	−0,70	1,00	0,58	−0,58
Taxa de Leitura	−0,70	1,00	−0,69	−0,42	0,39
Uso de memória	1,00	−0,69	1,00	0,52	−0,52
Falta de página	0,58	−0,42	0,52	1,00	−0,95
Uso de CPU	−0,58	0,39	−0,52	−0,95	1,00

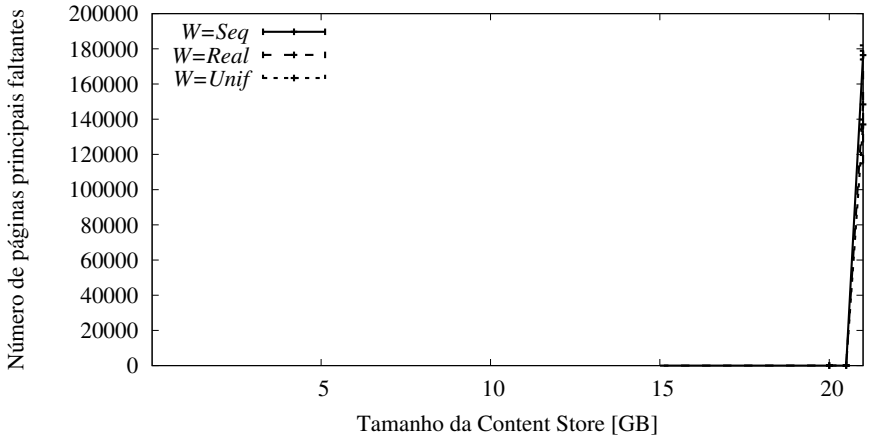
Fonte: o autor

Em resumo, os resultados dessa avaliação demonstram que uma CS de único nível pode escalar bem até atingir o espaço disponível na memória principal. A partir desse limiar, o gerenciamento de memória nativo do sistema operacional pode mover o gargalo da CPU para E/S (mesmo para tamanhos relativamente pequenos de CS) e tornar o desempenho do sistema instável. Esses dois fatores motivam o desenvolvimento de uma CS baseada em memórias hierárquicas como a investigada nesta tese.

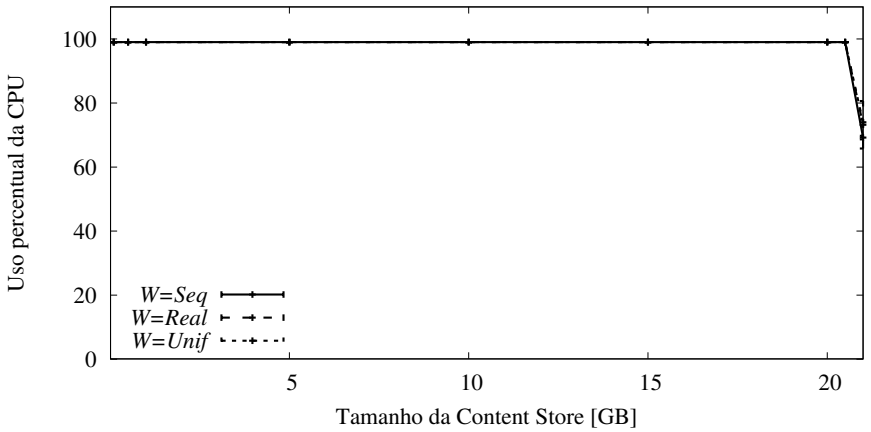
Figura 4.3: Impacto do tamanho do NFD-CS em três recursos do sistema



(a) Uso de memória



(b) Número de páginas faltantes



(c) Uso de CPU

Fonte: o autor

4.3 Sistemas Hierárquicos de Única *Thread*

Esta seção está organizada em quatro subseções. A Subseção 4.3.1 valida as técnicas de emulação para sistemas hierárquicos. A Subseção 4.3.2 valida os resultados obtidos pelo sistema hierárquico emulado e mede o desempenho do NFD-HCS executando em uma única thread. A Subseção 4.3.3 avalia o impacto de dois fatores relacionados à L2 (vazão e tamanho) no desempenho do sistema. Por fim, a Subseção 4.3.4 discute os resultados.

4.3.1 Validação das Técnicas de Emulação de Sistemas Hieráquicos

O objetivo desta avaliação é validar as técnicas propostas para emular sistemas hierárquicos. Primeiramente, foram introduzidas duas técnicas de emulação de atraso $D \in \{Busy, Sleep\}$ para controlar a taxa de dados externa τ_{L2} dos dispositivos L2 emulados. Em segundo lugar, foram propostos dois métodos de alocação de memória para L2, $M \in \{Static, Dynamic\}$. Nesta avaliação, considera-se a carga de trabalho $W = Real$ e um conjunto de configurações de componentes L2 que resulta da combinação das técnicas para implementação de atraso e alocação de memória. O tamanho do NFD-HCS foi fixado em $|L1| = 1$ GB e $|L2| = 10$ GB. A vazão da L2 foi variada no intervalo de valores $\tau_{L2} \in [1, 64]$ Gbps.

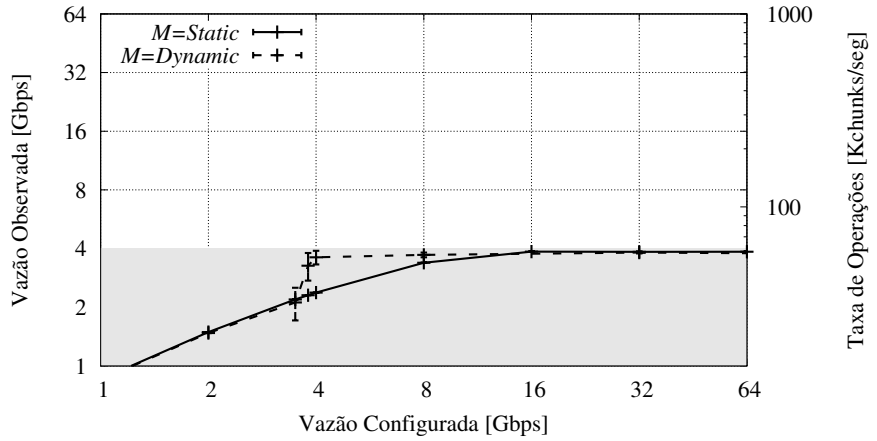
A Figura 4.4 mostra a vazão observada da L2 como função da vazão configurada da L2 (mostrada em escala logarítmica). Portanto, o resultado esperado (ou seja, que demonstra que a combinação de técnicas de emulação é capaz de sustentar a vazão estipulada) é uma função identidade ($f(x) = x$), para qualquer par de parâmetros (M, D). Para fins de referência, os gráficos apresentam uma região sombreada, a qual corresponde ao desempenho do NFD tradicional, conforme avaliado na seção anterior. Caso a vazão da HCS seja igual ou maior do que a região sombreada, conclui-se que as técnicas de emulação não representariam um gargalo para o roteador completo.

Observe-se na Figura 4.4(a) que a implementação de espera $D = Sleep$ oferece resultados insatisfatórios (independentemente da técnica de alocação de memória M). Pode-se concluir que não é factível realizar chamadas de sistemas em frequências tão altas como as necessárias para este estudo. Consequentemente, a técnica de espera $D = Busy$ foi selecionada para emular a vazão da L2 no restante desta tese. Salienta-se que essa escolha oferece resultados conservadores: o núcleo de CPU utilizado durante espera ocupada poderia ser utilizado por outras threads de processamento caso fosse utilizada uma de chamada de sistema. Esse fator é particularmente relevante em avaliações de arquiteturas paralelas, como aquelas apresentadas na próxima seção.

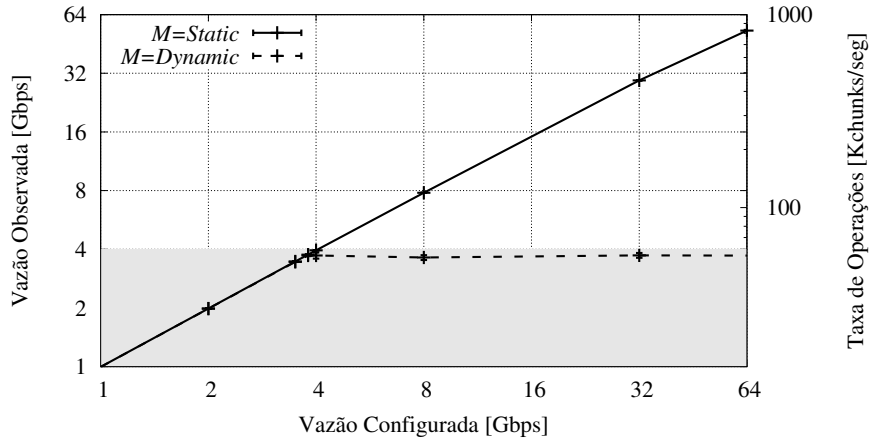
Note-se na Figura 4.4(b) que a técnica $D = Busy$ permite emular de forma confiável a vazão de L2 até $\tau_{L2} \approx 64$ Gbps somente quando combinada com a técnica de alocação de memória $M = Static$. A taxa de transferência satura em cerca de $\tau_{L2} \approx 4$ Gbps quando é utilizada a técnica de alocação de memória $M = Dynamic$.

Portanto, o par de técnicas $(D, M) = (Busy, Static)$ permite emular L2 com características praticamente irrestritas em termos de vazão, porém com tamanho limitado (pela quantidade de memória DRAM disponível). Por sua vez, o par $(D, M) = (Busy, Dynamic)$ permite emular L2 com características de tamanho irrestritos, porém com vazão limitada (mais precisamente, cerca de 4 Gbps). No que se segue, utiliza-se uma das duas combinações acima, dependendo do parâmetro sob investigação. Por uma questão de simplificação, o par $(D, M) = (Busy, Static)$ é adotado como configuração padrão, salvo indicação distinta, expressada explicitamente.

Figura 4.4: Precisão das técnicas de alocação de memória para L2 $M = \{Static, Dynamic\}$ e espera emulada da L2 $D \in \{Sleep, Busy\}$ ($|L1| = 1\,\text{GB}$, $|L2| = 10\,\text{GB}$, $W = Real$)



(a) $D = Sleep$



(b) $D = Busy$

Fonte: o autor

4.3.2 Validação do Sistema Hierárquico de Única Thread

O desempenho do sistema hierárquico de única thread é validado por uma metodologia complementar, especificamente a modelagem analítica. Intuitivamente, o desempenho da memória hierárquica é afetado, fundamentalmente, pela taxa de acertos na L1. Por isso, a apresentação do modelo analítico é iniciada pela probabilidade (P) de ocorrer acerto na L1 para as três cargas de trabalho consideradas. Para a carga de trabalho uniformemente aleatória ($Unif$), a probabilidade de acerto na L1 é estimada trivialmente como a fração do catálogo de chunks que cabe em L1:

$$\mathbb{E}[P_{\text{Unif}}] = |L1|/C \tag{4.1}$$

Para a carga de trabalho sequencial (Seq), em geral, a requisição do primeiro chunk de um batch sempre resulta em falha na L1, enquanto as requisições para os chunks subsequentes do mesmo batch sempre resultam em acerto devido ao processo de transferência de chunks agrupados em batch. No caso particular onde $L1 = L2$, a probabilidade de acerto é 1. Assim, a probabilidade de acerto na L1 para a carga de trabalho Seq é trivial-

mente estimada por:

$$\mathbb{E}[P_{\text{Seq}}] = \begin{cases} 1, \text{ se } |L1| = |L2| \\ 1 - 1/B, \text{ outros casos} \end{cases} \tag{4.2}$$

No caso da carga de trabalho $Real$, o primeiro chunk de um batch é encontrado em L1 com uma probabilidade P_{Real}^{1st}. Os outros chunks do mesmo batch são encontrados em L1 seguindo a intuição do caso anterior. Assim, a probabilidade de acerto em L1 para a carga de trabalho realística é estimada por:

$$\mathbb{E}[P_{Real}] = \frac{P_{real}^{1st}}{B} + \left(1 - \frac{1}{B}\right) \tag{4.3}$$

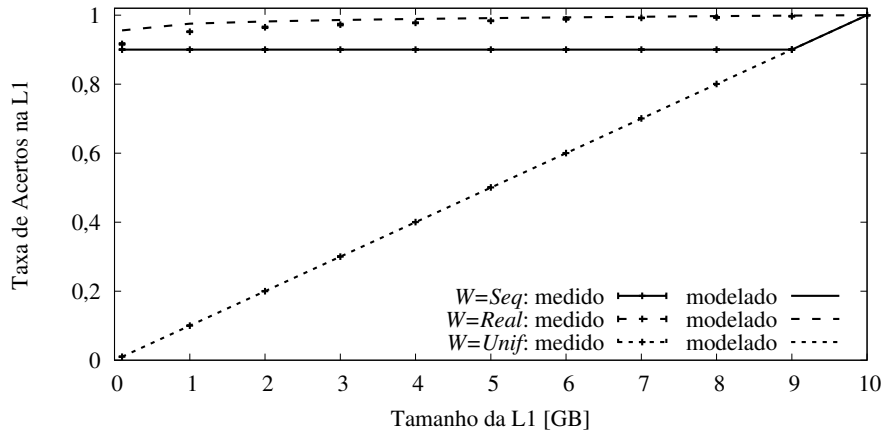
O valor de P_{Real}^{1st} pode ser estimado numericamente usando a extensão da aproximação de Che para cache FIFO (MARTINA; GARETTO; LEONARDI, 2014), explicada na Seção 2.3, considerando $C = |L1|$.

$$P_{Real}^{1st} = P_{FIFO} \tag{4.4}$$

A avaliação é baseada no seguinte cenário. O NFD-HCS foi configurado com valores fixos de tamanho da $|L2| = 10\,\text{GB}$ e de taxa de leitura da L2 $\tau_{L2} = 4\,\text{Gbps}$. O tamanho da L1 foi variado na faixa entre $|L1| \in [100\,\text{MB}, 10\,\text{GB}]$. Essas instâncias do NFD-HCS foram alimentadas com as três cargas de trabalho sintéticas.

A Figura 4.5 apresenta as taxas de acertos na L1 resultantes da emulação e do modelo analítico. A superposição entre os resultados da emulação e do modelo analítico, para cada uma das três cargas de trabalho, valida o sistema implementado de maneira visual. Analiticamente, foram obtidos valores baixos para a Raiz do Erro Quadrático Médio (REQM) das três cargas de trabalho: $REQM_{Seq} = 0{,}0000$, $REQM_{Real} = 0{,}0127$ e $REQM_{Unif} = 0{,}0004$.

Figura 4.5: Taxa de acerto na L1: comparação entre medições e modelo analítico ($|L2| = 10\,\text{GB}$, $\tau_{L2} = 4\,\text{Gbps}$, $M = Static$, $D = Busy$)



Fonte: o autor

O passo seguinte à validação do NFD-HCS é a avaliação do seu desempenho. O desempenho do sistema foi estimado por meio de modelagem matemática. Isso porque considerou-se que qualquer instrumentação do código do NFD para medir o tempo de duração das suas operações ($d_{\text{L1.lookup}}$ e $d_{\text{L1.insert}}$) proporcionaria resultados enviesados. A

precisão dos relógios da CPU não permite monitorar cada operação individualmente e tal procedimento provavelmente interferiria no desempenho do sistema.

O desempenho do NFD-HCS é estimado através do seguinte modelo matemático:

$$\mathbb{E}[Throughput] = |c|/\mathbb{E}[d] \tag{4.5}$$

onde $\mathbb{E}[d]$ é o tempo médio de leitura de cada chunk, que pode ser estimado como:

$$\mathbb{E}[d] = Pd_{\text{hit}} + (1 - P)d_{\text{miss}} \tag{4.6}$$

onde P é calculado como (4.1), (4.2), ou (4.3), dependendo da carga de trabalho, enquanto d_{hit} e d_{miss} representam o tempo de duração das operações em casos de acerto e falha na L1, respectivamente. Em caso de acerto na L1, o valor é estimado como sendo o tempo necessário para acessar um chunk em L1 (ou seja, encontrar um ponteiro para o chunk em L1 e acessar o local de memória):

$$d_{\text{hit}} = d_{\text{L1.lookup}} + d_{\text{L1.read}} \approx d_{\text{L1.lookup}} \tag{4.7}$$

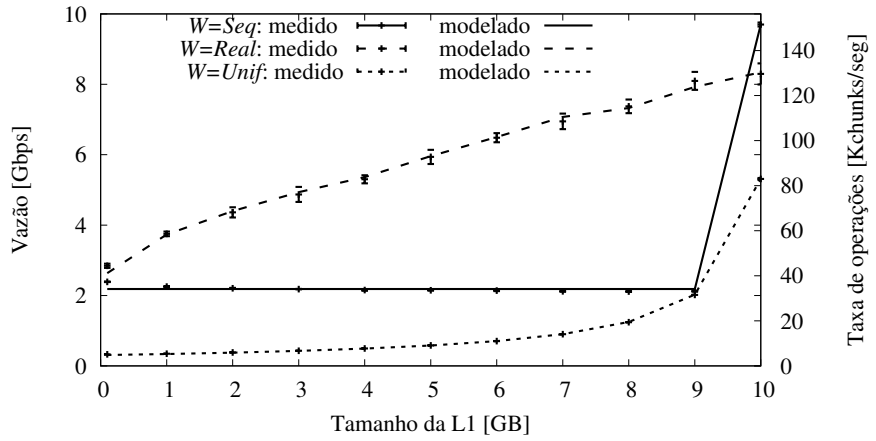
Em caso de falha na L1, o tempo de acesso a um chunk armazenado na L2 do NFD-HCS é estimado pela soma de três termos:

$$d_{\text{miss}} = d_{\text{L1.lookup}} + d_{\text{L2.read}} + d_{\text{L1.insert}} \tag{4.8}$$

O primeiro termo ($d_{\text{L1.lookup}}$) modela o tempo necessário para o NFD-HCS reconhecer que o conteúdo não está na L1. O segundo termo ($d_{\text{L2.read}}$) representa o tempo necessário para ler o conteúdo da L2. O último termo ($d_{\text{L1.insert}}$) modela o tempo necessário para inserir o batch completo na L1.

Utilizando-se a probabilidade estimada de acerto em L1 e a taxa de leitura medida do sistema é possível estimar o tempo consumido pelas operações $d_{L1.lookup}$ e $d_{L1.insert}$. Os resultados dessa estimativa são mostrados na Figura 4.6 e na Tabela 4.6.

Figura 4.6: Taxa de transferência do NFD-HCS: resultados da emulação e da aproximação analítica ($|L2| = 10\,\text{GB}$, $\tau_{L2} = 4\,\text{Gbps}$, $D = Busy$, $M = Static$)



Fonte: o autor

A Tabela 4.6 permite a realização de três observações. Em primeiro lugar, notam-se erros assintóticos pequenos (especialmente para a operação $d_{L1.lookup}$). Em segundo lugar, uma comparação entre as durações $d_{L1.lookup}$ e $d_{L1.insert}$ sugere que a realização de

Tabela 4.6: Taxa de transferência da HCS: comparação analítica entre valores medidos e previstos

Carga de Trabalho (W)	**Variável**	**Valor** (μs)	**Erro assintótico**	**REQM**
Sequencial (*Seq*)	$d_{\text{L1.lookup}}$	$6{,}6 \pm 0{,}06$	$0{,}9\,\%$	$0{,}076$
	$d_{\text{L1.insert}}$	$67{,}2 \pm 3{,}63$	$5{,}4\,\%$	
Real (*Real*)	$d_{\text{L1.lookup}}$	$7{,}4 \pm 0{,}06$	$0{,}9\,\%$	$0{,}098$
	$d_{\text{L1.insert}}$	$42{,}1 \pm 5{,}31$	$12{,}6\,\%$	
Uniforme (*Uni*)	$d^{uni}_{\text{L1.lookup}}$	$12{,}1 \pm 0{,}02$	$0{,}1\,\%$	$0{,}007$
	$d^{uni}_{\text{L1.insert}}$	$37{,}0 \pm 0{,}74$	$1{,}9\,\%$	

Fonte: o autor

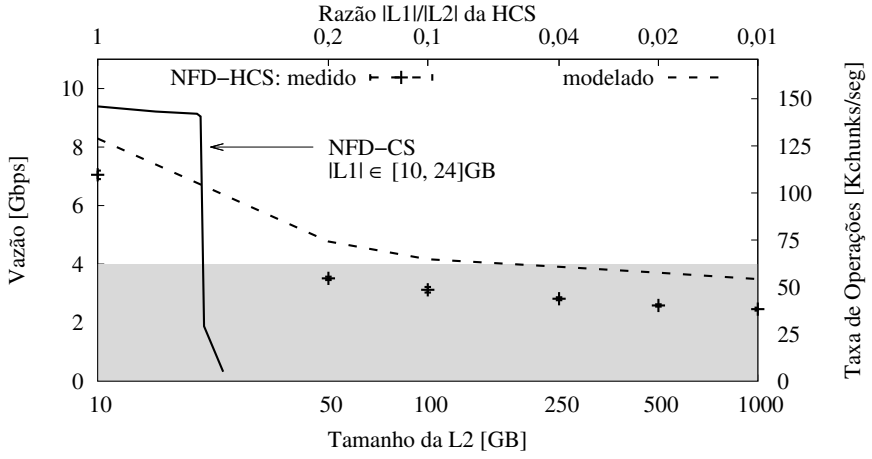
operações consecutivas para chunks de um mesmo batch na estrutura de dados Skiplist proporciona ganhos em termos de gerenciamento de memória. Intuitivamente, os registros de memória utilizados para a inserção do primeiro chunk são úteis para inserção dos chunks subsequentes do batch. Em terceiro lugar, observe-se que a duração de pesquisa $d_{L1.lookup}$ tem ordem de 10 μs, o que não permitiria a sustentação da taxa de operação na ordem de 10 Gbps. Ou seja, a sobrecarga de gerenciamento de memória L1 é cerca de 3 ordens de magnitude maior do que o tempo de acesso à DRAM (ordem de 10 ns). Isso confirma que a indexação da CS pode se tornar um gargalo de software, assim como reportado em (PERINO; VARVELLO, 2011).

4.3.3 Impacto da L2 no Desempenho do Sistema Hierárquico de Única *Thread*

Nesta avaliação, o NFD-HCS foi configurado com $\tau_{L2} = 4$ Gbps, $|L1| = 10$ GB e um intervalo de tamanho de L2 $|L2| \in [10, 1000]$ GB. A alocação de memória dinâmica ($M = Dynamic$) foi adotada para superar as restrições de memória impostas pela plataforma de testes.

A Figura 4.7 mostra o desempenho do sistema NFD-HCS como função do tamanho da L2 (em escala logarítmica). Para fins de referência, o desempenho do NFD-CS (de única camada) também é apresentado.

Figura 4.7: Impacto do tamanho da L2 no desempenho do NFD-HCS ($|L1| = 10$ GB, $\tau_{L2} = 4$ Gbps, $W = Real$, $M = Dynamic$, $D = Busy$)



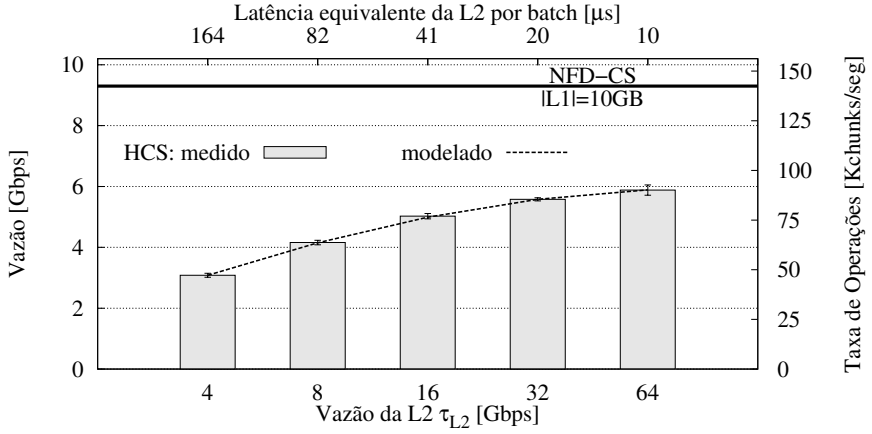
Fonte: o autor

Três observações emergem a partir da Figura 4.7. Em primeiro lugar, observe-se que para o menor tamanho de L2 ($|L2| = 10\,\text{GB}$), o NFD-HCS exibe alguma sobrecarga em relação ao NFD-CS (de apenas uma camada de memória). Essa penalidade é esperada e ainda mais exacerbada pelo uso da técnica de alocação $M = Dynamic$. Em segundo lugar, note-se que o NFD-CS é inviável para tamanhos a partir de cerca de $24\,\text{GB}$, ao passo que o NFD-HCS é viável. Finalmente, observe-se que à medida que o tamanho da L2 aumenta, o desempenho do sistema diminui logaritmicamente, considerando a inclinação linear da curva e a escala logarítmica do eixo horizontal, especialmente a partir de cerca de $50\,\text{GB}$. O decréscimo se explica pelo fato de que como $O = |L2|$, a taxa de acerto em L1 diminui e, consequentemente, o fluxo de solicitações a serem satisfeitas por L2 cresce. Contudo, mesmo para uma L2 de tamanho $1\,\text{TB}$, o rendimento global de NFD-HCS ainda está próximo da taxa de transferência L2 $\tau_{L2} = 4\,\text{Gbps}$, que é a taxa de encaminhamento do NFD tradicional (região sombreada). Portanto, é plausível esperar que uma implementação paralelizada de roteadores ICN munidos de caches hierárquicas com espaço na ordem de $1\,\text{TB}$ sejam capazes de atender a solicitações de caches em taxa de linha.

Para aumentar o desempenho do NFD-HCS, uma opção intuitiva é aumentar a taxa de leitura da L2 (ou seja, explorar a escalabilidade vertical). Os limites tecnológicos poderiam ser superados, por exemplo, empregando-se múltiplas unidades SSD em paralelo. Para avaliar essa hipótese, mediu-se o desempenho do NFD-HCS considerando um tamanho fixo ($|L1| = 1\,\text{GB}$ e $|L2| = 10\,\text{GB}$), diversas taxas de leitura de L2 ($\tau_{L2} \in [1, 64]\,\text{Gbps}$), e a carga de trabalho $W = Real$.

A Figura 4.8 mostra os resultados desse experimento e, adicionalmente, o desempenho do NFD-CS com tamanho $|L1| = 10\,\text{GB}$ para fins de referência. Duas observações são destacadas. Em primeiro lugar, considerando a escala logarítmica do eixo horizontal, a inclinação linear sugere um retorno logarítmico para o desempenho do NFD-HCS como função da taxa de leitura da L2. Em segundo lugar, o desempenho do NFD-HCS não alcança o desempenho do NFD-CS de tamanho similar. Essa diferença deve-se, provavelmente, à existência gargalos de software impostos pelo gerenciamento da segunda camada de memória.

Figura 4.8: Impacto da taxa de transferência da L2 no desempenho do NFD-HCS ($|L1| = 1\,\text{GB}$, $|L2| = 10\,\text{GB}$, $W = Real$, $M = Static$, $D = Busy$)



Fonte: o autor

4.3.4 Discussão

Em resumo, as seguintes lições principais foram aprendidas a partir das avalições apresentadas nesta seção:

- algumas simplificações são necessárias para desenvolver técnicas capazes de emular tecnologias para a camada L2 ora maiores, ora mais rápidas. O conjunto de avaliações das estratégias de emulação permitem identificar cenários onde cada estratégia pode ser confiavelmente aplicada, considerando a plataforma de testes utilizada.

- O desempenho do NFD-HCS foi validado por modelos analíticos. Os modelos analíticos, adicionalmente, permitem compreender características fundamentais do sistema que seriam difíceis de serem medidas diretamente (por exemplo, duração da pesquisa nas estruturas de dados do NFD).

- O sistema hierárquico atende requisitos de espaço, mas não de vazão. O NFD-HCS suporta tamanhos de cache da ordem dos terabytes. Porém, o NFD-HCS não é capaz de atender a taxa de linha necessária, mesmo que possíveis aperfeiçoamentos de hardware (explorando, assim, a escalabilidade vertical, como o aumento da vazão da L2) estivessem disponíveis.

Em suma, os resultados desta seção reforçam a motivação para o desenvolvimento de arquiteturas paralelas (que exploram a escalabilidade horizontal) para unidades de cache hierárquicas em roteadores ICN. As arquiteturas propostas para esse fim são alvo das avaliações apresentadas na próxima seção.

4.4 Sistemas Hierárquicos de Múltiplas *Threads*

Esta seção investiga questões relacionadas aos sistemas NFD-HCS de múltiplas threads. As subseções 4.4.3 e 4.4.2 avaliam as arquiteturas Pipeline e Escalar, respectivamente. Em seguida, a Subseção 4.4.3 refina a avaliação de desempenho da arquitetura Escalar comparando dois esquemas de distribuição da carga de trabalho entre as threads. Por fim, a Subseção 4.4.4 resume as principais conclusões.

4.4.1 Desempenho da Arquitetura Pipeline

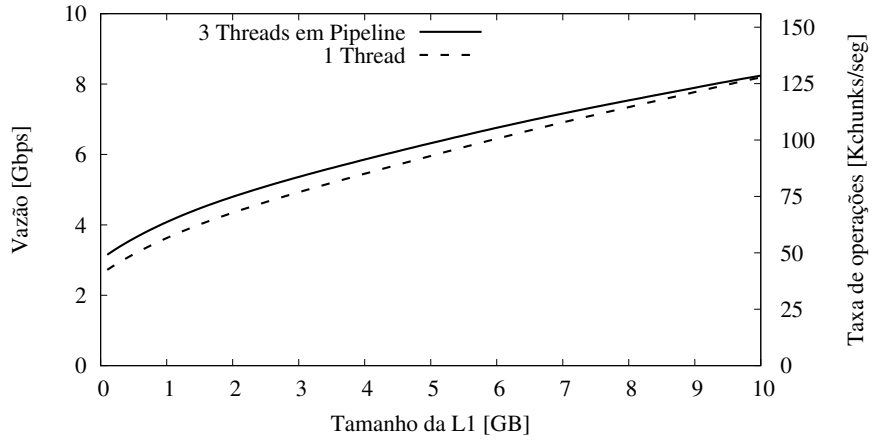
A primeira opção considerada para paralelizar o NFD-HCS é a arquitetura Pipeline. No algoritmo dessa arquitetura, os estágios do caminho crítico ($L1.lookup$, $L1.insert$, $L2.read$) são executados em paralelo. Os ganhos permitidos pelo algoritmo Pipeline em relação ao algoritmo sequencial são avaliados analiticamente através de um modelo matemático, descrito a seguir.

A seção anterior mostra que o desempenho do sistema hierárquico pode ser estimado pela taxa de acerto na camada L1 e os atrasos correspondentes em cada caso (acerto ou falha). Esse modelo de estimativa foi estendido para permitir uma comparação analítica de desempenho dos algoritmos sequencial e Pipeline. No caso de acerto na L1, o atraso médio, denotado por d_{hit}, é o mesmo tanto para o algoritmo sequencial como o Pipeline. Em caso de falha na L1, o atraso depende da abordagem: no caso de um algoritmo sequencial, o atraso d_{miss}^{seq} é dado por (4.8), enquanto que no algoritmo em pipeline, o atraso corresponde ao valor máximo entre as operações:

$$d_{miss}^{par} = max\{d_{L1.lookup}, d_{L1.insert}, d_{L2.read}\} \tag{4.9}$$

A Figura 4.9 apresenta os resultados do modelo para uma arquitetura com três threads organizadas em Pipeline e da avaliação experimental da arquitetura de única thread, considerando a carga de trabalho $W = Real$. É possível observar que, considerando os limites tecnológicos onde um único componente domina os outros, ou seja, $d_{miss}^{par} \approx d_{miss}^{seq} \approx d_{L2.read}$, o ganho real permitido por uma implementação paralela em pipeline é marginal. Assumindo 1 instância apenas de cada estágio, o ganho máximo teórico de $2/3$ pode ser alcançado caso os três componentes possuíssem desempenhos iguais, de modo que a evolução da tecnologia pode motivar uma reavaliação desse aspecto em um grão mais fino.

Figura 4.9: Comparação analítica entre arquiteturas hierárquicas de única thread e hierárquica de três threads organizadas em Pipeline ($|L2| = 10\,\text{GB}$, $\tau_{L2} = 4\,\text{Gbps}$, $W = Real$, $M = Static$, $D = Busy$)



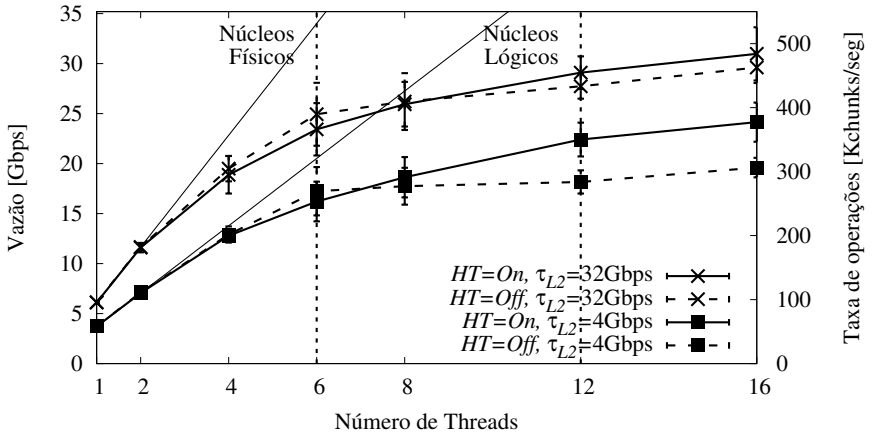
Fonte: o autor

4.4.2 Desempenho da Arquitetura Escalar

A segunda opção considerada para paralelizar o NFD-HCS é a arquitetura Escalar. Para avaliar essa alternativa, usou-se a carga de trabalho $Real$ e mediu-se o desempenho de um conjunto de instâncias do NFD-HCS resultante da combinação dos seguintes parâmetros: $|L1| = 1\,\text{GB}$, $|L2| = 10\,\text{GB}$, função hash $H = Content$, $\tau_{L2} \in \{4, 32\}\,\text{Gbps}$, e múltiplas quantidades de threads no intervalo $R \in [1, 16]$. Além disso, a ativação da opção hyper-threading do sistema subjacente foi variada ($HT \in \{On, Off\}$). Quando o HT está ativado (On), o sistema operacional possui a seu dispor um número de núcleos lógicos que é exatamente o dobro do número de núcleos físicos (opção Off).

A Figura 4.10 apresenta os resultados da avaliação e permite realizar quatro observações. Em primeiro lugar, a arquitetura Escalar oferece ganhos independentemente das propriedades físicas da L2 (embora o efeito do aumento do número de threads seja mais benéfico para sistemas com maior taxa de leitura da L2). Em segundo lugar, conforme esperado, o hyper-threading oferece maiores ganhos e, portanto, deve ser ativado por padrão. Em terceiro lugar, observa-se um joelho na curva onde o número de threads excede o número de núcleos, o que é especialmente visível no sistema mais restrito, ou seja, com a opção hyper-threading desativada e $\tau_{L2} = 4\,\text{Gbps}$.

Figura 4.10: Desempenho do NFD-HCS Escalar, considerando múltiplos graus de paralelismo, taxa de transferência para L2 e opções de hyper-threading ($|L1| = 1\,\text{GB}$, $|L2| = 10\,\text{GB}$, $W = Real$, $M = Static$, $D = Busy$, $H = Content$)



Fonte: o autor

Por fim, é interessante notar que a tendência de ganhos por número de threads apresentada na Figura 4.10 não estabiliza completamente, mesmo quando o número de threads excede o número de núcleos. Isto pode ser explicado por duas razões. Primeiro, o aumento do número de threads incrementa a taxa de leitura agregada da L2, superando gargalos de hardware. Segundo, a divisão da carga de trabalho em tarefas menores dimiui a carga na thread mais carregada, reduzindo assim os gargalos de software.

4.4.3 Impacto do Balanceamento de Carga no Desempenho da Arquitetura Escalar

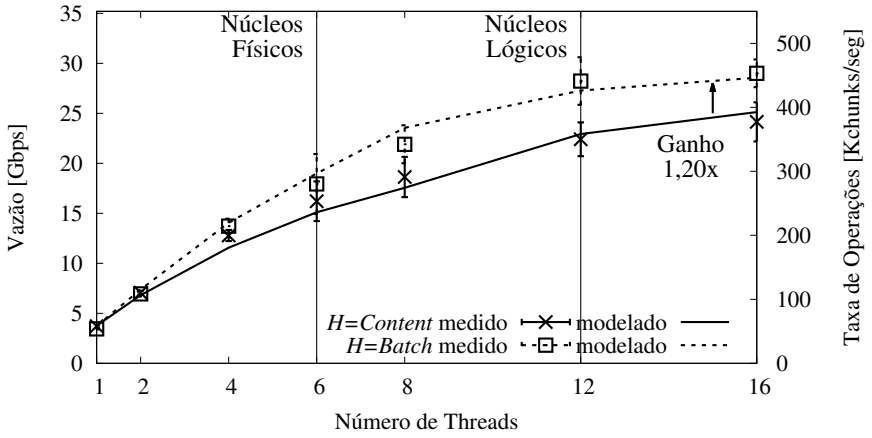
Para avaliar o impacto do balanceamento da carga de trabalho no desempenho da arquitetura Escalar, a carga de trabalho $W = Real$ foi usada para alimentar um conjunto sistemas resultante da combinação dos seguintes valores de parâmetros. Os tamanhos das camadas L1 e L2 foram fixados em $|L1| = 1\,\text{GB}$ e $|L2| = 10\,\text{GB}$, respectivamente. A vazão da L2 foi fixada em $\tau_{L2} = 4\,\text{Gbps}$. O grau de paralelismo foi variado no intervalo $R \in [1, 16]$. A carga de trabalho foi distribuída entre as threads usando os dois esquemas de hash propostos $H \in \{Content, Batch\}$.

Os resultados da avaliação são apresentados na Figura 4.11. Ela contém duas curvas, uma para cada função hash, e mostra o desempenho dos sistemas como função do grau de paralelismo. Como esperado, o desempenho do sistema baseado em função hash $H = Batch$ é superior. Esse comportamento se explica pelo balanceamento de carga, conforme justificado a seguir.

A Figura 4.12 apresenta o balanceamento de carga para quatro graus de paralelismo (R) diferentes: (a) 2 threads, (b) 4 threads, (c) 8 threads e (d) 16 threads. Em geral, observe-se que enquanto a função hash $H = Batch$ proporciona balanceamento próximo do ideal, a função hash $H = Content$, não. Por exemplo, quando $R = 16\,\text{threads}$, a diferença da carga para a thread mais carregada (primeira) entre os casos $H = Content$ e $H = Batch$ é cerca de 3 vezes. A razão para essa diferença está no impacto da distribuição Zipf na função $H = Content$, como explicado a seguir.

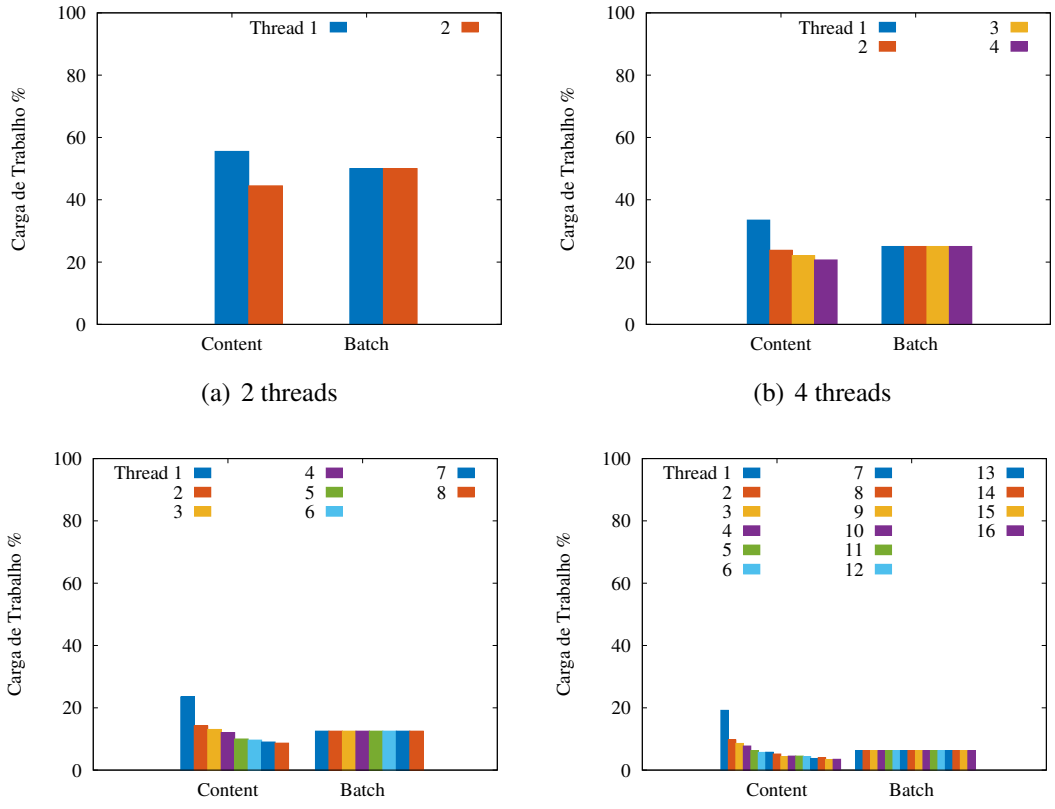
Para verificar a hipótese de que a distribuição Zipf interfere no balanceamento de carga da função hash baseada em conteúdo, foi realizada uma avaliação de propósito específico. Nessa avaliação, a função hash $H = Content$ foi configurada com quatro graus de para-

Figura 4.11: Impacto do balanceamento de carga no desempenho do sistema Escalar ($|L1| = 1\,\text{GB}$, $|L2| = 10\,\text{GB}$, $\tau_{L2} = 4\,\text{Gbps}$, $W = Real$, $M = Static$, $D = Busy$, $HT = On$)



Fonte: o autor

Figura 4.12: Impacto das funções hash ($H \in \{Content, Batch\}$) no balanceamento de carga em sistemas com diferentes graus de paralelismo ($R \in \{2, 4, 8, 16\}$)



(a) 2 threads

(b) 4 threads

(c) 8 threads

(d) 16 threads

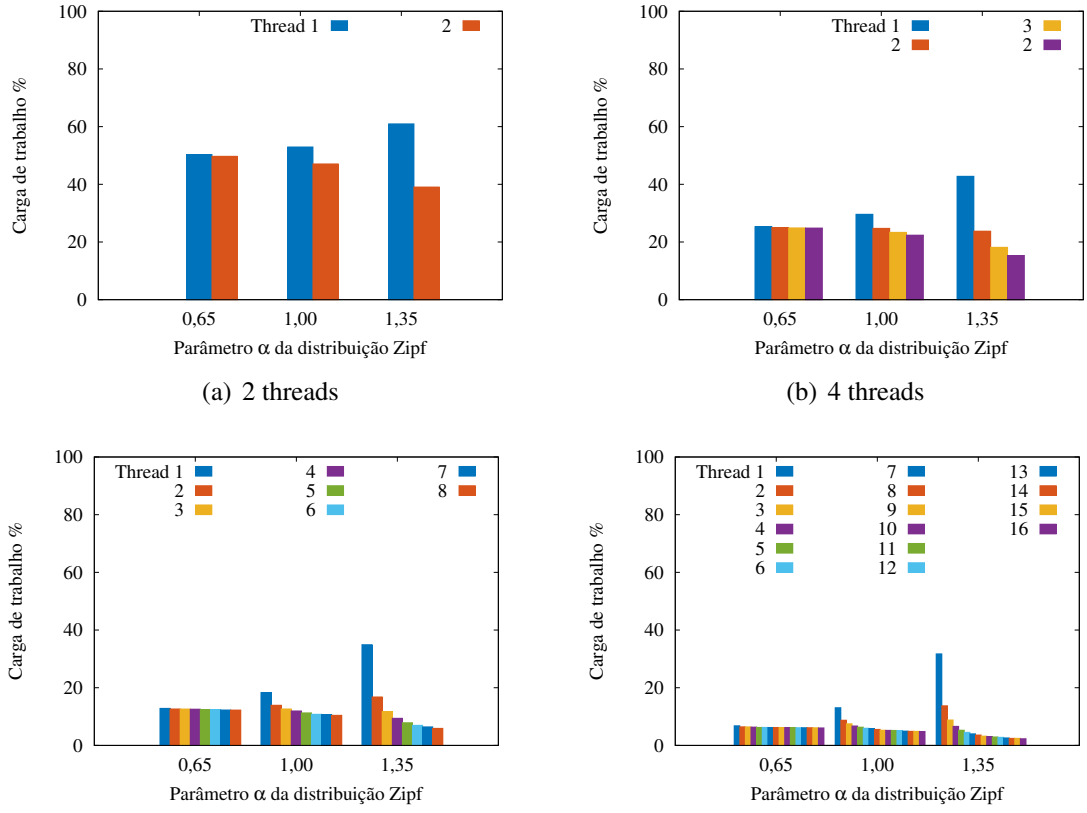
Fonte: o autor

lelismo distintos ($R \in \{2, 4, 8, 16\}$) e alimentada com três cargas de trabalho. As cargas de trabalho possuem catálogo e tamanho iguais $N = O = 1\,\text{TB}$ e são baseadas na carga $Real$ com valores diferentes para o parâmetro alpha $\alpha \in \{0{,}65,\ 1{,}00,\ 1{,}35\}$. O primeiro

valor corresponde ao valor medido do traço ($W = Trace$) e o segundo, ao valor da carga sintética realística ($W = Real$). O terceiro valor foi escolhido como limiar superior e é derivado dos valores anteriores $(1 + (1 - 0{,}65))$.

A Figura 4.13 apresenta o balanceamento de carga para quatro sistemas com diferentes graus de paralelismo $R \in \{2, 4, 8, 16\}$ e a função hash $H = Content$. Como esperado, de maneira geral, o desbalanceamento cresce conforme α aumenta. Note-se que para $\alpha = 1{,}65$, o balanceamento é próximo do ideal para todos os graus de paralelismo apresentados. Além disso, observa-se que o desbalanceamento é mais acentuado no caso onde $\alpha = 1{,}35$ e $R = 16$.

Figura 4.13: Impacto do parâmetro α da distribuição Zipf no balanceamento de carga da função Hash $H = Content$ em quatro sistemas com diferentes graus de paralelismo $R \in \{2, 4, 8, 16\}$



(a) 2 threads

(b) 4 threads

(c) 8 threads

(d) 16 threads

Fonte: o autor

Para fins de aferição, a Tabela 4.7 apresenta a quantidade e o percentual de requisições realizadas para os 16 conteúdos mais populares nas três cargas de trabalho. Os números confirmam quantativamente o impacto do parâmetro α na popularidade dos conteúdos.

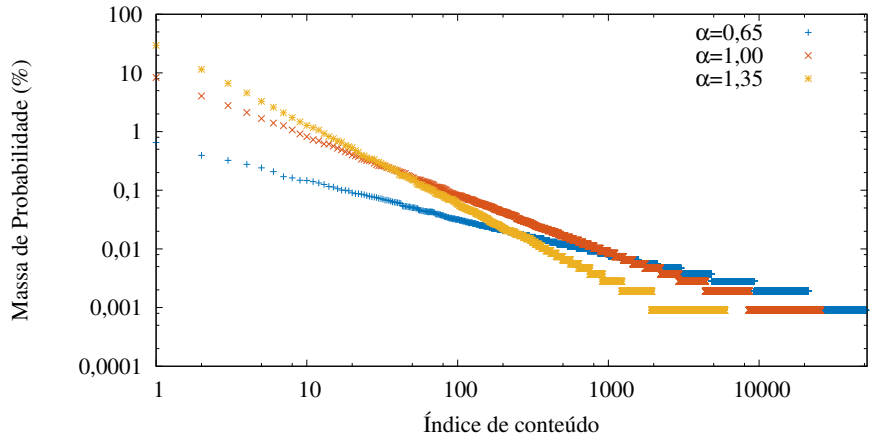
Novamente para fins de aferição, a Figura 4.14 apresenta a função massa de probabilidade para as três cargas de trabalho, em escala logarítmica. Como esperado, os ângulos das curvas crescem conforme α aumenta.

Tabela 4.7: Quantidade e percentual de requisições efetuadas para os 16 conteúdos mais populares para as três cargas de trabalho

Índice	$\alpha = 0{,}65$		$\alpha = 1{,}00$		$\alpha = 1{,}35$	
	# requisições	% requisições	# requisições	% requisições	# requisições	% requisições
1	896000	0,006516	11496960	0,083614	40569600	0,295051
2	541440	0,003938	5556480	0,040411	15792640	0,114855
3	444160	0,003230	3824640	0,027816	9160960	0,066625
4	380160	0,002765	2917120	0,021215	6269440	0,045596
5	331520	0,002411	2289920	0,016654	4501760	0,032740
6	285440	0,002076	1922560	0,013982	3580160	0,026037
7	234240	0,001704	1734400	0,012614	2877440	0,020927
8	222720	0,001620	1470720	0,010696	2383360	0,017334
9	203520	0,001480	1258240	0,009151	2018560	0,014680
10	202240	0,001471	1128960	0,008211	1749760	0,012726
11	193280	0,001406	999680	0,007270	1600000	0,011636
12	183040	0,001331	960000	0,006982	1461760	0,010631
13	172800	0,001257	849920	0,006181	1273600	0,009263
14	160000	0,001164	843520	0,006135	1131520	0,008229
15	156160	0,001136	776960	0,005651	1045760	0,007606
16	145920	0,001061	716800	0,005213	953600	0,006935

Fonte: o autor

Figura 4.14: Impacto do parâmetro α da distribuição Zipf na massa de probabilidade da carga de trabalho sintética realística ($O = 1\,\text{TB}$).



Fonte: o autor

4.4.4 Discussão

As avaliações apresentadas nesta seção permitem extrair as seguintes conclusões principais:

- Embora a arquitetura Pipeline possa gerar benefícios, intuitivamente, eles não compensam o esforço necessário para implementar a arquitetura. Por exemplo, considerando as restrições tecnológicas atuais, o ganho de uma arquitetura Pipeline sobre a implementação sequencial é limitado em cerca de $10\,\%$.

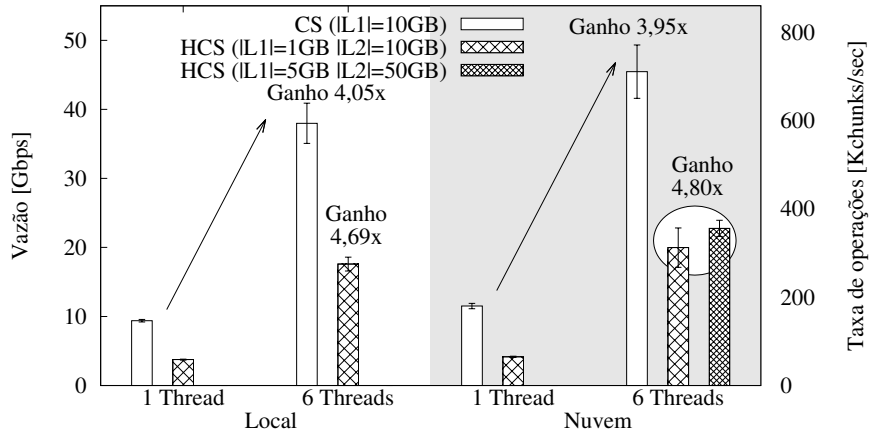
- O rendimento do sistema Escalar exibe um retorno logarítmico em função do grau de paralelismo, até o número dos núcleos (lógicos) disponíveis. Opções de hardware, como o Hyper-threading oferecem ganhos perceptíveis. Opções de software, tais como esquemas de hash que minimizam distorções da distribuição da carga de trabalho, também são desejáveis.

4.5 Impacto do Ambiente de Testes no Desempenho de Sistemas

Esta avaliação verifica a consistência entre os resultados gerados pelas duas plataformas de testes, que (conforme a Tabela 4.1) possuem especificações próximas, porém distintas. A carga de trabalho realística ($W = Real$) foi utilizada para alimentar instâncias do NFD-CS ($|L1| = 1$ GB) e do NFD-HCS ($|L1| = 1$ GB, $|L2| = 10$ GB e $\tau_{L2} = 4$ Gbps), baseados em um sistema de thread única ($R = 1$) e outro sistema com 6 threads ($R = 6$). Na nuvem, foi avaliado, adicionalmente, um NFD-HCS com mesmo grau de paralelismo ($R = 6$), mesma taxa de leitura de L2 $\tau_{L2} = 4$ Gbps e mesma proporção entre tamanhos das camadas $|L1|/|L2|$, porém com tamanhos cinco vezes maiores ($|L1| = 5$ GB e $|L2| = 50$ GB).

Os resultados são apresentados na Figura 4.15. Como esperado, as duas plataformas de testes retornam valores notavelmente próximos tanto em casos de thread única como no caso paralelizado. Esses resultados sugerem que a emulação não é distorcida pela plataforma subjacente de testes e que a comunidade científica pode utilizar a referida nuvem para reproduzir e estender a investigação apresentada nesta tese. Além disso, observa-se que o impacto da multiplicação do tamanho do NFD-HCS em cinco vezes é estatisticamente desprezível. Essa observação demonstra, ainda que de maneira limitada, a escalabilidade do NFD-HCS em termos de tamanho.

Figura 4.15: Medições realizadas em múltiplas configurações de hardware ($\tau_{L2} = 4$ Gbps, $W = Real$, $M = Static$, $D = Busy$, $H = Content$, $HT = Off$)



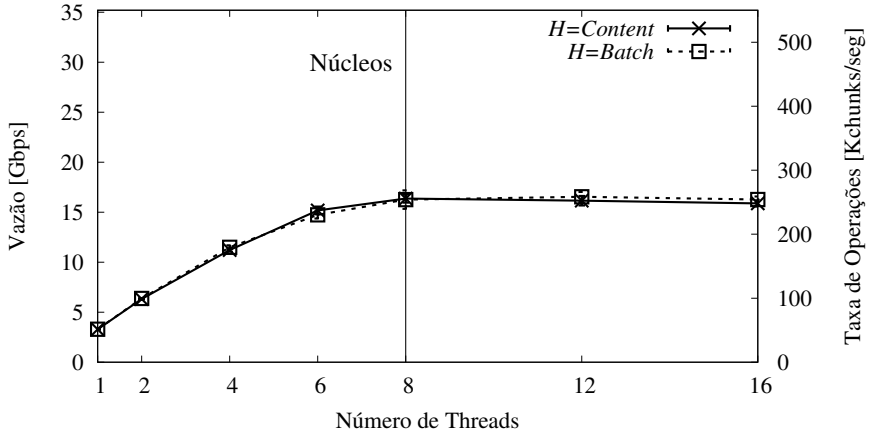
Fonte: o autor

4.6 Estudo de Caso

O objetivo do estudo de caso é verificar o desempenho da arquitetura paralela Escalar em um cenário particularmente fidedigno, ainda que de propósito específico. Para tanto, a carga de trabalho baseada em traço ($W = Trace$) foi utilizada para alimentar um conjunto de sistemas resultante da combinação dos seguintes valores de parâmetros. Os tamanhos das camadas L1 e L2 foram fixados em $|L1| = 1$ GB e $|L2| = 10$ GB, respectivamente. A vazão da L2 foi fixada em $\tau_{L2} = 4$ Gbps. O grau de paralelismo foi variado no intervalo $R \in [1,16]$. A carga de trabalho foi distribuída entre as threads usando os dois esquemas de hash propostos $H \in \{Content, Batch\}$. A estratégia de alocação de memória para L2 dinâmica ($M = Dynamic$) foi adotada para permitir a avaliação de conteúdos com tamanhos distintos. A avaliação foi executada na plataforma de testes alocada na nuvem.

A Figura 4.16 apresenta o desempenho do sistema como função do grau de paralelismo (R) e permite duas observações principais. Primeiro, observe-se que o desempenho se estabiliza mais rapidamente do que na avaliação similar anterior (Figura 4.11) conforme o grau de paralelismo aumenta. Esse fenômeno se explica pela sobrecarga imposta pela alocação de memória da L2 $M = Dynamic$. Mesmo assim, o NFD-HCS Escalar com grau de paralelismo $R = 4$ supera com folga a taxa de linha básica para as bordas da rede (10 Gbps). Segundo, observe-se que o desempenho do sistema é praticamente o mesmo para as duas funções hash. Isso ocorre porque o balanceamento de carga é próximo do ideal mesmo no caso onde $H = Content$, como explicado a seguir.

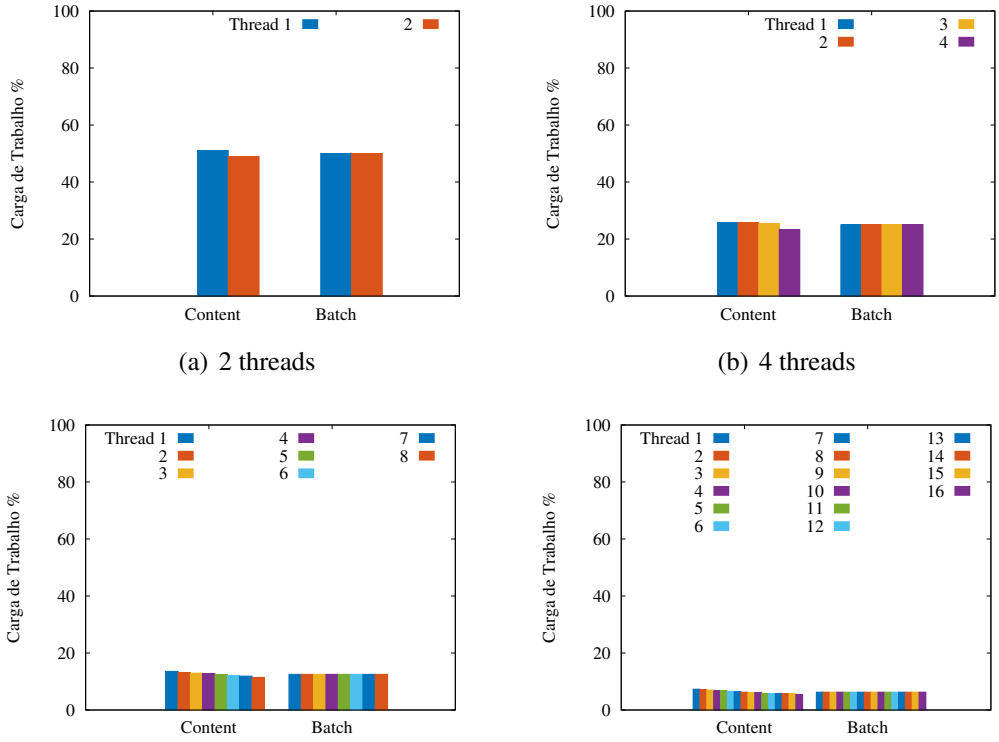
Figura 4.16: Desempenho do NFD-HCS Escalar alimentado com carga de trabalho ($|L1| = 1$ GB, $|L2| = 10$ GB, $\tau_{L2} = 4$ Gbps, $W = Trace$, $M = Static$, $D = Busy$)



Fonte: o autor

A Figura 4.17 mostra o balanceamento de carga os sistemas baseados nas duas funções hash e quatro graus de paralelismo (R) diferentes: (a) 2 threads, (b) 4 threads, (c) 8 threads e (d) 16 threads. Note-se que o balanceamento é próximo do ideal para as duas funções hash. O responsável por esse fenômeno é o parâmetro $\alpha = 0{,}65$ da distribuição Zipf observada no traço (conforme a Tabela 4.4), em contraste com $\alpha = 1{,}0$ da carga de trabalho $W = Real$ (conforme a Tabela 4.3). Portanto, esses resultados estão em acordo com as discussões anteriores sobre impacto do balanceamento de carga no desempenho do sistema e do parâmetro α no balanceamento de carga.

Figura 4.17: Impacto das Funções Hash ($H \in \{Content, Batch\}$) no balanceamento de carga em sistemas com diferentes graus de paralelismo ($R \in \{2, 4, 8, 16\}$)



(a) 2 threads

(b) 4 threads

(c) 8 threads

(d) 16 threads

Fonte: o autor

4.7 Discussão

Por um lado, o escopo limitado do sistema produz resultados que podem ser considerados limiares superiores do desempenho de um sistema completo, já que diversas operações (por exemplo, buscar pacotes de entrada a partir da NIC e encaminhar os pacotes de saída para a NIC) são abstraídas. Por outro lado, a implementação do NFD-HCS oferece muitas oportunidades para melhorias de software (por exemplo, usar uma estrutura de dados melhor do que a Skiplist), considerando que o NFD foi alegadamente projetado com foco em aspectos qualitativos como extensibilidade e facilidade de uso, em detrimento de desempenho.

Contudo, o estudo sobre o paralelismo é ortogonal a esses aspectos, e as conclusões gerais da comparação entre as arquiteturas paralelas propostas são evidentes. Implementar uma arquitetura Pipeline permitirias ganhos pequenos em relação ao desafio de se implementar tal arquitetura de software considerando os requisitos de taxa da linha. A arquitetura paralela Escalar, por sua vez, resulta em maior aceleração por thread ao mesmo tempo que possui arquitetura de software mais simples de ser implementada do que a arquitetura Pipeline.

5 CONSIDERAÇÕES FINAIS

Este capítulo resume as conclusões desta tese de doutorado (Seção 5.1) e motiva trabalhos futuros (Seção 5.2).

5.1 Conclusões

Em resumo, nesta tese, nós investigamos os desafios relacionados ao acoplamento de memórias em HCS. Para superar tais desafios, nós elaboramos a seguinte hipótese:

> *"Uma arquitetura de paralelização que acople as camadas de memória de maneira adequada pode viabilizar unidades de cache hierárquicas suficientemente grandes e rápidas para roteadores ICN."*

Para confirmar essa hipótese, inicialmente, nós propomos duas arquiteturas de paralelização para HCS, denominadas Pipeline e Escalar. Em seguida, nós propomos uma metodologia de avaliação para unidades de cache hierárquicas e as arquiteturas derivadas paralelas propostas.

A metodologia viabilizou três conjuntos de avaliações principais. O primeiro conjunto confirmou as limitações de desempenho das unidades de cache de camada única. O segundo conjunto esclareceu o impacto dos componentes de unidades de cache hierárquicas no desempenho do sistema. O terceiro conjunto de avaliações apresentou evidências sobre as vantagens da arquitetura Escalar em relação à arquitetura Pipeline.

Adicionalmente, nós realizamos duas avaliações complementares. A primeira delas demonstrou que o impacto da plataforma de testes nos resultados gerados pela emulação pode ser considerado negligível. A segunda demonstrou que a arquitetura paralela Pipeline atende os requisitos de desempenho necessários para as bordas de rede de uma ICN, em um cenário baseado em traços reais.

O estudo, que foi resumido acima, apresentado em (MANSILHA et al., 2015, 2017) e detalhado ao longo desta tese, apresenta duas conclusões principais:

Conclusão primária. Quanto ao esquema de paralelização para HCS.

Resumo. A arquitetura Escalar se mostrou factível e eficiente. Primeiramente, a implementação do NFD-HCS demonstrou a viabilidade de se executar múltiplas instâncias de HCS em paralelo sem necessitar sincronização. Em segundo lugar, os resultados das avaliações das arquiteturas paralelas sugerem que a escalabilidade da arquitetura Escalar é maior do que a escalabilidade da arquitetura Pipeline. Por fim, os resultados da avaliação da arquitetura Escalar sugerem que o grau de paralelismo gera ganhos (aproximadamente) lineares no desempenho do sistema, quando o número de threads é menor

ou igual ao número de núcleos do processador (sejam eles físicos ou lógicos). Os ganhos foram confirmados por um estudo de caso.

Conclusão secundária. Quanto à metodologia de avaliação de HCS.

Resumo. A metodologia proposta para avaliar o desempenho de HCS, explorando conjuntamente metodologias de emulação e modelagem analítica, se mostrou um instrumento adequado. A modelagem analítica validou os resultados obtidos pelo NFD-HCS, e a comparação dos resultados obtidos a partir de dois ambientes de testes (local e nuvem) demostrou que a emulação sofre interferência mínima da plataforma de hardware subjacente. Em segundo lugar, a emulação permitiu a investigação de uma gama de cenários muito mais rica do que foi possível pela metodologia baseada em prototipação apresentada em trabalhos relacionados anteriores.

Os resultados foram aferidos por duas metodologias distintas (analítica e emulacional), gerados por duas plataformas de testes e baseados em uma quantidade notável de cenários, além de serem coerentes com trabalhos experimentais relacionados. Portanto, nós esperamos que estas conclusões sirvam como primeiro passo (mesmo que possivelmente pequeno) de uma caminhada a ser percorrida futuramente, conforme discutido na próxima seção.

5.2 Trabalhos Futuros

Em termos de trabalhos futuros, o próximo passo é investigar as operações de inserção de conteúdos na L2. Para tanto, é necessário expandir a metodologia para permitir a avaliação de cenários onde o tamanho do catálogo é maior do que o tamanho da cache. Nesse contexto, são consideradas duas frentes de trabalho, que refletem a organização desta tese.

Quanto ao esquema de inserção on-line na L2 de HCS.

Resumo. Nesta área, propõe-se a exploração de políticas de inserção na L2. Uma abordagem intuitiva é explorar a política de substituição $n - LRU$. Desse modo, apenas os conteúdos realmente populares penalizariam o desempenho do sistema ao serem gravados na camada L2. Dependendo do local de adoção, restrições e orçamento, diferentes parâmetros como grau de LRU podem oferecer resultados mais otimizados.

Quanto à metodologia de avaliação de HCS.

Resumo. A metodologia precisa ser estendida para viabilizar a exploração de cenários onde o tamanho da carga de trabalho é maior que o tamanho da cache. Uma abordagem possível é emular o atraso da rede, de maneira similar à emulação do atraso da L2.

Em um contexto mais amplo, outros possíveis trabalhos incluem a aplicação dos resultados da avaliação e a exploração de outros níveis na hierarquia de memórias. Quanto ao primeiro caso, os resultados das avaliações podem embasar estudos de otimização de alocação de cache considerando novos fatores da memória hierárquica, como custo e relação $|L1|/|L2|$. Quando ao segundo caso, a metodologia poderia ser estendida para investigar a adoção de uma terceira camada de memória, baseada em, por exemplo, tecnologia HDD.

Nós esperamos que a pesquisa desenvolvida nesta tese, bem como trabalhos futuros vislumbrados, possa ajudar a pavimentar um caminho sólido em direção ao desenvol-

vimento e implantação de redes de cache transparentes e ubíquas. No horizonte desse caminho, nós vislumbramos a consolidação de ICN como alternativa para disseminação de conteúdos em larga escala.

REFERÊNCIAS

ABDALLAH, E. G.; HASSANEIN, H. S.; ZULKERNINE, M. A Survey of Security Attacks in Information-Centric Networking. **IEEE Communications Surveys & Tutorials**, Piscataway, EUA, v.17, n.3, p.1441–1454, 2015.

AGGARWAL, A.; ALPERN, B.; CHANDRA, A.; SNIR, M. A model for hierarchical memory. In: ACM ANNUAL SYMPOSIUM ON THEORY OF COMPUTING. **Proceedings** ACM, 1987. p.305–314. (STOC 1987).

AGYAPONG, P. K.; SIRBU, M. Economic incentives in information-centric networking: implications for protocol design and public policy. **IEEE Communications Magazine (ComMag)**, Piscataway, EUA, v.50, n.12, p.18–26, 2012.

AHLGREN, B.; DANNEWITZ, C.; IMBRENDA, C.; KUTSCHER, D.; OHLMAN, B. A survey of information-centric networking. **IEEE Communications Magazine (ComMag)**, Piscataway, EUA, v.50, n.7, p.26–36, 2012.

ARIANFAR, S.; NIKANDER, P. Packet-level Caching for Information-centric Networking. In: ACM SIGCOMM REARCH WORKSHOP, Nova Delhi, Índia. **Proceedings** ACM, 2010.

CAROFIGLIO, G.; GALLO, M.; MUSCARIELLO, L.; PERINO, D. Pending Interest Table Sizing in Named Data Networking. In: ACM INTERNATIONAL CONFERENCE ON INFORMATION-CENTRIC NETWORKING, São Francisco, EUA. **Proceedings** ACM, 2015. (ACM-ICN'15).

CAROFIGLIO, G.; MORABITO, G.; MUSCARIELLO, L.; SOLIS, I.; VARVELLO, M. From content delivery today to information centric networking. **Elsevier Computer Networks (COMNET)**, [S.l.], v.57, n.16, p.3116–3127, novembro 2013.

CHE, H.; TUNG, Y.; WANG, Z. Hierarchical Web caching systems: modeling, design and experimental results. **IEEE Journal on Selected Areas in Communication (JSAC)**, Piscataway, EUA, v.20, n.7, p.1305–1314, setembro 2002.

CHE, H.; WANG, Z.; TUNG, Y. Analysis and design of hierarchical Web caching systems. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATIONS, Anchorage, EUA. **Proceedings** IEEE, 2001. p.1416–1424. (INFOCOM 2001).

CISCO. **Cisco Visual Networking Index: forecast and methodology, 2015-2020**. Disponível em: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf >. Acesso em: 01/10/2016.

CISCO. **Forecast Widget**. Disponível em: <http://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-widget/forecast-widget/index.html >. Acesso em: 02/01/2016.

Corsair. Disponível em: <http://www.corsair.com >. Acesso em: 01/02/2017.

DAI, H.; LIU, B. CONSERT: constructing optimal name-based routing tables. **Elsevier Computer Networks (COMNET)**, Amsterdam, Holanda, v.94, p.62–79, janeiro 2016.

DANNEWITZ, C.; KUTSCHER, D.; OHLMAN, B.; FARRELL, S.; AHLGREN, B.; KARL, H. Network of information (NetInf) - an information-centric network. **Elsevier Computer Communications (COMCOM)**, Amsterdam, Holanda, v.36, n.7, p.721–735, abril 2013.

FOTIOU, N.; NIKANDER, P.; TROSSEN, D.; POLYZOS, G. C. Developing information networking further: from PSIRP to PURSUIT. In: INTERNATIONAL ICST CONFERENCE, Atenas, Grécia. **Proceedings** Springer, 2012. p.1–13. (BROADNETS 2010).

FRICKER, C.; ROBERT, P.; ROBERTS, J. A versatile and accurate approximation for LRU cache performance. In: INTERNATIONAL TELETRAFFIC CONGRESS, Cracóvia, Polônia. **Proceedings** ITC, 2012. p.8:1–8:8. (ITC 2012).

FRICKER, C.; ROBERT, P.; ROBERTS, J.; SBIHI, N. Impact of traffic mix on caching performance in a content-centric network. In: IEEE WORKSHOP ON EMERGING DESIGN CHOICES IN NAME-ORIENTED NETWORKING, Orlando, EUA. **Proceedings** IEEE, 2012. p.310–315. (NOMEN 2012).

GHODSI, A.; SHENKER, S.; KOPONEN, T.; SINGLA, A.; RAGHAVAN, B.; WILCOX, J. Information-centric Networking: seeing the forest for the trees. In: ACM WORKSHOP ON HOT TOPICS IN NETWORKS, Cambridge, EUA. **Proceedings** ACM, 2011. p.1:1–1:6. (HotNets-X 2011).

HASEGAWA, T.; NAKAI, Y.; OHSUGI, K.; TAKEMASA, J.; KOIZUMI, Y.; PSARAS, I. Empirically modeling how a multicore software ICN router and an ICN network consume power. In: ACM INTERNATIONAL CONFERENCE ON INFORMATION-CENTRIC NETWORKING, Paris, França. **Proceedings** ACM, 2014. p.157–166. (ACM-ICN'14).

ICNRG. **CCN/NDN convergence effort**. Disponível em: <https://trac.tools.ietf.org/group/irtf/trac/wiki/icnrg/convergence >. Acesso em: 12/09/2016.

IRCache. **IRCache Traces**. Disponível em: <ftp://ircache.net/Traces/DITL-2007-01-09/ >. Acesso em: 03/01/2016.

JACOBSON, V.; SMETTERS, D. K.; THORNTON, J. D.; PLASS, M. F.; BRIGGS, N. H.; BRAYNARD, R. L. Networking Named Content. In: INTERNATIONAL CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES, 5., Roma, Itália. **Proceedings** ACM, 2009. p.1–12. (CoNEXT'09).

JAMSHED, M. A.; LEE, J.; MOON, S.; YUN, I.; KIM, D.; LEE, S.; YI, Y.; PARK, K. Kargus: a highly-scalable software-based intrusion detection system. In: ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, Raleigh, EUA. **Proceedings** ACM, 2012. p.317–328. (CCS'12).

KATSAROS, K.; XYLOMENOS, G.; POLYZOS, G. C. MultiCache: an overlay architecture for information-centric networking. **Computer Netwtorks (COMNET)**, Amsterdam, Holanda, v.55, n.4, p.936–947, março 2011.

KIM, N.; CHOI, G.; CHOI, J. A scalable carrier-grade DPI system architecture using synchronization of flow information. **IEEE Journal on Selected Areas in Communications (JSAC)**, Piscataway, EUA, v.32, n.10, p.1834–1848, setembro 2014.

KIM, T.; KIM, E.-J. Hybrid storage-based caching strategy for content delivery network services. **Springer Multimedia Tools and Applications**, Berlim, Alemanha, v.74, n.5, p.1697–1709, março 2015.

KIRCHNER, D.; FERDOUS, R.; CIGNO, R. L.; MACCARI, L.; GALLO, M.; PERINO, D.; SAINO, L. Augustus: a CCN router for programmable networks. In: ACM INTERNATIONAL CONFERENCE ON INFORMATION-CENTRIC NETWORKING, Kyoto, Japão. **Proceedings** ACM, 2016. p.31–39. (ACM-ICN'16).

KOPONEN, T.; CHAWLA, M.; CHUN, B.-G.; ERMOLINSKIY, A.; KIM, K. H.; SHENKER, S.; STOICA, I. A Data-oriented (and Beyond) Network Architecture. **SIGCOMM Computer Communication Review**, Nova Iorque, EUA, v.37, n.4, p.181–192, agosto 2007.

LAOUTARIS, N.; SYNTILA, S.; STAVRAKAKIS, I. Meta algorithms for hierarchical Web caches. In: IEEE INTERNATIONAL CONFERENCE ON PERFORMANCE, COMPUTING, AND COMMUNICATIONS, Phoenix, EUA. **Proceedings** IEEE, 2004. p.445–452.

LAOUTARIS, N.; ZISSIMOPOULOS, V.; STAVRAKAKIS, I. On the optimization of storage capacity allocation for content distribution. **Elsevier Computer Networks (COMNET)**, Amsterdam, Holanda, v.47, n.3, p.409–428, fevereiro 2005.

Linux User's Manual. **Time - a simple command or give resource usage**. Disponível em: <http://man7.org/linux/man-pages/man1/time.1.html >. Acesso em: 03/06/2016.

MANSILHA, R. B.; BARCELLOS, M. P.; LEONARDI, E.; ROSSI, D. Exploiting parallelism in hierarchical content stores for high-speed ICN routers. **Elsevier Computer Networks (COMNET)**, Amsterdam, Holanda, 2017. DOI http://dx.doi.org/10.1016/j.comnet.2017.04.041.

MANSILHA, R. B.; SAINO, L.; BARCELLOS, M. P.; GALLO, M.; LEONARDI, E.; PERINO, D.; ROSSI, D. Hierarchical content stores in high-speed ICN Routers: emulation and prototype implementation. In: ACM INTERNATIONAL CONFERENCE ON INFORMATION-CENTRIC NETWORKING, São Francisco, EUA. **Proceedings** ACM, 2015. p.59–68. (ACM-ICN'15). http://doi.acm.org/10.1145/2810156.2810159.

MARCHAL, X.; CHOLEZ, T.; FESTOR, O. Server-side Performance Evaluation of NDN. In: ACM INTERNATIONAL CONFERENCE ON INFORMATION-CENTRIC NETWORKING, Kyoto, Japão. **Proceedings** ACM, 2016. (ACM-ICN'16).

MARTINA, V.; GARETTO, M.; LEONARDI, E. A unified approach to the performance analysis of caching systems. In: IEEE CONFERENCE ON COMPUTER COMMUNICATIONS, Toronto, Canada. **Proceedings** IEEE, 2014. p.2040–2048. (INFOCOM 2014).

NDN Project Team. **NFD – Named Data Networking Forwarding Daemon**. Disponível em: <http://named-data.net/doc/NFD>. Acesso em: 01/02/2016.

PAPALINI, M.; KHAZAEI, K.; CARZANIGA, A.; ROGORA, D. High throughput forwarding for ICN with descriptors and locators. In: SYMPOSIUM ON ARCHITECTURES FOR NETWORKING AND COMMUNICATIONS SYSTEMS, Santa Clara, EUA. **Proceedings** ACM, 2016. p.43–54. (ANCS'16).

PARC. **CCNx**. Disponível em: <http://www.ccnx.org>. Acesso em: 01/02/2016.

PASSARELLA, A. A survey on content-centric technologies for the current Internet: CDN and P2P solutions. **Elsevier Computer Communications (COMCOM)**, Amsterdam, Holanda, v.35, n.1, p.1–32, janeiro 2012.

PENTIKOUSIS, K.; OHLMAN, B.; DAVIES, E.; SPIROU, S.; BOGGIA, G.; MAHADEVAN, P. **Information-centric Networking: evaluation methodology**. Disponível em: <https://tools.ietf.org/html/draft-irtf-icnrg-evaluation-methodology-05>. Acesso em: 24/06/2016. 1–32p.

PERINO, D.; VARVELLO, M. A reality check for content centric networking. In: ACM SIGCOMM WORKSHOP ON INFORMATION-CENTRIC NETWORKING, Toronto, Canada. **Proceedings** ACM, 2011. p.44–49. (ACM-ICN'11).

ROSSINI, G.; ROSSI, D.; GARETTO, M.; LEONARDI, E. Multi-Terabyte and multi-Gbps information centric routers. In: IEEE CONFERENCE ON COMPUTER COMMUNICATIONS, Toronto, Canada. **Proceedings** IEEE, 2014. p.181–189. (INFOCOM 2014).

SHI, J.; LIANG, T.; WU, H.; LIU, B.; ZHANG, B. NDN-NIC: name-based filtering on network interface card. In: ACM INTERNATIONAL CONFERENCE ON INFORMATION-CENTRIC NETWORKING, Kyoto, Japão. **Proceedings** ACM, 2016. p.40–49. (ACM-ICN'16).

SO, W.; CHUNG, T.; YUAN, H.; ORAN, D.; STAPP, M. Toward terabyte-scale caching with SSD in a Named Data networking router. In: ACM/IEEE SYMPOSIUM ON ARCHITECTURES FOR NETWORKING AND COMMUNICATIONS SYSTEMS, Los Angeles, EUA. **Proceedings** ACM, 2014. p.241–242. (ANCS'14, Poster session).

SO, W.; NARAYANAN, A.; ORAN, D. Named Data Networking on a router: fast and DoS-resistant forwarding with hash tables. In: ACM/IEEE SYMPOSIUM ON ARCHITECTURES FOR NETWORKING AND COMMUNICATIONS SYSTEMS, San Jose, EUA. **Proceedings** IEEE Press, 2013. p.215–226. (ANCS'13).

SO, W.; NARAYANAN, A.; ORAN, D.; STAPP, M. Named Data Networking on a Router: forwarding at 20Gbps and beyond. In: ACM SIGCOMM CONFERENCE, Hong Kong, China. **Proceedings** ACM, 2013. p.495–496. (SIGCOMM 2013, Poster session).

SONG, T.; YUAN, H.; CROWLEY, P.; ZHANG, B. Scalable Name-Based Packet Forwarding: from millions to billions. In: ACM INTERNATIONAL CONFERENCE ON INFORMATION-CENTRIC NETWORKING, São Francisco, EUA. **Proceedings** ACM, 2015. (ACM-ICN'15).

Sony Oficial News. **Sony Pictures Launching ULTRA 4K Streaming Service on April 4TH**. Disponível em: <https://blog.sony.com/press/sony-pictures-launching-ultra-4k-streaming-service-on-april-4th-2/ >. Acesso em: 08/09/2016.

TANIGUCHI, K.; TAKEMASA, J.; KOIZUMI, Y.; HASEGAWA, T. A method for designing high-speed software NDN routers. In: ACM CONFERENCE ON INFORMATION-CENTRIC NETWORKING, Kyoto, Japão. **Proceedings** ACM, 2016. p.203–204. (ACM ICN'16, Poster Session).

The Guardian. **PlayStation 4 Pro review – powerful, impressive and yet to really come into its own**. Disponível em: <https://www.theguardian.com/technology/2016/nov/10/playstation-4-pro-review-impressive-performance >. Acesso em: 04/01/2017.

Tom's Hardware. **SSD Best Picks**. Disponível em: <http://www.tomshardware.com/reviews/ssd-recommendation-benchmark,3269.html >. Acesso em: 01/02/2016.

TROSSEN, D.; KOSTOPOULOS, A. Techno-economic aspects of information-centric networking. **Penn State University Press Journal of Information Policy**, Pennsylvania, EUA, v.2, p.26–50, março 2012.

TYSON, G.; SASTRY, N.; CUEVAS, R.; RIMAC, I.; MAUTHE, A. A survey of mobility in information-centric networks. **Communications of the ACM**, Nova Iorque, EUA, v.56, n.12, p.90–98, dezembro 2013.

VARVELLO, M.; PERINO, D.; LINGUAGLOSSA, L. On the design and implementation of a wire-speed Pending Interest Table. In: IEEE INFOCOM WORKSHOP ON EMERGING DESIGN CHOICES IN NAME-ORIENTED NETWORKING, Turim, Itália. **Proceedings** IEEE, 2013. p.1–6. (NOMEN 2013).

WANG, Y.; LI, Z.; TYSON, G.; UHLIG, S.; XIE, G. Design and evaluation of the optimal cache allocation for content-centric networking. **IEEE Transactions on Computers**, Los Alamitos, EUA, v.65, n.1, p.95–107, janeiro 2016.

WOLFF, R. W. Poisson Arrivals See Time Averages. **INFORMS Operations Research**, Catonsville, EUA, v.30, n.2, p.223–231, 1982.

YI, C.; AFANASYEV, A.; MOISEENKO, I.; WANG, L.; ZHANG, B.; ZHANG, L. A case for stateful forwarding plane. **Elsevier Computer Communications**, Amsterdam, Holanda, v.36, n.7, p.779–791, abril 2013.

YOU, W.; MATHIEU, B.; TRUONG, P.; PELTIER, J.-F.; SIMON, G. Realistic storage of pending requests in content-centric network routers. In: IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS IN CHINA, Beijing, China. **Proceedings** IEEE, 2012. (ICCC'12).

YOU, W.; MATHIEU, B.; TRUONG, P.; PELTIER, J.-F.; SIMON, G. DiPIT: a distributed Bloom-filter based PIT table for CCN nodes. In: INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATION NETWORKS. **Proceedings** IEEE, 2012. p.1–7. (ICCCN 2012).

ZERFOS, P.; SRIVATSA, M.; YU, H.; DENNERLINE, D.; FRANKE, H.; AGRAWAL, D. Platform and applications for massive-scale streaming network analytics. **IBM Journal of Research and Development**, Nova Iorque, EUA, v.57, n.3/4, p.11:1–11:13, maio 2013.

ZHANG, G.; LI, Y.; LIN, T. Caching in information centric networking: a survey. **Elsevier Computer Networks (COMNET)**, Amsterdam, Holanda, v.57, n.16, p.3128–3141, novembro 2013.

ZHANG, L.; ESTRIN, D.; BURKE, J.; JACOBSON, V.; THORNTON, J.; SMETTERS, D.; ZHANG, B.; TSUDIK, G.; MASSEY, D.; PAPADOPOULOS, C. **Named data networking (NDN) project**. Disponível em: <http://named-data.net/techreport/TR001ndn-proj.pdf >. Acesso em: 01/06/2015.